로봇으로 쉽게 배우는

C 프로그래밍

C PROGRAMMING

- Windows GCC 컴파일러를 활용한 C 언어 익히기
- C 언어를 활용하여 로봇 제어하기

동부로봇 자료실

(매뉴얼 및 어플리케이션을 다운로드 받으십시오.)

www.dongburobot.com

Dongbu Robot

머릿말

로봇 제작과정을 간단히 요약하면 로봇 기획 → 로봇 디자인 → 기구설계 → 하드웨어설계 → 소프트웨어 프로그래밍으로 최종 마무리합니다. 본 교재는 소프트웨어 파트입니다. 로봇을제어할 수 있는 소프트웨어는 가장 기본적으로 펌웨어에 속하는 C 언어와 어플리케이션 파트에서는 C++ 을 활용하는 Visual C++, 로봇전용 스크립트 언어인 MSRDS 등이 있고,최근에는Java 를 활용한 Android 프로그래밍도 로봇 프로그래밍 언어로 각광받고 있습니다.

로봇에는 지능형 로봇과 산업용 로봇으로 크게 구분되는데, 여기에서 다루고자 하는 소프트웨어는 지능형 로봇의 소프트웨어입니다. 하드웨어 설계까지 끝나면 로봇의 두뇌역할을 하는 제어기의 펌웨어를 작성해야합니다. 그 펌웨어를 거의 대부분 C 언어로 구현하게 됩니다. 그래서 로봇에서 C 프로그래밍은 무엇보다도 중요하다고 할 수 있습니다. 하지만 본 교재로는 펌웨어를 습득할 수 없습니다.. 펌웨어를 익히려면 C 언어는 필수로 익숙해야하고, ATMega 128 등의 마이크로컨트롤러에 대한 지식이 있어야 합니다. 본 교재는 펌웨어중에 C 언어를 익숙하게 다루는데 촛점을 맞췄습니다.

로봇뿐 아니라 C 언어는 모든 프로그래머 입문자에게는 기초 언어로 습득한 후 다른 어플리케이션 프로그래밍을 익히게 됩니다. 본 C 언어 교재는 로봇 입문자와 C 프로그래밍 입문자에게 적합합니다. 초급 C 언어 기초는 시중에 많은 C 언어책를 보신 후, 약간의 기초를 갖추었다면 이 책을 통해 바로 로봇을 제어할 수 있는 프로그래밍이 가능합니다.

본 책의 구성은 C 문법별 예제 –〉 문법 요약 –〉 예제 프로그래밍 계획 및 설계 –〉 예제 프로그래밍 작성 –〉 컴파일/실행 순으로 진행될 것입니다. 여기에 쓰이는 C 언어 편집기는 java 와 Linux 에서 많이 활용되는 무료 배포 편집기인 Eclips 에서 작성을 하고, Windows GCC 컴파일러에서 컴파일합니다.

사실 로봇에서 쓰이는 C 언어라는 개념은 펌웨어를 가르킵니다. 하지만, 펌웨어는 입문자가 익히기에는 하드웨어 지식이 많이 필요한 것이 사실입니다. 따라서 펌웨어는 이 교재에서 C 언어가 익숙해진 후에 하드웨어 공부를 하고 본격적으로 펌웨어 프로그래밍을 하는 방법을 추천합니다. 여기에서 제공하는 함수와 예제만으로도 C 언어를 통한 휴머노이드 제어는 무난하게 진행할 수 있고, 또한 C 언어를 정리하는데 도움이 될 것입니다. 휴머노이드 제어기 펌웨어를 더 깊이 학습하고 싶은 분은 동부로봇에서 편찬한 "로봇으로 배우는 AVR C 프로그래밍"을 통하여 로봇 펌웨어 학습하기를 추천합니다.
감사합니다.

목차

01 C 언어 개요

컴퓨터는 저장과 계산을 할 수 있는 일종의 지능적인 기계입니다. 그렇다면 그러한 지능적인 기계를 활용하기 위해선 기계와 인간이 의사소통을 할 수 있어야 할 것입니다. 그 의사소통을 가능하게 한 것이 바로 프로그래밍입니다. 프로그래밍 중에서도 C 언어는 가장 기초적인 프로그래밍 언어라고 볼 수 있습니다. 해서 모든 프로그래머는 입문시 C 언어를 필수로 습득하게 됩니다. C 언어를 간단히 요약하면, 메모리에 데이터를 저장하는 변수,포인터,구조체 등과 흐름과 로직을 구성하는 분기문과 반복, 기능을 정의하는 함수로 구성되어있습니다. 본 교재는 사용자가 아주 기초적인 문법은 습득하였다는 것을 가정하고, 각 문법별로 맞는 로봇 예제를 간추려 로봇에 응용하는데 촛점을 맞추었습니다. 아래 로봇 예제를 통해 C 언어의 전체적인 윤곽을 살펴봅니다.

1.1 C 언어 훑어보기

아래 예제는 전체적인 C 언어의 개요를 설명하기 위한 로봇프로그래밍입니다.

hhello.c

```
/*
파일 명              : hhello.c
제어로봇 형태      : 16dof 휴머노이드
설명 : C 언어 전반적인 설명을 위한 예제로 전처리기, 변수, 반복, 분기, 함수호출이 포함
되어 로봇의 한 팔을 들어올리는 예제. (motor 0번을 789 값을 넣어주면 됨)
*/
#include 〈stdio.h〉
#include "hovis.h"          // (1) 전처리기 – 선행처리

int main()              //(2) 함수 – 기능정의
{
        int nResult;

        int i;          //(3) 변수 – 메모리관리 –〉 포인터



        nResult = initialize(L"COM2", 115200);
        if(nResult == 0){               // (4) 조건분기 – 선택 실행
```

```
                printf("Initialization Failed₩n");
                fflush(stdout);
                return 0;

        }


        for(i=0;i<16;i++){          // (5) 반복
                g_fMotorPos[i]=0;

        }
        g_fMotorPos[0] = -90;

        g_fMotorPos[1] = 90;

        g_fMotorPos[3] = -90;

        g_fMotorPos[4] = 90;

        run(1000);
        delay(1000);



        g_fMotorPos[0] = 90;

        run(1000);

        delay(1000);



        goto _LABEL_1;

_LABEL_0:

        printf("hello, c!₩n");
        fflush(stdout);
_LABEL_1:

        printf("hello, hovis!₩n");
        fflush(stdout);

        terminate();

        return 0;
}
```

위 예제는 로봇 손을 드는 가장 간단한 동작을 실행한 예제입니다. 위 예제를 통해서 C 프로그래밍 문법이 어떻게 구성되는지 간략히 요약되어있습니다.

C 언어는 크게 두 가지요소를 가집니다. 바로 변수와 함수입니다.
(3) 변수는 데이터를 저장하고자 하는 목적으로 메모리를 관리하고, 그 메모리는 각각 주소가 존재하는데, 그 주소를 저장하는 것을 포인터라고 부릅니다. 포인터 또한 변수의 일종입니다. 흔히 C 언어하면 포인터에 대한 난해성이 많은 언어로 규정짓기도 합니다.

(2) 함수는 동작시키는 기능을 말하는 것으로서, 프로그래밍은 함수들의 결합이라고 말할 수 있습니다.
프로그래밍을 하는 가장 큰 이유는 컴퓨터가 인간이 할 수 있는 계산 능력을 훨씬 뛰어넘기 때문입니다. 그것을 가장 잘 활용하는 것이 바로, 제어와 분기와 반복입니다.
(4) 조건분기와 (5) 반복은 C 언어뿐 아니라, 모든 프로그래밍 언어의 문법이 거의 동일하다고 보면 됩니다.
(1) 전처리기는 프로그래밍이 시작되면 모든 함수와 변수를 선행처리하는 기능을 가지고 있습니다.

본 예제는 이클립스와 컴파일러를 설치한 후 실행하면서 다시 설명하겠습니다.

◆ C 언어 요약

C 프로그래밍 언어는 프로그램의 흐름을 제어하는 플로우 와 메모리에 데이터를 관리하는 메모리 파트, 그리고 기능을 부여하는 함수로 구성됩니다.

프로그램 작성 및 실행순서는 아래와 같습니다.

1) 프로그램 작성 합니다.
2) 컴파일 합니다.
3) 링크 : 연결합니다.
4) 실행파일이 생성됩니다.

1.2 이클립스와 MinG 설치

본 교재는 이클립스라는 언어편집기와 GCC 컴파일러를 활용하여 컴파일을 합니다. 보통은 Microsoft 사의 Visual C++을 활용하여 편집과 컴파일을 합니다. 하지만 Visual C++이 무료배포 Studio가 아니기 때문에 본 교재의 기본툴은 이클립스와 GCC를 활용할 예정입니다. 본 예제는 Visual C++에서도 호환이 가능합니다.

[1.2.1] 개발환경 이클립스

01 이클립스 다운로드 사이트

http://www.eclipse.org/downloads/ 인터넷 주소창에 주소를 입력합니다.

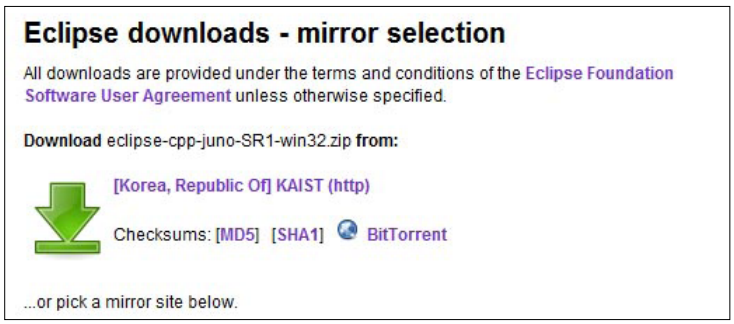
02 다운로드

Eclipse IDE for C/C++ Developers 를 클릭합니다.

03 다운로드2

컴퓨터 사양에 따라 우측에 32bit 나 64bit 를 클릭합니다/

04 다운로드3

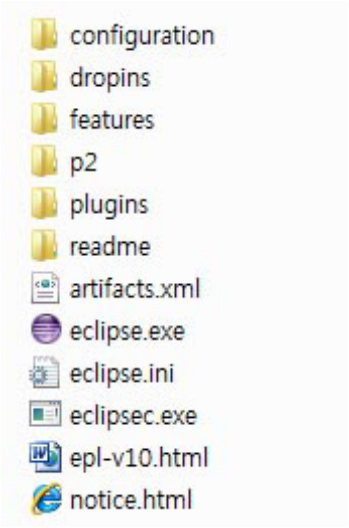


다운로드 화살표를 클릭하여 다운로드 받습니다.

05 다운로드4

저장 버튼을 클릭한 후, 원하는 폴더에 다운로드합니다

06 이클립스 폴더



zip 을 풀면 위와 같은 이클립스 폴더가 생성됩니다.
파일 중 eclipse.exe 를 클릭하면 eclips 가 실행됩니다.

[1.2.2] 컴파일러 GCC

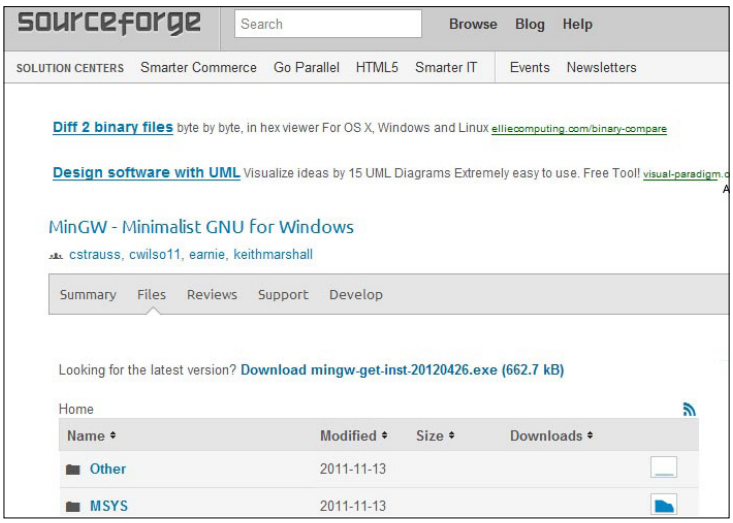
01 컴파일러 사이트

http://www.mingw.org/ 인터넷 주소창에 주소를 입력합니다.

02 다운로드 클릭

Navigation 의 Downloads 를 클릭합니다.

03 Source Forge 에서 다운로드



http://sourceforge.net/projects/mingw/files/ 로 넘어갑니다.

Sourceforge 사이트에서 Download mingw-get...exe 파일을 클릭합니다.

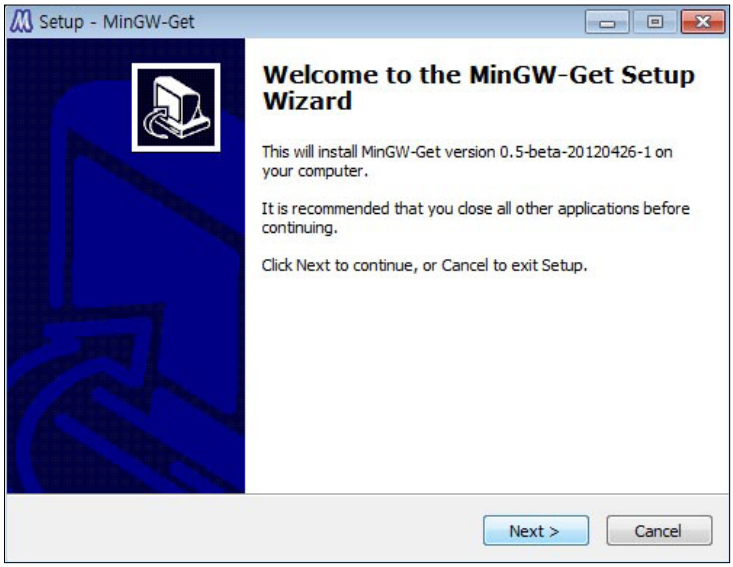
04 다운로드2

저장 버튼을 누르고 원하는 폴더에 저장합니다.

05 실행

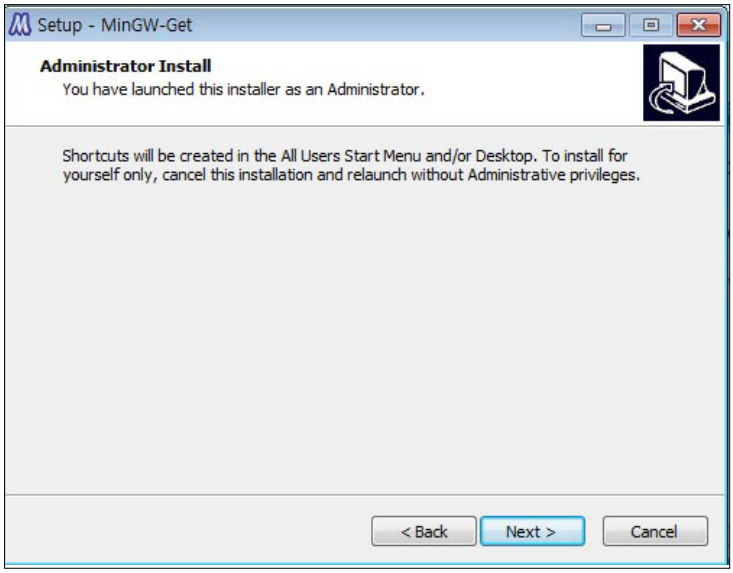
mingw 실행파일을 클릭합니다.

06 실행1



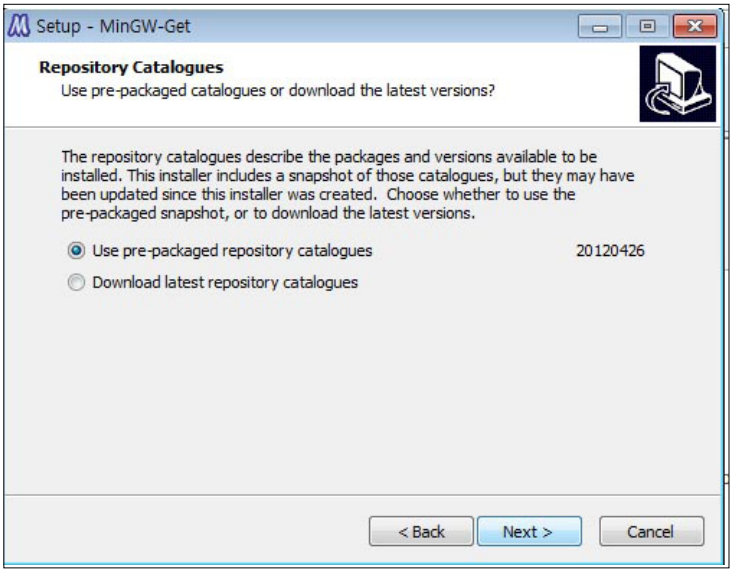
Next 를 클릭합니다.

07 실행2



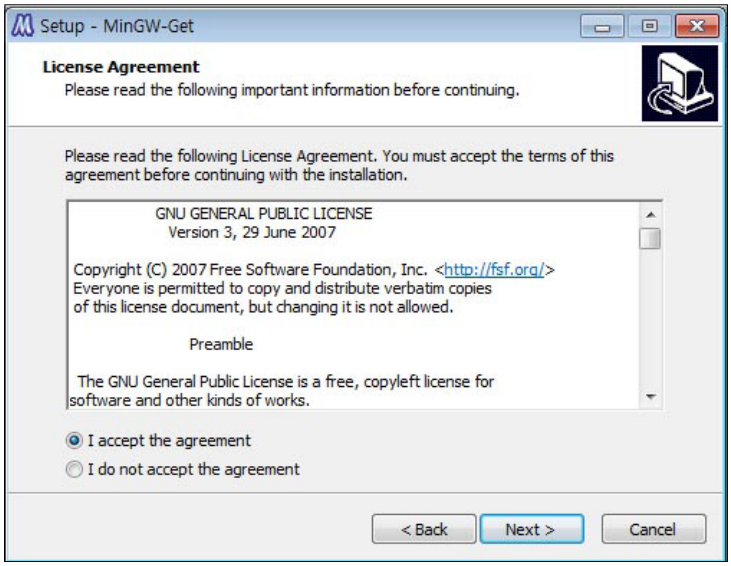
Next 를 클릭합니다.

08 실행3



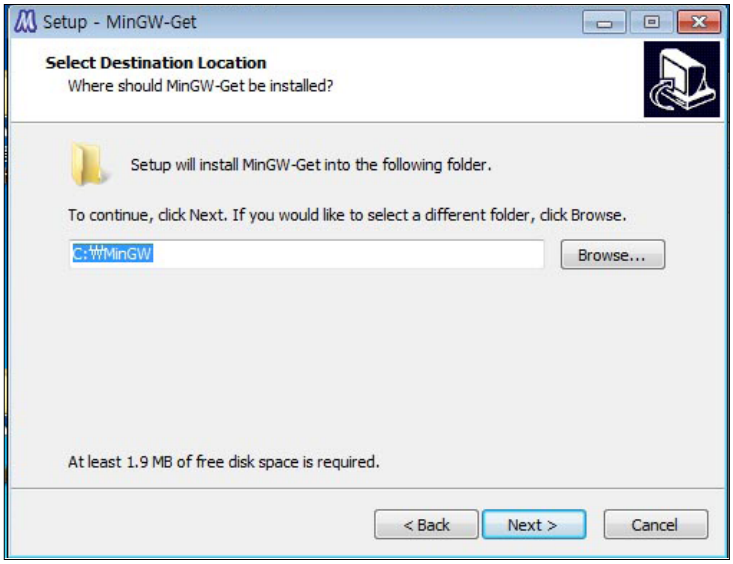
Next 를 클릭합니다.

09 실행4



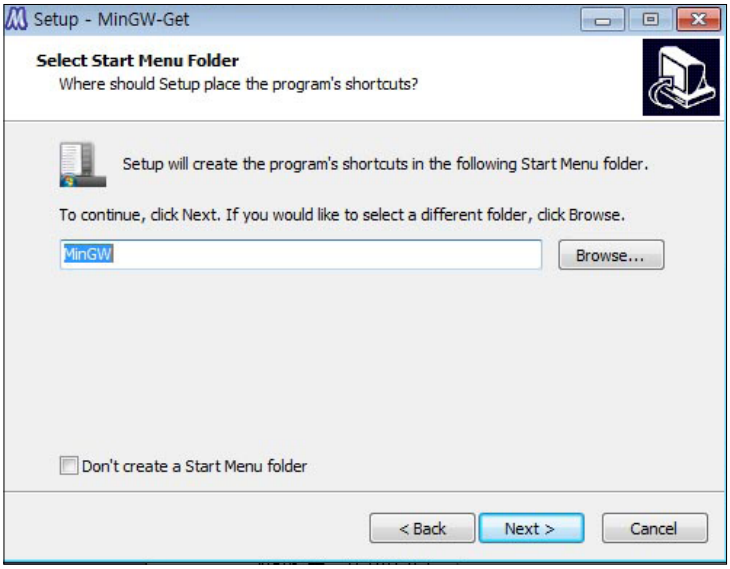
I accept 를 선택하고 Next 를 클릭합니다

10 실행5



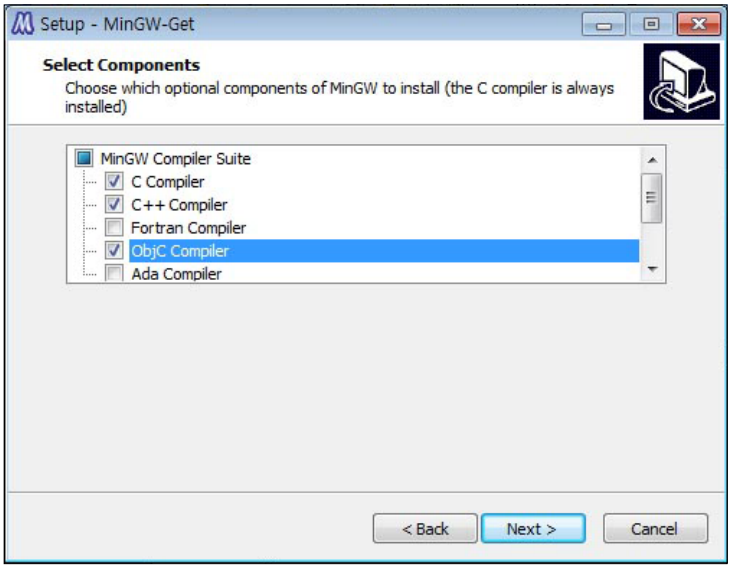
Next 를 클릭합니다.

11 실행6



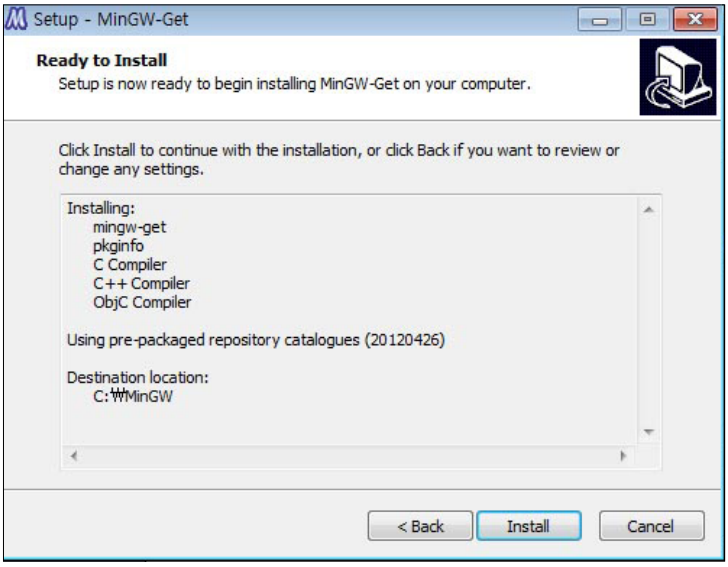
Next 를 클릭합니다.

12 실행7



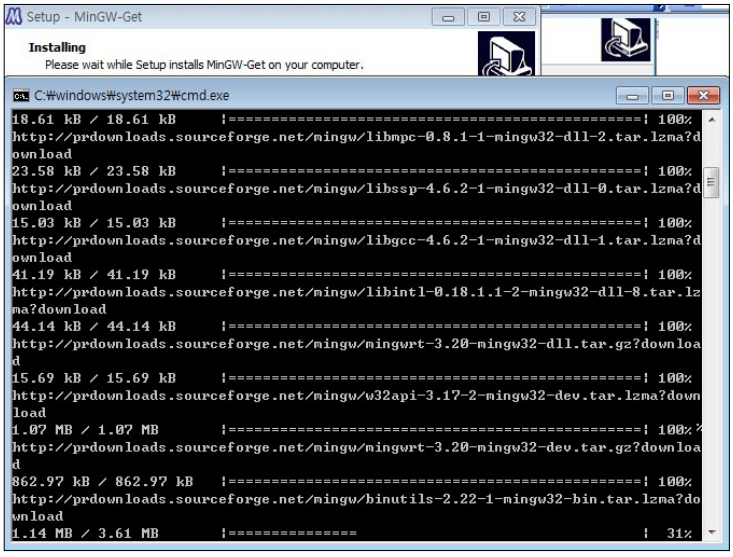
C, C++, ObjC Compiler 를 선택하고, Next 를 클릭합니다.

13 실행8



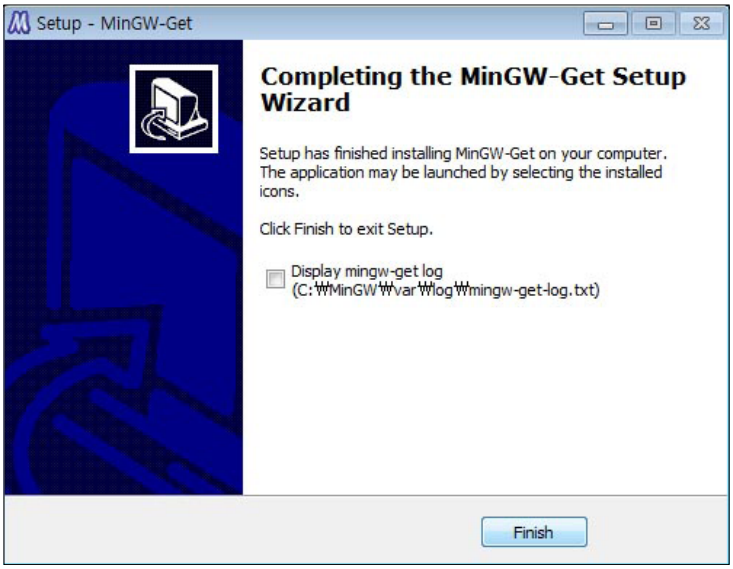
Install 을 클릭합니다

14 실행9



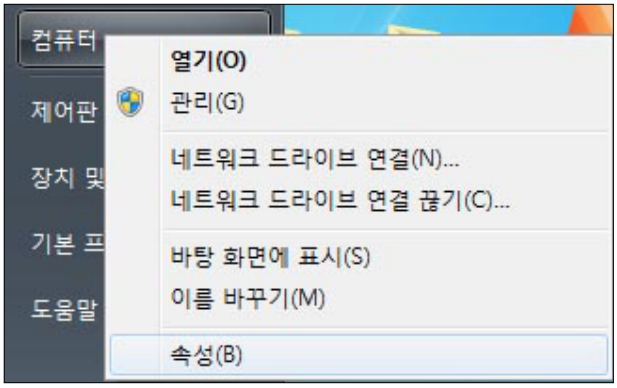
Install 하면 위와 같이 설치됩니다.

15 실행10



Finish 를 클릭하면 설치가 완료 됩니다.

16 설정1

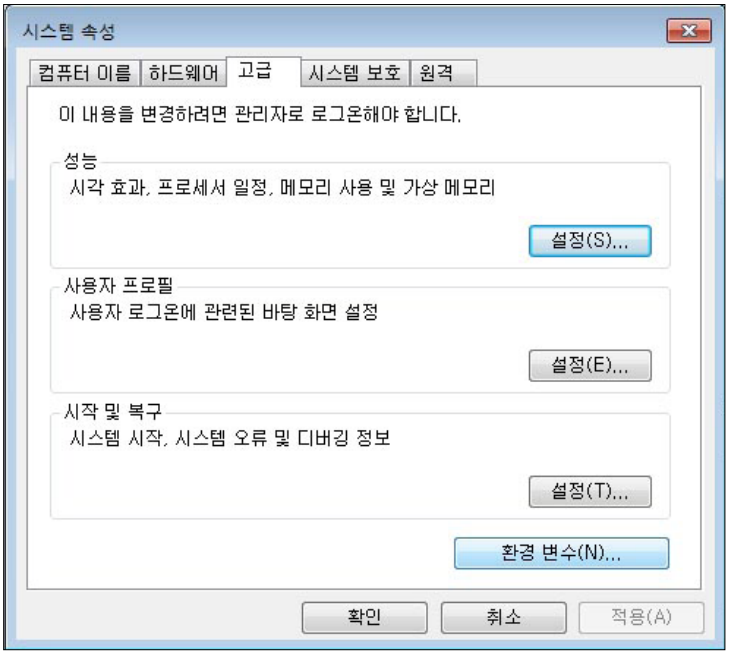


내 컴퓨터에서 우측 마우스를 클릭합니다. 속성을 클릭합니다.

17 설정2

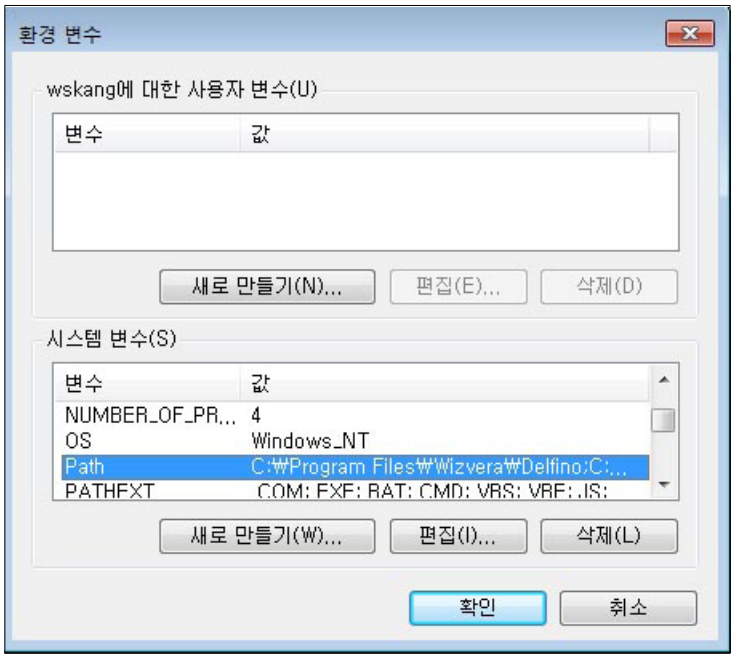
고급 시스템 설정을 클릭합니다.

18 설정3



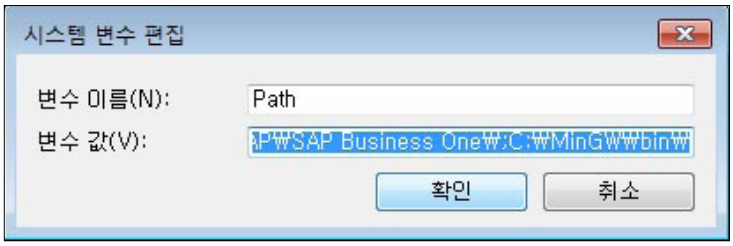
고급탭에서 환경변수를 클릭합니다.

19 설정4



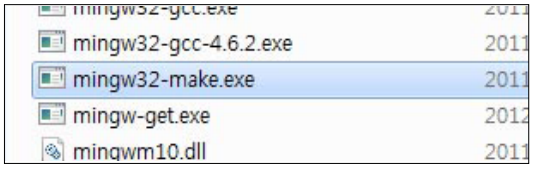
Path 를 선택하고, 편집 버튼을 누릅니다.

20 설정5



;C:₩MinGW₩bin₩ 라고 입력하고 확인 버튼을 누릅니다.

21 이클립스에서 make 파일 쓰기



이클립스에서는 make.ee 파일을 make 할 때 쓰이는데, MinGW 는 mingw32-make.exe 파일이 make 파일입니다. 그래서 이름을 변경해줘야 합니다.

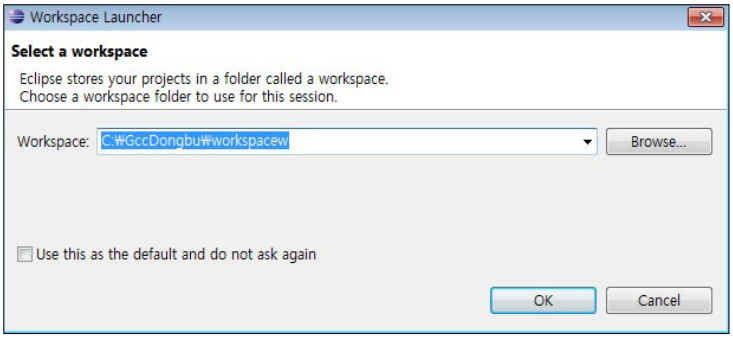
C://MinG/bin 폴더에 들어가서 mingw32-make.exe 파일을 바깥에 복사하여, 이름을 make.exe 로 바꿔서 저장하고, 이 파일을 bin 폴더에 그대로 복사해 넣습니다. 그러면 bin 폴더에는 make.exe 파일이 존재하게 됩니다.

[1.2.3] 개발환경 이클립스

01 이클립스 실행

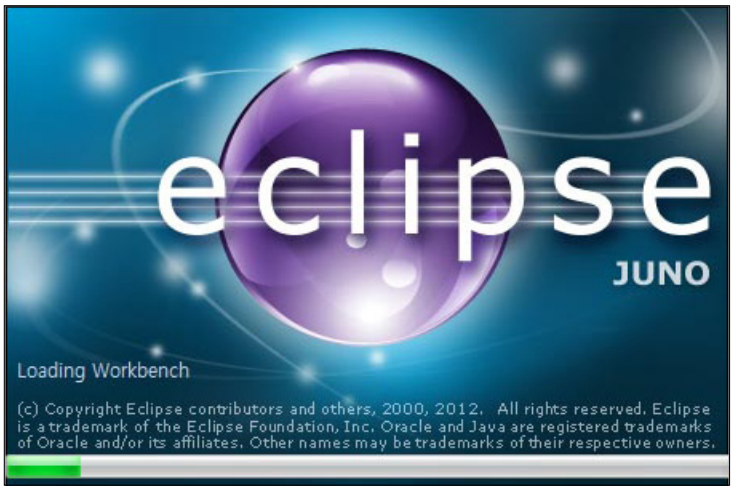
이클립스 실행 버튼을 누릅니다.

02 workspace



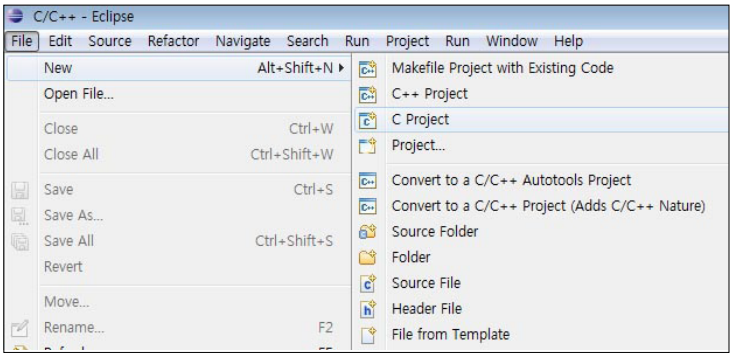
작업하고자 하는 폴더를 지정합니다. Browse 를 클릭하여 폴더를 지정하고 OK 버튼을 누릅니다.

03 이클립스 로딩 화면



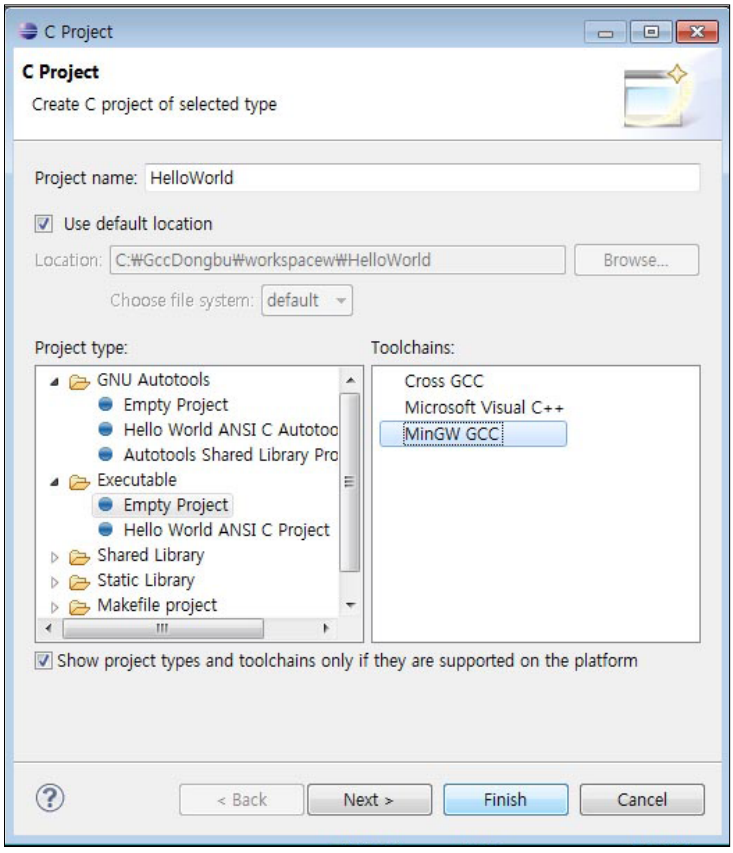
이클립스 로딩 화면입니다.

04 프로젝트 생성



이클립스 편집창 메뉴에서 File 〉 New 〉 C project 를 클릭합니다.

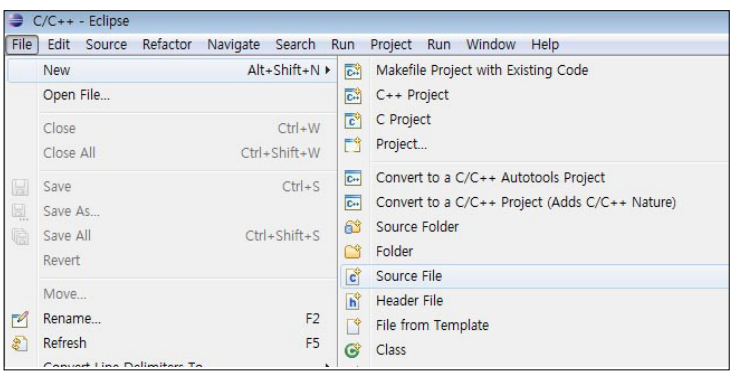
05 프로젝트 이름



Projec Name 을 HelloWorld 라고 입력하고, Project type 에서 Exectable 에서 Empty Project 를 선택합니다.

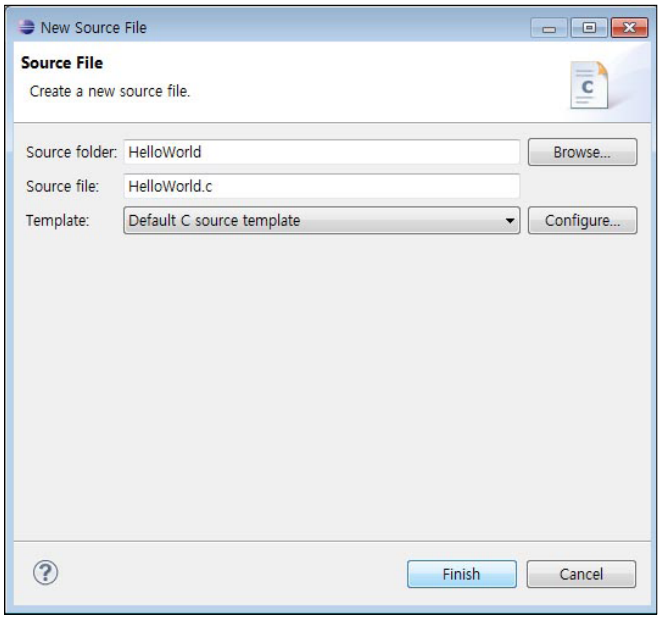
Toolchians 에서 MinGW GCC 를 선택하고 Finish 버튼을 누릅니다.

06 소스파일 작성



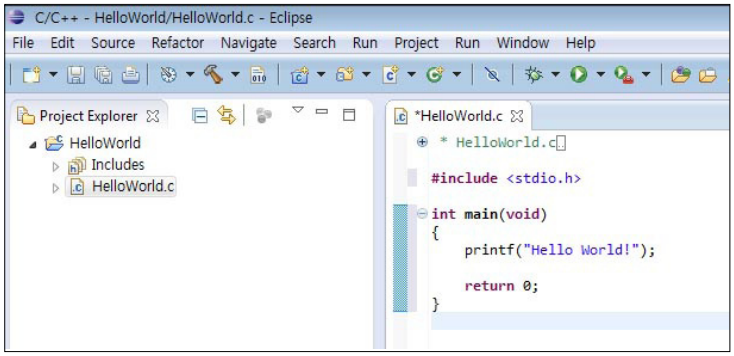
File 〉 New 〉 Source File 을 클릭합니다.

07 소스파일 이름



HelloWorld.c 라고 입력하고 Finish 버튼을 누릅니다.

08 Hello World 작성

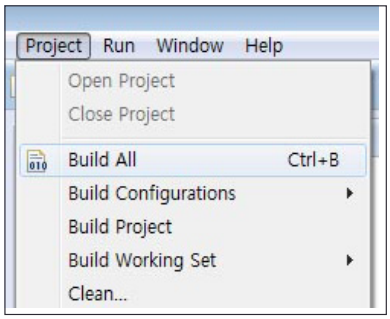


간단한 print 문을 작성합니다.

helloworld.c

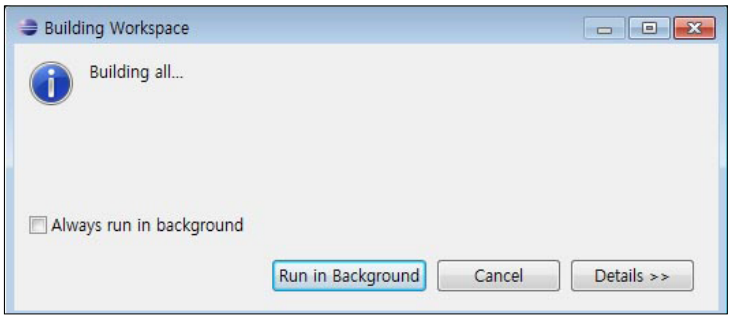
```
#include 〈stdio.h〉
int main(void)
{
    printf(“Hello World!”);
    fflush(stdout);
    return 0;
}
```

09 빌드



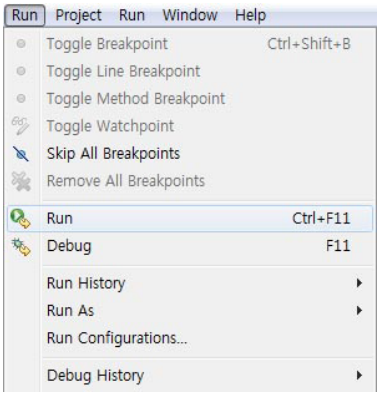
Projec 〉Build All 을 클릭합니다. 단축키 Ctrl+B 를 눌러도 결과는 마찬가지 입니다.

10 빌드중



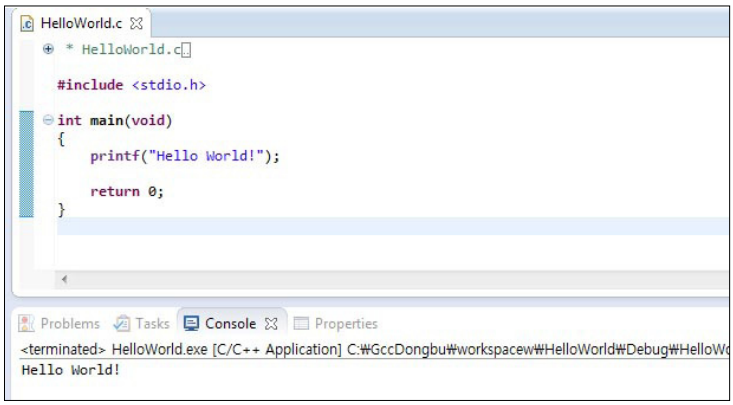
빌드 팝업입니다. 그대로 두면 완료 후 사라집니다.

11 Run 실행



Run 〉Run 을 클릭합니다. Ctrl+F11 단축키도 동일합니다.

12 실행화면



하단 Console 박스를 보면 Hello World! 가 정확히 찍힌 것을 확인합니다.

1.3 훑어본 소스 실행

본 예제는 C 언어의 전반적인 설명을 위해 작성된 것입니다. 로봇은 간단히 팔을 올리는 동작입니다. 문법의 전체 개요은 앞부분에서 간단히 요약했으므로 생략하겠습니다.

코딩계획

```
// 로봇초기화
// 모터사용 – 차려자세
// 동작
// 분기
// 종료
```

프로그래밍 세부설계

```
int main()
{
        // 결과값을 리턴받을 변수
        // for 문을 사용하기 위해 선언한 변수

        // 초기화 – 로봇제어를 하기 위해 반드시 앞 부분에 선언한다.
        // 각 변수들을 초기화 한다.
        // 열고 난 이후 이상이 있는 지 확인한다.
        // 이상이 있다면 에러 출력
        // 프로그램 종료

        // 이제부터 모터를 사용 할 수 있다.
        // 똑바로 일어선다. (차렷자세)

        // 모든 모터(16개 모터)를 0 위치로 설정함
        // 0 번 모터(오른쪽 어깨)
        // 1 번 모터(오른쪽 윗팔)
        // 3 번 모터(왼쪽 어깨)
        // 4 번 모터(왼쪽 윗팔)
        // 모터 동작(해당 자세를 1000ms 동안 동작)
        // 동작 대기(1000ms)

        //// 동작
        // 0 번 모터(오른쪽 어깨). 팔을 위로 들어올리는 값을 넣는다.
        // 모터 동작(해당 동작 전체를 1000 ms 동안 동작한다)
        // 동작 대기(1000ms)

        // 분기
        // 분기를 위한 goto문. _LABEL_1로 이동한다.
 // _LABEL_0은 goto문 때문에 건너뛰어진다.

// goto문을 사용하면 _LABEL_1로 바로 이동한다.


        // 프로그램 종료
        // 제어 종료 함수를 실행한다.
```

프로그래밍 작성

```
/*
        파일 명          : hhello.c
        제어로봇 형태    : 16dof 휴머노이드
        설명             : C 언어 전반적인 설명을 위한 예제로 전처리기, 변수, 반복,
분기, 함수호출이 포함되어 로봇의 한 팔을 들어올리는 예제. (motor 0번을 789 값을 넣어주면 됨)
*/
#include <stdio.h>
#include "hovis.h"
        // 동부로봇의 api 를 사용하기위해 반드시 이렇게 선언 해 두어야 한다.

int main()
{
        int nResult;
                // 결과값을 리턴받을 변수
        int i;      // for 문을 사용하기 위해 선언한 변수

        // 초기화 – 로봇제어를 하기위해 반드시 앞 부분에 선언한다.
        nResult = initialize(L"COM2", 115200);          // 각 변수들을 초기화 한다.
        if(nResult == 0){                               // 열고 난 이후 이상이 있는 지 확인한다.
                printf("Initialization Failed₩n");      // 이상이 있다면 에러 출력
                fflush(stdout);
                return 0;                               // 프로그램 종료
        }

        // 이제부터 모터를 사용 할 수 있다.
        // 똑바로 일어선다.         (차렷자세)
        for(i=0;i<16;i++){
                g_fMotorPos[i]=0;
        // 모든 모터(16개 모터)를 0 위치로 설정함
        }
        g_fMotorPos[0] = –90;                           // 0 번 모터(오른쪽 어깨)
        g_fMotorPos[1] = 90;                            // 1 번 모터(오른쪽 윗팔)
        g_fMotorPos[3] = –90;                           // 3 번 모터(왼쪽 어깨)
        g_fMotorPos[4] = 90;                            // 4 번 모터(왼쪽 윗팔)
        run(1000);                  // 모터 동작(해당 자세를 1000ms 동안 동작)
        delay(1000);                // 동작 대기(1000ms)
```

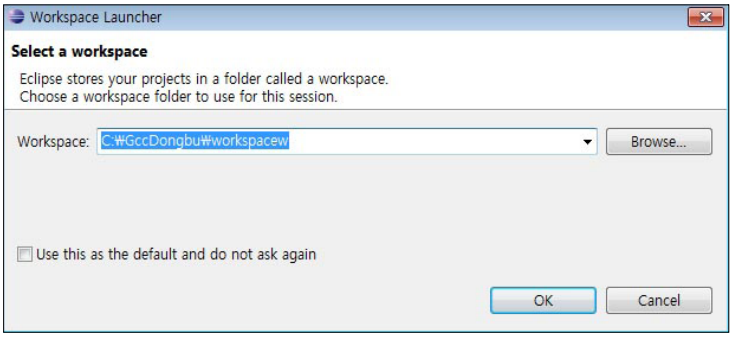
```
        //// 동작
        g_fMotorPos[0] = 90; // 0 번 모터(오른쪽 어깨). 팔을 위로 들어올리는 값을 넣는다.
        run(1000);          // 모터 동작(해당 동작 전체를 1000 ms 동안 동작한다)
        delay(1000);        // 동작 대기(1000ms)

        // 분기
        goto _LABEL_1;              // 분기를 위한 goto문. _LABEL_1로 이동한다.
        _LABEL_0:                   // _LABEL_0은 goto문 때문에 건너뛰어진다.
        printf("hello, c!₩n");
        fflush(stdout);
        _LABEL_1:                   // goto문을 사용하면 _LABEL_1로 바로 이동한다.
        printf("hello, hovis!₩n");
        fflush(stdout);

        // 프로그램 종료
        terminate();                // 제어 종료 함수를 실행한다.
        return 0;
}
```

빌드 및 실행

01 이클립스 실행

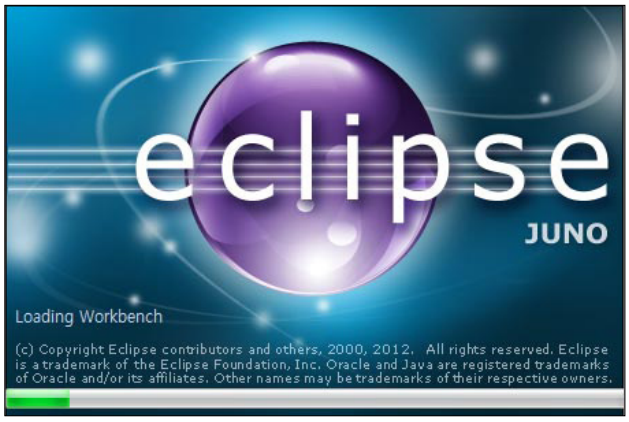
이클립스 실행 버튼을 누릅니다.

02 Workspace



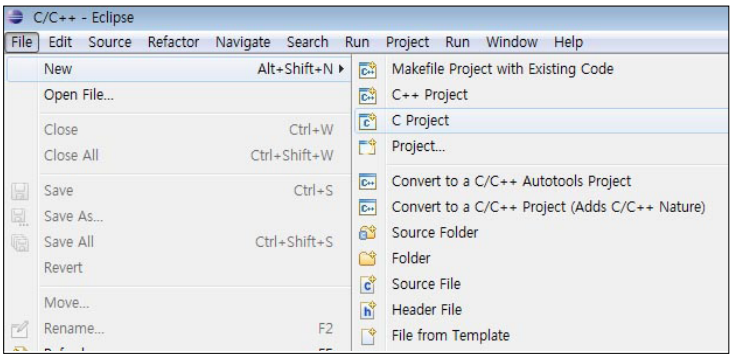
작업하고자 하는 폴더를 지정합니다. Browse 를 클릭하여 폴더를 지정하고 OK 버튼을 누릅니다.

03 이클립스 로딩 화면



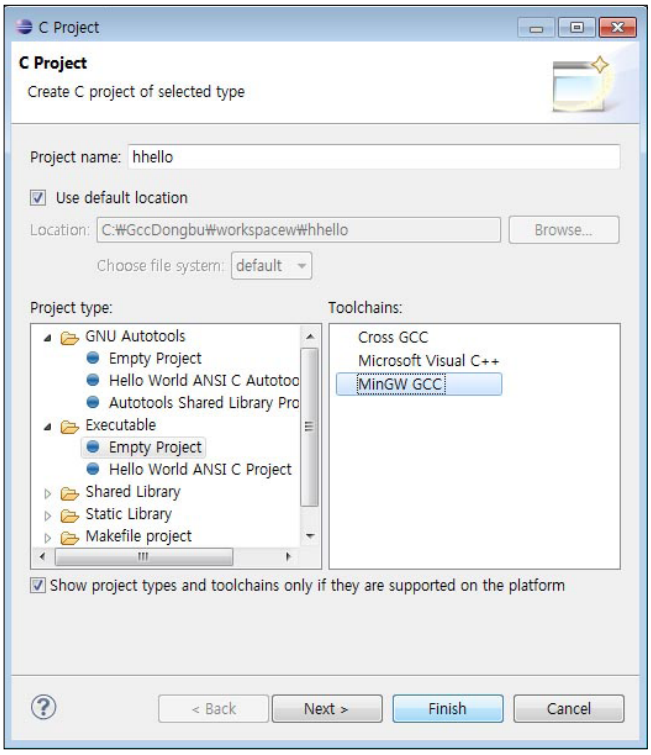
이클립스 로딩 화면입니다.

04 프로젝트 생성



이클립스 편집창 메뉴에서 File 〉 New 〉 C project 를 클릭합니다.

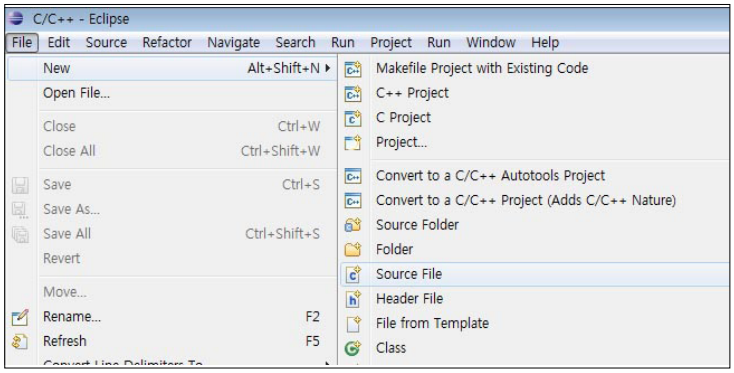
05 프로젝트 이름



Projec Name 을 hhello 라고 입력하고, Project type 에서 Exectable 에서 Empty Project 를 선택합니다.

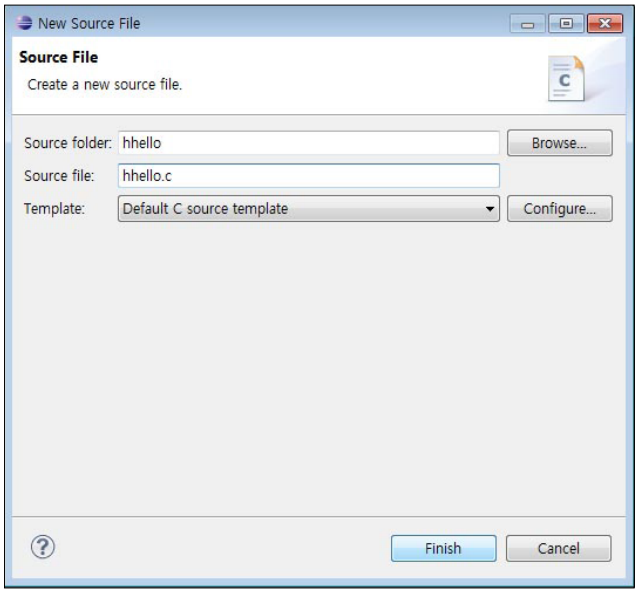
Toolchians 에서 MinGW GCC 를 선택하고 Finish 버튼을 누릅니다

06 소스파일 작성



File 〉New 〉Source File 을 클릭합니다.

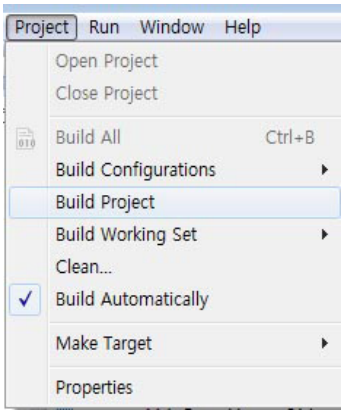
07 소스파일 이름



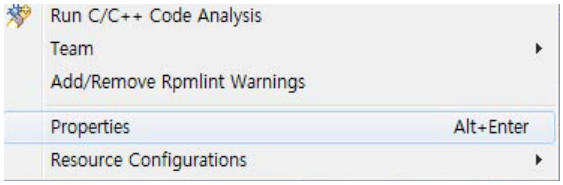
hhello.c 라고 입력하고 Finish 버튼을 누릅니다.

08 소스작성

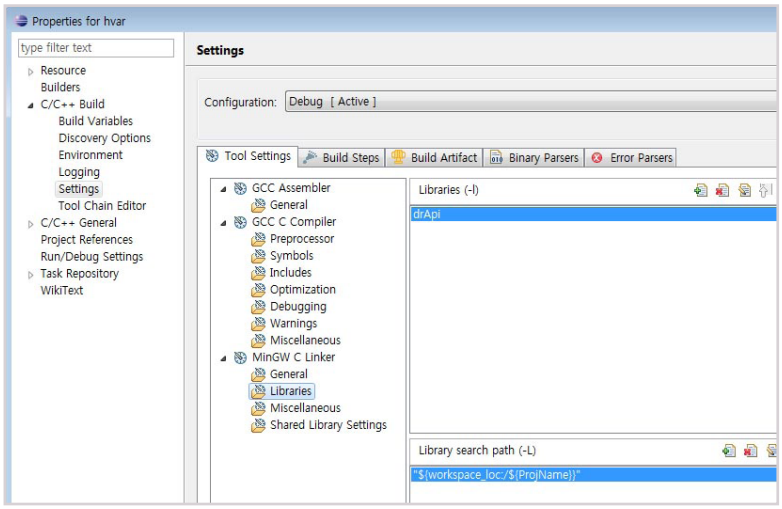
본 교재에게 C 프로그래밍을 하기 위해선 기본 제공되는 라이브러리 파일을 먼저 추가해주어야 올바른 컴파일을 할 수 있습니다. 필요한 파일은 drApi.dll, drApi.h, hovis.c, hovis.h 등 총 4가지 입니다. 파일 추가 전에 먼저 Property 에서 라이브러리를 설정해주어야 합니다. 파일 생성을 한 후에 아무 코딩도 하지 말고, 바로 Build project 를 실행하세요. 그래야 라이브러리 추가가 가능합니다.



Build Project 를 클릭합니다.

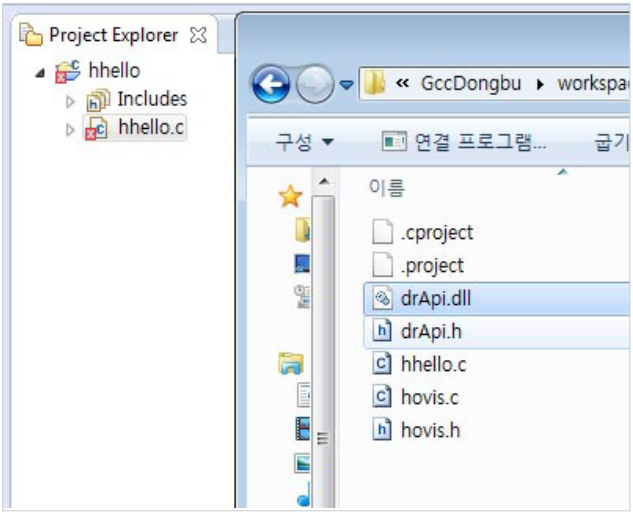


좌측 프로젝트 이름에 커서를 놓고 오른쪽 마우스키를 누릅니다.
제일 하단에 Propertis 를 클릭합니다. Alt+Enter 도 동일합니다.

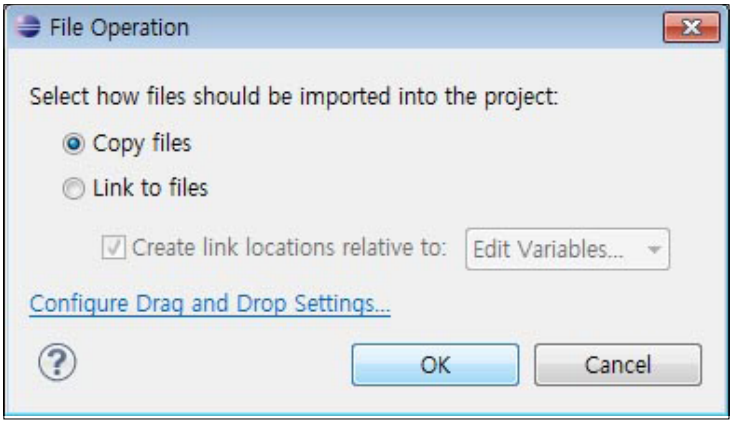


C/C++ Build 〉 Settings 를 클릭하고, 우측에 MinGW C Linker 〉 Libraties 를 클릭합니다. 우측에 Libraies 에 + 마크를 클릭하고 팝업창이 뜨면 drApi 라고 입력합니다. Library search path 에 + 마크를 클릭하고 팝업창이 뜨면 Workspace 를 클릭하고, Folder selection 에서 프로젝트 이름과 동일한 것을 클릭하고 OK 버튼을 누릅니다.

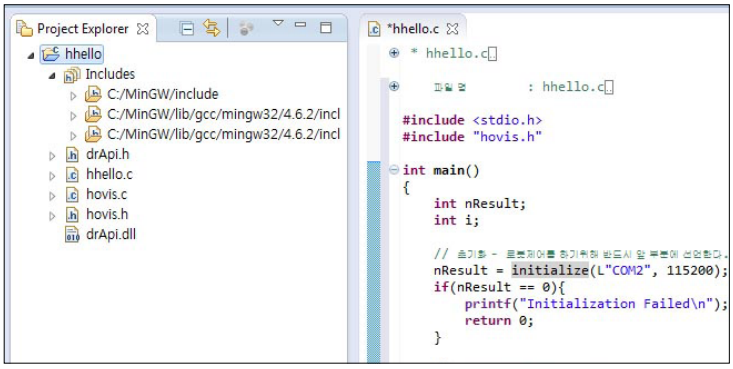
로봇을 동작시키기 위해선 위 네 개 파일을 프로젝트에 포함시켜야 합니다.
위 파일이 있는 파일탐색기 폴더에서 드래그 하는 방식으로 프로젝트에 포함시킵니다.



파일을 끌어다가 드래그해서 왼쪽 프로젝트에 추가합니다.

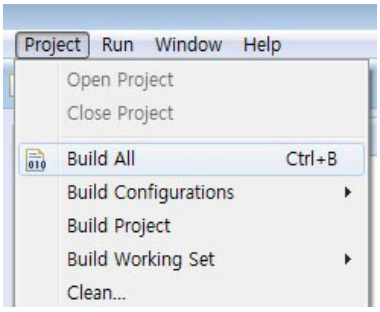


OK 버튼을 누릅니다.



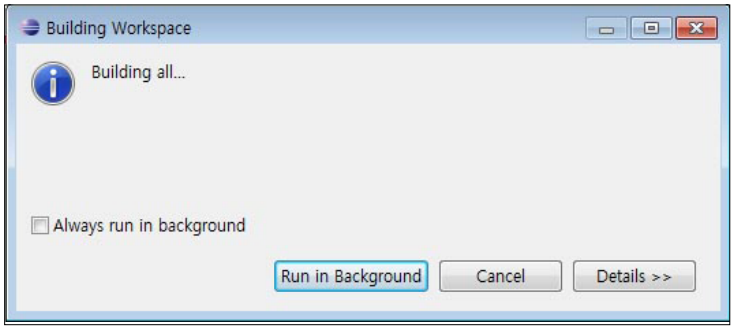
소스를 작성합니다.

09 빌드



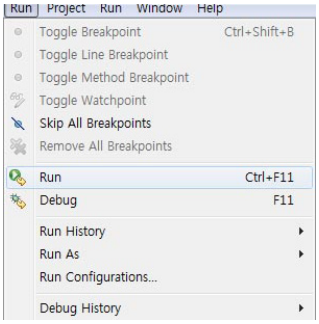
Projec 〉 Build All 을 클릭합니다. 단축키 Ctrl+B 를 눌러도 결과는 마찬가지 입니다.

10 빌드중



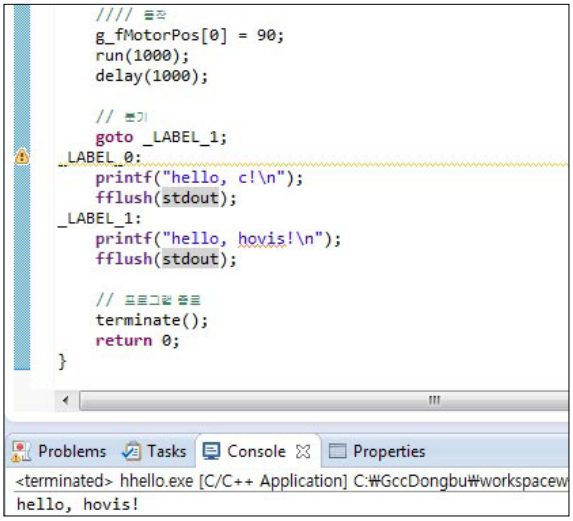
빌드 팝업입니다. 그대로 두면 완료 후 사라집니다.

11 Run 실행



Projec 〉Build All 을 클릭합니다. 단축키 Ctrl+B 를 눌러도 결과는 마찬가지 입니다.

12 실행화면



하단 Console 박스를 보면 Hello World! 가 정확히 찍힌 것을 확인합니다.

13 로봇동작

로봇이 오른팔을 듭니다.

1.4 로봇제어함수

C 로봇 프로그래밍을 위해서는 동부로봇에서 제공하는 라이브러리와 오픈 함수를 활용해야 합니다. www.hovis.co.kr/guide 에 기본 API 를 제공하지만, GCC 컴파일러를 활용한 C 프로그래밍을 위해서 간편성과 편의성이 추가된 오픈 소스 함수를 제공합니다. 본 장에서는 함수를 간략히 설명하고, 부록에서 상세히 소스코드를 설명합니다. 이후 프로그래밍 예제는 여기에서 제시된 함수를 활용하여 프로그래밍되어 있습니다.

 # 함수명 : int initialize(const TCHAR* pszPort, DWORD dwBaudRate)

함수설명

HOVIS LITE 16DOF를 사용하기 위한 초기화 함수입니다.
nPortNum의 번호를 가진 시리얼 포트를 nBaudRate의 속도로 열고, 모터의 방향을 설정합니다.

 # 함수명 : void run(int nTime)

함수설명

HOVIS LITE 16DOF를 실제로 움직이는 함수입니다.
위에 선언된 g_fMotorPos[16], g_ucMotorGreenLed[16], g_ucMotorBlueLed[16], g_ucMotorRedLed[16], g_ucMotorStop[16], g_nDrcMelody에 값을 넣고 run() 함수를 실행하면 실제 로 로봇이 그 값에 따라서 동작하게 됩니다.
static 변수들을 선언해, 직전에 run() 함수를 호출 했을 때의 전역 변수 값들을 저장해 놓고 다음 호출 시에는 이전과 비교해 변화된 모터만 반영하여 실행합니다. 단, 맨 처음 호출한 경우에는 모두 반영하여 실행합니다. nTime의 범위는 0~2844까지다. 그 이상의 시간 동안은 동작시킬 수 없으며 이 범위를 벗어난 경우는 자동으로 범위 내로 수정해서 동작됩니다.

 # 함수명 : void motion(int nMotionNum, int nReady)

함수설명

HOVIS LITE 16DOF에 저장된 모션을 실행하는 함수입니다.
nMotionNum은 저장된 모션의 번호이고, 0~127의 값을 가집니다.
nReady는 준비자세 실행 여부로, 0~1의 값을 가집니다.
nReady가 1로 입력된 경우 모션의 첫 프레임만 천천히 실행합니다.
nReady가 0으로 입력된 경우, 모션의 모든 프레임을 실행합니다.
범위 밖의 값이 입력된 경우 실행하지 않습니다.

 # 함수명 : void read()

함수설명

HOVIS LITE 16DOF의 DRC로부터 센서 값을 읽어서 센서 변수에 업데이트 합니다.
read() 함수를 실행하면 g_nButton, g_nBattery, g_nAdcDist[2] 등의 전역 변수가 새로 업데이트 된 값으로 바뀌게 됩니다.

 # 함수명 : void delay(int nTime)

함수설명

nTime(ms)의 시간동안 아무 일도 하지 않고 대기합니다.

 # 함수명 : void terminate()

함수설명

제어를 종료하는 함수다. g_hDr 핸들의 시리얼 통신 연결을 해제하고 모터와 관련된 메모리를 해제 후 g_hDr 핸들의 메모리까지 해제 후 g_hDr 값을 NULL로 만들어 줍니다.

02 변수와 연산자

2.1 변수와 연산자

변수란 데이터를 저장할 수 있는 메모리 공간에 붙여진 이름을 말합니다. 모든 프로그램은 첫 번째는 변수 선언부터 시작합니다. 결과값이나 반복을 위해 항상 사용되어지기 때문입니다. 연산자란 연산을 요구할 때 사용되는 기호를 말합니다. 예를 들면 +, −, *, / 등 흔히 알고 있는 수학적 기호입니다.

아래 예제는 모터를 하나씩 제어해서 웨이브 동작을 만드는 프로그램입니다.

hvar.c

```
#include <stdio.h>
#include "hovis.h"

int main()
{
        int nResult;
        int i;

        nResult = initialize(L"COM2", 115200);
        if(nResult == 0){
                printf("Initialization Failed₩n");
                fflush(stdout);
                return 0;

        }

        for(i=0;i<16;i++){
                g_fMotorPos[i]=0;

        }
        g_fMotorPos[0] = -90;
        g_fMotorPos[1] = 90;
        g_fMotorPos[3] = -90;
        g_fMotorPos[4] = 90;
        run(1000);

        delay(1500);
```

```
        g_fMotorPos[1] = 45;
        g_fMotorPos[4] = 45;
        run(100);
delay(1000);
g_fMotorPos[1] = 0;
g_fMotorPos[4] = 0;
run(100);
        delay(1000);
        g_fMotorPos[4] = –20;
        g_fMotorPos[5] = 55;
        run(400);
delay(300);
        g_fMotorPos[2] = –25;
        g_fMotorPos[4] = 31;
        g_fMotorPos[5] = –31;
        run(400);
delay(300);
        g_fMotorPos[1] = 31;
        g_fMotorPos[2] = –31;
        g_fMotorPos[4] = –25;
        g_fMotorPos[5] = 0;
        run(400);
delay(300);
g_fMotorPos[1] = –20;
g_fMotorPos[2] = 55;
g_fMotorPos[4] = 0;
        run(400);
        delay(300);
                g_fMotorPos[1] = 0;
                        g_fMotorPos[2] = 0;
                        run(400);
                delay(500);
                g_fMotorPos[1] = 45;
                g_fMotorPos[4] = 45;
                run(100);
        delay(200);
        g_fMotorPos[1] = 90;
        g_fMotorPos[4] = 90;
        run(100);
delay(200);
terminate();
return 0;
```

위 코드중에 진하게 처리된 변수 선언부분을 살펴봅니다.

```
int nResult;
int i;
```

int nResult;는 결과값을 리턴받을 변수입니다.
int i; 는 for 문을 사용하기 위해 선언한 변수 입니다.

앞으로 제시되는 전체 소스코드는 가볍게 한번 훑어보고, 핵심적인 사항이 전체 코드에서 어떻게 쓰이는지 살펴보라는 의미입니다. 엔지니어마다 코딩 방식이 다르지만, 이 책은 전체적인 소스를 보고 눈에 익힌 후, 한글로 윤곽이 되는 사항을 적고, 세부설계를 한 후 코딩을 하는 방식을 취하겠습니다.

문법요약

◆ 변수

ex) ledbutton.dts 중 일부

C-like 보기

```
1  void main()
2  {
3          Red=false
4          Green=false
5          Blue=false
6          BtnEnd=false
7          while( true )
8          {
9                  if( ( ( MPSU_ButtonStat == 0x04 ) && ( !BtnEnd ) ) )
10                 {
11                         Red=( !Red )
12                         BtnEnd=true
13                 }
14                 else
15                 {
16                 }
```

※ 변수란 데이터를 저장할 수 있는 메모리 공간에 붙여진 이름을 의미합니다.

변경이 가능합니다. 단, 상수는 변경이 불가능합니다.

※ 다향한 형태(자료형) 의 변수는 메모리 공간 쓰임새에 따라서 아래와 같이 간략히 나뉩니다.

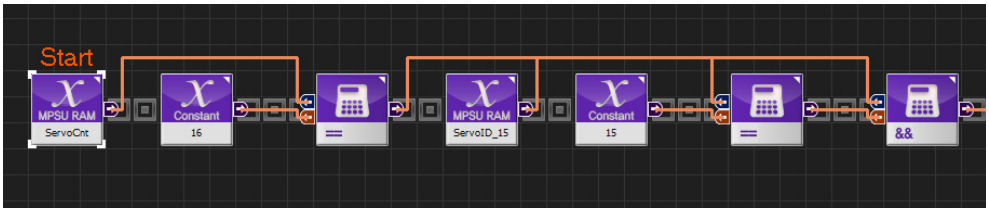
정수형 : char, int, long 3

실수형 : float, double 3.1

※ 변수의 선언 및 대입 예시입니다.

```
Int main(void)
{
        Int val; //int형 변수 val 의 선언
                        // → int 형 변수 val 이름 지어주고, 메모리 공간 할당
        Val = 20; //변수 val 에 20을 저장
```

ex) ledbutton.dts 중 일부



C-like 보기

```
 9  void main()
10  {
11          if( ( ( MPSU_ServoCnt == 16 ) && ( MPSU_ServoID_15 == 15 ) ) )
12          {
13                  melody( 1 )
14                  SERVO_TorqCtrl[254]=96
15                  InitPosition16()
16                  delay( 2000 )
17                  CurrentMotion=-1
18                  MotionAfterStop=-1
19                  RmcEnd=false
20                  MtnReady=false
```

◆ 리터럴 상수의 기본 자료형

상수도 메모리 공간에 저장되기 위해서 자료형이 결정됩니다.

Char c = 'A' / 문자 상수 → 문자를 쓸 경우 char 형으로 인식하고, 메모리에 넣는다는 의미입니다.

Int I = 5 //정수상수

Double d = 3.15 // 실수상수

왼쪽은 변수, 오른쪽은 상수, 둘은 독립적인 관계입니다.

◆ 심볼릭(symbolic) 상수

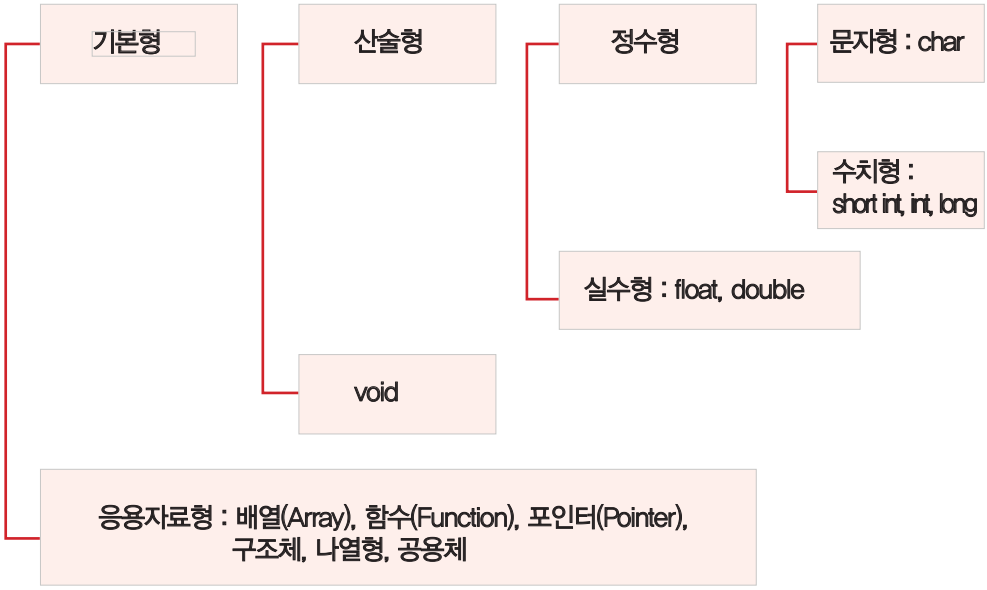
이름을 지니는 상수입니다. 심볼릭 상수를 정의하는 방법은 아래와 같습니다.

const 키워드 를 통한 변수의 상수화
const int MAX = 100 // 변수가 상수가 됨
MAX = 102 // 허용 안됨, MAX 는 상수이기 때문에 상수 변수 선언시 대문자로 써주는 게 관례입니다.

◆ 접미사에 따른 다양한 상수의 표현

접미사	자료형	사용 예
u or U	unsigned int	304U
l or L	long	304L
ul or UL	unsigned long	304UL
f or F	float	3.15F
l or L	long double	3.15L

◆ 자료형 종류



◆ 자료형 범위

구분	자료형	범위	byte
문자형	char	–128~127	1
	unsigned char	0~255	
정수형	int	–2147483648 ~ 2147483637	4
	unsigned int	0 ~ 4294967295	
	short	–32768 ~ 32767	2
	unsigned short	0 ~ 65535	
	long	–2147483648 ~ 2147483637	4
	unsigned long	0 ~ 4294967295	
실수형	float	±3.4 x 10–38 ~ ±3.4 x 1038	4
	double	±1.7 x 10–308 ~ ±1.7 x 10308	8

◆ 확장문자열

표기	의미	기능
₩n	New line	줄바꿈을 한다.
₩t	Tab	다음 탭 위치로 옮긴다.
₩b	Back space	현재 줄의 첫번째 칸으로 간다
₩r	Carriage return	한 칸 뒤로 옮긴다.
₩a	Alert	경보음을 낸다
₩”	” 표시 출력	큰 따옴표를 출력
₩’	’ 표시 출력	작은 따옴표를 출력

◆ printf 변환 문자열

구분	변환문자열	출력형태
정수형	%d	부호 있는 10진수
	%u	부호 없는 10진수
	%o	부호 없는 8진수
	%x, %X	부호 없는 16진수
실수형	%lf, %f	부호 있는 소수점(double, float)
	%le, %e	부호 있는 지수형(double, float)
문자형	%c	하나의 문자
문자열	%s	문자열

◆ printf 변환 문자열 옵션

변경자	의 미	
숫자	출력 자릿수 확보	
	%10d	화면에 10자리를 확보 오른쪽 정렬
	%–10d	화면에 10자리를 확보 왼쪽 정렬
	%10.2lf	화면에 10자리를 확보, 소수점 이하 2자리 까지 출력
h	Short 형임을 의미 예) %hu	
L , l	Long형임을 의미 예) %ld	

◆ Scanf 변환 문자열

구분	변환문자열	출력형태
정수형	%d, %u	10진수
	%o	8진수
	%x	16진수
실수형	%lf, %f	소수점 또는 지수형
	%le, %e	부호 있는 지수형(double, float)
문자형	%c	하나의 문자 (char 형 변수)
문자열	%s	문자열 (char 배열)

◆ C 언어의 키워드 와 설명 1

1.asm : 인라인 어셈블리 코드를 나타내는 키워드
2.auto : 기본적인 변수의 저장방식을 나타내는 키워드
3.break : for,while,switch,do...while문을 조건없이 마치는 명령
4.case : switch문 내에서 사용되는 명령
5.char : 가장 간단한 데이터형
6.const : 변수가 변경되지 않도록 방지하는 데이터 지정자
7.continue : for,while,switch,do...while문을 다음 반복동작으로 진행시키는 명령
8.default : case문에 일치하지 않는 경우를 처리하기 위해 switch문에서 사용되는 명령
9.do : while문과 함께 사용되는 순환명령.순환문은 최소한 한번 실행됨.
10.double : 배정도 부동 소수형값을 저장할 수 있는 데이터형
11.else : if문이 FALSE로 평가될 때 실행되는 선택적인 문장을 나타내는 명령
12.enum :변수가 특정값만을 받아들이도록 해주는 데이터형
13.extern : 변수가 프로그램의 다른 부분에서 선언된다는 것을 알려주는 데이터 지정자
14.float : 부동 소수형 숫자값을 저장하기 위해 사용되는 데이터형
15.for : 초기화,증가,조건 부분을 가지는 순환명령

◆ C 언어의 키워드 와 설명 2

16.goto :　　정의되어 있는 레이블로 이동시키는 명령
17.if : TRUE/FALSE의 결과에 따라 프로그램의 제어를 변경하는데 사용되는 명령
18.int : 정수형 값을 저장하는 데 사용되는 데이터형
19.long : int형보다 큰 정수형 값을 저장하는 데 사용되는 데이터형
20.register : 가능하다면 변수를 레지스터에 저장하도록 지정하는 저장형태 지정자
21.return : 현재의 함수를 마치고 호출한함수로 프로그램의 제어를 돌려주는 명령.
함수에서 값을 돌려주기 위해서 사용됨.
22.short : 정수형 값을 저장하는 데 사용되는 데이터형. 자주 사용되지는 않지만
대부분의 컴퓨터에서 int형과 동일한 크기를 가짐.
23.signed : 변수가 양수와 음수값을 모두 저장할 수 있다는 것을 지정하기 위해서
사용되는 지정자
24.sizeof : 항목의 크기를 바이트 단위로 알려주는 연산자
25.static :컴파일러가 변수의 값을 보존해야 한다는 것을 지정하는데 사용되는 지정자
26.struct :C에서 어떤 데이터형의 변수를 함께 결합시키는 데 사용되는 키워드
27.switch :여러 가지 조건을 통해서 프로그램의 흐름을 변경하는 데 사용되는 명령.
case문과 함께 사용됨.
28.typedef : 이미 존재하는 변수와 함수의 형태를 새로운 이름으로 변경하는 데
사용되는 지정자
29.union :여러 개의 변수가 동일한 메모리 영역을 공유하도록 해주는 데 사용되는 키워드
30.unsigned : 변수가 양수값만으르 저장할 수 있다는 것을 지정하는 데 사용되는 지정자.
31.void : 함수가 어떤 값을 돌려주지 않거나, 또는 사용되는 포인터가 범용
포인터이거나 모든 데이터형을 지적할 수 있다는 갓을 지정하는 데 사용되는 키워드
32.volatile : 변수가 변경될 수 없다는 것을 지정하는 지정자.
33.while : 지정된 조건이 TRUE로 평가되는 한 계속해서 포함된 문장을 실행하는 순환문

◆ 문자형

문자 표현을 위하여 ASCII 코드라는 것이 등장합니다.

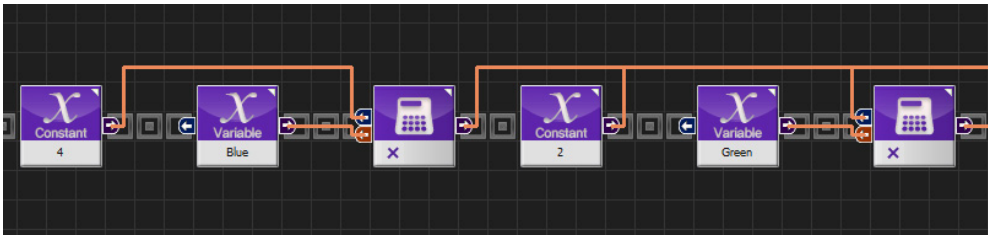
미국 표준 협회 (ANSI) 에 의해 정의, 컴퓨터를 통해서 문자를 표현하기 위한 표준으로서 컴퓨터는 문자를 표현하지 못하기때문에, 문자와 숫자의 연결 관계를 정의하였습니다. 문자 A 는 숫자 65, 문자 B 는 숫자 66 으로 매칭시킵니다.
문자, 컴퓨터는 2진수밖에 받지 못하기 때문에, 컴퓨터와 사람사이에 약속을 하기 시작하였습니다. 사람은 문자 표현 원하고, 문자와 숫자 사이에 매핑 관계를 유지하기를 원합니다. 에를 들어 A=65, B = 55 라고 약속, 사람이 A 입력, 컴퓨터는 65라고 저장합니다.

ANSI 에서 정의되었고, 숫자와 문자 매핑은 아래와 같습니다.

10진	16진	문자	10진	16진	문자	10진	16진	문자	10진	16진	문자	10진	16진	문자
0	0	Null	47	2F	/	68	44	D	89	59	Y	110	6E	n
7	7	Bell	48	30	0	69	45	E	90	5A	Z	111	6F	o
8	8	BS	49	31	1	70	46	F	91	5B	[	112	70	p
9	9	Tab	50	32	2	71	47	G	92	5C	₩	113	71	q
10	A	LF	51	33	3	72	48	H	93	5D	]	114	72	r
13	D	CR	52	34	4	73	49	I	94	5E	^	115	73	s
32	20	공백	53	35	5	74	4A	J	95	5F	_	116	74	t
33	21	!	54	36	6	75	4B	K	96	60	`	117	75	u
34	22	"	55	37	7	76	4C	L	97	61	a	118	76	v
35	23	#	56	38	8	77	4D	M	98	62	b	119	77	w
36	24	$	57	39	9	78	4E	N	99	63	c	120	78	x
37	25	%	58	3A	:	79	4F	O	100	64	d	121	79	y
38	26	&	59	3B	;	80	50	P	101	65	e	122	7A	z
39	27	'	60	3C	<	81	51	Q	102	66	f	123	7B	{
40	28	(	61	3D	=	82	52	R	103	67	g	124	7C	|
41	29	)	62	3E	>	83	53	S	104	68	h	125	7D	}
42	2A	*	63	3F	?	84	54	T	105	69	i	126	7E	~
43	2B	+	64	40	@	85	55	U	106	6A	j	127	7F	Del
44	2C	,	65	41	A	86	56	V	107	6B	k			
45	2D	–	66	42	B	87	57	W	108	6C	l			
46	2E	.	67	43	C	88	58	X	109	6D	m			

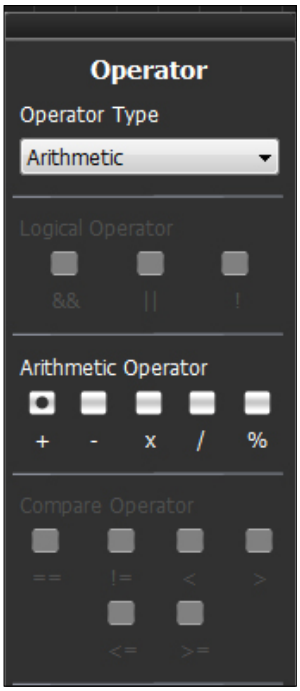
◆ 연산자

ex) ledbutton.dts 중 일부



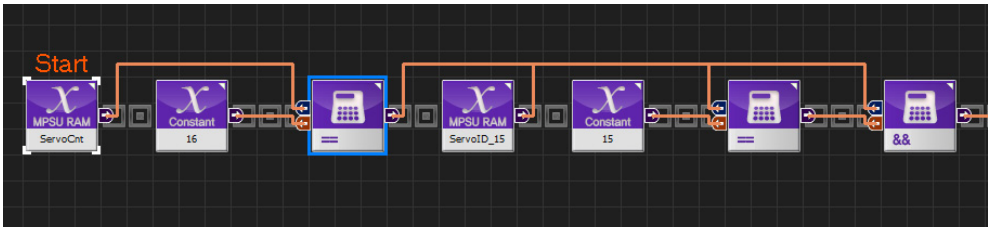
C-like 보기

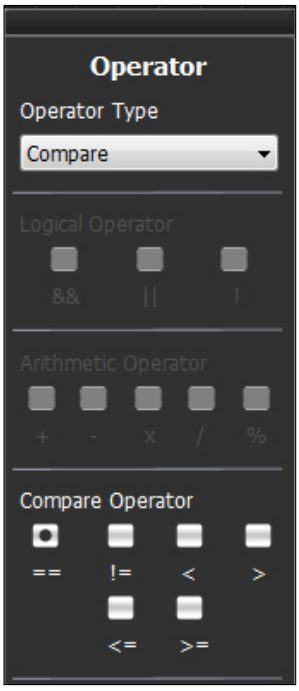
```
33  led( ( ( ( 4 * Blue ) + ( 2 * Green ) ) + Red ) )
34  if( ( ( MPSU_ButtonStat == 0x00 ) && BtnEnd ) )
35  {
36          BtnEnd=false
37  }
38  else
39  {
40  }
```

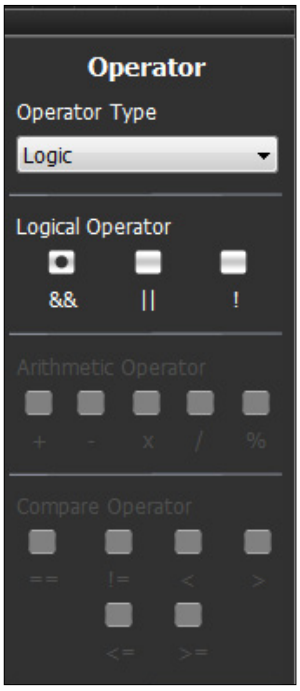


기능별 종류	연산자
산술 연산자	+ – * / %
부호 연산자	+ –
대입 연산자	= 복합 대입 연산자
관계 연산자	== != 〈= 〈 〉= 〉
증감 연산자	++ ––
포인터 연산자	* & []
구조체 연산자	. –〉
논리 연산자	|| && !
비트 연산자	| & ~ 〉〉 〈〈
삼항 조건 연산자	? :
쉼표 연산자	,
sizeof 연산자	sizeof
캐스트 연산자	(type) type()
괄호 연산자	()
C++ 연산자	new delete :: .* –〉*

ex) Remocon16.dts 중 일부







C-like 보기

```
 9  void main()
10  {
11          if( ( ( MPSU_ServoCnt == 16 ) && ( MPSU_ServoID_15 == 15 ) ) )
12          {
13                  melody( 1 )
14                  SERVO_TorqCtrl[254]=96
15                  InitPosition16()
16                  delay( 2000 )
```

◆ 연산자 우선순위

순위	명칭	연산자 종류	결합방향
1	1차 연산자	() {} . –>	→
2	단항 연산자	+ – ! ~ (type) sizeof ++ –– & *	←
3	승제	* / %	→
4	가감	+ –	
5	Shift 연산자	〈〈 〉〉	
6	관계연산자	〈 〉 〈= 〉=	
7	등가 연산자	== !=	
8	비트 곱	&	
9	비트 차	^	
10	비트 합	|	
11	논리곱	&&	
12	논리합	||	
13	조건 연산자	? :	→
14	대입연산자	= += –= *= /= %= 〈〈= 〉〉=&= |=	←
15	순차연산자	,	→

◆ 기억클래스

데이터 영역

힙(Heap) 영역
런타임에 메모리 할당

스택 영역
컴파일 타임에 메모리 할당

◆ 메모리 영역

지정자		유효범위	생존기간	메모리 위치	자동초기화
auto		선언된 블록 내	블록 종료 시	Stack	X
extern		한 모듈 전체 프로그램	프로그램 종료 시	(비)초기화 영역	O
static	내부 static	선언된 블록 내	프로그램 종료 시	(비)초기화 영역	O
	외부 static	선언된 모듈 내			
register			블록 종료 시	CPU 내 레지스터	X

코딩계획

```
// 결과값을 리턴받을 변수
// for 문을 사용하기 위해 선언한 변수
//초기화
//모터사용
//웨이브
```

프로그래밍 세부설계

```
int main()
{
        // 결과값을 리턴받을 변수
        // for 문을 사용하기 위해 선언한 변수
        // 초기화 – 로봇제어를 하기 위해 반드시 앞 부분에 선언한다.
        // 각 변수들을 초기화 한다.
        // 열고 난 이후 이상이 있는 지 확인한다.
                // 이상이 있다면 에러 출력
                // 프로그램 종료
        // 이제부터 모터를 사용 할 수 있다.
        // 똑바로 일어선다.          (차렷자세)
                // 모든 모터(16개 모터)를 0 위치로 설정함
```

```
// 0 번 모터(오른쪽 어깨)
// 1 번 모터(오른쪽 윗팔)
// 3 번 모터(왼쪽 어깨)
// 4 번 모터(왼쪽 윗팔)
// 모터 동작(해당 자세를 1000ms 동안 동작)
// 동작 대기(1500ms)
// 웨이브
// 동작 1
// 1 번 모터(오른쪽 윗팔)
// 4 번 모터(왼쪽 윗팔)
// 모터 동작(해당 자세를 100ms 동안 동작)
// 동작 대기(1000ms)
// 동작 2
// 1 번 모터(오른쪽 윗팔)
// 4 번 모터(왼쪽 윗팔)
// 모터 동작(해당 자세를 100ms 동안 동작)
// 동작 대기(1000ms)
// 동작 3
// 4 번 모터(왼쪽 윗팔)
// 5 번 모터(왼쪽 아랫팔)
// 모터 동작(해당 자세를 400ms 동안 동작)
// 동작 대기(300ms)
// 동작 4
// 2 번 모터(오른쪽 아랫팔)
// 4 번 모터(왼쪽 윗팔)
// 5 번 모터(왼쪽 아랫팔)
// 모터 동작(해당 자세를 400ms 동안 동작)
// 동작 대기(300ms)
// 동작 5
// 1 번 모터(오른쪽 윗팔)
// 2 번 모터(오른쪽 아랫팔)
// 4 번 모터(왼쪽 윗팔)
// 5 번 모터(왼쪽 아랫팔)
// 모터 동작(해당 자세를 400ms 동안 동작)
// 동작 대기(300ms)
// 동작 6
```

```
// 1 번 모터(오른쪽 윗팔)
// 2 번 모터(오른쪽 아랫팔)
// 4 번 모터(왼쪽 윗팔)
// 모터 동작(해당 자세를 400ms 동안 동작)
// 동작 대기(300ms)
// 동작 7
// 1 번 모터(오른쪽 윗팔)
// 2 번 모터(오른쪽 아랫팔)
// 모터 동작(해당 자세를 400ms 동안 동작)
// 동작 대기(500ms)
// 동작 8// 1 번 모터(오른쪽 윗팔)
// 4 번 모터(왼쪽 윗팔)
// 모터 동작(해당 자세를 100ms 동안 동작)
// 동작 대기(200ms)
// 동작 9
// 1 번 모터(오른쪽 윗팔)
// 4 번 모터(왼쪽 윗팔)
// 모터 동작(해당 자세를 100ms 동안 동작)
// 동작 대기(200ms)
// 프로그램 종료
// 제어 종료 함수를 실행한다.
```

프로그래밍 작성

```
/*
파일 명              : hvar.c
제어로봇 형태        : 16dof 휴머노이드
설명 : 모터를 하나씩 제어해서 웨이브 동작을 만드는 프로그램으로 Visual Logic Part6.
Page 87 "Motor 예제 따라하기" 와 동일한 소스
*/
#include <stdio.h>
#include "hovis.h" // 동부로봇의 api 를 사용하기위해 반드시 이렇게 선언 해 두어야 한다.
int main()
{
        int nResult;        // 결과값을 리턴받을 변수
        int i;              // for 문을 사용하기 위해 선언한 변수
```

```
// 초기화 – 로봇제어를 하기위해 반드시 앞 부분에 선언한다.
nResult = initialize(L"COM2", 115200);    // 각 변수들을 초기화 한다.
if(nResult == 0){   // 열고 난 이후 이상이 있는 지 확인한다.
        printf("Initialization Failed₩n");    // 이상이 있다면 에러 출력
        fflush(stdout);
        return 0;   // 프로그램 종료
}
// 이제부터 모터를 사용 할 수 있다.
// 똑바로 일어선다.          (차렷자세)
for(i=0;i<16;i++){
        g_fMotorPos[i]=0; // 모든 모터(16개 모터)를 0 위치로 설정
}
g_fMotorPos[0] = -90;      // 0 번 모터(오른쪽 어깨)
g_fMotorPos[1] = 90;       // 1 번 모터(오른쪽 윗팔)
g_fMotorPos[3] = -90;      // 3 번 모터(왼쪽 어깨)
g_fMotorPos[4] = 90;       // 4 번 모터(왼쪽 윗팔)
run(1000);          // 모터 동작(해당 자세를 1000ms 동안 동작)
delay(1500);                // 동작 대기(1500ms)
// 웨이브
// 동작 1
g_fMotorPos[1] = 45;                // 1 번 모터(오른쪽 윗팔)
g_fMotorPos[4] = 45;                // 4 번 모터(왼쪽 윗팔)
run(100); // 모터 동작(해당 자세를 100ms 동안 동작)
delay(1000);                        // 동작 대기(1000ms)
// 동작 2
g_fMotorPos[1] = 0;                 // 1 번 모터(오른쪽 윗팔)
g_fMotorPos[4] = 0;                 // 4 번 모터(왼쪽 윗팔)
run(100); // 모터 동작(해당 자세를 100ms 동안 동작)
delay(1000);                        // 동작 대기(1000ms)
// 동작 3
g_fMotorPos[4] = -20;               // 4 번 모터(왼쪽 윗팔)
g_fMotorPos[5] = 55;                // 5 번 모터(왼쪽 아랫팔)
run(400);// 모터 동작(해당 자세를 400ms 동안 동작)
delay(300);                         // 동작 대기(300ms)
// 동작 4
g_fMotorPos[2] = -25;               // 2 번 모터(오른쪽 아랫팔)
```

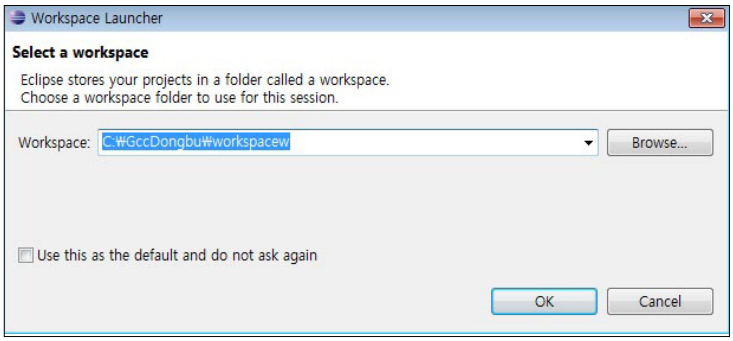
```
        g_fMotorPos[4] = 31;        // 4 번 모터(왼쪽 윗팔)
        g_fMotorPos[5] = -31;       // 5 번 모터(왼쪽 아랫팔)
        run(400);// 모터 동작(해당 자세를 400ms 동안 동작)
        delay(300);                 // 동작 대기(300ms)
        // 동작 5
        g_fMotorPos[1] = 31;        // 1 번 모터(오른쪽 윗팔)
        g_fMotorPos[2] = -31;       // 2 번 모터(오른쪽 아랫팔)
        g_fMotorPos[4] = -25;       // 4 번 모터(왼쪽 윗팔)
        g_fMotorPos[5] = 0;         // 5 번 모터(왼쪽 아랫팔)
        run(400);// 모터 동작(해당 자세를 400ms 동안 동작)
        delay(300);         // 동작 대기(300ms)
        // 동작 6
        g_fMotorPos[1] = -20;       // 1 번 모터(오른쪽 윗팔)
        g_fMotorPos[2] = 55;        // 2 번 모터(오른쪽 아랫팔)
        g_fMotorPos[4] = 0;         // 4 번 모터(왼쪽 윗팔)
        run(400);// 모터 동작(해당 자세를 400ms 동안 동작)
        delay(300);                 // 동작 대기(300ms)
        // 동작 7
        g_fMotorPos[1] = 0;         // 1 번 모터(오른쪽 윗팔)
        g_fMotorPos[2] = 0;         // 2 번 모터(오른쪽 아랫팔)
        run(400);// 모터 동작(해당 자세를 400ms 동안 동작)
        delay(500);                 // 동작 대기(500ms)
        // 동작 8
        g_fMotorPos[1] = 45;        // 1 번 모터(오른쪽 윗팔)
        g_fMotorPos[4] = 45;        // 4 번 모터(왼쪽 윗팔)
        run(100); // 모터 동작(해당 자세를 100ms 동안 동작)
        delay(200);                 // 동작 대기(200ms)
        // 동작 9
        g_fMotorPos[1] = 90;        // 1 번 모터(오른쪽 윗팔)
        g_fMotorPos[4] = 90;        // 4 번 모터(왼쪽 윗팔)
        run(100); // 모터 동작(해당 자세를 100ms 동안 동작)
        delay(200);         // 동작 대기(200ms)
        // 프로그램 종료
        terminate();        // 제어 종료 함수를 실행한다.
        return 0;
}
```

빌드 및 실행

01 이클립스 실행

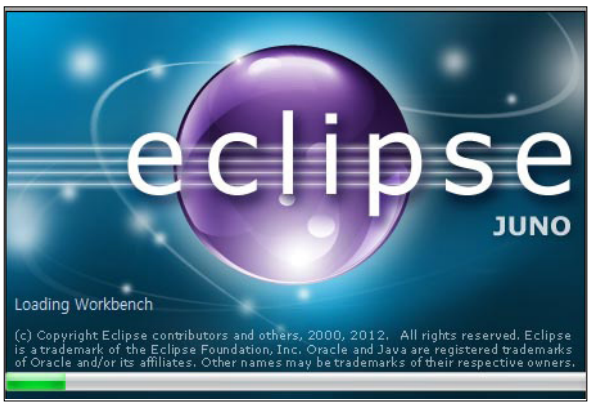
이클립스 실행 버튼을 누릅니다.

02 workspace



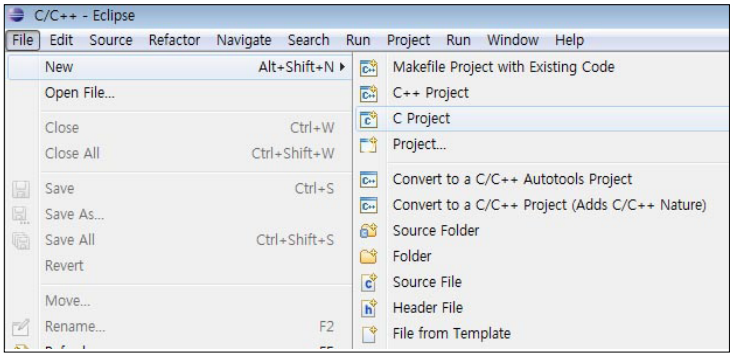
작업하고자 하는 폴더를 지정합니다. Browse 를 클릭하여 폴더를 지정하고 OK 버튼을 누릅니다.

03 이클립스 로딩 화면



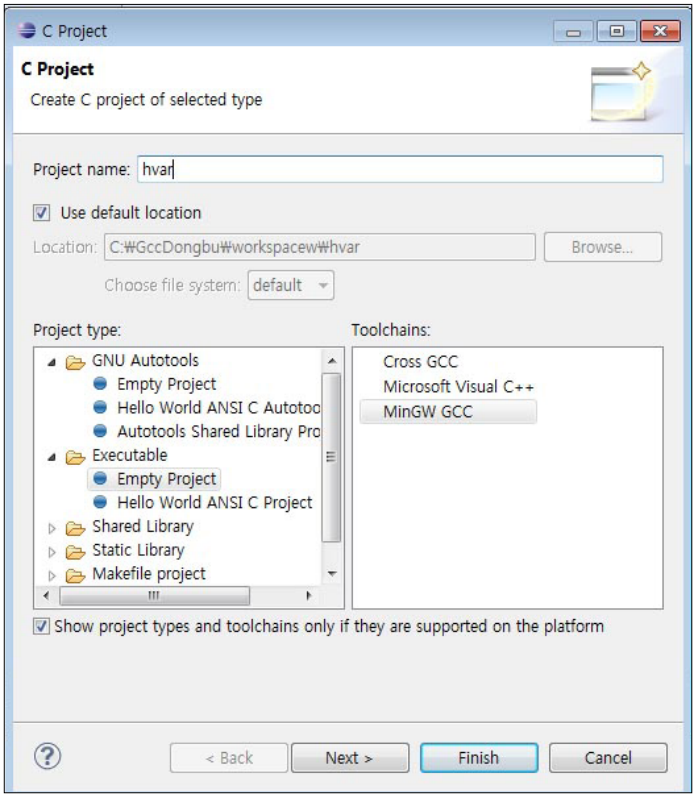
이클립스 로딩 화면입니다.

04 프로젝트 생성



이클립스 편집창 메뉴에서 File 〉 New 〉 C project 를 클릭합니다.

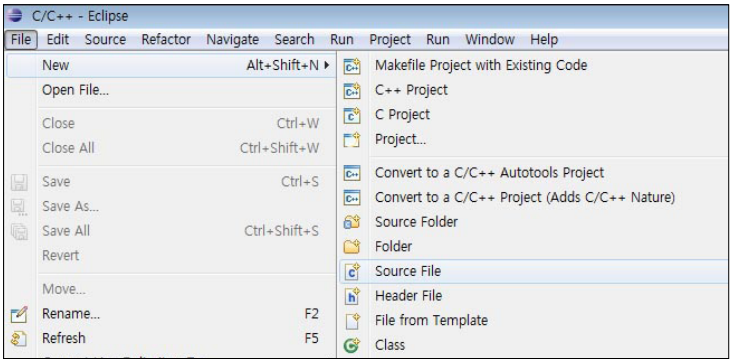
05 프로젝트 이름



Projec Name 을 hvar 라고 입력하고, Project type 에서 Exectable 에서 Empty Project 를 선택합니다.

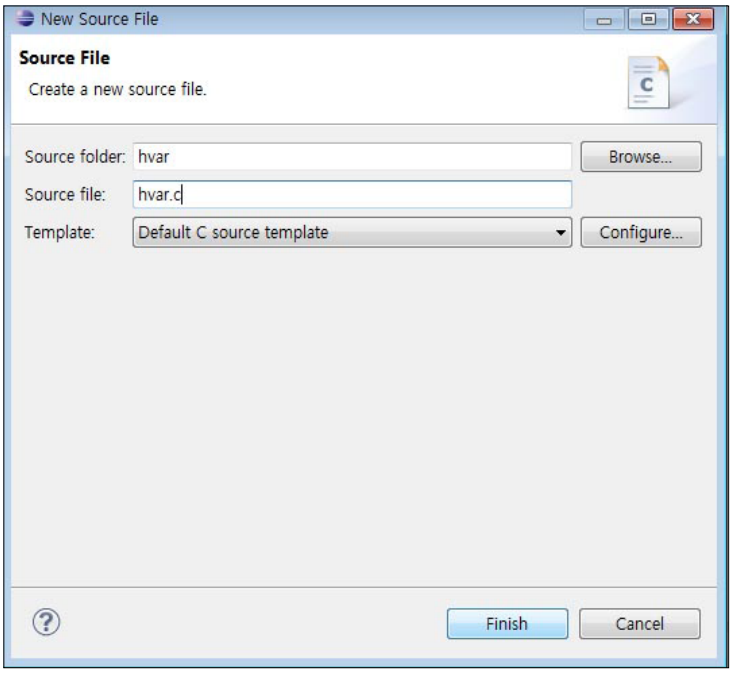
Toolchians 에서 MinGW GCC 를 선택하고 Finish 버튼을 누릅니다

06 소스파일 추가



File 〉New 〉Source File 을 클릭합니다.

07 소스파일 이름

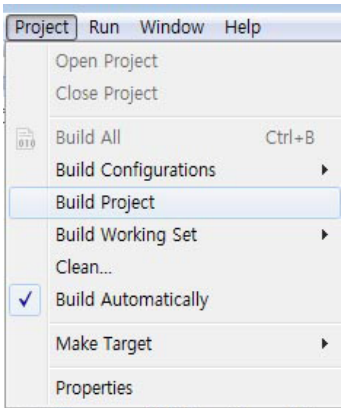


hvar.c 라고 입력하고 Finish 버튼을 누릅니다.

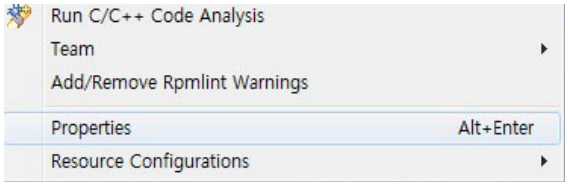
08 기본 제공 파일 추가

본 교재에게 C 프로그래밍을 하기 위해선 기본 제공되는 라이브러리 파일을 먼저 추가해주어야 올바른 컴파일을 할 수 있습니다. 필요한 파일은 drApi.dll, drApi.h, hovis.c, hovis.h 등 총 4가지 입니다.

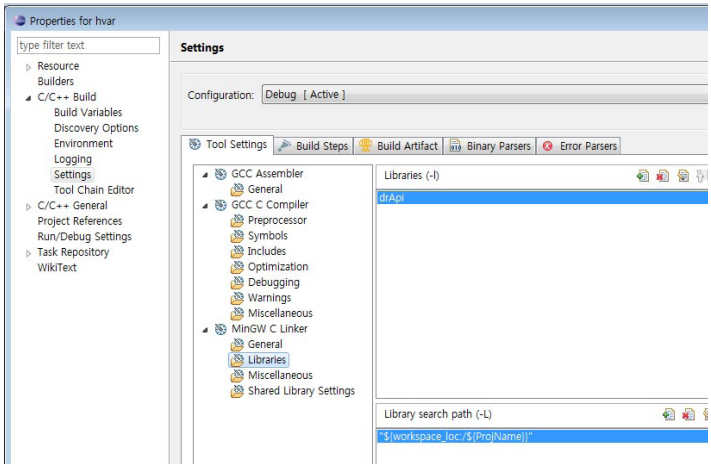
파일 추가 전에 먼저 Property 에서 라이브러리를 설정해주어야 합니다. 파일 생성을 한 후에 아무 코딩도 하지 말고, 바로 Build project 를 실행하세요. 그래야 라이브러리 추가가 가능합니다.



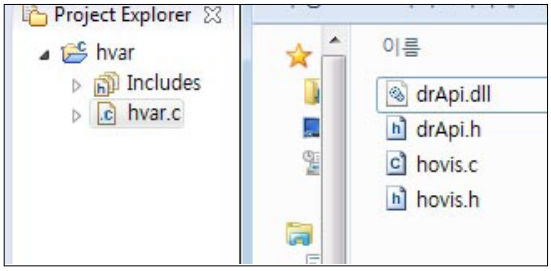
Build Project 를 클릭합니다.



좌측 프로젝트 이름에 커서를 놓고 오른쪽 마우스키를 누릅니다.
제일 하단에 Propertis 를 클릭합니다. Alt+Enter 도 동일합니다.

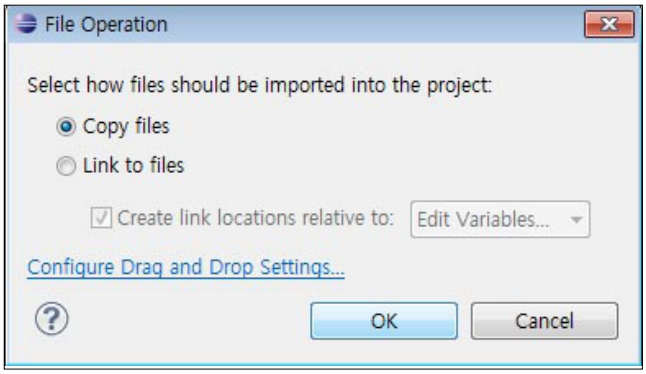


C/C++ Build 〉 Settings 를 클릭하고, 우측에 MinGW C Linker 〉 Libraties 를 클릭합니다. 우측에 Libraies 에 + 마크를 클릭하고 팝업창이 뜨면 drApi 라고 입력합니다. Library search path 에 + 마크를 클릭하고 팝업창이 뜨면 Workspace 를 클릭하고, Folder selection 에서 프로젝트 이름과 동일한 것을 클릭하고 OK 버튼을 누릅니다.



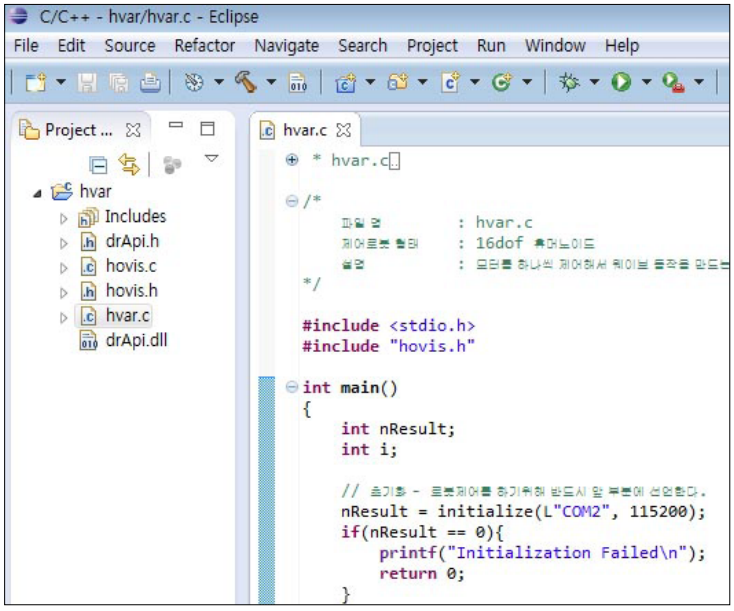
로봇을 동작시키기 위해선 위 네 개 파일을 프로젝트에 포함시켜야 합니다.
위 파일이 있는 파일탐색기 폴더에서 드래그 하는 방식으로 프로젝트에 포함시킵니다.
파일을 끌어다가 드래그해서 왼쪽 프로젝트에 추가합니다.

09 기본 파일 Import



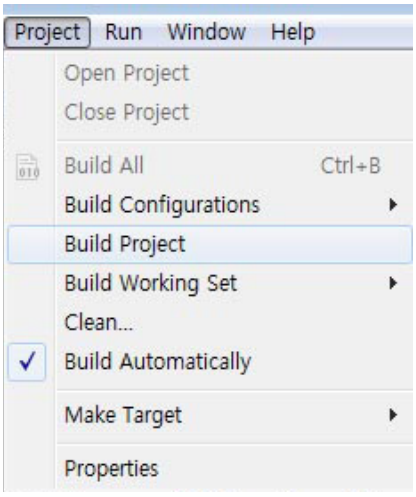
OK 버튼을 누릅니다.

10 소스 작성



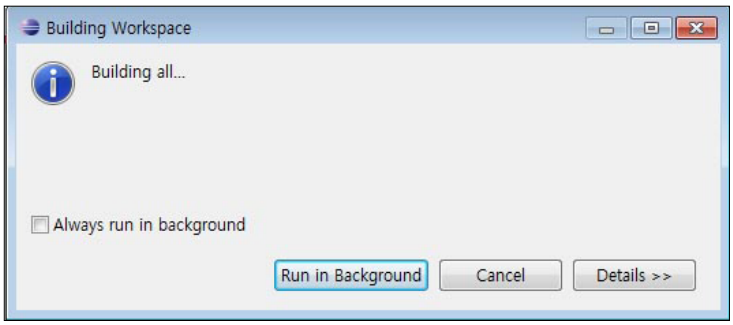
소스를 작성합니다.

11 빌드



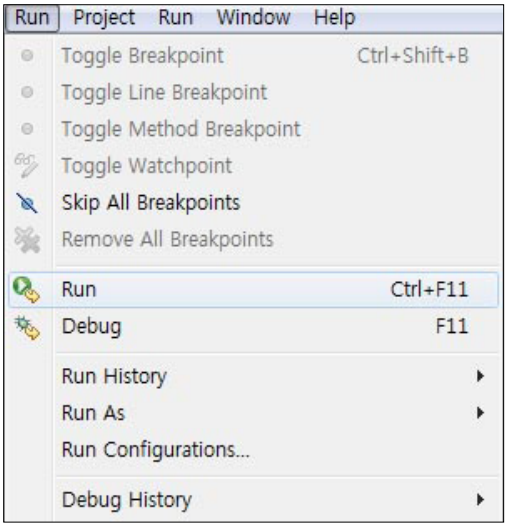
Projec 〉 Build Project 를 클릭합니다.

12 빌드중



빌드 팝업입니다. 그대로 두면 완료 후 사라집니다.

13 Run 실행



Run 〉Run 을 클릭합니다. Ctrl+F11 단축키도 동일합니다.

14 실행화면

```
        // 동작 8
        g_fMotorPos[1] = 45;
        g_fMotorPos[4] = 45;
        run(100);
        delay(200);

        // 동작 9
        g_fMotorPos[1] = 90;
        g_fMotorPos[4] = 90;
        run(100);
        delay(200);

        // 프로그램 종료
        terminate();
        return 0;
    }
```

Problems Tasks Console Properties

<terminated> hvar.exe [C/C++ Application] C:₩GccDongbu₩workspacew

15 로봇동작

로봇이 웨이브 동작합니다.

03 반복과 조건분기

반복문이란 특정 영역을 특정 조건이 만족하는 동안에 반복 실행하기 위한 문장입니니다. 반복문에는 while 문에 의한 반복, do~while 문에 의한 반복, for 문에 의한 반복 등 세가지가 있습니다.

3.1 while 반복문

while 문은 "반복의 조건"이 만족되는 동안" 반복 내용"을 반복 실행하라는 의미입니다.

```
while( 반복조건) <– true 일때 계속해서 반복내용을 반복 실행해주겠다.
{
반복내용
}
```

아래 예제는 앉았다 일어나는 모션을 기본으로 하되 한 번 일어날 때마다 일어난 숫자만큼 멜로디1을 반복 출력합니다.

hwhile.c

```
#include <stdio.h>
#include "hovis.h"
int main()
{
        int nResult;
        int i, j;
        nResult = initialize(L"COM2", 115200);
if(nResult == 0){
printf("Initialization Failed₩n");
        return 0;
}
        for(i=0;i<16;i++){
                g_fMotorPos[i]=0;
}
```

```
        g_fMotorPos[0] = –90;
        g_fMotorPos[1] = 90;
        g_fMotorPos[3] = –90;
        g_fMotorPos[4] = 90;
        run(1000);
        delay(1000);
        for(i = 0; i < 5; i++) {
                g_fMotorPos[7] = 60;
                g_fMotorPos[8] = 120;
                g_fMotorPos[9] = 60;
                g_fMotorPos[12] = 60;
                g_fMotorPos[13] = 120;
                g_fMotorPos[14] = 60;
                run(1000);
                delay(1500);
                // 일어서기 동작
                g_fMotorPos[7] = 0;
                g_fMotorPos[8] = 0;
                g_fMotorPos[9] = 0;
                g_fMotorPos[12] = 0;
                g_fMotorPos[13] = 0;
                g_fMotorPos[14] = 0;
                run(1000);
                delay(1000);
                j = i + 1;
        while(j > 0)
                {
                        g_nDrcMelody = 1;
                        run(0);
                        delay(300);
                        j--;
                }
        }
        terminate();
        return 0;
}
```

위 코드 중에 진하게 처리된 while 문을 살펴봅니다.

```
j = i + 1;
while(j > 0)
{
        g_nDrcMelody = 1;
        run(0);
        delay(300);
        j--;
        }
```

j = i + 1;일어선 횟수를 j 변수에 넣습니다. while(j>0) j가 0보다 클 동안 반복합니다. g_nDrcMelody = 1;는 Buzz 1 을 실행하도록 g_nDrcMelody 값을 설정한다는 의미입니다. 로봇을 실제로 동작 시키고 run(0); 멜로디가 울릴 동안 기다립니다. delay(300); j를 하나 감소시킵니다. j--;

문법요약

◆ 반복

ex) digital.dts 중 일부

C-like 보기

```
1  void main()
2  {
3          SERVO_TorqCtrl[254]=96
4          motionready( 0 )
5          delay( 1500 )
6          while( true )
7          {
8                  if( ( MPSU_ADCType1 == 2 && MPSU_ADCVal1 == 1 ) )
```

반복문이란 ?

※ 반복문의 기능

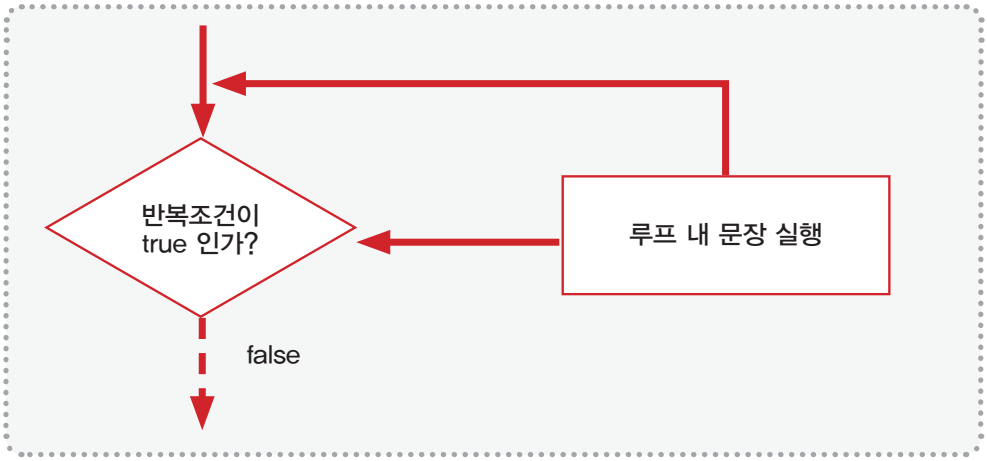
– 특정 영역을 특정 조건이 만족하는 동안에 반복 실행하기 위한 문장입니다.

※ 세가지 형태의 반복문

– while 문에 의한 반복
– do~while 문에 의한 반복
– for 문에 의한 반복

한가지만 다 이해해도 모두 잘 이해할 수 있습니다.

while 문의 순서도



일반형식

```
While(조건식) {
        반복한 문장;
}
```

● 선조건 비교 순환문입니다. (조건이 참이 아닐 경우 한번도 실행하지 못할 수 있습니다.)
● 조건식에 0(zero)가 아닌 값이 오면 C는 참으로 인식하여 무한 loop에 빠지므로 반드시 탈출문을 반복 블록 안에 포함시켜야 합니다.

코딩계획

```
// while 을 위한 변수 선언
//초기화
//모터사용
//while 문 이용 로봇 동작
```

프로그래밍 세부설계

```
/*
파일명              : hwhile.c
제어로봇 형태       : 16dof 휴머노이드
설명                : while 문을 사용하는 예제로 앉았다 일어나는 모션을 기본으로 하되
한 번 일어날 때마다 일어난 숫자만큼 멜로디1을 반복 출력한다.
*/
// 동부로봇의 api 를 사용하기 위해 반드시 이렇게 선언 해 두어야 한다.
int main()
{
        // 결과값을 리턴받을 변수
        // for, while 문을 사용하기 위해 선언한 변수

        // 초기화 – 로봇제어를 하기 위해 반드시 앞 부분에 선언한다.
        // 각 변수들을 초기화 한다.
        // 열고 난 이후 이상이 있는 지 확인한다.
                // 이상이 있다면 에러 출력
                // 프로그램 종료

        // 이제부터 모터를 사용 할 수 있다.
        // 똑바로 일어선다.          (차렷자세)

                // 모든 모터(16개 모터)를 0 위치로 설정함

        // 0 번 모터(오른쪽 어깨)
        // 1 번 모터(오른쪽 윗팔)
        // 3 번 모터(왼쪽 어깨)
        // 4 번 모터(왼쪽 윗팔)
```

```
// 모터 동작(해당 자세를 1000ms 동안 동작)
// 동작 대기(1000ms)

// for문을 사용하여 5번 반복

        // 앉기 동작
        // 7 번 모터(오른쪽 다리 앞뒤 방향)
        // 8 번 모터(오른쪽 다리 무릎)
        // 9 번 모터(오른쪽 발 앞뒤 방향)
        // 12 번 모터(왼쪽 다리 앞뒤 방향)
        // 13 번 모터(왼쪽 다리 무릎)
        // 13 번 모터(왼쪽 발 앞뒤 방향)
        // 모터 동작(해당 자세를 1000ms 동안 동작)
        // 동작 대기(1500ms)

        // 일어서기 동작
        // 7 번 모터(오른쪽 다리 앞뒤 방향)
        // 8 번 모터(오른쪽 다리 무릎)
        // 9 번 모터(오른쪽 발 앞뒤 방향)
        // 12 번 모터(왼쪽 다리 앞뒤 방향)
        // 13 번 모터(왼쪽 다리 무릎)
        // 13 번 모터(왼쪽 발 앞뒤 방향)
        // 모터 동작(해당 자세를 1000ms 동안 동작)
        // 동작 대기(1000ms)

        // 일어선 횟수를 j 변수에 넣는다.
        // j가 0보다 클 동안 반복한다.

                // Buzz 1 을 실행하도록 g_nDrcMelody 값 설정함.
                // 로봇을 실제로 동작 시킨다.
                // 멜로디가 울릴 동안 기다린다.
                // j를 하나 감소시킨다.


// 프로그램 종료
// 제어 종료 함수를 실행한다.
```

프로그래밍 작성

```
/*
파일명              : hwhile.c
제어로봇 형태       : 16dof 휴머노이드
설명 : while 문을 사용하는 예제로 앉았다 일어나는 모션을 기본으로 하되 한번 일어날 때 마
다 일어난 숫자만큼 멜로디1을 반복 출력한다.
*/

#include <stdio.h>
#include "hovis.h"  // 동부로봇의 api 를 사용하기 위해 반드시 이렇게 선언 해 두어야 한다.

int main()
{
        int nResult;                    // 결과값을 리턴받을 변수
        int i, j;               // for, while 문을 사용하기 위해 선언한 변수
        // 초기화 – 로봇제어를 하기위해 반드시 앞 부분에 선언한다.
        nResult = initialize(L"COM2", 115200);
    // 각 변수들을 초기화 한다.
        if(nResult == 0){   // 열고 난 이후 이상이 있는 지 확인한다.
                printf("Initialization Failed₩n");
                fflush(stdout);
           // 이상이 있다면 에러 출력
                return 0;
        // 프로그램 종료
        }

        // 이제부터 모터를 사용 할 수 있다.
        // 똑바로 일어선다.          (차렷자세)
        for(i=0;i<16;i++){
                g_fMotorPos[i]=0;
        // 모든 모터(16개 모터)를 0 위치로 설정함
        }
        g_fMotorPos[0] = -90;               // 0 번 모터(오른쪽 어깨)
        g_fMotorPos[1] = 90;                // 1 번 모터(오른쪽 윗팔)
        g_fMotorPos[3] = -90;               // 3 번 모터(왼쪽 어깨)
        g_fMotorPos[4] = 90;                // 4 번 모터(왼쪽 윗팔)
```

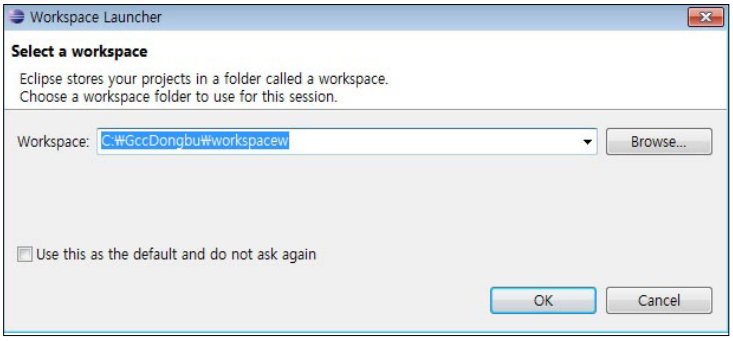
```
    run(1000);          // 모터 동작(해당 자세를 1000ms 동안 동작)
    delay(1000);        // 동작 대기(1000ms)

    // for문을 사용하여 5번 반복
    for(i = 0; i < 5; i++) {                    // 앉기 동작
        g_fMotorPos[7] = 60;// 7 번 모터(오른쪽 다리 앞뒤 방향)
        g_fMotorPos[8] = 120;       // 8 번 모터(오른쪽 다리 무릎)
        g_fMotorPos[9] = 60;        // 9 번 모터(오른쪽 발 앞뒤 방향)
        g_fMotorPos[12] = 60;       // 12 번 모터(왼쪽 다리 앞뒤 방향)
        g_fMotorPos[13] = 120;      // 13 번 모터(왼쪽 다리 무릎)
        g_fMotorPos[14] = 60;       // 13 번 모터(왼쪽 발 앞뒤 방향)
        run(1000);  // 모터 동작(해당 자세를 1000ms 동안 동작)
        delay(1500);                // 동작 대기(1500ms)

        // 일어서기 동작
        g_fMotorPos[7] = 0;  // 7 번 모터(오른쪽 다리 앞뒤 방향)
        g_fMotorPos[8] = 0;         // 8 번 모터(오른쪽 다리 무릎)
        g_fMotorPos[9] = 0;         // 9 번 모터(오른쪽 발 앞뒤 방향)
        g_fMotorPos[12] = 0;        // 12 번 모터(왼쪽 다리 앞뒤 방향)
        g_fMotorPos[13] = 0;        // 13 번 모터(왼쪽 다리 무릎)
        g_fMotorPos[14] = 0;        // 13 번 모터(왼쪽 발 앞뒤 방향)
        run(1000);   // 모터 동작(해당 자세를 1000ms 동안 동작)
        delay(1000);                // 동작 대기(1000ms)

        j = i + 1;          // 일어선 횟수를 j 변수에 넣는다.
        while(j > 0)                // j가 0보다 클 동안 반복한다.
        {
            g_nDrcMelody = 1;  // Buzz 1 을 실행하도록 g_nDrcMelody 값을 설정한다.
            run(0);             // 로봇을 실제로 동작 시킨다.
            delay(300);         // 멜로디가 울릴 동안 기다린다.
            j--;                // j를 하나 감소시킨다.
        }
    }
    // 프로그램 종료
    terminate();        // 제어 종료 함수를 실행한다.
    return 0;
}
```

빌드 및 실행

01 이클립스 실행

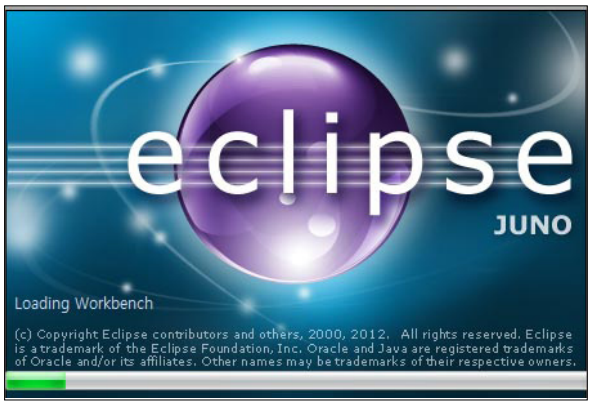
이클립스 실행 버튼을 누릅니다.

02 workspace



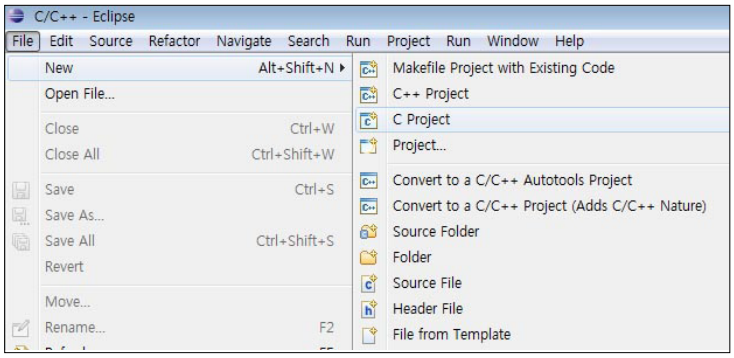
작업하고자 하는 폴더를 지정합니다. Browse 를 클릭하여 폴더를 지정하고 OK 버튼을 누릅니다.

03 이클립스 로딩 화면



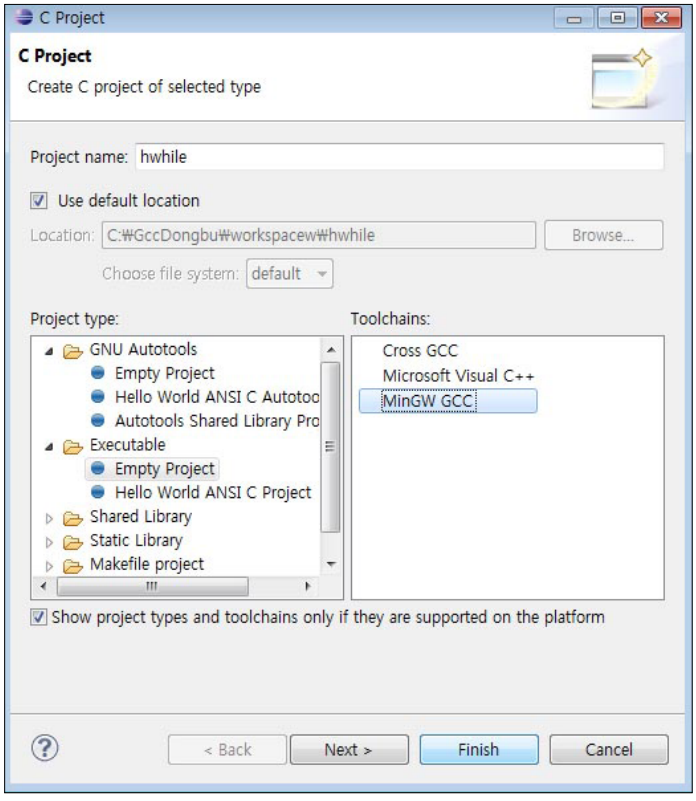
이클립스 로딩 화면입니다.

04 프로젝트 생성



이클립스 편집창 메뉴에서 File 〉 New 〉 C project 를 클릭합니다.

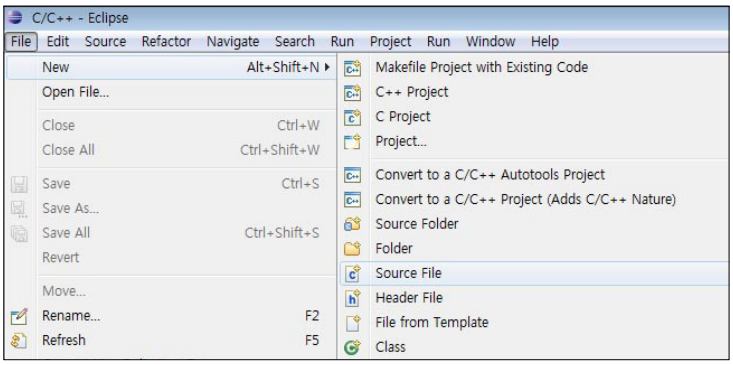
05 프로젝트 이름



Projec Name 을 hwhile 라고 입력하고, Project type 에서 Exectable 에서 Empty Project 를 선택합니다.

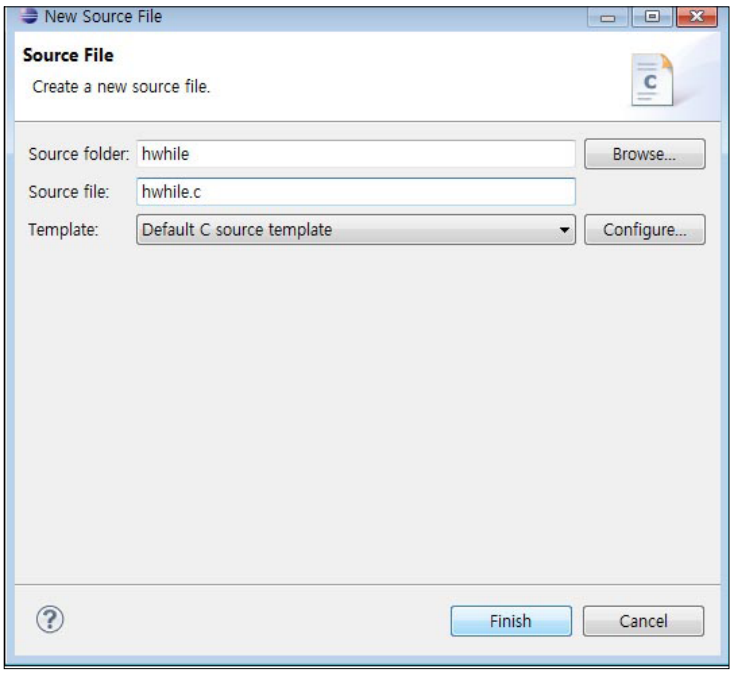
Toolchians 에서 MinGW GCC 를 선택하고 Finish 버튼을 누릅니다.

06 소스파일 추가



File 〉 New 〉 Source File 을 클릭합니다.

07 소스파일 이름

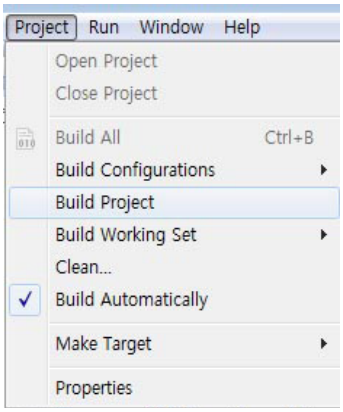


hwhile.c 라고 입력하고 Finish 버튼을 누릅니다.

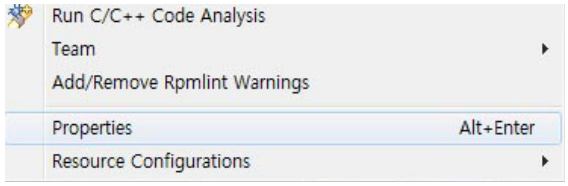
08 기본 제공 파일 추가

본 교재에게 C 프로그래밍을 하기 위해선 기본 제공되는 라이브러리 파일을 먼저 추가해주어야 올바른 컴파일을 할 수 있습니다. 필요한 파일은 drApi.dll, drApi.h, hovis.c, hovis.h 등 총 4가지 입니다.

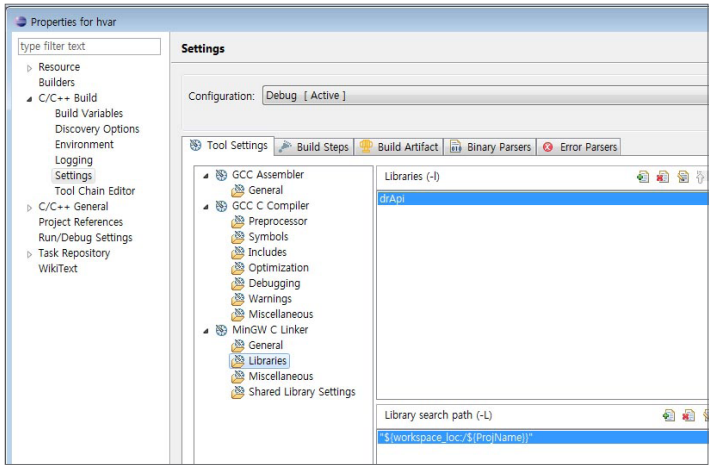
파일 추가 전에 먼저 Property 에서 라이브러리를 설정해주어야 합니다. 파일 생성을 한 후에 아무 코딩도 하지 말고, 바로 Build project 를 실행하세요. 그래야 라이브러리 추가가 가능합니다.



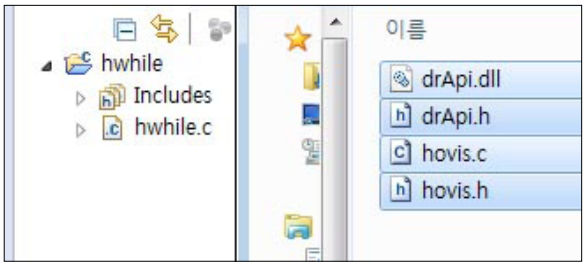
Build Project 를 클릭합니다.



좌측 프로젝트 이름에 커서를 놓고 오른쪽 마우스키를 누릅니다.
제일 하단에 Propertis 를 클릭합니다. Alt+Enter 도 동일합니다.

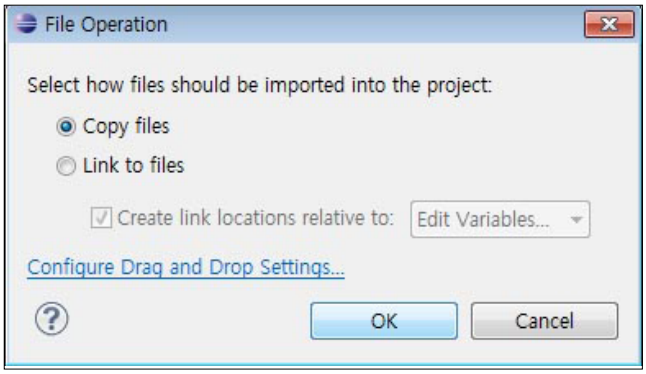


C/C++ Build 〉 Settings 를 클릭하고, 우측에 MinGW C Linker 〉 Libraties 를 클릭합니다. 우측에 Libraies 에 + 마크를 클릭하고 팝업창이 뜨면 drApi 라고 입력합니다. Library search path 에 + 마크를 클릭하고 팝업창이 뜨면 Workspace 를 클릭하고, Folder selection 에서 프로젝트 이름과 동일한 것을 클릭하고 OK 버튼을 누릅니다.



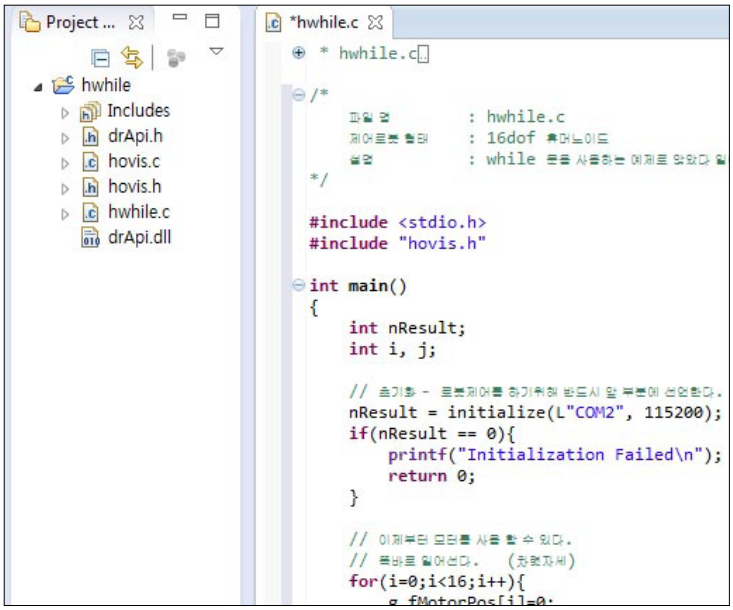
로봇을 동작시키기 위해선 위 네 개 파일을 프로젝트에 포함시켜야 합니다.
위 파일이 있는 파일탐색기 폴더에서 드래그 하는 방식으로 프로젝트에 포함시킵니다.
파일을 끌어다가 드래그해서 왼쪽 프로젝트에 추가합니다.

09 기본 파일 Import



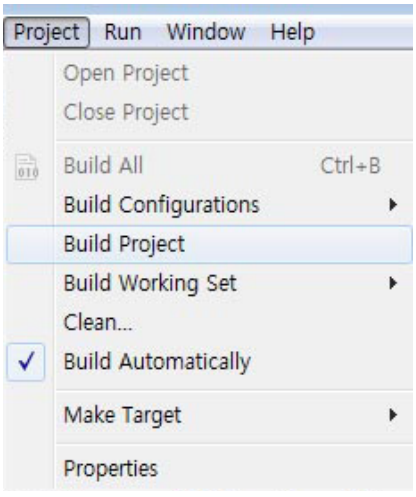
OK 버튼을 누릅니다.

10 소스 작성



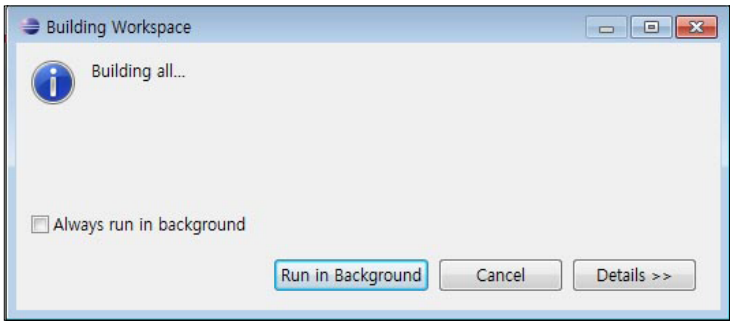
소스를 작성합니다.

11 빌드



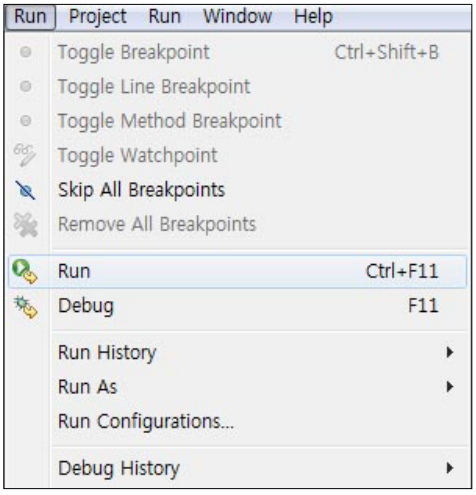
Project 〉 Build Project 를 클릭합니다.

12 빌드중



빌드 팝업입니다. 그대로 두면 완료 후 사라집니다.

13 Run 실행



Run 〉 Run 을 클릭합니다. Ctrl+F11 단축키도 동일합니다.

14 실행화면

```
        j = i + 1;
        while(j > 0)
        {
            g_nDrcMelody = 1;
            run(0);
            delay(300);
            j--;
        }
    }

    // 프로그램 종료
    terminate();
    return 0;
}
```

Problems Tasks Console Properties

<terminated> hwhile.exe [C/C++ Application] C:₩GccDongbu₩works

15 로봇동작

앉았다 일어나는 모션을 기본으로 하되 한 번 일어날 때마다 일어난 숫자만큼 멜로디1을 반복 출력합니다.

3.2 do ~ while 반복문

do ~ while 문과 while 문의 차이점은 do~while 문은 일단 한번 실행하고 나서 조건 검사를 진행하지만 while 조건검사가 안맞으면 처음부터 실행하지 않을 수 있습니다.
do~while 은 일단 먼저 반복하고 반복의 조건이 맞는지 봅니다.

```
do
{
반복 내용
} while (반복의 조건);
```

경우에 따라서는 먼저 실행하는게 자연스러운게 있습니다.

아래 예제는 조건과 상관없이 반드시 한번은 실행해야 하는 do ~ while 문을 사용하는 예제로 모션 중 원투펀치를 날리는 권투동작을 하면서 한번 실행시 좌측으로 이동하며, 총 5번 좌측으로 이동한 후 동작을 멈추는 함수입니다.

hdowhile.c

```
#include <stdio.h>
#include "hovis.h"

int main()
{
        int nResult;
        int i;
        nResult = initialize(L"COM2", 115200);
        if(nResult == 0){
                printf("Initialization Failed₩n");
                fflush(stdout);
                return 0;
        }

        for(i=0;i<16;i++){
                g_fMotorPos[i]=0;
        }
        g_fMotorPos[0] = -90;
        g_fMotorPos[1] = 90;
        g_fMotorPos[3] = -90;
        g_fMotorPos[4] = 90;
```

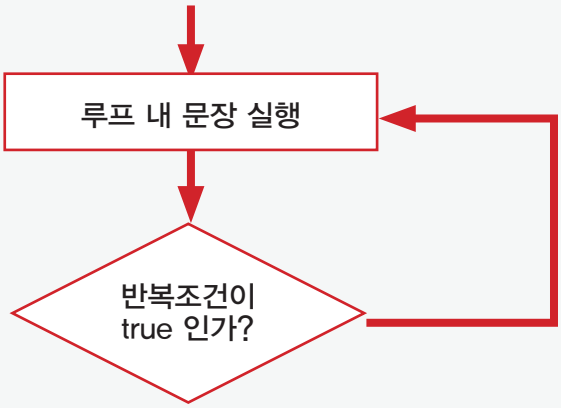
```
        run(1000);
        delay(1000);

        i = 4;
        do {
                motion(0, 0);
                motion_wait();
        }while(i-- > 0);
terminate();
return 0;
}
```

위 코드 중에 진하게 처리된 do~while 을 살펴봅니다.

```
do {
        motion(0, 0);       // 모션 0번을 실행(준비 자세 없이)
        motion_wait();      // 모션이 종료될 때까지 대기
}while(i-- > 0);
// i 값이 4, 3, 2, 1, 0 으로 계속 변경하나 동작 후 판단을 하기 때문에
   5회 반복을 하게 됩니다.
```

문법요약

do~while 문의 순서도

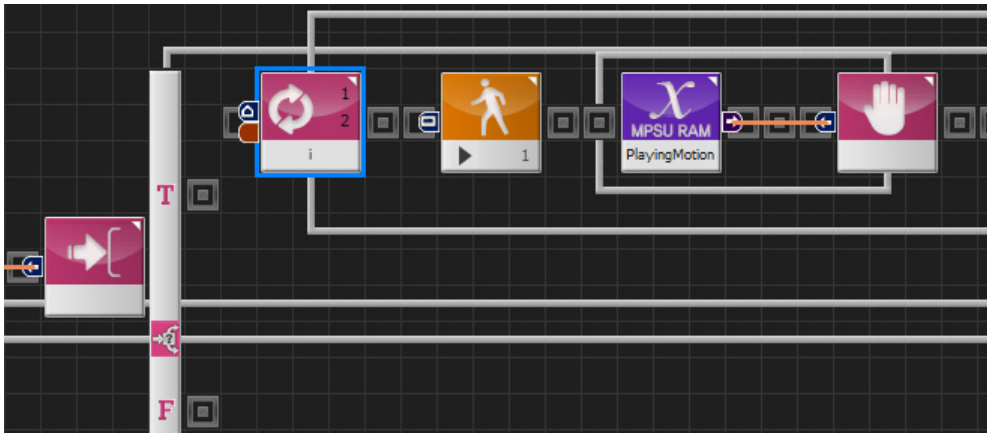


일반형식

```
do {
        반복한 문장;
} while(조건식);
```

● 후조건 비교 순환문이다. (조건이 참이 아닐 경우라도 한번은 실행한다.)

ex) digital.dts 중 일부



C-like 보기

```
if( ( MPSU_ADCType1 == 2 && MPSU_ADCVal1 == 0 ) )
{
        for( i = 1 ~ 2 )
        {
                motion( 1 )
```

코딩계획

```
// do~while 을 받기 위한 변수 선언
//초기화
//모터사용
//do~while 문 이용 로봇 동작
```

프로그래밍 세부설계

```
/*
파일 명              : hdowhile.c
제어로봇 형태        : 16dof 휴머노이드
설명                 : 조건과 상관없이 반드시 한번은 실행해야 하는 do ~ while 문을 사용하는 예제로 모션 중 원
투펀치를 날리는 권투동작을 하면서 한번 실행시 좌측으로 이동하며, 총 5번 좌측으로 이동한 후 동작을 멈추는 함수*/

// 동부로봇의 api 를 사용하기위해 반드시 이렇게 선언 해 두어야 한다.
int main()
{
        // 결과값을 리턴받을 변수
        // do-while 문을 사용하기 위해 선언한 변수
        // 초기화 – 로봇제어를 하기위해 반드시 앞 부분에 선언한다.
        // 각 변수들을 초기화 한다.
        // 열고 난 이후 이상이 있는 지 확인한다.
                // 이상이 있다면 에러 출력
                // 프로그램 종료

        // 이제부터 모터를 사용 할 수 있다.
        // 똑바로 일어선다.          (차렷자세)

                // 모든 모터(16개 모터)를 0 위치로 설정함

        // 0 번 모터(오른쪽 어깨)
        // 1 번 모터(오른쪽 윗팔)
        // 3 번 모터(왼쪽 어깨)
        // 4 번 모터(왼쪽 윗팔)
        // 모터 동작(해당 자세를 1000ms 동안 동작)
        // 동작 대기(1000ms)

                // 모션 0번을 실행(준비 자세 없이)
                // 모션이 종료될 때까지 대기

        // i 값이 4, 3, 2, 1, 0 으로 계속 변경하나 동작 후 판단을 하기 때문에 5회 반복을 하게 된다.

        // 프로그램 종료
        // 제어 종료 함수를 실행한다.
```

프로그래밍 작성

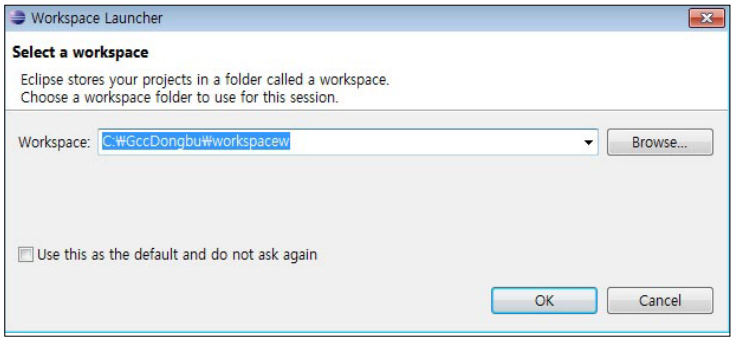
```
/*
파일 명              : hdowhile.c
제어로봇 형태        : 16dof 휴머노이드
설명                 : 조건과 상관없이 반드시 한번은 실행해야 하는 do ~ while 문을 사용하는 예제로 모션 중 원투
펀치를 날리는 권투동작을 하면서 한번 실행시 좌측으로 이동하며, 총 5번 좌측으로 이동한 후 동작을 멈추는 함수*/

#include 〈stdio.h〉
#include "hovis.h"
        // 동부로봇의 api 를 사용하기위해 반드시 이렇게 선언 해 두어야 한다.

int main()
{

        int nResult;                    // 결과값을 리턴받을 변수
        int i;                  // do-while 문을 사용하기 위해 선언한 변수

        // 초기화 - 로봇제어를 하기위해 반드시 앞 부분에 선언한다.
        nResult = initialize(L"COM2", 115200);          // 각 변수들을 초기화 한다.
        if(nResult == 0){   // 열고 난 이후 이상이 있는 지 확인한다.
                printf("Initialization Failed₩n");      // 이상이 있다면 에러 출력
                fflush(stdout);
                return 0; // 프로그램 종료
        }

        // 이제부터 모터를 사용 할 수 있다.
        // 똑바로 일어선다.         (차렷자세)
        for(i=0;i〈16;i++){
                g_fMotorPos[i]=0;// 모든 모터(16개 모터)를 0 위치로 설정
        }
        g_fMotorPos[0] = -90;                   // 0 번 모터(오른쪽 어깨)
        g_fMotorPos[1] = 90;                    // 1 번 모터(오른쪽 윗팔)
        g_fMotorPos[3] = -90;                   // 3 번 모터(왼쪽 어깨)
        g_fMotorPos[4] = 90;                    // 4 번 모터(왼쪽 윗팔)
        run(1000);          // 모터 동작(해당 자세를 1000ms 동안 동작)
        delay(1000);                            // 동작 대기(1000ms)

        i = 4;
        do {
                motion(0, 0);                   // 모션 0번을 실행(준비 자세 없이)
                motion_wait();                  // 모션이 종료될 때까지 대기
        }while(i-- 〉 0);    // i 값이 4, 3, 2, 1, 0 으로 계속 변경하나 동작 후 판단을 하기 때문에 5회 반복을 하게 된다.

        // 프로그램 종료
        terminate();        // 제어 종료 함수를 실행한다.
        return 0;
}
```

빌드 및 실행

01 이클립스 실행

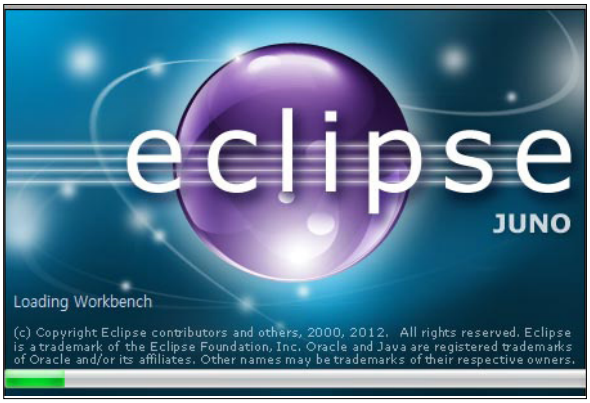
이클립스 실행 버튼을 누릅니다.

02 workspace



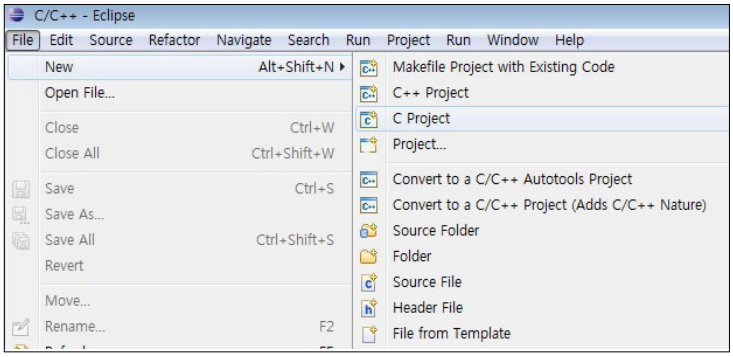
작업하고자 하는 폴더를 지정합니다. Browse 를 클릭하여 폴더를 지정하고 OK 버튼을 누릅니다.

03 이클립스 로딩 화면



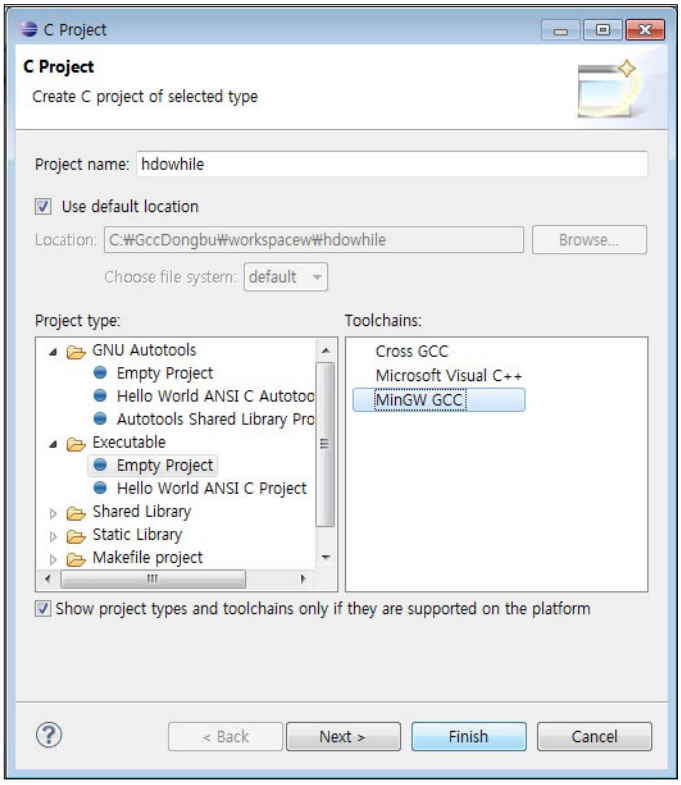
이클립스 로딩 화면입니다.

04 프로젝트 생성



이클립스 편집창 메뉴에서 File 〉 New 〉 C project 를 클릭합니다.

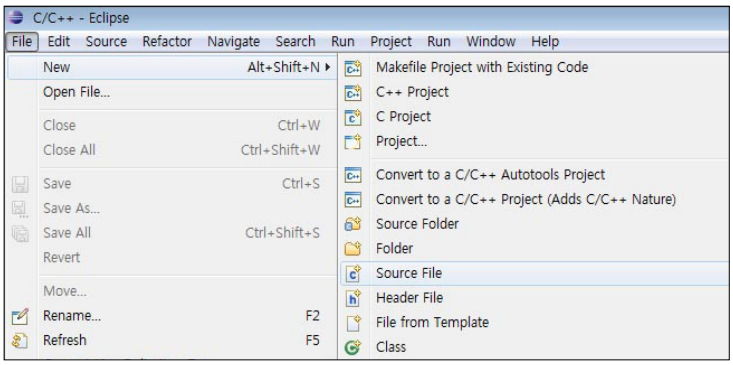
05 프로젝트 이름



Projec Name 을 hdowhile 라고 입력하고, Project type 에서 Exectable 에서 Empty Project 를 선택합니다.

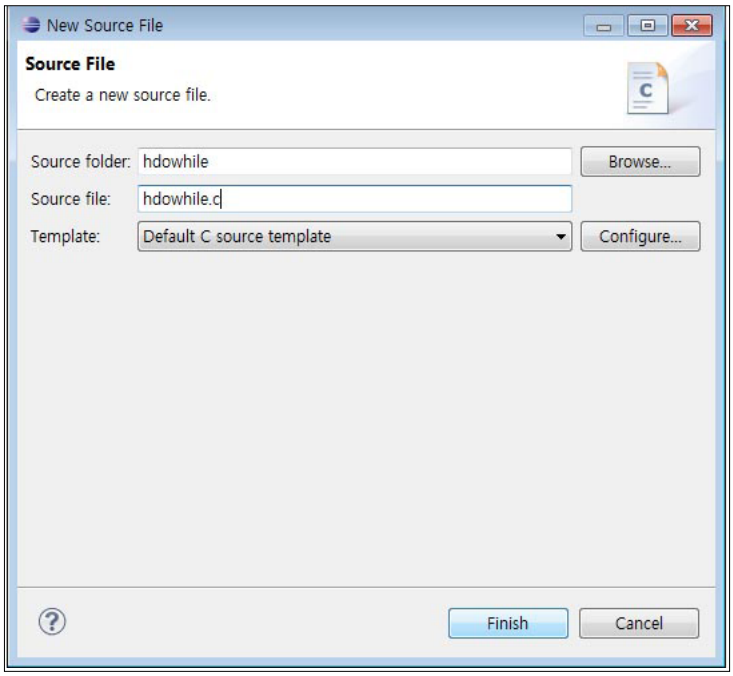
Toolchians 에서 MinGW GCC 를 선택하고 Finish 버튼을 누릅니다

06 소스파일 추가



File 〉New 〉Source File 을 클릭합니다.

07 소스파일 이름

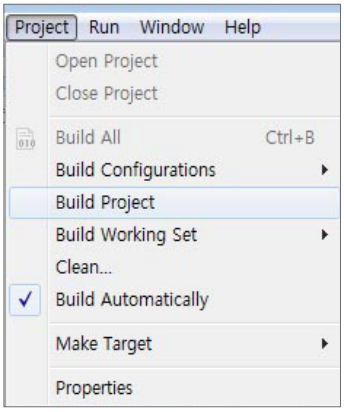


hdowhile.c 라고 입력하고 Finish 버튼을 누릅니다.

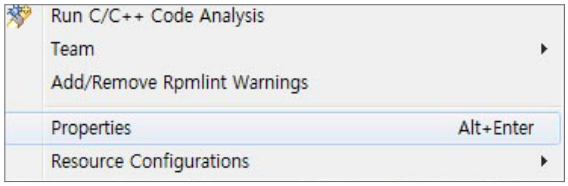
08 기본 제공 파일 추가

본 교재에게 C 프로그래밍을 하기 위해선 기본 제공되는 라이브러리 파일을 먼저 추가해주어야 올바른 컴파일을 할 수 있습니다. 필요한 파일은 drApi.dll, drApi.h, hovis.c, hovis.h 등 총 4가지 입니다.

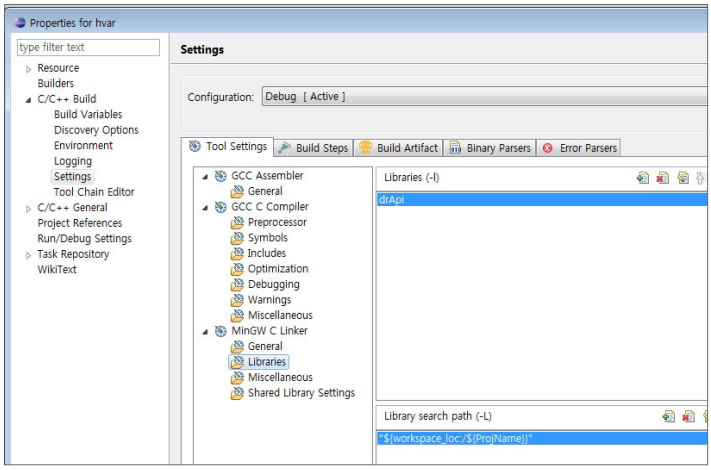
파일 추가 전에 먼저 Property 에서 라이브러리를 설정해주어야 합니다. 파일 생성을 한 후에 아무 코딩도 하지 말고, 바로 Build project 를 실행하세요. 그래야 라이브러리 추가가 가능합니다.



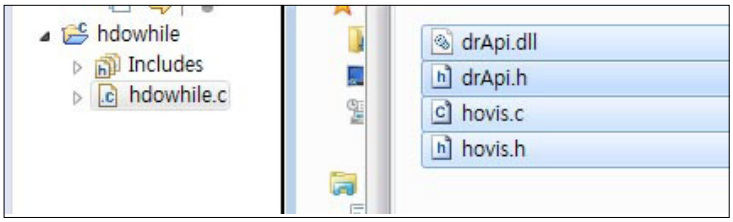
Build Project 를 클릭합니다.



좌측 프로젝트 이름에 커서를 놓고 오른쪽 마우스키를 누릅니다.
제일 하단에 Propertis 를 클릭합니다. Alt+Enter 도 동일합니다.

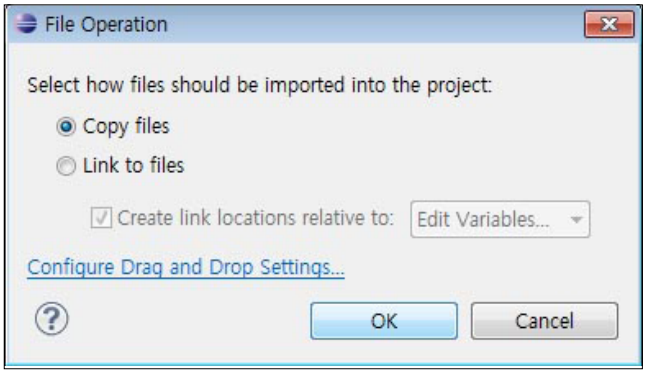


C/C++ Build 〉 Settings 를 클릭하고, 우측에 MinGW C Linker 〉 Libraties 를 클릭합니다. 우측에 Libraies 에 + 마크를 클릭하고 팝업창이 뜨면 drApi 라고 입력합니다. Library search path 에 + 마크를 클릭하고 팝업창이 뜨면 Workspace 를 클릭하고, Folder selection 에서 프로젝트 이름과 동일한 것을 클릭하고 OK 버튼을 누릅니다.



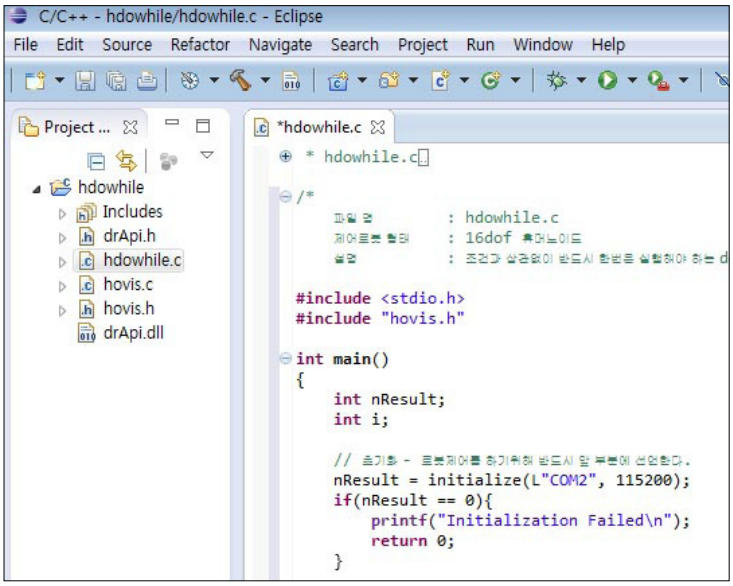
로봇을 동작시키기 위해선 위 네 개 파일을 프로젝트에 포함시켜야 합니다.
위 파일이 있는 파일탐색기 폴더에서 드래그 하는 방식으로 프로젝트에 포함시킵니다.
파일을 끌어다가 드래그해서 왼쪽 프로젝트에 추가합니다.

09 기본 파일 Import



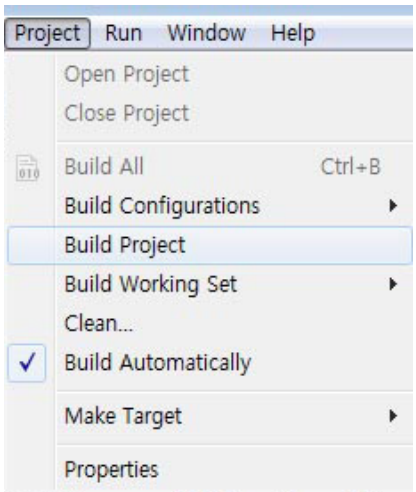
OK 버튼을 누릅니다.

10 소스 작성



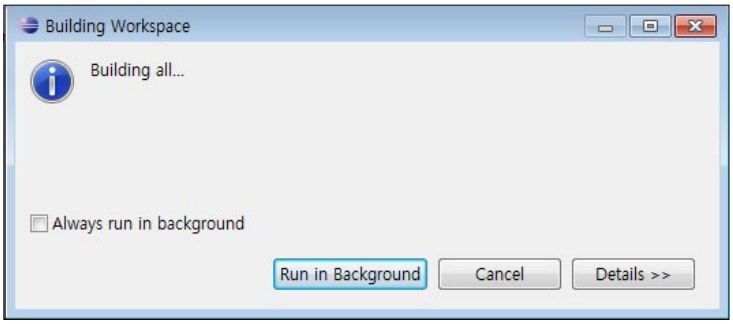
소스를 작성합니다.

11 빌드



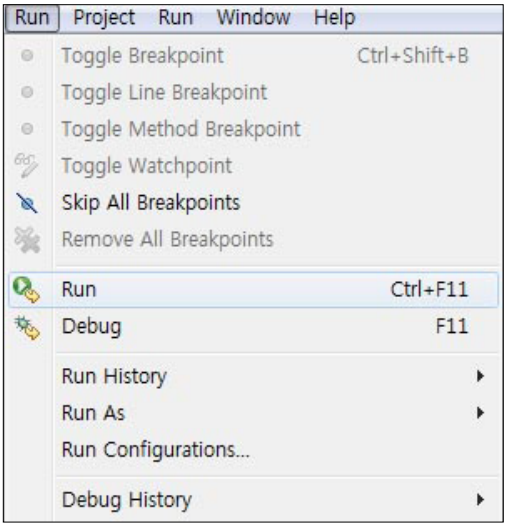
Projec 〉 Build Project 를 클릭합니다.

12 빌드중



빌드 팝업입니다. 그대로 두면 완료 후 사라집니다.

13 Run 실행



Run 〉 Run 을 클릭합니다. Ctrl+F11 단축키도 동일합니다.

14 실행화면

```
    i = 4;
    do {
        motion(0, 0);
        motion_wait();
    }while(i-- > 0);

    // 프로그램 종료
    terminate();
    return 0;
}
```

Problems Tasks Console Properties

hdowhile.exe [C/C++ Application] C:₩GccDongbu₩workspace

15 로봇동작

모션 중 원투펀치를 날리는 권투동작을 하면서 한 번 실행 시 좌측으로 이동하며, 총 5번 좌측으로 이동한 후 동작을 멈추는 동작을 합니다.

3.3 for 반복문

for 문은 가장 많이 사용되는 반복문입니다. 초기문, 조건문, 증감문 모두를 기본적으로 포함있습니다.

```
for(초기문;조건문;증감문)
{
반복하고자 하는 내용
}
```

아래 예제는 중첩 for 문을 써서 모션과 소리가 출력되게 만드는 프로그램입니다.

hfor.c

```
#include 〈stdio.h〉
#include "hovis.h"

int main()
{
        int nResult;
        int i, j;
        nResult = initialize(L"COM2", 115200);
        if(nResult == 0){
                printf("Initialization Failed₩n");
                 fflush(stdout);
                return 0;
        }
        for(i=0;i〈16;i++){
                g_fMotorPos[i]=0;
        }
        g_fMotorPos[0] = -90;
        g_fMotorPos[1] = 90;
        g_fMotorPos[3] = -90;
        g_fMotorPos[4] = 90;
        run(1000);
        delay(1000);
        for (i = 0; i 〈 10; i++)
        {
                motion(0, 0);
                motion_wait();
```

```
            for (j = 0; j < 2; j++)
            {
            g_nDrcMelody = 1;
                    run(0);
                    dr_wait_delay(300);
            }
        }
        terminate();
        return 0;
}
```

위 코드 중 진하게 처리된 for 문을 살펴봅니다.

총 10번 반복 위한 for 문입니다. for (i = 0; i < 10; i++)

멜로디 출력 2회를 위한 for 문입니다. for (j = 0; j < 2; j++)

문법요약

for 문

일반형식

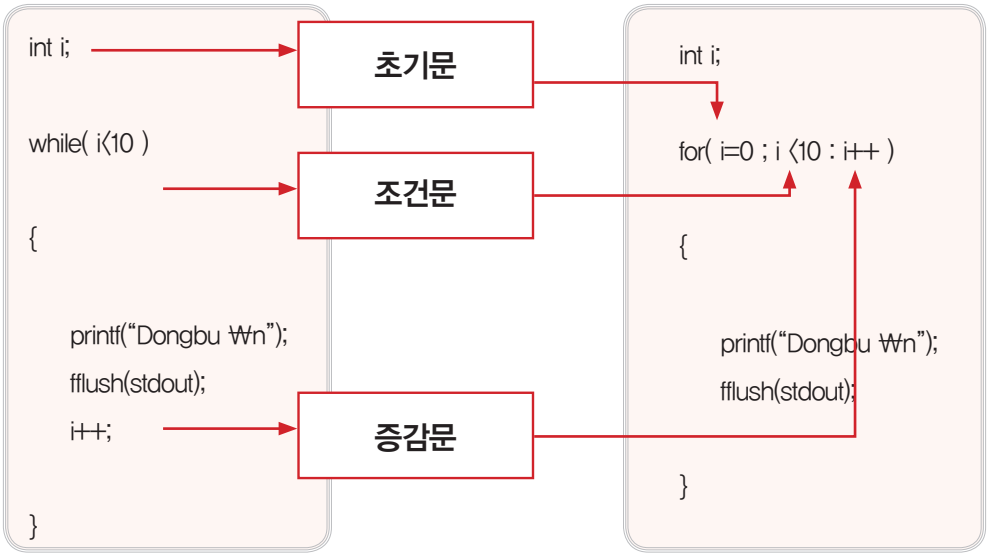
```
for (초기식 ; 조건식 ; 증감식) {
        반복할 문장;
}
```

- 초기식
  – Loop제어변수를 초기화 한다.(처음 한번만 수행)
- 조건식
  – Loop제어변수의 범위를 검사하여 반복여부를 결정한다.
- 증감식
  – Loop제어변수를 증가 또는 감소한다.

다중 for 문

```
for (초기식 ; 조건식 ; 증감식) {
        for (초기식 ; 조건식 ; 증감식) {
                반복할 문장 1;
        }
        반복할 문장 2;
}
```

for 와 while 문 비교




```
for(i=0;i<n+1;i++)
        total+=i;

 while 문으로 바꾸면

i=0;
while(i<n+1)
{
        total+=i;
        i++;
}
```

코딩계획

```
// for 문을 받기 위한 변수 선언
//초기화
//모터사용
//모션실행, 멜로디 실행
```

프로그래밍 세부설계

```
/*
파일명              : hfor.c
제어로봇 형태       : 16dof 휴머노이드
설명                : 중첩 for 문을 써서 모션과 소리가 출력되게 만드는 프로그램.
*/

// 동부로봇의 api 를 사용하기위해 반드시 이렇게 선언 해 두어야 한다.
int main()
{

        // 결과값을 리턴받을 변수
           // for 문을 사용하기 위해 선언한 변수

           // 초기화 – 로봇제어를 하기 위해 반드시 앞 부분에 선언한다.
           // 각 변수들을 초기화 한다.
           // 열고 난 이후 이상이 있는 지 확인한다.
                    // 이상이 있다면 에러 출력
                    // 프로그램 종료

           // 이제부터 모터를 사용 할 수 있다.
           // 똑바로 일어선다.          (차렷자세)
                    // 모든 모터(16개 모터)를 0 위치로 설정함

           // 0 번 모터(오른쪽 어깨)
           // 1 번 모터(오른쪽 윗팔)
           // 3 번 모터(왼쪽 어깨)
           // 4 번 모터(왼쪽 윗팔)
           // 모터 동작(해당 자세를 1000ms 동안 동작)
           // 동작 대기(1000ms)
           //총 10번 반복

                    // 권투모션의 실행
                    // 모션 0번을 실행(두 번째 파라미터는 준비 자세만 실행하는 모드인
지 여부이므로 0)
                    // 모션이 종료될 때까지 대기

                    // 멜로디 출력 2회

                              // Buzz 1 을 실행하도록 g_nDrcMelody 값을 설정.
                              // 로봇을 실제로 동작 시킨다.
                              // 멜로디가 울릴 동안 기다린다.

           // 프로그램 종료
           // 제어 종료 함수를 실행한다.
```

프로그래밍 작성

```
/*
파일 명              : hfor.c
제어로봇 형태        : 16dof 휴머노이드
설명                 : 중첩 for 문을 써서 모션과 소리가 출력되게 만드는 프로그램.
*/

#include <stdio.h>
#include "hovis.h"// 동부로봇의 api 를 사용하기 위해 반드시 이렇게 선언 해 두어야 한다.

int main()
{
        int nResult;                    // 결과값을 리턴받을 변수
        int i;                  // for 문을 사용하기 위해 선언한 변수

        // 초기화 – 로봇제어를 하기 위해 반드시 앞 부분에 선언한다.
        nResult = initialize(L"COM2", 115200);          // 각 변수들을 초기화 한다.
        if(nResult == 0){   // 열고 난 이후 이상이 있는 지 확인한다.
                printf("Initialization Failed₩n");          // 이상이 있다면 에러 출력
                fflush(stdout);
                return 0; // 프로그램 종료
        }

        // 이제부터 모터를 사용 할 수 있다.
        // 똑바로 일어선다.          (차렷자세)
        for(i=0;i<16;i++){
                g_fMotorPos[i]=0;// 모든 모터(16개 모터)를 0 위치로 설정
        }
        g_fMotorPos[0] = -90;                   // 0 번 모터(오른쪽 어깨)
        g_fMotorPos[1] = 90;                    // 1 번 모터(오른쪽 윗팔)
        g_fMotorPos[3] = -90;                   // 3 번 모터(왼쪽 어깨)
        g_fMotorPos[4] = 90;                    // 4 번 모터(왼쪽 윗팔)
        run(1000);          // 모터 동작(해당 자세를 1000ms 동안 동작)
        delay(1000);                            // 동작 대기(1000ms)

        //총 10번 반복
        for (i = 0; i < 10; i++)
        {
                // 권투모션의 실행
                motion(0, 0);                   // 모션 0번을 실행(두 번째 파라미터는 준
비 자세만 실행하는 모드인지 여부이므로 0)
                motion_wait();                  // 모션이 종료될 때까지 대기

                // 멜로디 출력 2회
                for (j = 0; j < 2; j++)
```

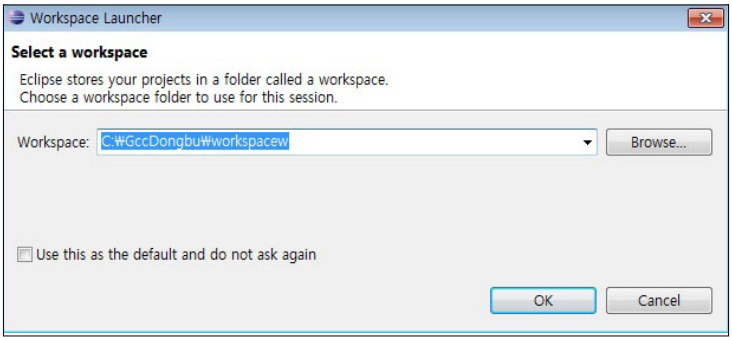
```
                {
                        g_nDrcMelody = 1;
                // Buzz 1 을 실행하도록 g_nDrcMelody 값을 설정한다.
                        run(0);    // 로봇을 실제로 동작 시킨다.
                                   // 로봇을 실제로 동작 시킨다.

                                   dr_wait_delay(300);
                        // 멜로디가 울릴 동안 기다린다.
                }
        }
        terminate();        // 제어 종료 함수를 실행한다.
        return 0;
}
```

빌드 및 실행

01 이클립스 실행

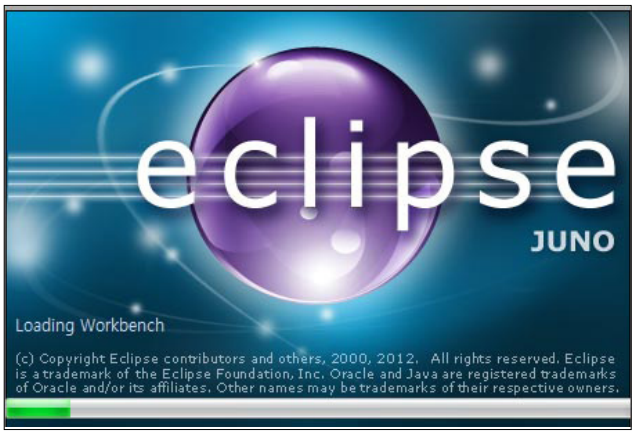
이클립스 실행 버튼을 누릅니다.

02 workspace



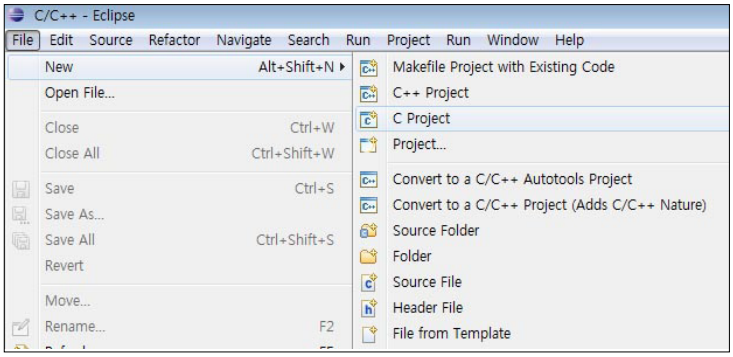
작업하고자 하는 폴더를 지정합니다. Browse 를 클릭하여 폴더를 지정하고 OK 버튼을 누릅니다.

03 이클립스 로딩 화면



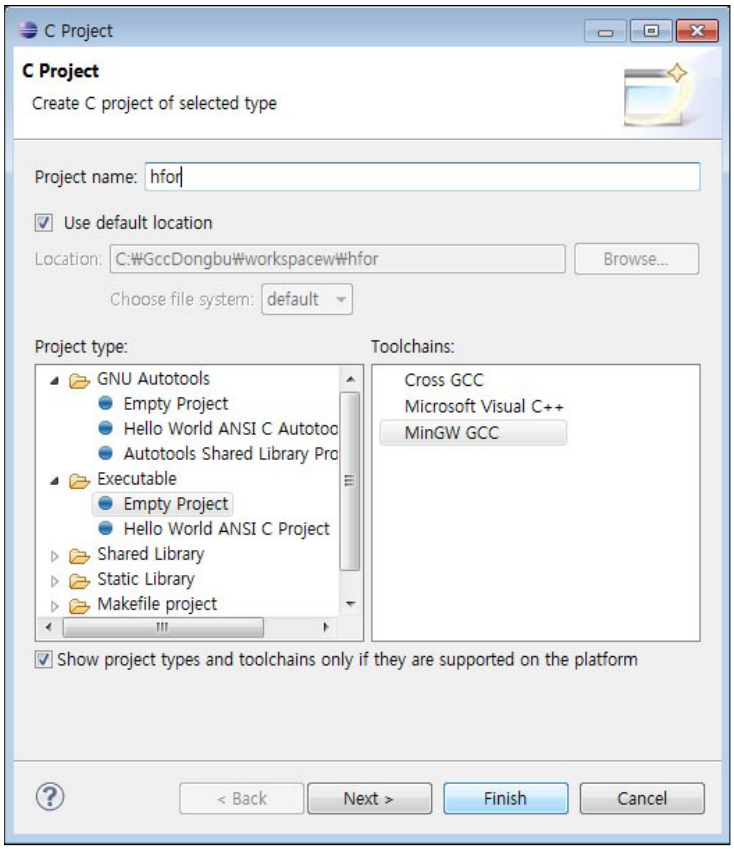
이클립스 로딩 화면입니다.

04 프로젝트 생성



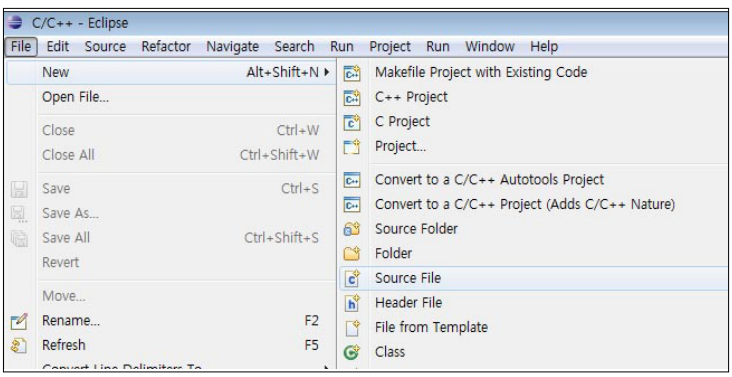
이클립스 편집창 메뉴에서 File 〉 New 〉 C project 를 클릭합니다.

05 프로젝트 이름



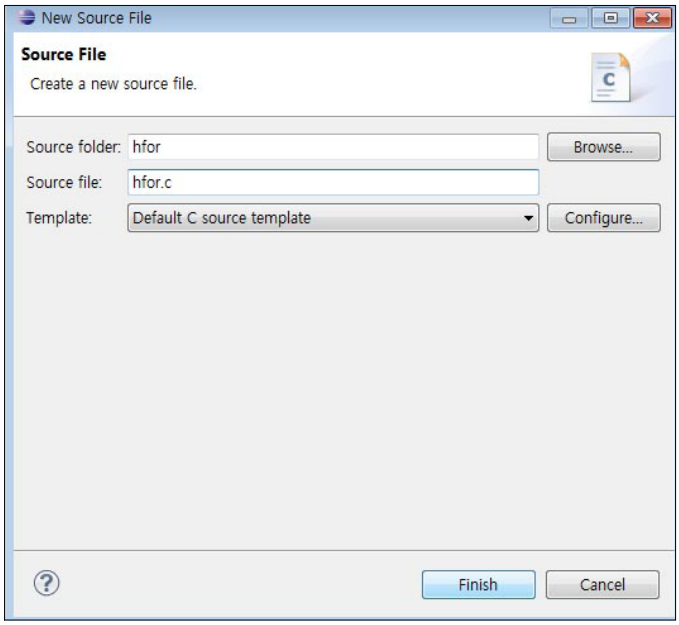
Projec Name 을 hfor 라고 입력하고, Project type 에서 Exectable 에서 Empty Project 를 선택합니다.
Toolchians 에서 MinGW GCC 를 선택하고 Finish 버튼을 누릅니다.

06 소스파일 추가



File 〉New 〉Source File 을 클릭합니다.

07 소스파일 이름

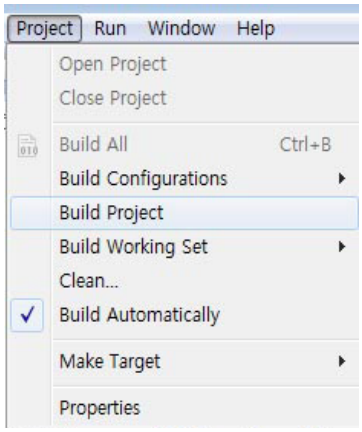


hfor.c 라고 입력하고 Finish 버튼을 누릅니다.

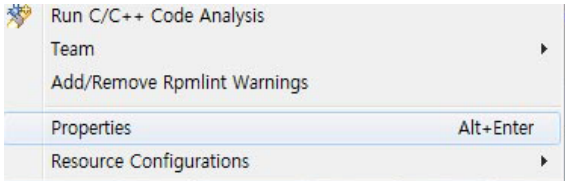
08 기본 제공 파일 추가

본 교재에게 C 프로그래밍을 하기 위해선 기본 제공되는 라이브러리 파일을 먼저 추가해주어야 올바른 컴파일을 할 수 있습니다. 필요한 파일은 drApi.dll, drApi.h, hovis.c, hovis.h 등 총 4가지 입니다.

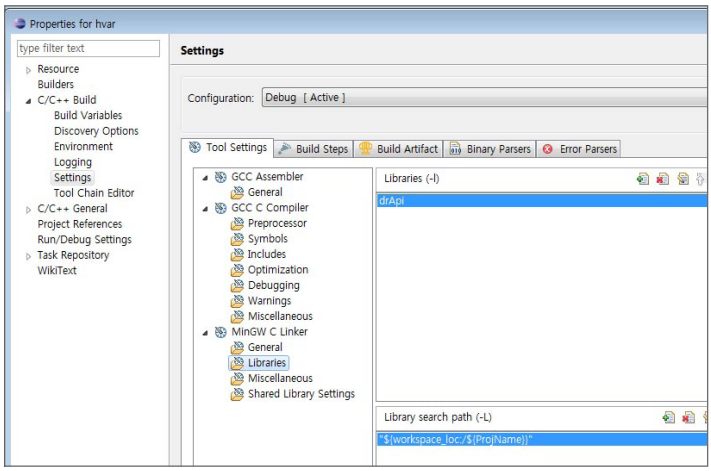
파일 추가 전에 먼저 Property 에서 라이브러리를 설정해주어야 합니다. 파일 생성을 한 후에 아무 코딩도 하지 말고, 바로 Build project 를 실행하세요. 그래야 라이브러리 추가가 가능합니다.



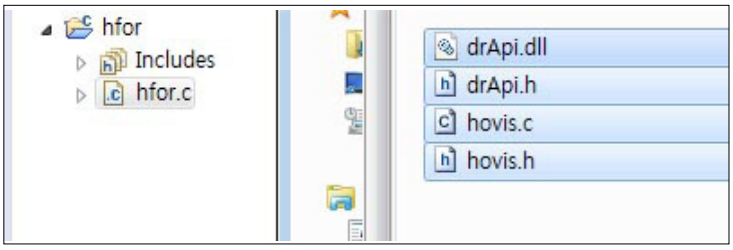
Build Project 를 클릭합니다.



좌측 프로젝트 이름에 커서를 놓고 오른쪽 마우스키를 누릅니다.
제일 하단에 Propertis 를 클릭합니다. Alt+Enter 도 동일합니다.

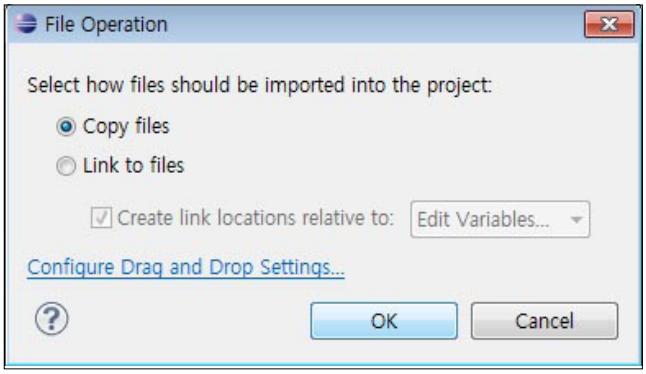


C/C++ Build 〉Settings 를 클릭하고, 우측에 MinGW C Linker 〉Libraties 를 클릭합니다. 우측에 Libraies 에 + 마크를 클릭하고 팝업창이 뜨면 drApi 라고 입력합니다. Library search path 에 + 마크를 클릭하고 팝업창이 뜨면 Workspace 를 클릭하고, Folder selection 에서 프로젝트 이름과 동일한 것을 클릭하고 OK 버튼을 누릅니다.



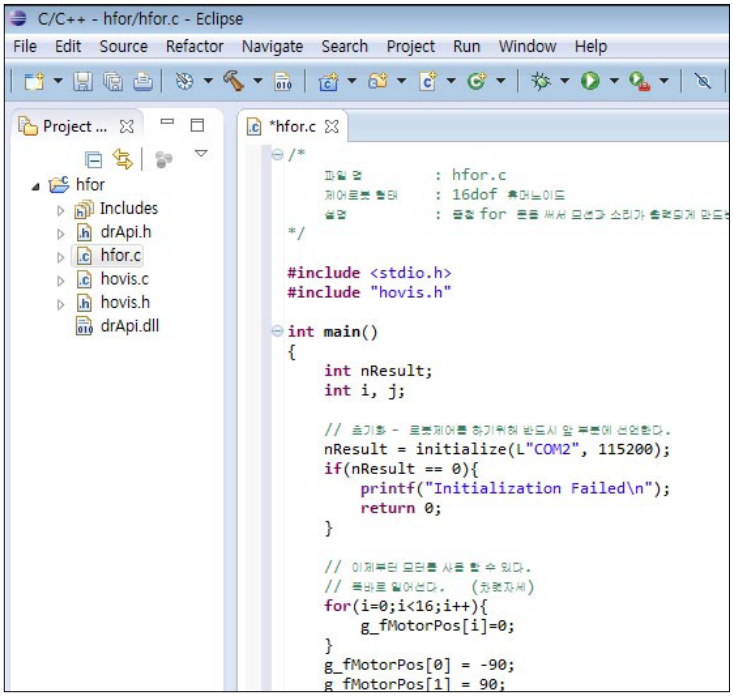
로봇을 동작시키기 위해선 위 네 개 파일을 프로젝트에 포함시켜야 합니다.
위 파일이 있는 파일탐색기 폴더에서 드래그 하는 방식으로 프로젝트에 포함시킵니다.
파일을 끌어다가 드래그해서 왼쪽 프로젝트에 추가합니다.

09 기본 파일 Import



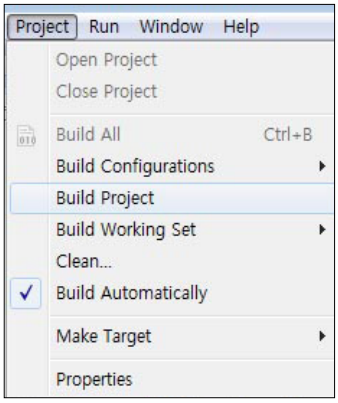
OK 버튼을 누릅니다.

10 소스 작성



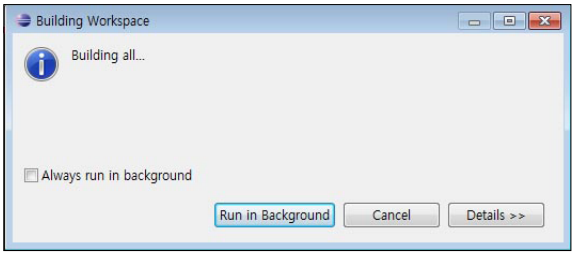
소스를 작성합니다.

11 빌드



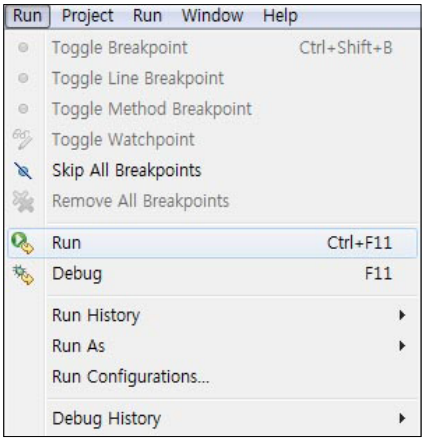
Projec 〉Build Project 를 클릭합니다.

12 빌드중



빌드 팝업입니다. 그대로 두면 완료 후 사라집니다.

13 Run 실행



Run 〉Run 을 클릭합니다. Ctrl+F11 단축키도 동일합니다.

14 실행화면

```
    //총 10번 반복
    for (i = 0; i < 10; i++)
    {
        // 원투모션의 실행
        motion(0, 0);
        motion_wait();

        // 멜로디 출력 2회
        for (j = 0; j < 2; j++)
        {
            g_nDrcMelody = 1;
            run(0);
            dr_wait_delay(300);
        }
    }

    // 프로그램 종료
    terminate();
    return 0;
}
```

Problems Tasks Console Properties

hfor.exe [C/C++ Application] C:₩GccDongbu₩workspacew₩ht

15 로봇동작

로봇에서 모션과 소리가 출력됩니다.

3.4 if 조건문

조건에 따른 흐름의 분기가 필요한 이유는 상황에 따라 프로그램의 유연성을 부여하기 위함입니다. 사용자의 입력에 따라서, 즉 어떤 조건에 의해서 프로그램의 흐름을 변경하게 만드는 것이 조건분기입니다.
if 문은 조건이 만족되는 경우에 한해서 실행되고,if (실행의조건) 가 true 라고 가정하면 실행하고자 하는 문장이 한 문장일 때 중괄호 생략이 가능합니다.

아래 예제는 PSD 앞에 무언가가 얼마나 가까운지를 화면상으로 출력하고 10 cm 이내로 접근 시 권투 모션을 실행하고 종료하는 프로그램입니다.

hif.c

```
#include <stdio.h>
#include "hovis.h"

int main()
{

        int nResult;
        int i;
        int nDistance;

        nResult = initialize(L"COM2", 115200);
if(nResult == 0){
printf("Initialization Failed₩n");
fflush(stdout);
return 0;
}

        for(i=0;i<16;i++){
                g_fMotorPos[i]=0;

}
        g_fMotorPos[0] = -90;
        g_fMotorPos[1] = 90;
        g_fMotorPos[3] = -90;
        g_fMotorPos[4] = 90;
```

```
        run(1000);

        delay(1000);

        read();
        if (g_nAdcType[0] == 1) {
                printf("PSD On The Left Side₩n");
                 fflush(stdout);
        }
        else if (g_nAdcType[1] == 1) {
                printf("PSD On The Right Side₩n");
                 fflush(stdout);
        }
        else {

                printf("PSD Not Connected₩n");
                 fflush(stdout);
        terminate();

        return 0;

        }

        do{
                read();

                if(g_nAdcType[0] == 1){
                nDistance = g_nAdcDist[0];
                printf("PSD = %d cm₩n", nDistance );
                 fflush(stdout);
                }
                else if(g_nAdcType[1] == 1){
                        nDistance = g_nAdcDist[1];
                printf("PSD = %d cm₩n", nDistance );
                 fflush(stdout);
                }
                else {
                        printf("PSD Not Connected₩n");
                        fflush(stdout);
                terminate();
                return 0;
                 }

} while (nDistance 〉 10);
motion(0, 0);

motion_wait();

terminate();

return 0;
}
```

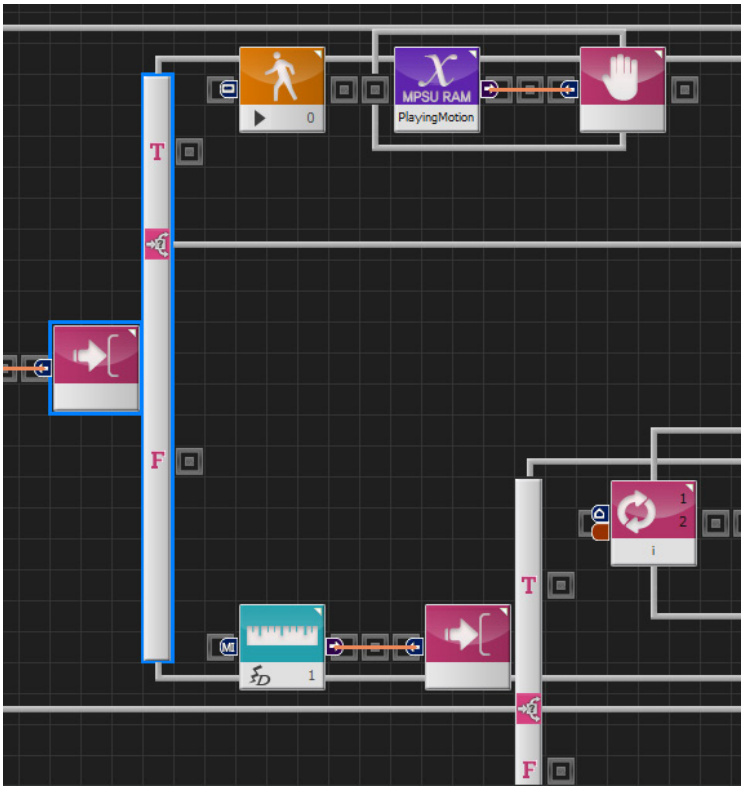
위 코드 중에 진하게 처리된 if 문장을 살펴봅니다.

```
nResult = initialize(L"COM2", 115200);     // 각 변수들을 초기화 합니다.
if(nResult == 0){                          // 열고 난 이후 이상이 있는 지 확인하는 문장입니다.
```

문법요약

◆ 조건분기

ex) digital.dts 중 일부



C-like 보기

```
8   if( ( MPSU_ADCType1 == 2 && MPSU_ADCVal1 == 1 ) )
9   {
10      motion( 0 )
11      waitwhile( MPSU_PlayingMotion )
12  }
13  else
14  {
15      if( ( MPSU_ADCType1 == 2 && MPSU_ADCVal1 == 0 ) )
16      {
```

◆ 조건 분기문

조건분기문은 if, if else, switch case 등 세가지 형태이지만, 기능은 동일하기 때문에 visual logic 에서는 if else 로보 표현했습니다.

◆ if

if문에 의한 조건적 실행 : 조건이 만족되는 경우에 한해서 실행합니다.

if(실행의조건) → true 라고 가정하면
{ – → 실행하고자 하는 문장이 한 문장일 때 중괄호 생략이 가능합니다.
실행하고자 하는 내용
}

```
int main(void)
{
  int val;
  printf("정수를 하나 입력하세요")
  scanf(%d, &val);

  if(val<0) //val <0 이 true이면…
  {
    printf("입력값은 0보다 작다"
  } → 문장이 둘 이상이면 반드시 넣어줘야함
  if(val>0)
    printf("입력값은 0보다 크다"
  if(val==0)
    printf("입력값은 0이다."

  return 0;
}
```

만약 1 을 입력하면, 두번째에서 True 이다, 따라서 세번째 조건 검사는 할 필요가 없습니다.
→ 이런 문제점을 해결하기 위해 else 를 넣습니다.

코딩계획

```
// for, 거리값 저장 변수 선언
//초기화
//모터사용
//PSD 사용
//권투모션 실행
```

프로그래밍 세부설계

```
/*
파일 명              : hif.c
제어로봇 형태        : 16dof 휴머노이드
설명                 : PSD 앞에 무언가가 얼마나 가까운 지를 화면상으로 출력하고 10 cm
이내로 접근 시 권투 모션을 실행하고 종료
*/

// 동부로봇의 api 를 사용하기 위해 반드시 이렇게 선언 해 두어야 한다.

int main()
{

        // 결과값을 리턴받을 변수
        // for 문을 사용하기 위해 선언한 변수
        // 거리 값을 저장하기 위한 변수

        // 초기화 – 로봇제어를 하기위해 반드시 앞 부분에 선언한다.
        // 각 변수들을 초기화 한다.
        // 열고 난 이후 이상이 있는 지 확인한다.
                // 이상이 있다면 에러 출력
                // 프로그램 종료

        // 이제부터 모터를 사용 할 수 있다.
        // 똑바로 일어선다.          (차렷자세)

                // 모든 모터(16개 모터)를 0 위치로 설정함

        // 0 번 모터(오른쪽 어깨)
        // 1 번 모터(오른쪽 윗팔)
        // 3 번 모터(왼쪽 어깨)
        // 4 번 모터(왼쪽 윗팔)
        // 모터 동작(해당 자세를 1000ms 동안 동작)
        // 동작 대기(1000ms)
```

```
        // PSD 가 DRC 의 어느 쪽 포트에 연결되었는지 확인
        // read() 함수를 실행하면 센서 변수들이 자동으로 업데이트 된다.
        // PSD 가 어느 부분에 조립되었는지 확인 하기 위해 통신요청.
        // g_nAdcType[0], g_nAdcType[1]은 각각 왼쪽, 오른쪽의 ADC 포트에 어떤 센서가
부착 되었는지를 나타낸다.
        // 1일 경우 PSD가 장착 되었다는 뜻이고, 0일 경우 센서가 없다는 뜻이다.
        // PSD 가 왼쪽에 위치한 컨트롤러 포트에 연결되었다면
                // 왼쪽을 나타내는 메세지를 출력한다.

        // PSD 가 오른쪽에 위치한 컨트롤러 포트에 연결되었다면
                // 오른쪽을 나타내는 메세지를 출력한다.

        // 2가지 경우가 전부 아니라면(PSD 가 연결되지 않았다면)
                // 에러 메시지를 출력한 후
                // 제어 종료 함수를 실행한다.
                // 프로그램을 종료한다.

        // PSD 값을 읽어서 10cm 이내일 경우 탈출한다.

                // PSD 센서 값을 읽기 위해 통신 요청.
                // PSD 가 왼쪽에 연결되었다면
                        // 해당 PSD 값을 nDistance에 넣는다.
                        // PSD 값을 출력한다.

                // PSD 가 오른쪽에 연결되었다면
                        // 해당 PSD 값을 nDistance에 넣는다.
                        // PSD 값을 출력한다.

                // 2가지 경우가 전부 아니라면(PSD 가 연결되지 않았다면)
                        // 에러 메시지를 출력한 후
                        // 제어 종료 함수를 실행한다.
                        // 프로그램을 종료한다.

        // 10 cm 이내로 들어온 경우 루프를 탈출한다.

        // 권투모션의 실행
        // 모션 0번을 실행(두 번째 파라미터는 준비 자세만 실행하는 모드인지 여부이므로 0)
        // 모션이 종료될 때까지 대기

        // 프로그램 종료
        // 제어 종료 함수를 실행한다.
```

프로그래밍 작성

```
/*
파일 명              : hif.c
제어로봇 형태        : 16dof 휴머노이드
설명                 : PSD 앞에 무언가가 얼마나 가까운 지를 화면상으로 출력하고 10 cm
이내로 접근 시 권투 모션을 실행하고 종료
*/

#include <stdio.h>
#include "hovis.h" // 동부로봇의 api 를 사용하기 위해 반드시 이렇게 선언 해 두어야 한다.

int main()
{

        int nResult;                    // 결과값을 리턴받을 변수
        int i;                  // for 문을 사용하기 위해 선언한 변수
        int nDistance;      // 거리 값을 저장하기 위한 변수

        // 초기화 – 로봇제어를 하기 위해 반드시 앞 부분에 선언한다.
        nResult = initialize(L"COM2", 115200);          // 각 변수들을 초기화 한다.
        if(nResult == 0){   // 열고 난 이후 이상이 있는 지 확인한다.
                printf("Initialization Failed₩n");          // 이상이 있다면 에러 출력
                fflush(stdout);
                return 0; // 프로그램 종료
        }

        // 이제부터 모터를 사용 할 수 있다.
        // 똑바로 일어선다.          (차렷자세)
        for(i=0;i<16;i++){
                g_fMotorPos[i]=0;// 모든 모터(16개 모터)를 0 위치로 설정
        }
        g_fMotorPos[0] = –90;                   // 0 번 모터(오른쪽 어깨)
        g_fMotorPos[1] = 90;                    // 1 번 모터(오른쪽 윗팔)
        g_fMotorPos[3] = –90;                   // 3 번 모터(왼쪽 어깨)
        g_fMotorPos[4] = 90;                    // 4 번 모터(왼쪽 윗팔)
        run(1000);          // 모터 동작(해당 자세를 1000ms 동안 동작)
        delay(1000);                            // 동작 대기(1000ms)
        // PSD 가 DRC 의 어느 쪽 포트에 연결되었는지 확인
        // read() 함수를 실행하면 센서 변수들이 자동으로 업데이트 된다.
```

```
read();    // PSD 가 어느 부분에 조립되었는지 확인 위해 통신요청.
// g_nAdcType[0], g_nAdcType[1]은 각각 왼쪽, 오른쪽의 ADC 포트에 어떤 센서가 부착 되었는지를 나타낸다.
// 1일 경우 PSD가 장착되었다는 뜻이고, 0일 경우 센서가 없다는 뜻이다.
if (g_nAdcType[0] == 1) { // PSD 가 왼쪽에 위치한 컨트롤러 포트에 연결되었다면
        printf("PSD On The Left Side₩n");    // 왼쪽을 나타내는 메세지를 출력한다.
        fflush(stdout);
}
else if (g_nAdcType[1] == 1) { // PSD 가 오른쪽에 위치한 컨트롤러 포트에 연결되었다면
        printf("PSD On The Right Side₩n"); // 오른쪽을 나타내는 메세지를 출력한다.
        fflush(stdout);
}
else {    // 2가지 경우가 전부 아니라면(PSD 가 연결되지 않았다면)
        printf("PSD Not Connected₩n");      // 에러 메시지를 출력한 후
        terminate();                        // 제어 종료 함수를 실행한다.
        return 0;              // 프로그램을 종료한다.
}

// PSD 값을 읽어서 10cm 이내일 경우 탈출한다.
do{
        read();                // PSD 센서 값을 읽기 위해 통신 요청.
        if(g_nAdcType[0] == 1){// PSD 가 왼쪽에 연결되었다면
                nDistance = g_nAdcDist[0]; // 해당 PSD 값을 nDistance에 넣는다.
                printf("PSD = %d cm₩n", nDistance ); // PSD 값을 출력한다.
                fflush(stdout);
        }
        else if(g_nAdcType[1] == 1){ // PSD 가 오른쪽에 연결되었다면
                nDistance = g_nAdcDist[1]; // 해당 PSD 값을 nDistance에 넣는다.
                printf("PSD = %d cm₩n", nDistance ); // PSD 값을 출력한다.
                fflush(stdout);
        }
        else {
        // 2가지 경우가 전부 아니라면(PSD 가 연결되지 않았다면)
                printf("PSD Not Connected₩n"); // 에러 메시지를 출력한 후
                fflush(stdout);
                terminate();          // 제어 종료 함수를 실행한다.
                return 0; // 프로그램을 종료한다.
        }
```

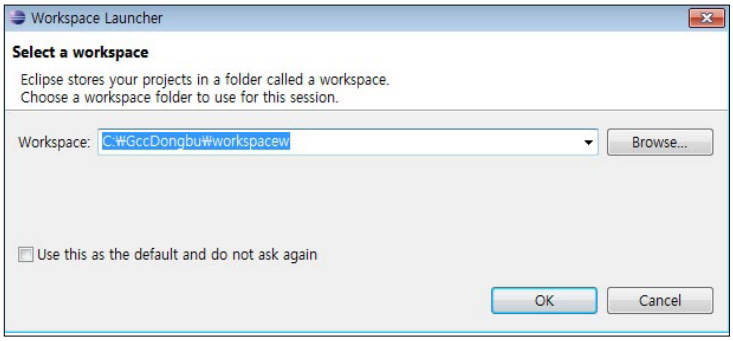
```
        } while (nDistance > 10);     // 10 cm 이내로 들어온 경우 루프를 탈출한다.

        // 권투모션의 실행
        motion(0, 0);  // 모션 0번을 실행(두 번째 파라미터는 준비 자세만 실행하는 모드인지 여부이므로 0)
        motion_wait();                // 모션이 종료될 때까지 대기

        // 프로그램 종료
        terminate();                  // 제어 종료 함수를 실행한다.
        return 0;
}
```

빌드 및 실행

01 이클립스 실행

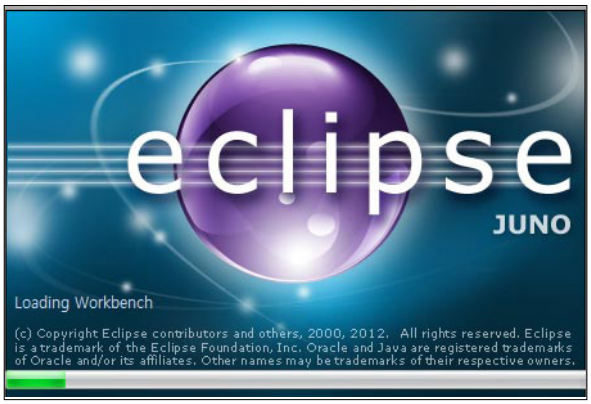
이클립스 실행 버튼을 누릅니다.

02 workspace



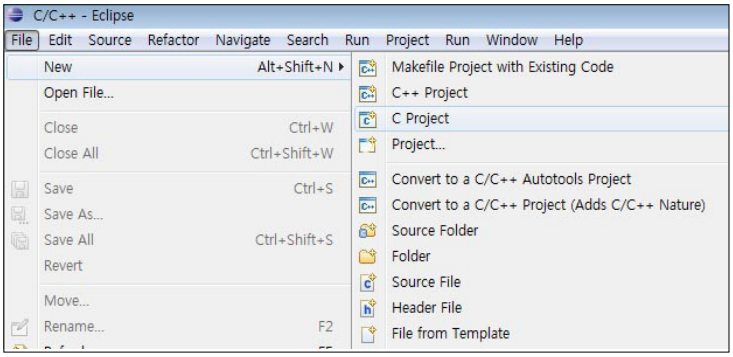
작업하고자 하는 폴더를 지정합니다. Browse 를 클릭하여 폴더를 지정하고 OK 버튼을 누릅니다.

03 이클립스 로딩 화면



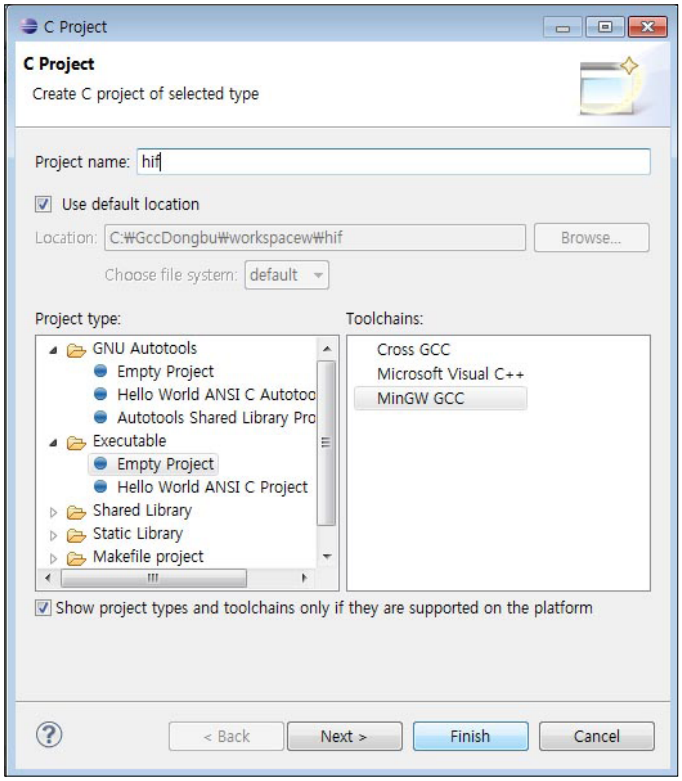
이클립스 로딩 화면입니다.

04 프로젝트 생성



이클립스 편집창 메뉴에서 File 〉 New 〉 C project 를 클릭합니다.

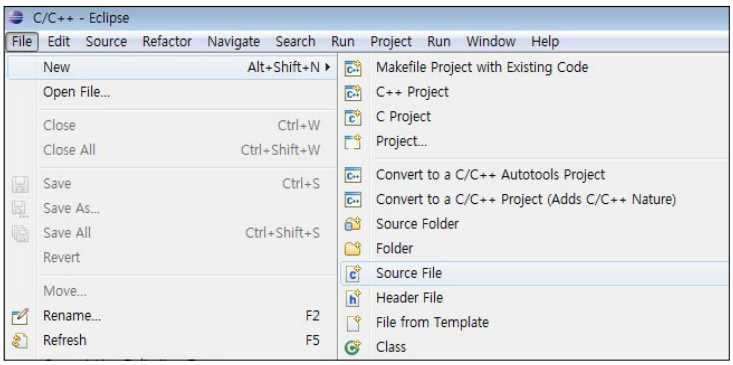
05 프로젝트 이름



Projec Name 을 hif 라고 입력하고, Project type 에서 Exectable 에서 Empty Project 를 선택합니다.

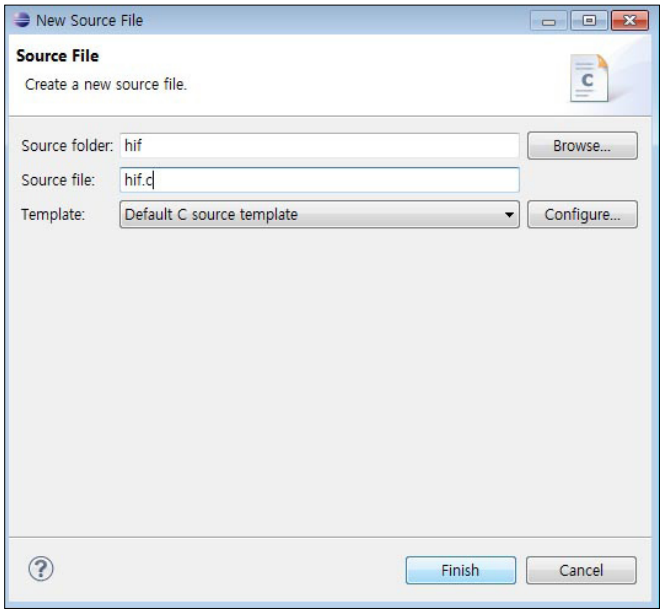
Toolchians 에서 MinGW GCC 를 선택하고 Finish 버튼을 누릅니다.

06 소스파일 추가



File 〉New 〉Source File 을 클릭합니다.

07 소스파일 이름

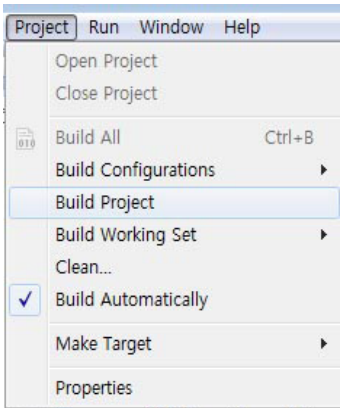


hif.c 라고 입력하고 Finish 버튼을 누릅니다.

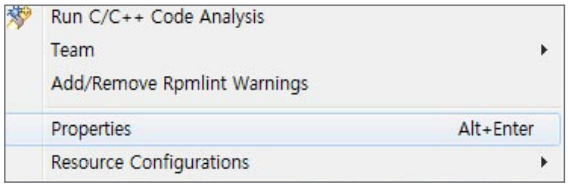
08 기본 제공 파일 추가

본 교재에게 C 프로그래밍을 하기 위해선 기본 제공되는 라이브러리 파일을 먼저 추가해주어야 올바른 컴파일을 할 수 있습니다. 필요한 파일은 drApi.dll, drApi.h, hovis.c, hovis.h 등 총 4가지 입니다.

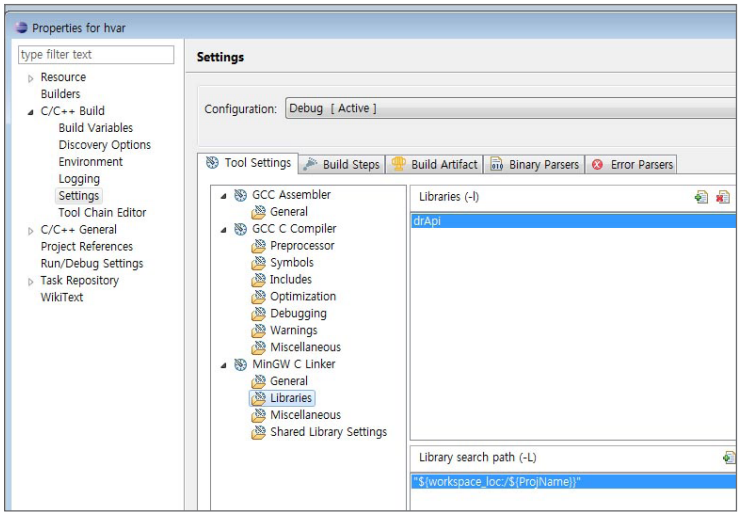
파일 추가 전에 먼저 Property 에서 라이브러리를 설정해주어야 합니다. 파일 생성을 한 후에 아무 코딩도 하지 말고, 바로 Build project 를 실행하세요. 그래야 라이브러리 추가가 가능합니다.



Build Project 를 클릭합니다.



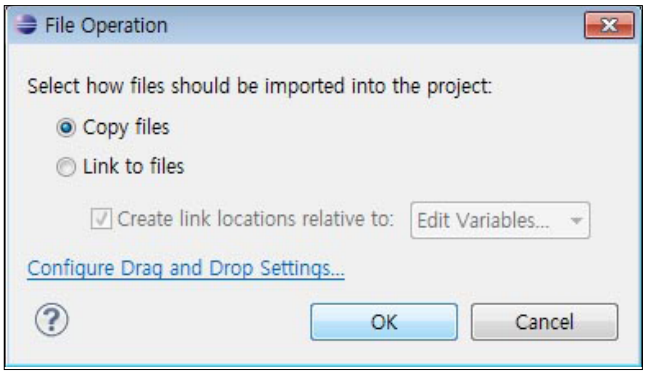
좌측 프로젝트 이름에 커서를 놓고 오른쪽 마우스키를 누릅니다.
제일 하단에 Propertis 를 클릭합니다. Alt+Enter 도 동일합니다.



C/C++ Build 〉Settings 를 클릭하고, 우측에 MinGW C Linker 〉Libraties 를 클릭합니다. 우측에 Libraies 에 + 마크를 클릭하고 팝업창이 뜨면 drApi 라고 입력합니다. Library search path 에 + 마크를 클릭하고 팝업창이 뜨면 Workspace 를 클릭하고, Folder selection 에서 프로젝트 이름과 동일한 것을 클릭하고 OK 버튼을 누릅니다.

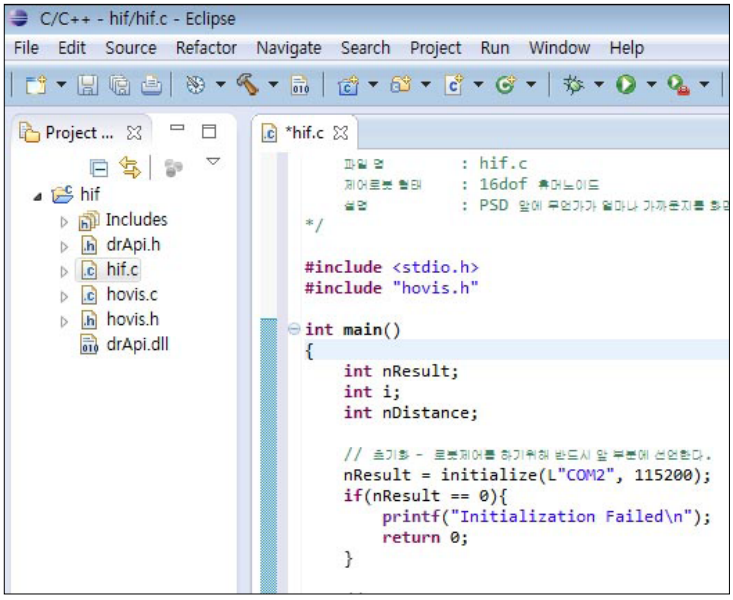
로봇을 동작시키기 위해선 위 네 개 파일을 프로젝트에 포함시켜야 합니다.
위 파일이 있는 파일탐색기 폴더에서 드래그 하는 방식으로 프로젝트에 포함시킵니다.
파일을 끌어다가 드래그해서 왼쪽 프로젝트에 추가합니다.

09 기본 파일 Import



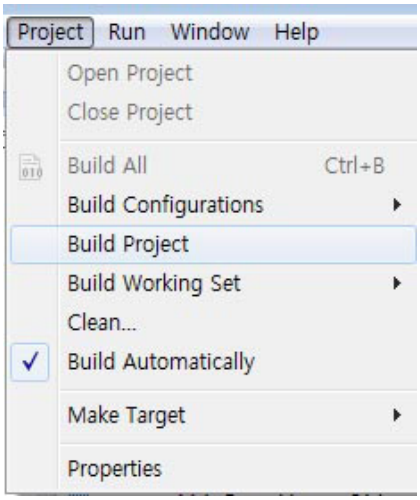
OK 버튼을 누릅니다.

10 소스 작성



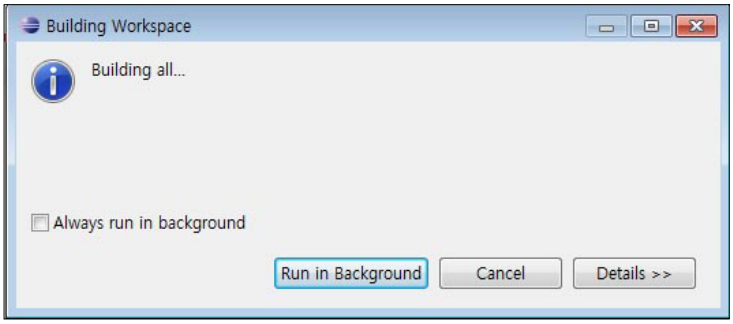
소스를 작성합니다.

11 빌드



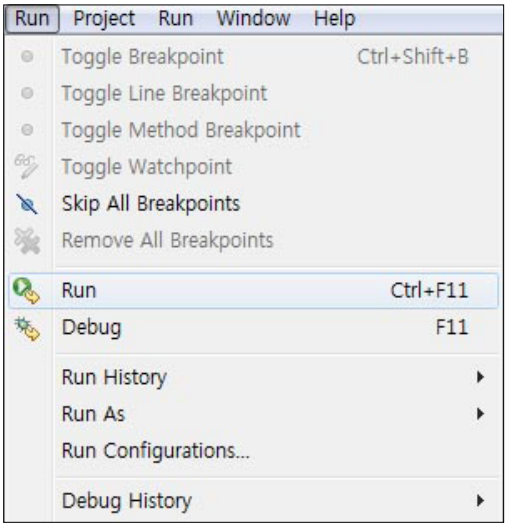
Projec 〉Build Project 를 클릭합니다.

12 빌드중



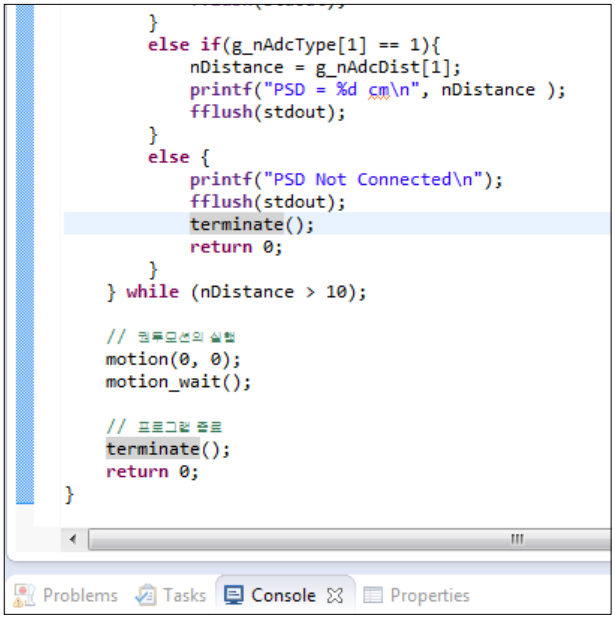
빌드 팝업입니다. 그대로 두면 완료 후 사라집니다.

13 Run 실행



Run 〉 Run 을 클릭합니다. Ctrl+F11 단축키도 동일합니다.

14 실행화면



15 로봇동작

PSD 앞에 무언가가 얼마나 가까운 지를 화면상으로 출력하고 10 cm 이내로 접근 시 권투 모션을 실행합니다.

3.5 if~else 조건문

if 는 때로 앞조건이 만족하는데도, 그 아래 문자의 조건을 검사하게 되서 불필요한 비교연산을 많이 하게 됩니다. 이를 해결하기 위해 if else 를 쓰게 됩니다.

```
If(조건)
{
        조건 만족시 실행
}
Else
{
        조건 불만족시 실행
}
```

아래 예제는 CDS를 손가락으로 가리면 오른쪽 팔을 하늘로 올리면서 모터의 LED를 파란색으로(3초), 떼면 반대동작으로 LED를 빨간색으로(3초) 변하는 프로그램입니다.

hifelse.c

```
#include <stdio.h>
#include "hovis.h"

int main()
{

        int nResult;

        int i;

        int nBright;

        nResult = initialize(L"COM2", 115200);
if(nResult == 0){
printf("Initialization Failed₩n");
fflush(stdout);
return 0;
}

        for(i=0;i<16;i++){
                g_fMotorPos[i]=0;
        }
        g_fMotorPos[0] = -90;
        g_fMotorPos[1] = 90;
```

```
        g_fMotorPos[3] = –90;
        g_fMotorPos[4] = 90;
        run(1000);

        delay(1000);

        read();

        if(g_nLight 〉= 100)
                nBright = 1;
        else
                nBright = 0;

        for (i = 0; i 〈 10; i++) {
         while(1){

                 read();
                if(nBright == 1 && g_nLight 〈 100)
                        break;
                 else if(nBright == 0 && g_nLight 〉= 100)
                        break;
           }

          if (g_nLight 〉= 100) {

                for (j = 0; j 〈 16; j++){
                        g_ucMotorRedLed[j] = 1;
                        g_ucMotorBlueLed[j] = 0;
           }
                g_fMotorPos[3] = 90;
                g_fMotorPos[0] = –90;
                run(1000);
                delay(3000);
                nBright = 1;
          }
        else {

                for (j = 0; j 〈 16; j++){
                          g_ucMotorBlueLed[j] = 1;
                          g_ucMotorRedLed[j] = 0;
                  }
                g_fMotorPos[0] = 90;
                g_fMotorPos[3] = –90;
                run(1000);
                delay(3000);
                nBright = 0;
                }
        }

        terminate();

        return 0;
}
```

위 코드 중에 진하게 처리된 if else 문을 살펴봅니다.

```
if(g_nLight 〉= 100)          // 빛 센서 값이 50 이상이라면
        nBright = 1;          // nBright에 1 대입(밝은 상태)
else                         // 빛 센서 값이 50 보다 작다면
        nBright = 0;          // nBright에 0 대입(어두운 상태)
```

문법요약

◆ if~else

앞조건이 만족하는데도, 그 아래 문자의 조건을 검사하게 됩니다.
불필요한 비교연산을 많이 하게되고, 그것을 해결하기위해서 if else 를 씁니다.

```
if(조건)
{
        조건 만족시 실행 일명 “이거”
}
else
{
        조건 불만족시 실행 일명”저거”
}
```

코딩계획

```
// for, 밝은상태 저장 변수 선언
//초기화
//모터사용
//빛센서 읽기
//10번 실행
//오른팔, 왼팔 들기
```

프로그래밍 세부설계

```
/*
파일 명            : hifelse.c
제어로봇 형태    : 16dof 휴머노이드
설명               : CDS를 손가락으로 가리면 오른쪽 팔을 하늘로 올리면서 모터의
LED를 파란색으로(3초), 떼면 반대동작으로 LED를 빨간색으로(3초)
*/

// 동부로봇의 api 를 사용하기 위해 반드시 이렇게 선언 해 두어야 한다.

int main()
{
        // 결과값을 리턴받을 변수
        // for 문을 사용하기 위해 선언한 변수
        // 현재 상태가 밝은 지 어두운 지 저장하는 변수

        // 초기화 – 로봇제어를 하기위해 반드시 앞 부분에 선언한다.
        // 각 변수들을 초기화 한다.
        // 열고 난 이후 이상이 있는 지 확인한다.
                // 이상이 있다면 에러 출력
                // 프로그램 종료

        // 이제부터 모터를 사용 할 수 있다.
        // 똑바로 일어선다.        (차렷자세)

                // 모든 모터(16개 모터)를 0 위치로 설정함

        // 0 번 모터(오른쪽 어깨)
        // 1 번 모터(오른쪽 윗팔)
        // 3 번 모터(왼쪽 어깨)
```

```
// 4 번 모터(왼쪽 윗팔)
// 모터 동작(해당 자세를 1000ms 동안 동작)
// 동작 대기(1000ms)

// 빛 센서 읽기
// read() 함수를 실행하면 센서 변수들이 자동으로 업데이트 된다.
// 빛 센서를 읽기 위해 통신 요청.

// 초기 빛 상태에 따라서 nBright 값을 저장
// 빛 센서 값이 50 이상이라면
        // nBright에 1 대입(밝은 상태)
// 빛 센서 값이 50 보다 작다면
        // nBright에 0 대입(어두운 상태)

// 총 10번 실행한다.
        // 탈출하기 전까지 무한 반복
                // 빛 센서를 읽기 위해 통신 요청.

// nBright가 1(밝은 상태)인데 빛 센서 값이 50보다 작은 경우
                        // while문을 탈출한다.
// nBright가 0(어두운 상태)인데 빛 센서 값이 50 이상인 경우
                        // while문을 탈출한다.

                // 동작 – 왼팔들기
                // j가 0~15까지 반복(16축)
                        // 모든 모터의 빨간 색 LED 를 켠다.
                        // 모든 모터의 파란 색 LED 를 끈다.

// 3 번 모터(왼쪽 어깨). 팔을 위로 들어올리는 값을 넣는다.
// 0 번 모터(오른쪽 어깨). 팔을 차렷자세로 하는 값을 넣는다.
                // 모터 동작(해당 자세를 1000ms 동안 동작)
                //동작 대기(3000ms)
                // nBright에 1 대입(밝은 상태)

                // 동작 – 오른팔들기
                // j가 0~15까지 반복(16축)
                        // 모든 모터의 파란 색 LED 를 켠다.
                        // 모든 모터의 빨간 색 LED 를 끈다.
// 0 번 모터(오른쪽 어깨). 팔을 위로 들어올리는 값을 넣는다.
// 3 번 모터(왼쪽 어깨). 팔을 차렷자세로 하는 값을 넣는다.
                // 모터 동작(해당 자세를 1000ms 동안 동작)
                // 동작 대기(3000ms)
                // nBright에 0 대입(어두운 상태)
// 프로그램 종료
// 제어 종료 함수를 실행한다.
```

프로그래밍 작성

```
/*
파일 명            : hifelse.c
제어로봇 형태      : 16dof 휴머노이드
설명               : CDS를 손가락으로 가리면 오른쪽 팔을 하늘로 올리면서 모터의
LED를 파란색으로(3초), 떼면 반대동작으로 LED를 빨간색으로(3초)
*/

#include <stdio.h>
#include "hovis.h"
// 동부로봇의 api 를 사용하기 위해 반드시 이렇게 선언 해 두어야 한다.

int main()
{
        int nResult;        // 결과값을 리턴받을 변수
        int i;              // for 문을 사용하기 위해 선언한 변수
        int nBright;        // 현재 상태가 밝은 지 어두운 지 저장하는 변수
        // 초기화 – 로봇제어를 하기 위해 반드시 앞 부분에 선언한다.
        nResult = initialize(L"COM2", 115200);      // 각 변수들을 초기화 한다.
        if(nResult == 0){ // 열고 난 이후 이상이 있는 지 확인한다.
                printf("Initialization Failed₩n");      // 이상이 있다면 에러 출력
                fflush(stdout);
                return 0;// 프로그램 종료
        }

        // 이제부터 모터를 사용 할 수 있다.
        // 똑바로 일어선다.        (차렷자세)
        for(i=0;i<16;i++){
                g_fMotorPos[i]=0;// 모든 모터(16개 모터)를 0 위치로 설정
        }
        g_fMotorPos[0] = -90;               // 0 번 모터(오른쪽 어깨)
        g_fMotorPos[1] = 90;                // 1 번 모터(오른쪽 윗팔)
        g_fMotorPos[3] = -90;               // 3 번 모터(왼쪽 어깨)
        g_fMotorPos[4] = 90;                // 4 번 모터(왼쪽 윗팔)
        run(1000);          // 모터 동작(해당 자세를 1000ms 동안 동작)
        delay(1000);                        // 동작 대기(1000ms)

        // 빛 센서 읽기
        // read() 함수를 실행하면 센서 변수들이 자동으로 업데이트 된다.
        read();                             // 빛 센서를 읽기 위해 통신 요청.
```

```
        // 초기 빛 상태에 따라서 nBright 값을 저장
        if(g_nLight >= 100)                    // 빛 센서 값이 50 이상이라면
                nBright = 1;                   // nBright에 1 대입(밝은 상태)
        else                                   // 빛 센서 값이 50 보다 작다면
                nBright = 0;                   // nBright에 0 대입(어두운 상태)

        for (i = 0; i < 10; i++) {     // 총 10번 실행한다.
                while(1){              // 탈출하기 전까지 무한 반복
                        read();                // 빛 센서를 읽기 위해 통신 요청.

                        if(nBright == 1 && g_nLight < 100)
// nBright가 1(밝은 상태)인데 빛 센서 값이 50보다 작은 경우
                                break;         // while문을 탈출한다.
                        else if(nBright == 0 && g_nLight >= 100)
// nBright가 0(어두운 상태)인데 빛 센서 값이 50 이상인 경우
                                break;         // while문을 탈출한다.
                }

                if (g_nLight >= 100) {                 // 동작 – 왼팔들기
                        for (j = 0; j < 16; j++){
// j가 0~15까지 반복(16축)
                                g_ucMotorRedLed[j] = 1;
        // 모든 모터의 빨간 색 LED 를 켠다.
                                g_ucMotorBlueLed[j] = 0;
// 모든 모터의 파란 색 LED 를 끈다.
                        }
                        g_fMotorPos[3] = 90;
        // 3 번 모터(왼쪽 어깨). 팔을 위로 들어올리는 값을 넣는다.
                        g_fMotorPos[0] = -90;
        // 0 번 모터(오른쪽 어깨). 팔을 차렷자세로 하는 값을 넣는다.
                        run(1000);
                        // 모터 동작(해당 자세를 1000ms 동안 동작)
                        delay(3000);
                // 동작 대기(3000ms)
                        nBright = 1;
                // nBright에 1 대입(밝은 상태)
                }
                else {
                        // 동작 – 오른팔들기
                        for (j = 0; j < 16; j++){
          // j가 0~15까지 반복(16축)
```

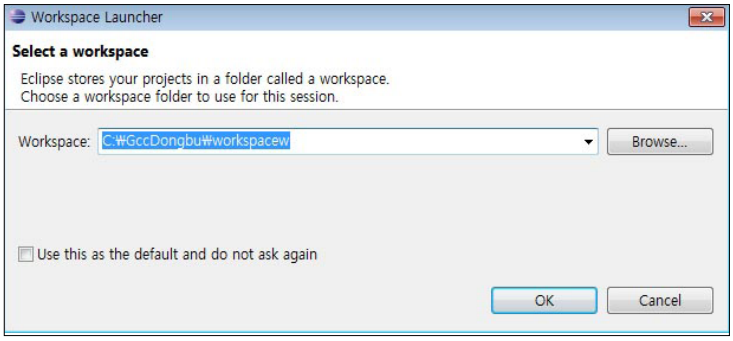
```
                                    g_ucMotorBlueLed[j] = 1;
// 모든 모터의 파란 색 LED 를 켠다.
                                    g_ucMotorRedLed[j] = 0;
        // 모든 모터의 빨간 색 LED 를 끈다.
                        }
                        g_fMotorPos[0] = 90;
        // 0 번 모터(오른쪽 어깨). 팔을 위로 들어올리는 값을 넣는다.
                        g_fMotorPos[3] = -90;
        // 3 번 모터(왼쪽 어깨). 팔을 차렷자세로 하는 값을 넣는다.
                        run(1000);
                        // 모터 동작(해당 자세를 1000ms 동안 동작)
                        delay(3000);                // 동작 대기(3000ms)
                        nBright = 0;       // nBright에 0 대입(어두운 상태)
                }
        }

        // 프로그램 종료
        terminate();                        // 제어 종료 함수를 실행한다.
        return 0;
}
```

빌드 및 실행

01 이클립스 실행

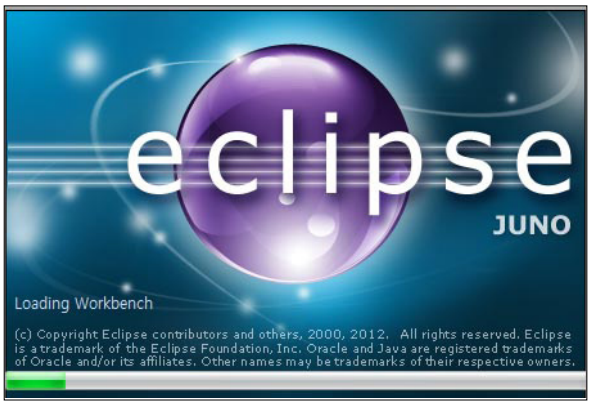
이클립스 실행 버튼을 누릅니다.

02 workspace



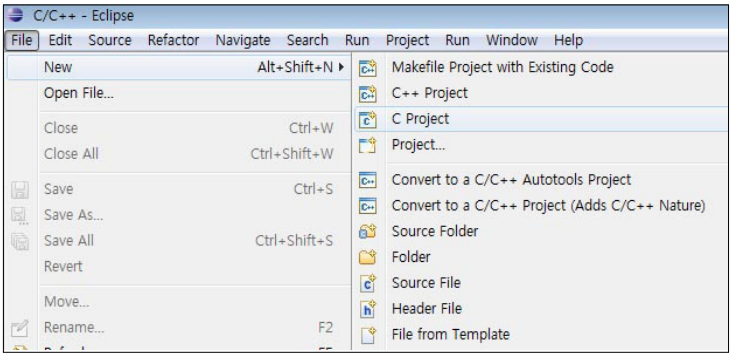
작업하고자 하는 폴더를 지정합니다. Browse 를 클릭하여 폴더를 지정하고 OK 버튼을 누릅니다.

03 이클립스 로딩 화면



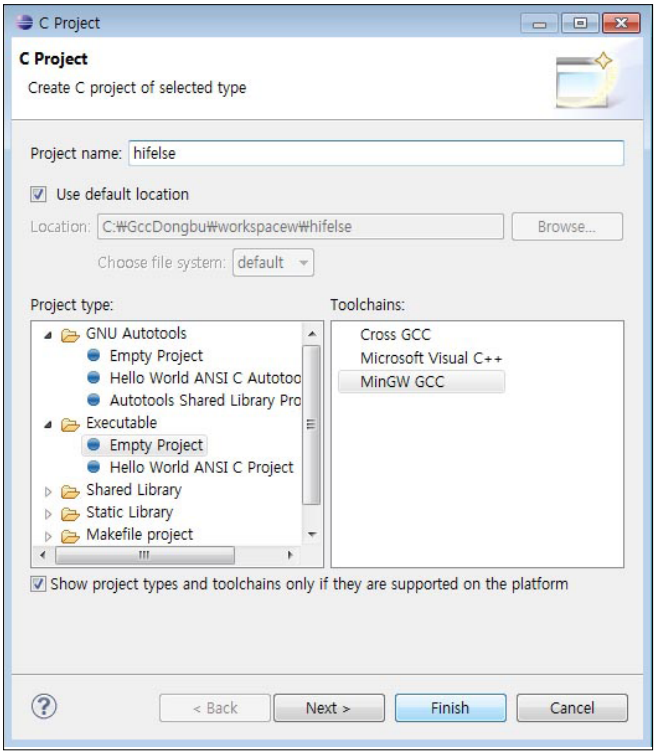
이클립스 로딩 화면입니다.

04 프로젝트 생성



이클립스 편집창 메뉴에서 File 〉 New 〉 C project 를 클릭합니다.

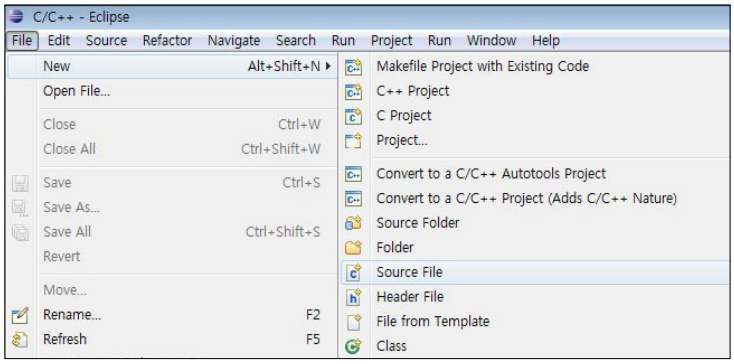
05 프로젝트 이름



Projec Name 을 hifelse 라고 입력하고, Project type 에서 Exectable 에서 Empty Project 를 선택합니다.

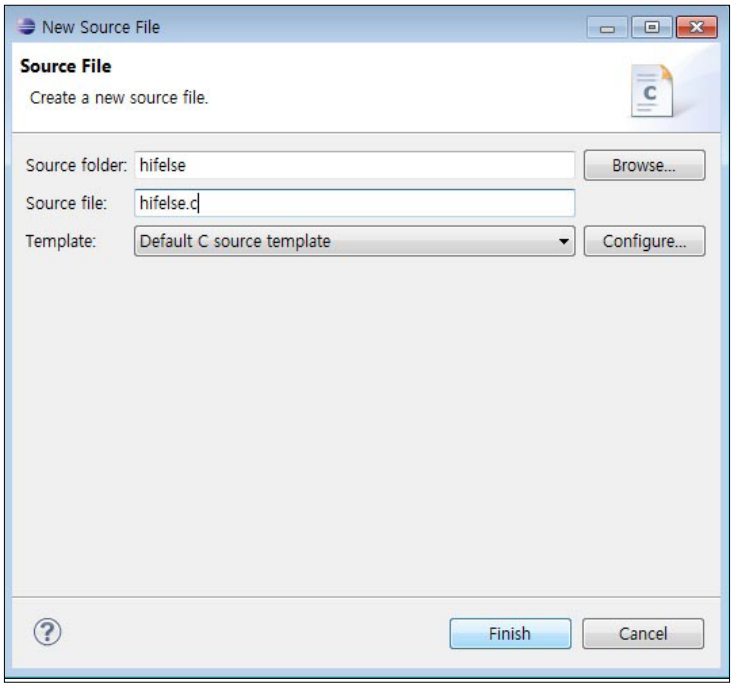
Toolchians 에서 MinGW GCC 를 선택하고 Finish 버튼을 누릅니다.

06 소스파일 추가



File〉New〉Source File 을 클릭합니다.

07 소스파일 이름

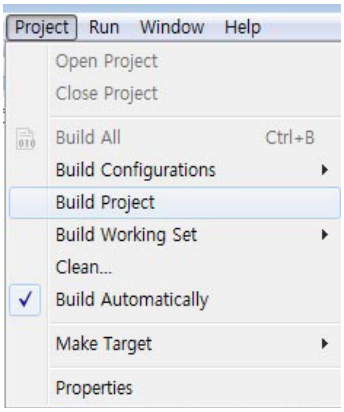


hifelse.c 라고 입력하고 Finish 버튼을 누릅니다.

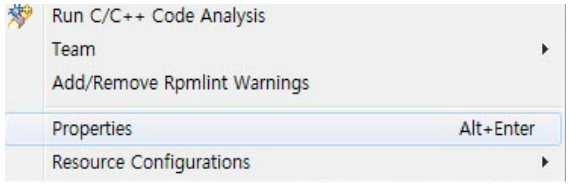
08 기본 제공 파일 추가

본 교재에게 C 프로그래밍을 하기 위해선 기본 제공되는 라이브러리 파일을 먼저 추가해주어야 올바른 컴파일을 할 수 있습니다. 필요한 파일은 drApi.dll, drApi.h, hovis.c, hovis.h 등 총 4가지 입니다.

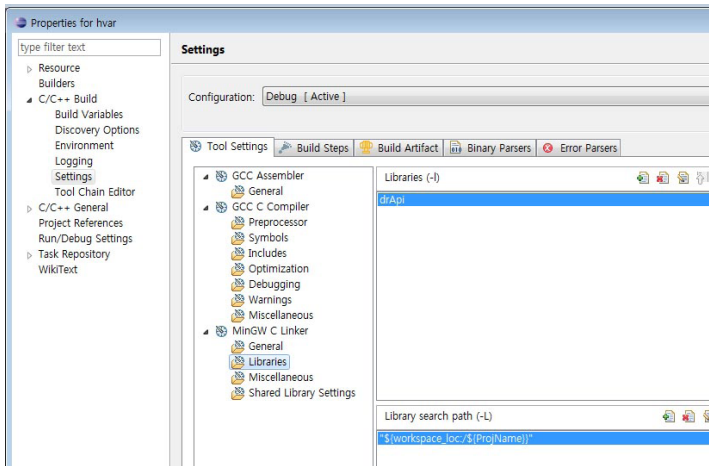
파일 추가 전에 먼저 Property 에서 라이브러리를 설정해주어야 합니다. 파일 생성을 한 후에 아무 코딩도 하지 말고, 바로 Build project 를 실행하세요. 그래야 라이브러리 추가가 가능합니다.



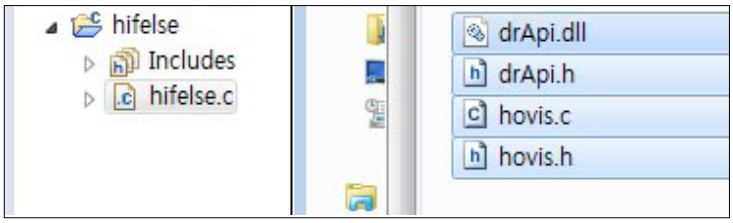
Build Project 를 클릭합니다.



좌측 프로젝트 이름에 커서를 놓고 오른쪽 마우스키를 누릅니다.
제일 하단에 Propertis 를 클릭합니다. Alt+Enter 도 동일합니다.

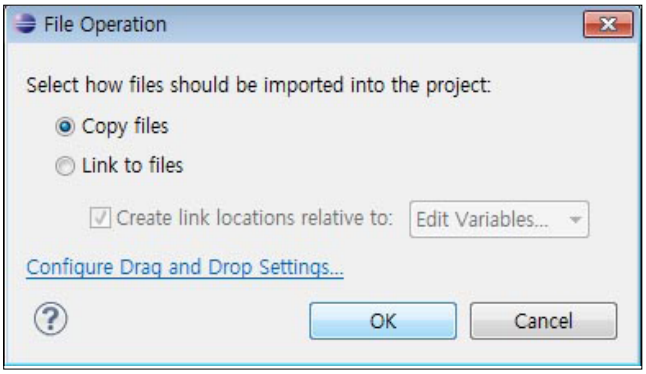


C/C++ Build 〉 Settings 를 클릭하고, 우측에 MinGW C Linker 〉 Libraties 를 클릭합니다. 우측에 Libraies 에 + 마크를 클릭하고 팝업창이 뜨면 drApi 라고 입력합니다. Library search path 에 + 마크를 클릭하고 팝업창이 뜨면 Workspace 를 클릭하고, Folder selection 에서 프로젝트 이름과 동일한 것을 클릭하고 OK 버튼을 누릅니다.



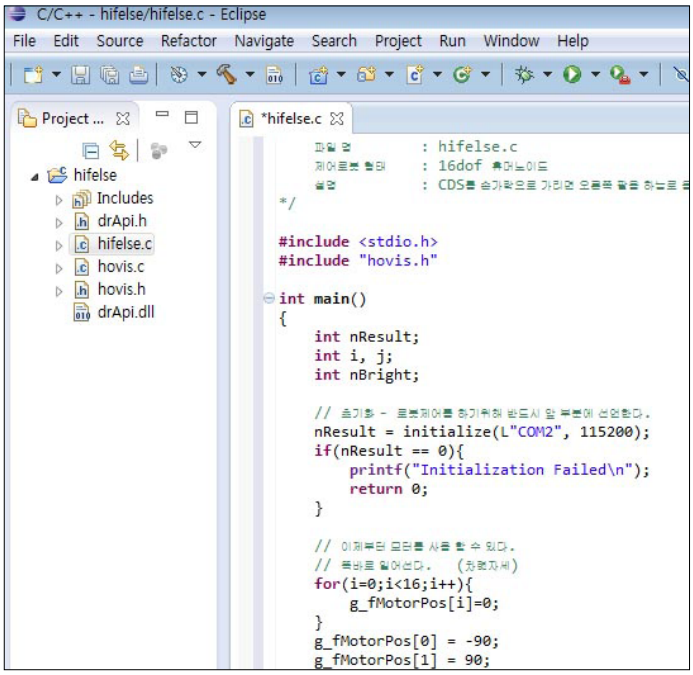
로봇을 동작시키기 위해선 위 네 개 파일을 프로젝트에 포함시켜야 합니다.
위 파일이 있는 파일탐색기 폴더에서 드래그 하는 방식으로 프로젝트에 포함시킵니다.
파일을 끌어다가 드래그해서 왼쪽 프로젝트에 추가합니다.

09 기본 파일 Import



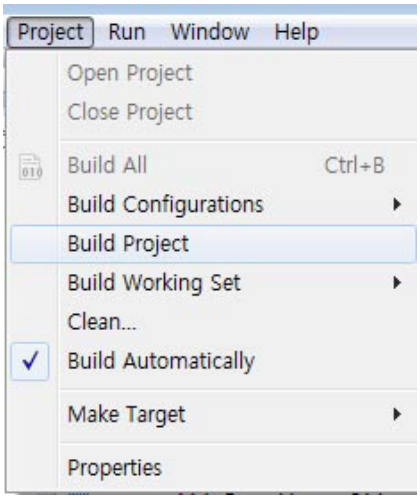
OK 버튼을 누릅니다.

10 소스 작성



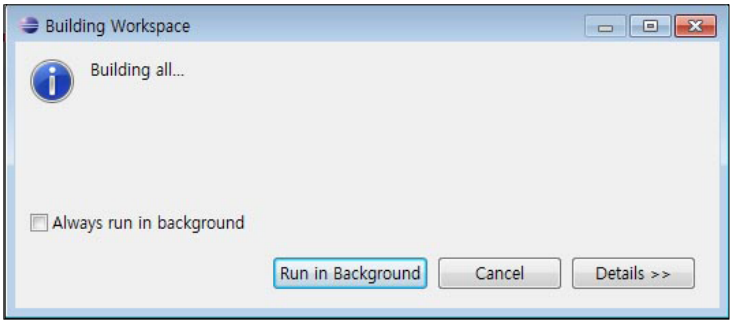
소스를 작성합니다.

11 빌드



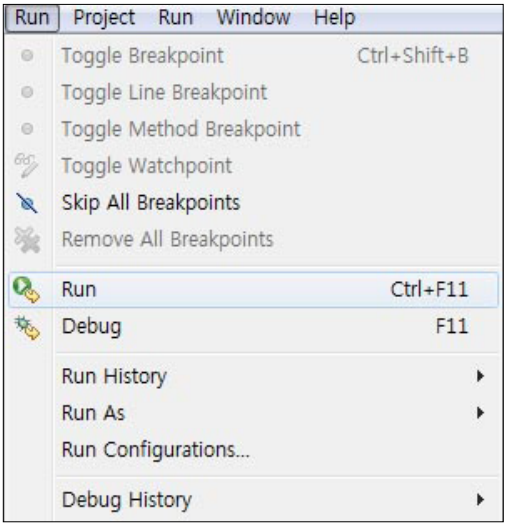
Projec 〉 Build Project 를 클릭합니다.

12 빌드중



빌드 팝업입니다. 그대로 두면 완료 후 사라집니다.

13 Run 실행



Run 〉 Run 을 클릭합니다. Ctrl+F11 단축키도 동일합니다.

14 실행화면

```
        }
        else {
            // 동작 - 오른팔올리기
            for (j = 0; j < 16; j++){
                g_ucMotorBlueLed[j] = 1;
                g_ucMotorRedLed[j] = 0;
            }
            g_fMotorPos[0] = 90;
            g_fMotorPos[3] = -90;
            run(1000);
            delay(3000);
            nBright = 0;
        }
    }

    // 프로그램 종료
    terminate();
    return 0;
}
```

Problems Tasks Console Properties

hifelse.exe [C/C++ Application] C:₩GccDongbu₩workspacew₩h

15 로봇동작

제어기의 CDS를 손가락으로 가리면 오른쪽 팔을 하늘로 올리면서 모터의 LED를 파란색으로(3초), 떼면 반대동작으로 LED를 빨간색으로(3초) 변합니다.

3.6 if~else if~else 조건문

if~else if~else 조건문은, 만약 조건 A가 만족하면 그 문장 실행 후 마지막 else 까지도 완전히 건너뛰게 됩니다. 예를 들어 조건 B가 만족된다면, A는 건너뛰고, B 를 실행한후 나머지는 다 건너띄어버리게 됩니다. 만약 조건 C 가 만족한다면, else 문장은 보지도 않고 건너뜁니다. 이 문장의 효과는 불필요한 연산을 극복할 수 있다는 것입니다.

아래 예제는 왼쪽에서 박수치면 왼쪽 손을 들고, 오른쪽에서 박수 치면 오른쪽 손을 든다. 그 외에는 차렷자세를 유지하는 프로그램입니다.

hifelseif.c

```
#include <stdio.h>
#include "hovis.h"

int main()
{

        int nResult;

        int i;

        nResult = initialize(L"COM2", 115200);
        if(nResult == 0){

                printf("Initialization Failed₩n");
                fflush(stdout);
                return 0;

        }

        for(i=0;i<16;i++){
                g_fMotorPos[i]=0;

        }
        g_fMotorPos[0] = -90;
        g_fMotorPos[1] = 90;
        g_fMotorPos[3] = -90;
        g_fMotorPos[4] = 90;
        run(1000);
        delay(1000);
```

```
        while(1){

                printf("Input Sound₩n");
                fflush(stdout);
                while (1) {
                        read();
                        if(g_nSoundCount)
                                break;

                }
                printf("Direction = %d₩n", g_nSoundDirection);
                fflush(stdout);

                if (g_nSoundDirection == -2) {
                        g_fMotorPos[0] = -90;
                        g_fMotorPos[3] = 90;
                        run(1000);
                        delay(1000);
                        g_fMotorPos[0] = -90;
                        g_fMotorPos[3] = -90;
                        run(1000);
                        delay(3000);
                }
                else if (g_nSoundDirection == 2) {

                        g_fMotorPos[0] = 90;
                        g_fMotorPos[3] = -90;
                        run(1000);
                        delay(1000);

                        g_fMotorPos[0] = -90;
                        g_fMotorPos[3] = -90;
                        run(1000);
                        delay(3000);
                }
                else {

                        g_fMotorPos[0] = -90;
                        g_fMotorPos[3] = -90;
                        run(1000);
                        delay(3000);
                }
        }

        terminate();
        return 0;
}
```

위 코드 중에 진하게 처리된 if else if else 문장을 살펴봅니다.

```
if (g_nSoundDirection == -2) {    // -2인 경우 : 왼쪽에서 소리가 난 경우
                                  → 동작 – 왼팔들기
else if (g_nSoundDirection == 2) { // 2인 경우 : 오른쪽에서 소리가 난 경우
                                  → 동작 – 오른팔들기
else {
                                  → 동작 – 차렷자세
```

문법요약

◆ if, else if, else

if(조건 A) → case1: 조건 A가 만족 "요거" 실행후 마지막 else 까지도
완전히 건너뜁니다.
{
조건 A 만족시 실행 일명 " 요거"
}
else if(조건 B) → case2: 조건 A가 만족 하지 않았다.
따라서 조건 B 가 만족되는지 확인합니다.
조건B는 만족!" 이거" 실행후 마지막 else 까지도 완전히 건너뜁니다.
{
조건 B만족시 실행 일명 "이거"
}
else if(조건 C)
조건 C 만족시 실행 일명 " 그거"
}
else → case3: 조건 A,B,C가 모두 만족되지 않는다.
Else 문 안에 있는 "저거" 실행합니다.
{
조건 ABC 불만족시 실행 일명 "저거"
}

예를 들어 조건 B가 만족된다면, A는 건너뛰고, B 를 실행한후 나머지는 다 건너띄어 버립니다.
만약 조건 C 가 만족한다면, else 문장은 보지도 않고 건너뜀뜁니다.
→ 불필요한 연산을 극복할 수 있습니다.

코딩계획

```
// for, 결과값 저장 변수 선언
//초기화
//모터사용
//소리감지
//오른팔, 왼팔 들기
```

프로그래밍 세부설계

```
/*
파일 명            : hifelseif.c
제어로봇 형태    : 16dof 휴머노이드
설명               : 왼쪽에서 박수치면 왼쪽 손을 들고, 오른쪽에서 박수 치면 오른
쪽 손을 든다. 그 외에는 차렷자세
*/

// 동부로봇의 api 를 사용하기 위해 반드시 이렇게 선언 해 두어야 한다.

int main()
{
        // 결과값을 리턴받을 변수
        // for 문을 사용하기 위해 선언한 변수

        // 초기화 – 로봇제어를 하기위해 반드시 앞 부분에 선언한다.
        // 각 변수들을 초기화 한다.
        // 열고 난 이후 이상이 있는 지 확인한다.
                // 이상이 있다면 에러 출력
                // 프로그램 종료

        // 이제부터 모터를 사용 할 수 있다.
        // 똑바로 일어선다.        (차렷자세)

                // 모든 모터(16개 모터)를 0 위치로 설정함
        // 0 번 모터(오른쪽 어깨)
        // 1 번 모터(오른쪽 윗팔)
        // 3 번 모터(왼쪽 어깨)
        // 4 번 모터(왼쪽 윗팔)
        // 모터 동작(해당 자세를 1000ms 동안 동작)
        // 동작 대기(1000ms)
        // 무한 반복한다.
```

```
// 소리를 입력하라는 메시지 출력
// 소리 감지 대기
// 탈출하기 전까지 무한 반복
        // 소리 센서를 읽기 위해 통신 요청.
        // 소리 감지 횟수가 0이 아닌 경우
                // while문을 탈출한다.

// 감지 후
// 소리 감지 방향을 출력한다.

// 소리 감지 방향에 따라서 다른 동작을 수행
// –2인 경우 : 왼쪽에서 소리가 난 경우
        // 동작 – 왼팔들기
        // 0 번 모터(오른쪽 어깨). 팔을 차렷자세로 하는 값을 넣는다.
        // 3 번 모터(왼쪽 어깨). 팔을 위로 들어올리는 값을 넣는다.
        // 모터 동작(해당 자세를 1000ms 동안 동작)
        // 동작 대기(1000ms)
        // 동작 – 차렷자세
        // 0 번 모터(오른쪽 어깨). 팔을 차렷자세로 하는 값을 넣는다.
        // 3 번 모터(왼쪽 어깨). 팔을 차렷자세로 하는 값을 넣는다.
        // 모터 동작(해당 자세를 1000ms 동안 동작)
        // 동작 대기(3000ms)

// 2인 경우 : 오른쪽에서 소리가 난 경우
        // 동작 – 오른팔들기
        // 0 번 모터(오른쪽 어깨). 팔을 위로 들어올리는 값을 넣는다.
        // 3 번 모터(왼쪽 어깨). 팔을 차렷자세로 하는 값을 넣는다.
        // 모터 동작(해당 자세를 1000ms 동안 동작)
        // 동작 대기(1000ms)
        // 동작 – 차렷자세
        // 0 번 모터(오른쪽 어깨). 팔을 차렷자세로 하는 값을 넣는다.
        // 3 번 모터(왼쪽 어깨). 팔을 차렷자세로 하는 값을 넣는다.
        // 모터 동작(해당 자세를 1000ms 동안 동작)
        // 동작 대기(3000ms)

        // 동작 – 차렷자세
// 0 번 모터(오른쪽 어깨). 팔을 차렷자세로 하는 값을 넣는다.
// 3 번 모터(왼쪽 어깨). 팔을 차렷자세로 하는 값을 넣는다.
        // 모터 동작(해당 자세를 1000ms 동안 동작)
        // 동작 대기(3000ms)

// 프로그램 종료
// 제어 종료 함수를 실행한다.
```

프로그래밍 작성

```
/*
파일명              : hifelseif.c
제어로봇 형태    : 16dof 휴머노이드
설명                : 왼쪽에서 박수치면 왼쪽 손을 들고, 오른쪽에서 박수 치면 오른
쪽 손을 든다. 그 외에는 차렷자세
*/
#include <stdio.h>
#include "hovis.h"
// 동부로봇의 api 를 사용하기 위해 반드시 이렇게 선언 해 두어야 한다.
int main()
{
        int nResult;        // 결과값을 리턴받을 변수
        int i;              // for 문을 사용하기 위해 선언한 변수
        int nBright;        // 현재 상태가 밝은 지 어두운 지 저장하는 변수
        // 초기화 – 로봇제어를 하기 위해 반드시 앞 부분에 선언한다.
        nResult = initialize(L"COM2", 115200);      // 각 변수들을 초기화 한다.
        if(nResult == 0){ // 열고 난 이후 이상이 있는 지 확인한다.
                printf("Initialization Failed₩n");      // 이상이 있다면 에러 출력
                fflush(stdout);
                return 0;// 프로그램 종료
        }
        // 이제부터 모터를 사용 할 수 있다.
        // 똑바로 일어선다.        (차렷자세)
        for(i=0;i<16;i++){
                g_fMotorPos[i]=0;// 모든 모터(16개 모터)를 0 위치로 설정
        }
        g_fMotorPos[0] = -90;               // 0 번 모터(오른쪽 어깨)
        g_fMotorPos[1] = 90;                // 1 번 모터(오른쪽 윗팔)
        g_fMotorPos[3] = -90;               // 3 번 모터(왼쪽 어깨)
        g_fMotorPos[4] = 90;                // 4 번 모터(왼쪽 윗팔)
        run(1000);          // 모터 동작(해당 자세를 1000ms 동안 동작)
        delay(1000);                        // 동작 대기(1000ms)
        while(1){                       // 무한 반복한다.
```

```
                printf("Input Sound₩n");
                fflush(stdout);
                                        // 소리를 입력하라는 메시지 출력
                // 소리 감지 대기
                while (1) {             // 탈출하기 전까지 무한 반복
                        read();  // 소리 센서를 읽기 위해 통신 요청.
                        if(g_nSoundCount)
                        // 소리 감지 횟수가 0이 아닌 경우
                                break;          // while문을 탈출한다.
                }
                // 감지 후
                printf("Direction = %d₩n", g_nSoundDirection);
                fflush(stdout);
                // 소리 감지 방향을 출력한다.
                // 소리 감지 방향에 따라서 다른 동작을 수행
                if (g_nSoundDirection == -2) {
// -2인 경우 : 왼쪽에서 소리가 난 경우
                        // 동작 - 왼팔들기
                        g_fMotorPos[0] = -90;
        // 0 번 모터(오른쪽 어깨). 팔을 차렷자세로 하는 값을 넣는다.
                        g_fMotorPos[3] = 90;
        // 3 번 모터(왼쪽 어깨). 팔을 위로 들어올리는 값을 넣는다.
                        run(1000);
                        // 모터 동작(해당 자세를 1000ms 동안 동작)
                        delay(1000);            // 동작 대기(1000ms)
                        // 동작 - 차렷자세
                        g_fMotorPos[0] = -90;
        // 0 번 모터(오른쪽 어깨). 팔을 차렷자세로 하는 값을 넣는다.
                        g_fMotorPos[3] = -90;
        // 3 번 모터(왼쪽 어깨). 팔을 차렷자세로 하는 값을 넣는다.
                        run(1000);
                        // 모터 동작(해당 자세를 1000ms 동안 동작)
                        delay(3000);            // 동작 대기(3000ms)
                }
                else if (g_nSoundDirection == 2) {
        // 2인 경우 : 오른쪽에서 소리가 난 경우
                        // 동작 - 오른팔들기
                        g_fMotorPos[0] = 90;
```

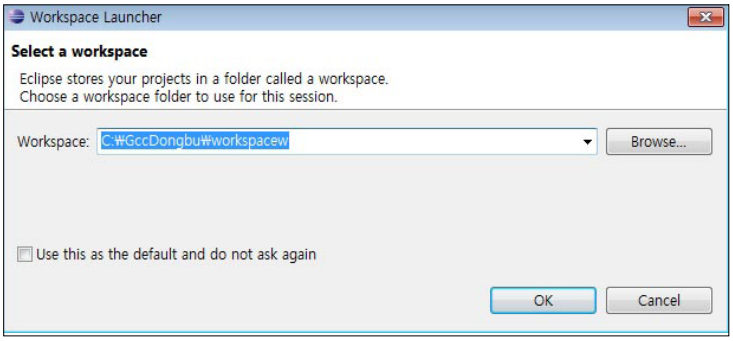
```
                                        // 0 번 모터(오른쪽 어깨). 팔을 위로
들어올리는 값을 넣는다.
                        g_fMotorPos[3] = –90;
        // 3 번 모터(왼쪽 어깨). 팔을 차렷자세로 하는 값을 넣는다.
                        run(1000);
                        // 모터 동작(해당 자세를 1000ms 동안 동작)
                        delay(1000);        // 동작 대기(1000ms)
                        // 동작 – 차렷자세
                        g_fMotorPos[0] = –90;
        // 0 번 모터(오른쪽 어깨). 팔을 차렷자세로 하는 값을 넣는다.
                        g_fMotorPos[3] = –90;
        // 3 번 모터(왼쪽 어깨). 팔을 차렷자세로 하는 값을 넣는다.
                        run(1000);
                        // 모터 동작(해당 자세를 1000ms 동안 동작)
                        delay(3000);        // 동작 대기(3000ms)
                }
                else {
                        // 동작 – 차렷자세
                        g_fMotorPos[0] = –90;
        // 0 번 모터(오른쪽 어깨). 팔을 차렷자세로 하는 값을 넣는다.
                        g_fMotorPos[3] = –90;
        // 3 번 모터(왼쪽 어깨). 팔을 차렷자세로 하는 값을 넣는다.
                run(1000);
                // 모터 동작(해당 자세를 1000ms 동안 동작)
                        delay(3000);
                // 동작 대기(3000ms)
                }
        }

        // 프로그램 종료
        terminate();
                // 제어 종료 함수를 실행한다.
        return 0;
}
        // 프로그램 종료
        terminate();                    // 제어 종료 함수를 실행한다.
        return 0;
}
```

빌드 및 실행

01 이클립스 실행

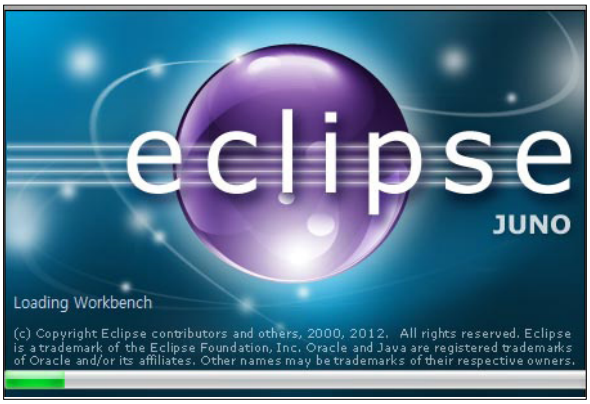
이클립스 실행 버튼을 누릅니다.

02 workspace



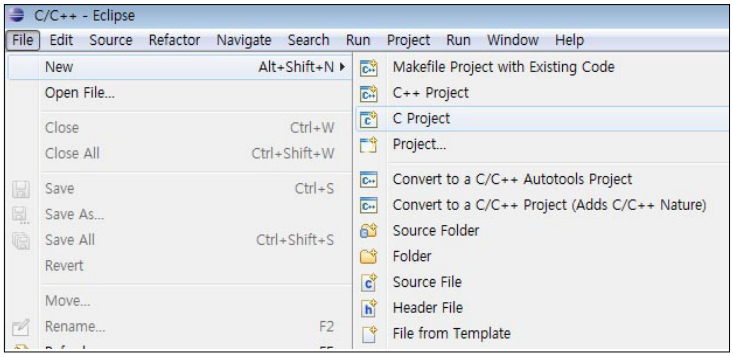
작업하고자 하는 폴더를 지정합니다. Browse 를 클릭하여 폴더를 지정하고 OK 버튼을 누릅니다.

03 이클립스 로딩 화면



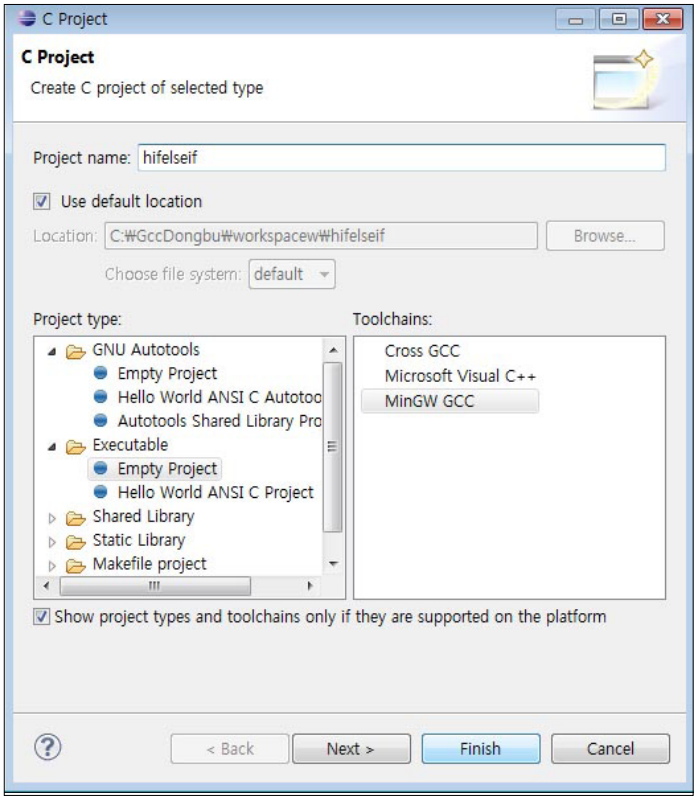
이클립스 로딩 화면입니다.

04 프로젝트 생성



이클립스 편집창 메뉴에서 File 〉 New 〉 C project 를 클릭합니다.

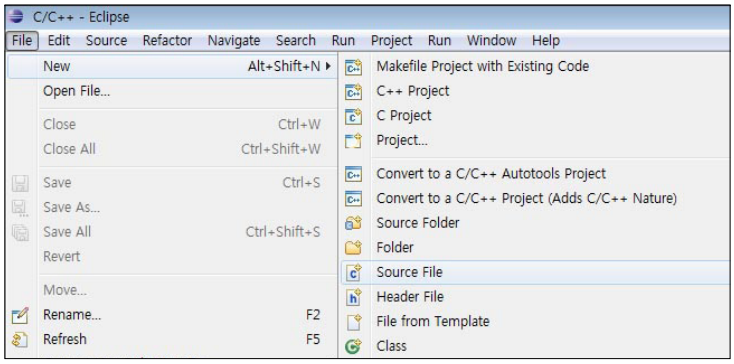
05 프로젝트 이름



Projec Name 을 hifelseif 라고 입력하고, Project type 에서 Exectable 에서 Empty Project 를 선택합니다.

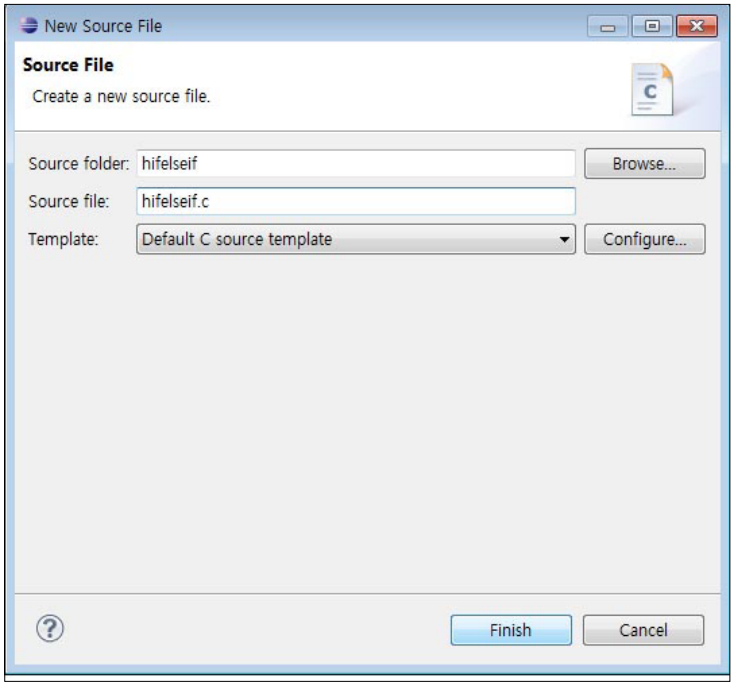
Toolchians 에서 MinGW GCC 를 선택하고 Finish 버튼을 누릅니다.

06 소스파일 추가



File 〉New 〉Source File 을 클릭합니다.

07 소스파일 이름

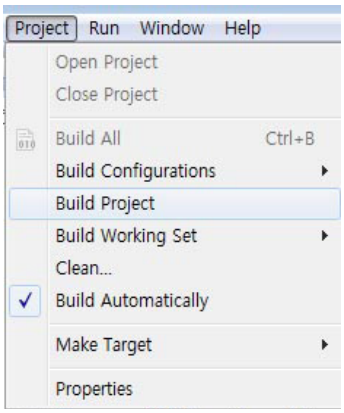


hifelseif.c 라고 입력하고 Finish 버튼을 누릅니다.

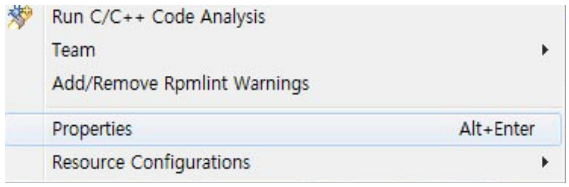
08 기본 제공 파일 추가

본 교재에게 C 프로그래밍을 하기 위해선 기본 제공되는 라이브러리 파일을 먼저 추가해주어야 올바른 컴파일을 할 수 있습니다. 필요한 파일은 drApi.dll, drApi.h, hovis.c, hovis.h 등 총 4가지 입니다.

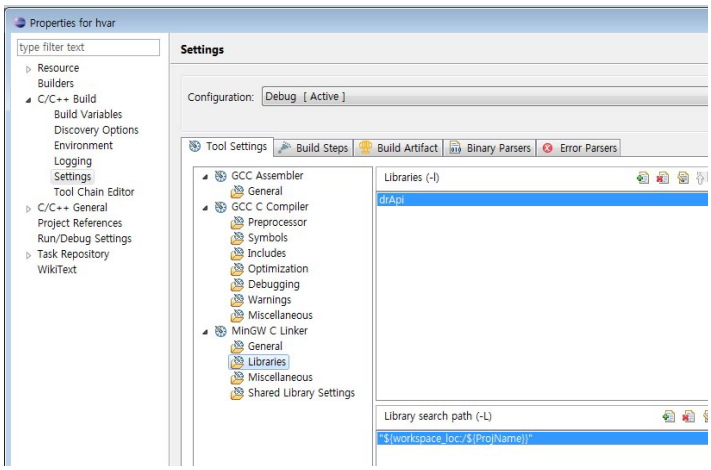
파일 추가 전에 먼저 Property 에서 라이브러리를 설정해주어야 합니다. 파일 생성을 한 후에 아무 코딩도 하지 말고, 바로 Build project 를 실행하세요. 그래야 라이브러리 추가가 가능합니다.



Build Project 를 클릭합니다.



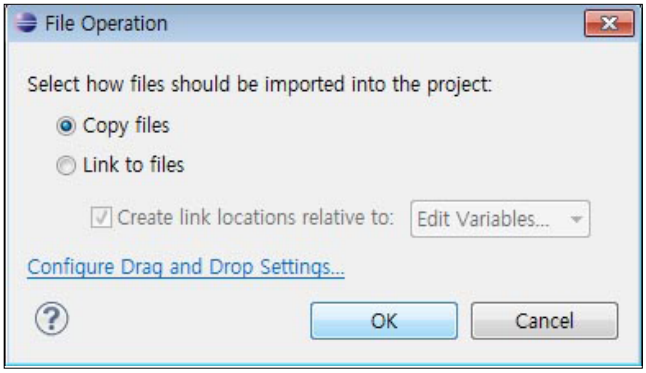
좌측 프로젝트 이름에 커서를 놓고 오른쪽 마우스키를 누릅니다.
제일 하단에 Propertis 를 클릭합니다. Alt+Enter 도 동일합니다.



C/C++ Build 〉 Settings 를 클릭하고, 우측에 MinGW C Linker 〉 Libraties 를 클릭합니다. 우측에 Libraies 에 + 마크를 클릭하고 팝업창이 뜨면 drApi 라고 입력합니다. Library search path 에 + 마크를 클릭하고 팝업창이 뜨면 Workspace 를 클릭하고, Folder selection 에서 프로젝트 이름과 동일한 것을 클릭하고 OK 버튼을 누릅니다.

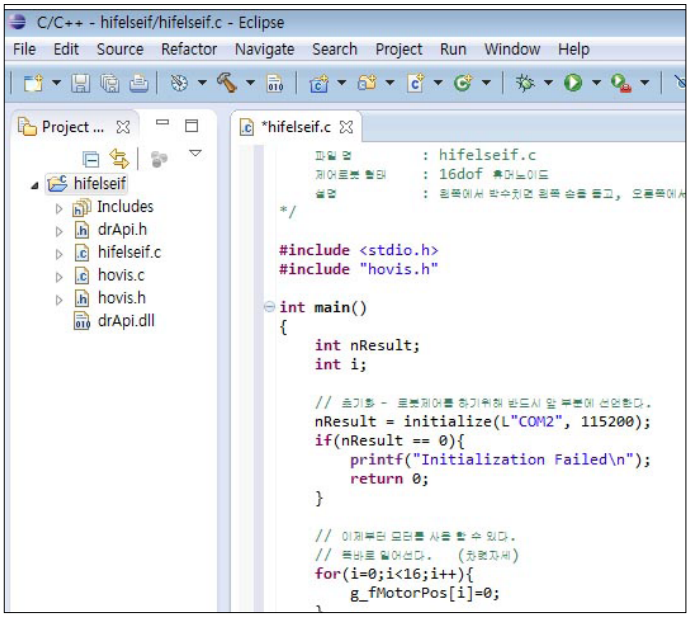
로봇을 동작시키기 위해선 위 네 개 파일을 프로젝트에 포함시켜야 합니다.
위 파일이 있는 파일탐색기 폴더에서 드래그 하는 방식으로 프로젝트에 포함시킵니다.
파일을 끌어다가 드래그해서 왼쪽 프로젝트에 추가합니다.

09 기본 파일 Import



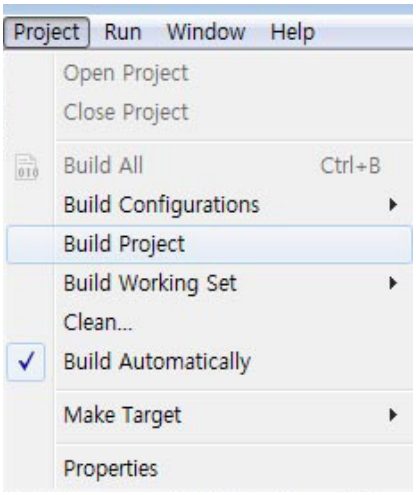
OK 버튼을 누릅니다.

10 소스 작성



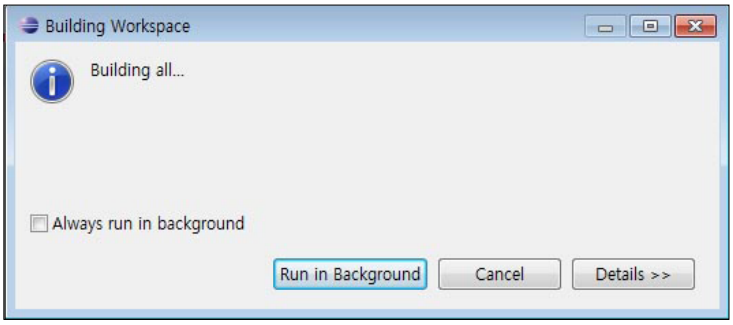
소스를 작성합니다.

11 빌드



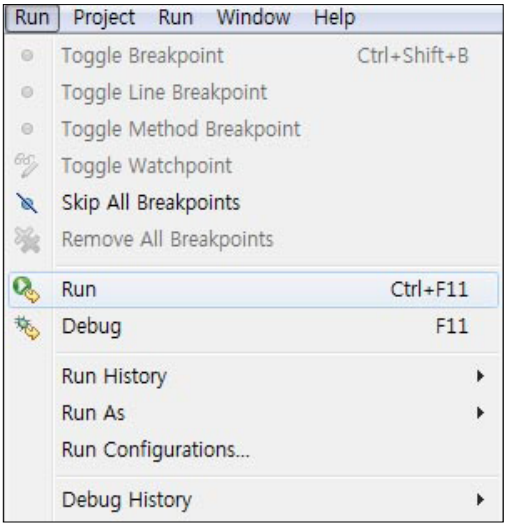
Projec 〉 Build Project 를 클릭합니다.

12 빌드중



빌드 팝업입니다. 그대로 두면 완료 후 사라집니다.

13 Run 실행



Run 〉 Run 을 클릭합니다. Ctrl+F11 단축키도 동일합니다.

14 실행화면

```
int main()
{
    int nResult;
    int i;

    // 초기화 - 로봇제어를 하기위해 반드시 맨 부분에 선언한다.
    nResult = initialize(L"COM3", 115200);
    if(nResult == 0){
        printf("Initialization Failed\n");
        fflush(stdout);
        return 0;
    }

    // 이제부터 모터를 사용 할 수 있다.
    // 똑바로 일어선다.   (차렷자세)
    for(i=0;i<16;i++){
        g_fMotorPos[i]=0;
    }
```

Problems Tasks Console Properties

hifelseif.exe [C/C++ Application] C:₩GccDongbu₩workspacew₩hi

```
Input Sound
Direction = -2
Input Sound
```

15 로봇동작

왼쪽에서 박수 치면 왼쪽 손을 들고, 오른쪽에서 박수 치면 오른쪽 손을 든다. 그 외에는 차렷자세를 유지합니다.

3.7 switch~case 조건문

switch 문의 구조는 아래와 같습니다.

```
switch(n)
        case1:   –〉 n 이 1인 경우
                 break;
        case2:
                 break;
        case3:
                 break;
default:
```

n=2 인 경우, case1 은 아니므로 건너뜁니다. case2는 맞으니까 실행하게됩니다. break 문은 switch 를 빠져나가라는 소리입니다. 만약 break 가 없으면, 2부터 그 아래는 모두 실행시킵니다.

아래 예제는 키보드로 모션 번호를 입력 받아 번호에 따라서 다른 모션을 실행하는 프로그램입니다.

hswitch.c

```
#include 〈stdio.h〉
#include “hovis.h”

int main()
{

        int nResult;

        int i;

        int nInputNum;

        nResult = initialize(L”COM2”, 115200);
if(nResult == 0){
printf(“Initialization Failed₩n”);
fflush(stdout);
return 0;
}
        for(i=0;i〈16;i++){
                g_fMotorPos[i]=0;
}
```

```
g_fMotorPos[0] = -90;
g_fMotorPos[1] = 90;
g_fMotorPos[3] = -90;
g_fMotorPos[4] = 90;
run(1000);

delay(1000);

while(1){

        printf("Input Number₩n");
        fflush(stdout);
        scanf("%d", &nInputNum);

        switch(nInputNum){

         case 0:
                printf("Motion 0₩n");
                fflush(stdout);
                motion(0, 0);
                motion_wait();
                break;

         case 2:
                printf("Motion 2₩n");
                fflush(stdout);
                motion(2, 0);
                motion_wait();
                break;

         case 3:
                printf("Motion 3₩n");
                fflush(stdout);
                motion(3, 0);
                motion_wait();
                break;

         case 6:

         case 7:

         case 8:
                printf("Motion 6₩n");
                fflush(stdout);
                motion(6, 0);
                motion_wait();
                break;

         case 9:

         case 10:

         case 11:

         case 12:
         printf("Motion 7₩n");
         fflush(stdout);
         motion(7, 0);
         motion_wait();
```

```
                    break;
                default:
                    printf("Wrong Number₩n");
                    fflush(stdout);
                    break;

            }
        }

        terminate();

        return 0;
}
```

위 코드 중에 진하게 처리된 switch ~ case 문장을 살펴봅니다.
switch, case 문은 총 12개로 키보드 1번에서 12번까지 입력할 수 있습니다.

문법요약

◆ switch ~ case

int 형 char 형 인자를 한 개 전달할 수 있음

switch(n)
    Case1: → n 이 1인 경우
        Break;
    Case2:
        Break;
    Case3:
        Break;
default:
n=2 인 경우, case1 은 아니므로 건너뜁니다.
case2는 맞으니까 실행합니다. break 가 있으면 break 문은 switch 를 빠져나가라는 의미입니다.
만약 break 가 없으면, 2부터 그 아래는 모두 실행시킵니다.

코딩계획

```
// for, 결과값, 입력번호 저장 변수 선언
//초기화
//모터사용
//번호에 따라 동작 바꾸기
```

프로그래밍 세부설계

```
/*
파일명              : hswitch.c
제어로봇 형태       : 16dof 휴머노이드
설명                : 키보드로 모션 번호를 입력 받아 번호에 따라서 다른 모션을 실행함.
*/

// 동부로봇의 api 를 사용하기 위해 반드시 이렇게 선언 해 두어야 한다.

int main()
{
        // 결과값을 리턴받을 변수
        // for 문을 사용하기 위해 선언한 변수
        // 입력 번호가 저장될 변수

        // 초기화 – 로봇제어를 하기 위해 반드시 앞 부분에 선언한다.
        // 각 변수들을 초기화 한다.
        // 열고 난 이후 이상이 있는 지 확인한다.
                // 이상이 있다면 에러 출력
                // 프로그램 종료

        // 이제부터 모터를 사용 할 수 있다.
        // 똑바로 일어선다.          (차렷자세)
                // 모든 모터(16개 모터)를 0 위치로 설정함
        // 0 번 모터(오른쪽 어깨)
        // 1 번 모터(오른쪽 윗팔)
        // 3 번 모터(왼쪽 어깨)
        // 4 번 모터(왼쪽 윗팔)
        // 모터 동작(해당 자세를 1000ms 동안 동작)
        // 동작 대기(1000ms)

        // 무한 반복한다.
                // 번호를 입력하라는 메시지 출력
                // 번호 값 입력 받기
```

```
// 번호에 따라서 switch문 분기

        // 0일 경우
                // 모션 0번 메시지 출력
                // 0번 모션 실행
                // 모션 끝날 때 까지 대기
                // switch-case문 탈출
        // 2일 경우
                // 모션 2번 메시지 출력
                // 2번 모션 실행
                // 모션 끝날 때 까지 대기
                // switch-case문 탈출
        // 3일 경우
                // 모션 3번 메시지 출력
                // 3번 모션 실행
                // 모션 끝날 때 까지 대기
                // switch-case문 탈출
        // 6일 경우
        // 7일 경우
        // 8일 경우
                // 모션 6번 메시지 출력
                // 6번 모션 실행
                // 모션 끝날 때 까지 대기
                // switch-case문 탈출
        // 9일 경우
        // 10일 경우
        // 11일 경우
        // 12일 경우
                // 모션 7번 메시지 출력
                // 7번 모션 실행
                // 모션 끝날 때 까지 대기
                // switch-case문 탈출
        // 그 외의 경우
                // 잘못된 모션 번호 메시지 출력
                // switch-case문 탈출

// 프로그램 종료
// 제어 종료 함수를 실행한다.
```

프로그래밍 작성

```
/*
파일명              : hswitch.c
제어로봇 형태       : 16dof 휴머노이드
설명                : 키보드로 모션 번호를 입력 받아 번호에 따라서 다른 모션을 실행함.
*/

#include <stdio.h>
#include "hovis.h" // 동부로봇의 api 를 사용하기 위해 반드시 이렇게 선언 해 두어야 한다.

int main()
{
        int nResult;        // 결과값을 리턴받을 변수
        int i;              // for 문을 사용하기 위해 선언한 변수
        int nInputNum;      // 입력 번호가 저장될 변수
        // 초기화 – 로봇제어를 하기 위해 반드시 앞 부분에 선언한다.
        nResult = initialize(L"COM2", 115200);  // 각 변수들을 초기화 한다.
        if(nResult == 0){   // 열고 난 이후 이상이 있는 지 확인한다.
                printf("Initialization Failed₩n");          // 이상이 있다면 에러 출력
                fflush(stdout);
                return 0; // 프로그램 종료
        }

        // 이제부터 모터를 사용 할 수 있다.
        // 똑바로 일어선다.          (차렷자세)
        for(i=0;i<16;i++){
                g_fMotorPos[i]=0;// 모든 모터(16개 모터)를 0 위치로 설정
        }
        g_fMotorPos[0] = -90;               // 0 번 모터(오른쪽 어깨)
        g_fMotorPos[1] = 90;                // 1 번 모터(오른쪽 윗팔)
        g_fMotorPos[3] = -90;               // 3 번 모터(왼쪽 어깨)
        g_fMotorPos[4] = 90;                // 4 번 모터(왼쪽 윗팔)
        run(1000);          // 모터 동작(해당 자세를 1000ms 동안 동작)
        delay(1000);                        // 동작 대기(1000ms)

        while(1){                   // 무한 반복한다.
```

```
        printf("Input Number₩n");// 번호입력하라는 메시지 출력
        fflush(stdout);
        scanf("%d", &nInputNum);  // 번호 값 입력 받기

        switch(nInputNum){          // 번호에 따라서 switch문 분기
                case 0:             // 0일 경우
                        printf("Motion 0₩n");
                        fflush(stdout);
                // 모션 0번 메시지 출력
                        motion(0, 0);       // 0번 모션 실행
                        motion_wait();      // 모션 끝날 때 까지 대기
                        break;              // switch-case문 탈출
                case 2:                     // 2일 경우
                        printf("Motion 2₩n");
                        fflush(stdout);
                        // 모션 2번 메시지 출력
                        motion(2, 0);       // 2번 모션 실행
                        motion_wait();      // 모션 끝날 때 까지 대기
                        break;              // switch-case문 탈출
                case 3:                     // 3일 경우
                        printf("Motion 3₩n");
                        fflush(stdout);
// 모션 3번 메시지 출력
                        motion(3, 0);       // 3번 모션 실행
                        motion_wait();      // 모션 끝날 때 까지 대기
                        break;              // switch-case문 탈출
                case 6:                             // 6일 경우
                case 7:                             // 7일 경우
                case 8:                             // 8일 경우
                        printf("Motion 6₩n");
                        fflush(stdout);
// 모션 6번 메시지 출력
                        motion(6, 0);               // 6번 모션 실행
                        motion_wait();      // 모션 끝날 때 까지 대기
                        break;              // switch-case문 탈출

                case 9:                             // 9일 경우
                case 10:                    // 10일 경우
                case 11:                    // 11일 경우
                case 12:                    // 12일 경우
                        printf("Motion 7₩n");
                        fflush(stdout);
// 모션 7번 메시지 출력
                        motion(7, 0);               // 7번 모션 실행
```

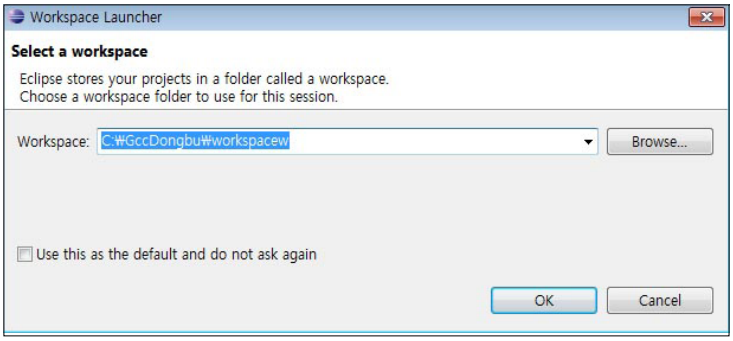
```
                    motion_wait();      // 모션 끝날 때 까지 대기
                    break;              // switch-case문 탈출
                default:                                // 그 외의 경우
                    printf("Wrong Number₩n");
                    fflush(stdout);
        // 잘못된 모션 번호 메시지 출력
                    break;              // switch-case문 탈출
            }
        }
        // 프로그램 종료
        terminate();                    // 제어 종료 함수를 실행한다.
        return 0;
}

        // 프로그램 종료
        terminate();                    // 제어 종료 함수를 실행한다.
        return 0;
}
```

빌드 및 실행

01 이클립스 실행

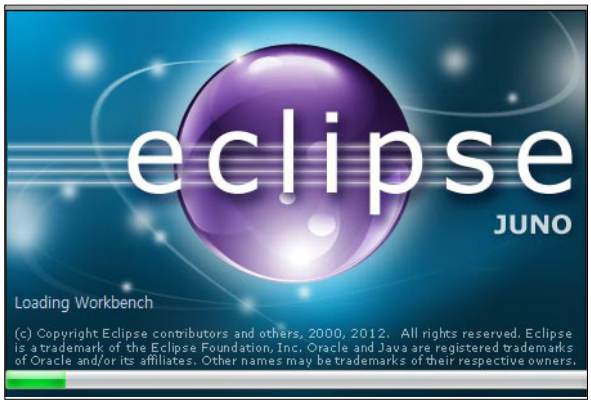
이클립스 실행 버튼을 누릅니다.

02 workspace



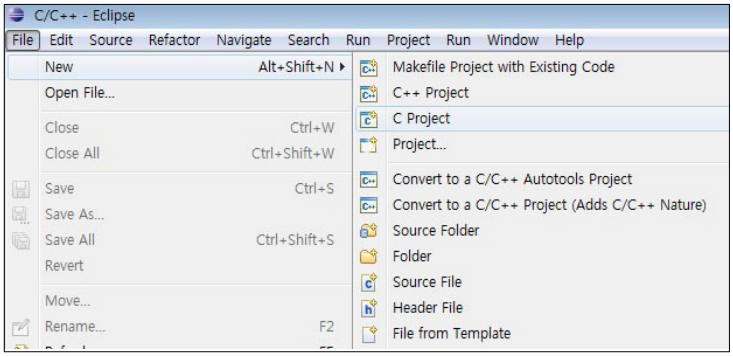
작업하고자 하는 폴더를 지정합니다. Browse 를 클릭하여 폴더를 지정하고 OK 버튼을 누릅니다.

03 이클립스 로딩 화면



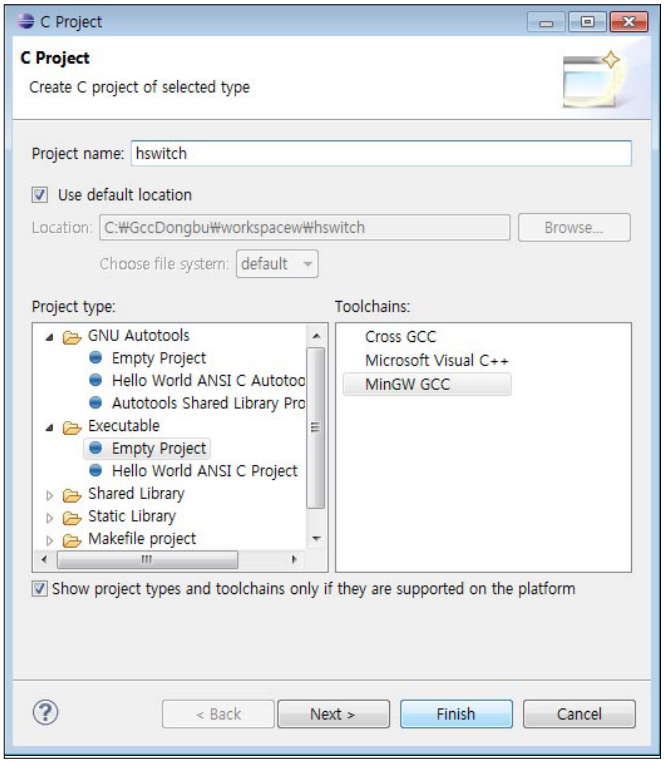
이클립스 로딩 화면입니다.

04 프로젝트 생성



이클립스 편집창 메뉴에서 File 〉 New 〉 C project 를 클릭합니다.

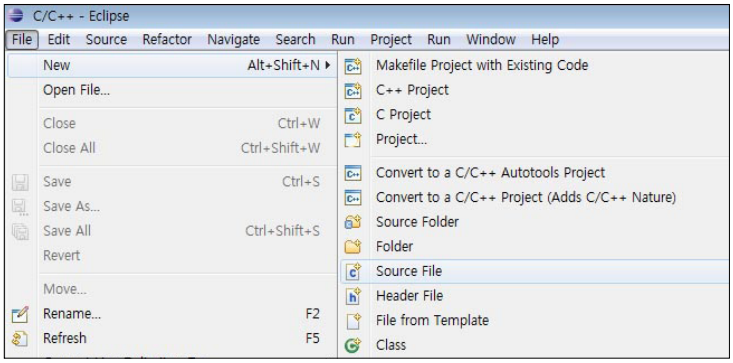
05 프로젝트 이름



Projec Name 을 hswitch 라고 입력하고, Project type 에서 Exectable 에서 Empty Project 를 선택합니다.

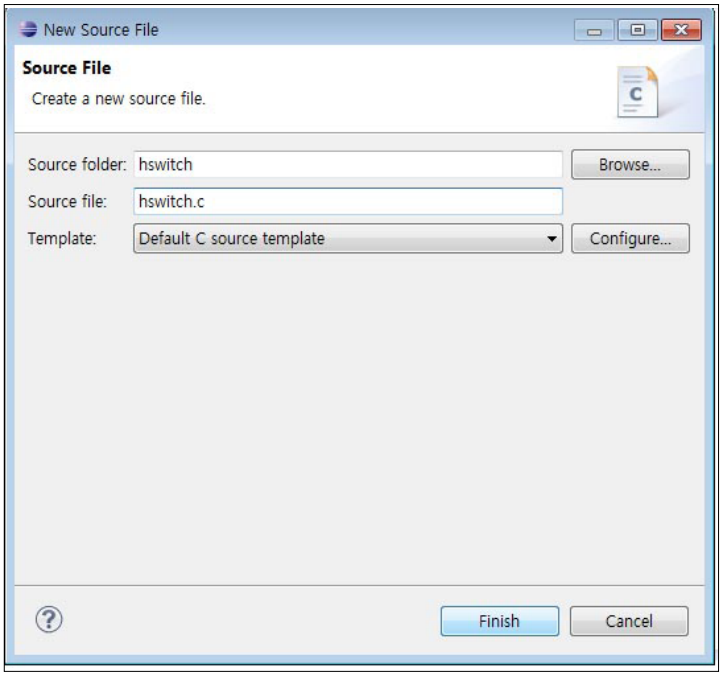
Toolchians 에서 MinGW GCC 를 선택하고 Finish 버튼을 누릅니다.

06 소스파일 추가



File 〉 New 〉 Source File 을 클릭합니다.

07 소스파일 이름

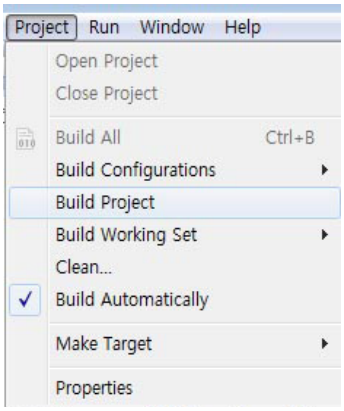


hswitch.c 라고 입력하고 Finish 버튼을 누릅니다.

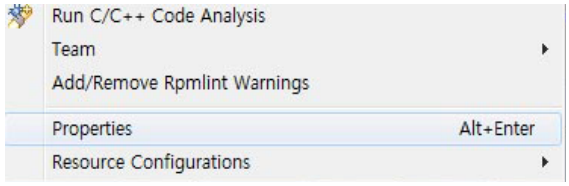
08 기본 제공 파일 추가

본 교재에게 C 프로그래밍을 하기 위해선 기본 제공되는 라이브러리 파일을 먼저 추가해주어야 올바른 컴파일을 할 수 있습니다. 필요한 파일은 drApi.dll, drApi.h, hovis.c, hovis.h 등 총 4가지 입니다.

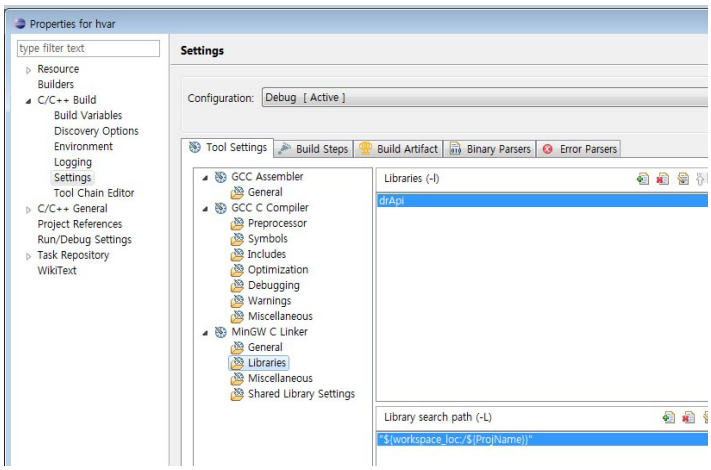
파일 추가 전에 먼저 Property 에서 라이브러리를 설정해주어야 합니다. 파일 생성을 한 후에 아무 코딩도 하지 말고, 바로 Build project 를 실행하세요. 그래야 라이브러리 추가가 가능합니다.



Build Project 를 클릭합니다.



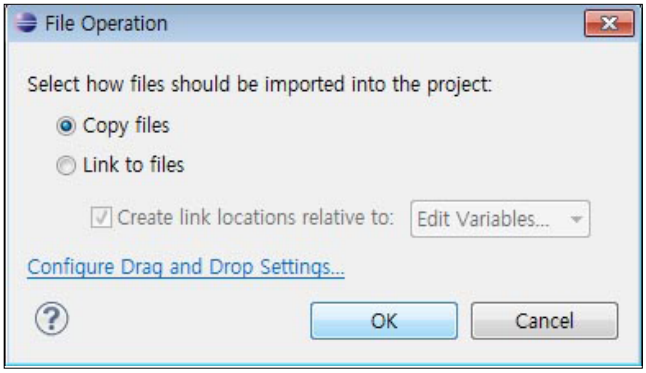
좌측 프로젝트 이름에 커서를 놓고 오른쪽 마우스키를 누릅니다.
제일 하단에 Propertis 를 클릭합니다. Alt+Enter 도 동일합니다.



C/C++ Build 〉 Settings 를 클릭하고, 우측에 MinGW C Linker 〉 Libraties 를 클릭합니다. 우측에 Libraies 에 + 마크를 클릭하고 팝업창이 뜨면 drApi 라고 입력합니다. Library search path 에 + 마크를 클릭하고 팝업창이 뜨면 Workspace 를 클릭하고, Folder selection 에서 프로젝트 이름과 동일한 것을 클릭하고 OK 버튼을 누릅니다.

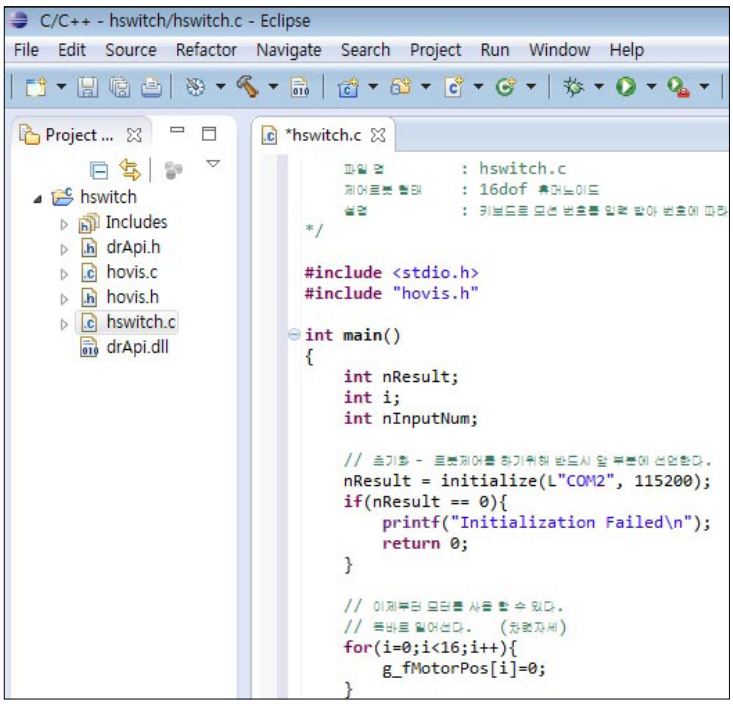
로봇을 동작시키기 위해선 위 네 개 파일을 프로젝트에 포함시켜야 합니다.
위 파일이 있는 파일탐색기 폴더에서 드래그 하는 방식으로 프로젝트에 포함시킵니다.
파일을 끌어다가 드래그해서 왼쪽 프로젝트에 추가합니다.

09 기본 파일 Import



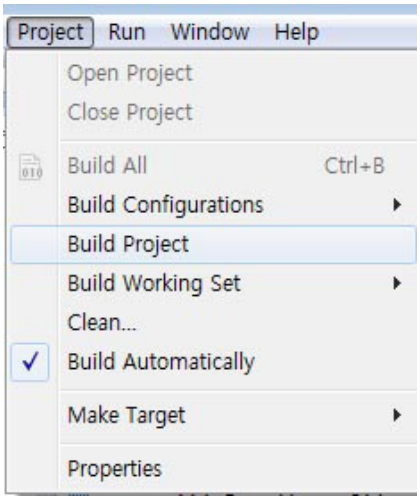
OK 버튼을 누릅니다.

10 소스 작성



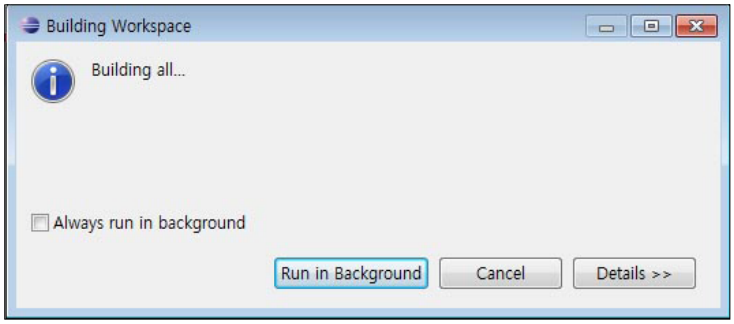
소스를 작성합니다.

11 빌드



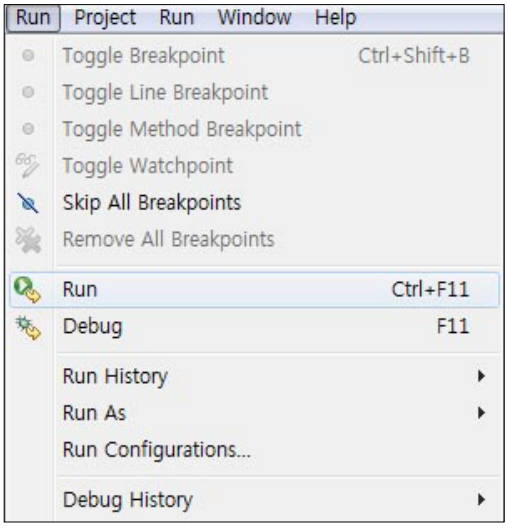
Projec 〉 Build Project 를 클릭합니다.

12 빌드중



빌드 팝업입니다. 그대로 두면 완료 후 사라집니다.

13 Run 실행



Run 〉 Run 을 클릭합니다. Ctrl+F11 단축키도 동일합니다.

14 실행화면

```
            case 9:
            case 10:
            case 11:
            case 12:
                printf("Motion 7\n");
                fflush(stdout);
                motion(7, 0);
                motion_wait();
                break;
            default:
                printf("Wrong Number\n");
                fflush(stdout);
                break;
        }
    }

    // 프로그램 종료
    terminate();
    return 0;
}
```

Problems Tasks Console Properties

hswitch Debug [C/C++ Application] C:₩GccDongbu₩workspacew₩hsv

```
Input Number
2
Motion 2
Input Number
```

15 로봇동작

컴퓨터 키보드로 모션 번호를 입력 받아 번호에 따라서 다른 모션을 실행합니다.

◆ 조건분기 전체 요약

if, if~else, if~else if~else, switch~case

```
if(조건)
{
  조건 만족시 실행”
}
```

```
if(조건 A)
{
    조건 A 만족시 실행
}
else if(조건 B)
{
    조건 B만족시 실행
}
else if(조건 C)
{
    조건 C 만족시 실행
}
else
{
    조건 ABC 불만족시 실행
}
```

```
if(조건)
{
  조건 만족시 실행”
}
else
{
  조건 불만족시 실행
}
```

```
switch(n)

   case1:   → n 이 1인 경우
            Break;
   case2:
            Break;
   case3:
      Break;

default:
```

04 함수

4.1 메인함수

프로그램에서 함수는 하는 일 즉, 기능에 대한 것을 말합니다. 따라서 C 언어는 함수에서 시작해서 함수로 끝난다고 해도 과언이 아닙니다.
main 함수는 함수의 기본 형태로서 아래와 같은 형식을 갖춥니다.

```
int main (void)
{  –〉 main 몸체의 시작
함수의 몸체
}  –〉 main 몸체 종료
```

함수는 이름을 가짐 : main
입력의 형태 : (void)
출력의 형태(반환형, 리턴형 이라는 말을 더 많이 함) : int
함수의 몸체 : 함수의 기능을 의미합니다.
A,B,C 함수를 많이 만들어놓고, 특정한 순서에 맞게 호출하는게 프로그래밍입니다.

아래 예제는 리모콘을 받을 수 있는 상태가 되는 함수를 정의 한 것입니다.
리모콘 파워버튼을 누르면 앉았다 일어나고 모든 모터의 LED가 세번 깜빡이는 프로그램입니다.

hfuncmain.c

```
#include 〈stdio.h〉
#include “hovis.h”
#include “hovis.h”

int main()
{
        int nResult;
        int i, j;
```

```
        nResult = initialize(L"COM2", 115200);
        if(nResult == 0){
        printf("Initialization Failed₩n");
         fflush(stdout);
                return 0;
}
        for(i=0;i<16;i++){
                g_fMotorPos[i]=0;
}
        g_fMotorPos[0] = -90;
        g_fMotorPos[1] = 90;
        g_fMotorPos[3] = -90;
        g_fMotorPos[4] = 90;
        run(1000);

        delay(1000);

        while(1){
                printf("Input Remocon₩n");
                fflush(stdout);
                 while (1) {

                        read();

                        if(g_nRemoconData != 0xFE)
                                break;

                }

                printf("Value = %d₩n", g_nRemoconData);
                fflush(stdout);
                if (g_nRemoconData == 0) {
                        g_fMotorPos[7] = 60;
                        g_fMotorPos[8] = 120;
                        g_fMotorPos[9] = 60;
                        g_fMotorPos[12] = 60;
                        g_fMotorPos[13] = 120;
                        g_fMotorPos[14] = 60;
                        run(1000);

                        delay(1500);
                        g_fMotorPos[7] = 0;
                        g_fMotorPos[8] = 0;
                        g_fMotorPos[9] = 0;
                        g_fMotorPos[12] = 0;
                        g_fMotorPos[13] = 0;
```

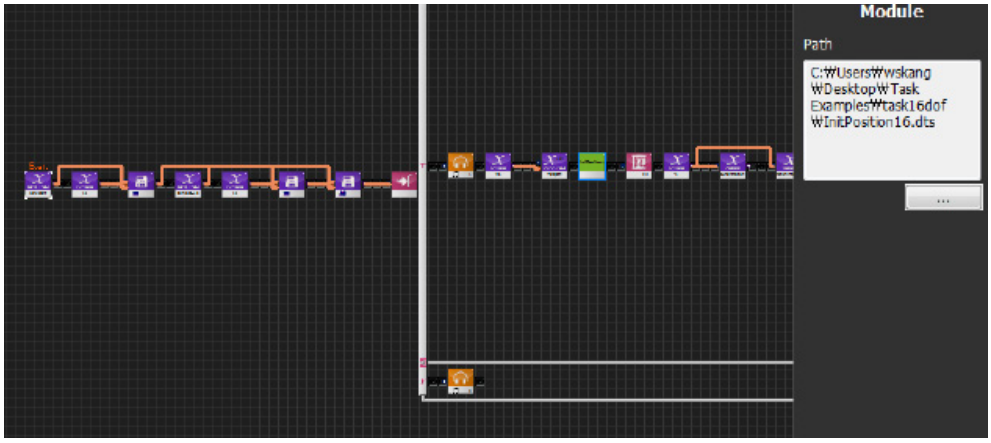
```
                        g_fMotorPos[14] = 0;
                        run(1000);
                        delay(1000);
                        for(i = 0; i < 3; i++) {
                                // On
                                for(j = 0; j < 16; j++){
                                        g_ucMotorRedLed[j] = 1;
                                }
                                run(0);
                                dr_wait_delay(1000);

                                // Off
                                for(j = 0; j < 16; j++){
                                g_ucMotorRedLed[j] = 0;
                                }
                                run(0);

                                dr_wait_delay(1000);

                        }
                }
                else{
                        delay(200);
                }
        }
        terminate();
        return 0;
}
```

위 코드 중에 진하게 처리된 main 문장을 살펴봅니다.
반환형이 int 로 main 이라는 이름을 가진 함수 입니다.

문법요약

◆ 메인함수

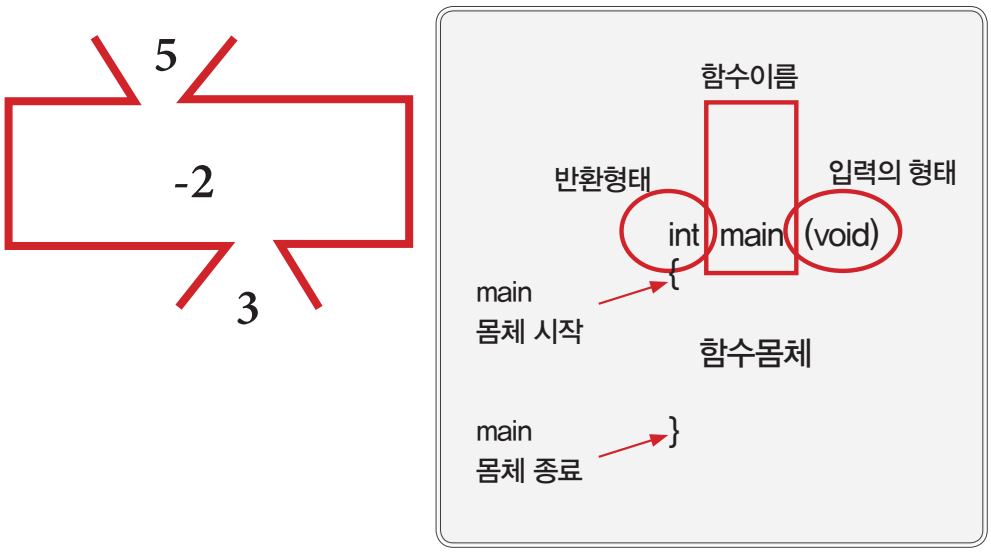
ex) Remocon16.dts 중 일부



C-like 보기

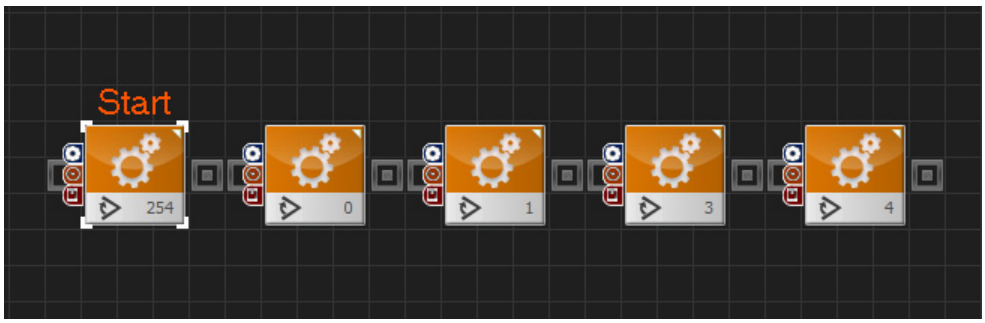
```
1  void InitPosition16()
2  {
3          jog( 512, 0, 254, 120 )
4          jog( 235, 0, 0, 120 )
5          jog( 235, 0, 1, 120 )
6          jog( 789, 0, 3, 120 )
7          jog( 789, 0, 4, 120 )
8  }
9  void main()
10 {
11         if( ( ( MPSU_ServoCnt == 16 ) && ( MPSU_ServoID_15 == 15 ) ) )
12         {
13                 melody( 1 )
14                 SERVO_TorqCtrl[254]=96
15                 InitPosition16()
16                 delay( 2000 )
17                 CurrentMotion=-1
18                 MotionAfterStop=-1
19                 RmcEnd=false
20                 MtnReady=false
21                 MtnStop=false
22                 Interruptable=true
23                 while( true )
```

※ C라는 언어로 구현된 프로그램은 함수로 시작해서, 함수로 끝난다고 해도 과언이 아닙니다. 함수라는 것은 C 언어를 이해하는데, 가장 중요한 일을 합니다. 포인터 가 어려운건 사실이지만, 함수가 훨씬 더 중요하다고 할 수 있습니다. 포인터는 공부하기가 어려운것이지 활용하는 것은 어렵지 않습니다.



첫째, 함수는 이름을 지닙니다. → main
들째, 입력의 형태가 있습니다. → (void)
셋째, 출력의 형태(반환형, 리턴형 이라는 말을 더 많이 함)가 있습니다. → int
넷째, 함수의 몸체 : 함수의 기능을 의미합니다.
A,B,C 함수를 많이 만들어놓고, 특정한 순서에 맞게 호출하는 것이 프로그래밍입니다.

ex)InitPosition16.dts 중 일부



C-like 보기

```
1  void main()
2  {
3          jog( 512, 0, 254, 120 )
4          jog( 235, 0, 0, 120 )
5          jog( 235, 0, 1, 120 )
6          jog( 789, 0, 3, 120 )
7          jog( 789, 0, 4, 120 )
8  }
```

◆ 함수선언

```
(1)  (2)    (3)
int minus (int i, int j)
{
    int result = i–j;
(4) return result;
}
```

(1) 반환 형
(2) 함수 이름
(3) 매개 변수
(4) 값의 반환

※ 4가지 형태의 함수

입력과 출력은 있을 수도 있고, 없을 수도 있습니다. 중요한 것은 기능입니다.
전달 인자 있음, 반환 값 있음
전달 인자 있음, 반환 값 없음
전달 인자 없음, 반환 값 있음
전달 인자 없음, 반환 값 없음

변수의 범위(scope)

※ 변수의 특성에 따른 분류

지역변수 (local variable) : 중괄호 내에 선언되는 변수
전역변수 (Global variable) : 함수 내에 선언되지 않는 변수
정적변수 (Static variable) : 함수 내부, 외부 모두 선언 가능
레지스터 변수(Register variable) : 선언에 제한이 많이 따름

static 변수

함수 내부 및 외부에 선언 가능합니다.
전역변수, 지역변수 앞에 static 키워드를 붙일 수 있습니다. → 파일 범위를 갖습니다.
↔ extern 한번만 초기화된다됩니다.: 전역변수의 특징
전역변수는 프로그램이 시작하면 올라갔다가, 종료되어야 내려갑니다.
지역변수는 함수가 호출될때마다 초기화됩니다.
함수 내부에서 선언될 경우 함수 내에서만 접근이 가능합니다. : 지역변수의 특징

코딩계획

```
// 결과값, for 를 저장하기 위한 변수 선언
//초기화
//모터사용
//무한반복
//리모콘 입력값, 출력값
//로봇 앉기 일어나기 동작
```

프로그래밍 세부설계

```
/*
파일명              : hfuncmain.c
제어로봇 형태       : 16dof 휴머노이드
설명                : 리모콘을 받을 수 있는 상태가 되는 함수 정의 – 리모콘 파워버튼을
누르면 앉았다 일어난다. 그리고 모든 모터의 LED가 세번 깜빡임
*/

// 동부로봇의 api 를 사용하기 위해 반드시 이렇게 선언 해 두어야 한다.

int main()
{
        // 결과값을 리턴받을 변수
        // for 문을 사용하기 위해 선언한 변수

        // 초기화 – 로봇제어를 하기 위해 반드시 앞 부분에 선언한다.
        // 각 변수들을 초기화 한다.
        // 열고 난 이후 이상이 있는 지 확인한다.
                // 이상이 있다면 에러 출력
                // 프로그램 종료

// 이제부터 모터를 사용 할 수 있다.
// 똑바로 일어선다.          (차렷자세)

        // 모든 모터(16개 모터)를 0 위치로 설정함
        // 0 번 모터(오른쪽 어깨)
        // 1 번 모터(오른쪽 윗팔)
        // 3 번 모터(왼쪽 어깨)
        // 4 번 모터(왼쪽 윗팔)
        // 모터 동작(해당 자세를 1000ms 동안 동작)
        // 동작 대기(1000ms)
```

```
// 무한 반복한다.
        // 리모컨을 입력하라는 메시지 출력
        // 리모컨 감지 대기
        // 탈출하기 전까지 무한 반복
                // 리모컨 입력값을 읽기 위해 통신 요청.
              // 리모컨 입력값이 0xFE가 아닌 경우(아무 입력 없을 때가 0xFE)
                        // while문을 탈출한다.
        // 감지 후
        // 리모컨 입력 값을 출력한다.
        // Power 버튼이 눌렸다면
                // 앉기 동작
                // 7 번 모터(오른쪽 다리 앞뒤 방향)
                // 8 번 모터(오른쪽 다리 무릎)
                // 9 번 모터(오른쪽 발 앞뒤 방향)
                // 12 번 모터(왼쪽 다리 앞뒤 방향)
                // 13 번 모터(왼쪽 다리 무릎)
                // 13 번 모터(왼쪽 발 앞뒤 방향)
                // 모터 동작(해당 자세를 1000ms 동안 동작)
                // 동작 대기(1500ms)
                // 일어서기 동작

                // 7 번 모터(오른쪽 다리 앞뒤 방향)
                // 8 번 모터(오른쪽 다리 무릎)
                // 9 번 모터(오른쪽 발 앞뒤 방향)
                // 12 번 모터(왼쪽 다리 앞뒤 방향)
                // 13 번 모터(왼쪽 다리 무릎)
                // 13 번 모터(왼쪽 발 앞뒤 방향)
                // 모터 동작(해당 자세를 1000ms 동안 동작)
                // 동작 대기(1000ms)

                // LED 동작
        // 3번 동안 LED를 켰다가 끄기 위해 for문으로 반복

                        // On
                        // 0 ~ 15번 모터를 설정하기 위해 for문으로 반복
                                // 모터의 빨간색 LED 를 켠다.
                        // 모터 동작(LED 동작)
                        // 대기(1000ms)

                        // Off
```

```
                        // 0 ~ 15번 모터를 설정하기 위해 for문으로 반복
                                // 모터의 빨간색 LED 를 다시 끈다.
                                // 모터 동작(LED 동작)
                                // 대기(1000ms)

                // 그 외의 키가 눌린 경우
                        // 200ms 동안 대기(리모컨 값이 초기화 될 때까지)

        // 프로그램 종료
        // 제어 종료 함수를 실행한다.
```

프로그래밍 작성

```
/*
파일명              : hfuncmain.c
제어로봇 형태       : 16dof 휴머노이드
설명 : 리모콘을 받을 수 있는 상태가 되는 함수 정의 – 리모콘 파워버튼을 누르면 앉았다
일어난다. 그리고 모든 모터의 LED가 세 번 깜빡임
*/

#include 〈stdio.h〉
#include "hovis.h" // 동부로봇의 api 를 사용하기 위해 반드시 이렇게 선언 해 두어야 한다.

int main()
{
        int nResult;                    // 결과값을 리턴받을 변수
        int i, j;               // for 문을 사용하기 위해 선언한 변수

        // 초기화 – 로봇제어를 하기 위해 반드시 앞 부분에 선언한다.
        nResult = initialize(L"COM2", 115200);// 각 변수들을 초기화함
        if(nResult == 0){   // 열고 난 이후 이상이 있는 지 확인한다.
                printf("Initialization Failed₩n");          // 이상이 있다면 에러 출력
                fflush(stdout);
                return 0;
        // 프로그램 종료
        }

        // 이제부터 모터를 사용 할 수 있다.
        // 똑바로 일어선다.             (차렷자세)
        for(i=0;i〈16;i++){
                g_fMotorPos[i]=0;
        // 모든 모터(16개 모터)를 0 위치로 설정함
        }
```

```
g_fMotorPos[0] = –90;                    // 0 번 모터(오른쪽 어깨)
g_fMotorPos[1] = 90;                     // 1 번 모터(오른쪽 윗팔)
g_fMotorPos[3] = –90;                    // 3 번 모터(왼쪽 어깨)
g_fMotorPos[4] = 90;                     // 4 번 모터(왼쪽 윗팔)
run(1000);          // 모터 동작(해당 자세를 1000ms 동안 동작)
delay(1000);                             // 동작 대기(1000ms)

while(1){                                // 무한 반복한다.
        // 무한 반복한다.
        printf("Input Remocon₩n"); // 리모컨을 입력하라는 메시지 출력
         fflush(stdout);
        // 리모컨 감지 대기
        while (1) {                      // 탈출하기 전까지 무한 반복
                read();    // 리모컨 입력값을 읽기 위해 통신 요청
                if(g_nRemoconData != 0xFE)
// 리모컨 입력값이 0xFE가 아닌 경우(아무 입력 없을 때가 0xFE)
                        break;              // while문을 탈출한다.
        }

        // 감지 후
        printf("Value = %d₩n", g_nRemoconData); // 리모컨 입력 값을 출력한다.
        fflush(stdout);
        if (g_nRemoconData == 0) {// Power 버튼이 눌렸다면
                // 앉기 동작
        else if(strcmp(str, "sit")==0){
                // 문자열이 sit와 같을 경우
                // 앉기 동작
                g_fMotorPos[7] = 60;
// 7 번 모터(오른쪽 다리 앞뒤 방향)
                g_fMotorPos[8] = 120;
// 8 번 모터(오른쪽 다리 무릎)
                g_fMotorPos[9] = 60;
// 9 번 모터(오른쪽 발 앞뒤 방향)
                g_fMotorPos[12] = 60;
// 12 번 모터(왼쪽 다리 앞뒤 방향)
                g_fMotorPos[13] = 120;
// 13 번 모터(왼쪽 다리 무릎)
                g_fMotorPos[14] = 60;
// 13 번 모터(왼쪽 발 앞뒤 방향)
```

```
                    run(1000);
                    // 모터 동작(해당 자세를 1000ms 동안 동작)
                    delay(1000);        // 동작 대기(1500ms)
                    // 일어서기 동작
                    g_fMotorPos[7] = 0;
                    // 7 번 모터(오른쪽 다리 앞뒤 방향)
                    g_fMotorPos[8] = 0;
                // 8 번 모터(오른쪽 다리 무릎)
                    g_fMotorPos[9] = 0;
                // 9 번 모터(오른쪽 발 앞뒤 방향)
                    g_fMotorPos[12] = 0;
        // 12 번 모터(왼쪽 다리 앞뒤 방향)
                    g_fMotorPos[13] = 0;
        // 13 번 모터(왼쪽 다리 무릎)
                    g_fMotorPos[14] = 0;
        // 13 번 모터(왼쪽 발 앞뒤 방향)
                    run(1000);
                    // 모터 동작(해당 자세를 1000ms 동안 동작)
                    delay(1000);                // 동작 대기(1000ms)
                    // LED 동작
                    for(i = 0; i < 3; i++) {
// 3번 동안 LED를 켰다가 끄기 위해 for문으로 반복
                        // On
                        for(j = 0; j < 16; j++){
// 0 ~ 15번 모터를 설정하기 위해 for문으로 반복
                            g_ucMotorRedLed[j] = 1;
        // 모터의 빨간색 LED 를 켠다.
                        }
                        run(0);             // 모터 동작(LED 동작)
                        dr_wait_delay(1000);
        // 대기(1000ms)
                        // Off
                        for(j = 0; j < 16; j++){
// 0 ~ 15번 모터를 설정하기 위해 for문으로 반복
                            g_ucMotorRedLed[j] = 0;
        // 모터의 빨간색 LED 를 다시 끈다.
                        }
```

```
                                        run(0);                 // 모터 동작(LED 동작)
                                        dr_wait_delay(1000);
        // 대기(1000ms)
                                }
                        }
                        else{
                                g_fMotorPos[8] = 0;
                        // 8 번 모터(오른쪽 다리 무릎)
                                g_fMotorPos[9] = 0;
                        // 9 번 모터(오른쪽 발 앞뒤 방향)
                                g_fMotorPos[12] = 0;
        // 12 번 모터(왼쪽 다리 앞뒤 방향)
                                g_fMotorPos[13] = 0;
        // 13 번 모터(왼쪽 다리 무릎)
                                g_fMotorPos[14] = 0;
        // 13 번 모터(왼쪽 발 앞뒤 방향)
                                run(1000);
                                // 모터 동작(해당 자세를 1000ms 동안 동작)
                                delay(1000);                    // 동작 대기(1000ms)

                                // LED 동작
                                for(i = 0; i < 3; i++) { // 3번 동안 LED를 켰다가 끄기 위해 for문으로 반복
                                        // On
                                        for(j = 0; j < 16; j++){ // 0 ~ 15번 모터를 설정하기 위해 for문으로 반복
                                                g_ucMotorRedLed[j] = 1; // 모터의 빨간색 LED 를 켠다.
                                        }
                                        run(0);                 // 모터 동작(LED 동작)
                                        dr_wait_delay(1000); // 대기(1000ms)

                                        // Off
                                        for(j = 0; j < 16; j++){
// 0 ~ 15번 모터를 설정하기 위해 for문으로 반복
                                                g_ucMotorRedLed[j] = 0;
        // 모터의 빨간색 LED 를 다시 끈다.
                                        }
                                        run(0);                 // 모터 동작(LED 동작)
                                        dr_wait_delay(1000);
```

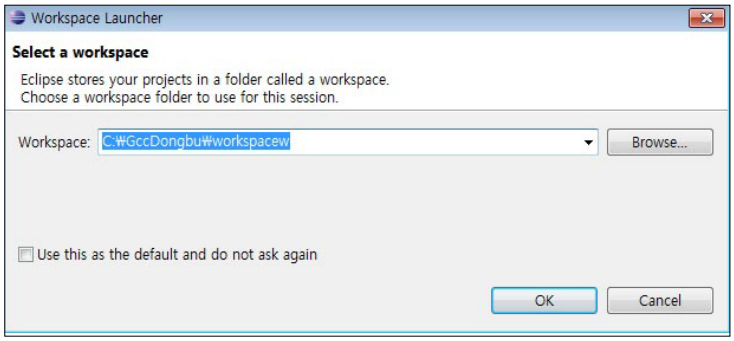
```
        // 대기(1000ms)
                        }
                }
                else{
                // 그 외의 키가 눌린 경우
                        delay(200);
                        // 200ms 동안 대기(리모컨 값이 초기화 될 때까지)
                }
        }

        // 프로그램 종료
        terminate();                    // 제어 종료 함수를 실행한다.
        return 0;
}
```

빌드 및 실행

01 이클립스 실행

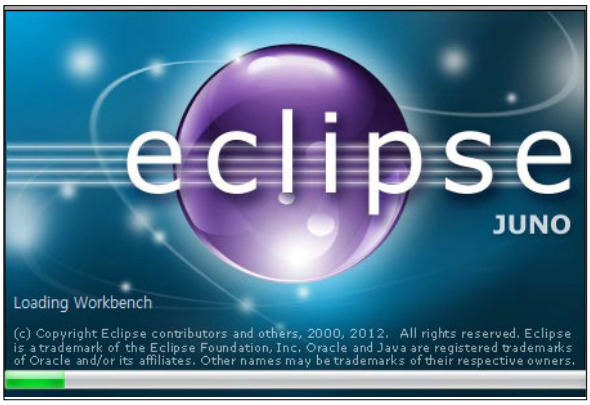
이클립스 실행 버튼을 누릅니다.

02 workspace



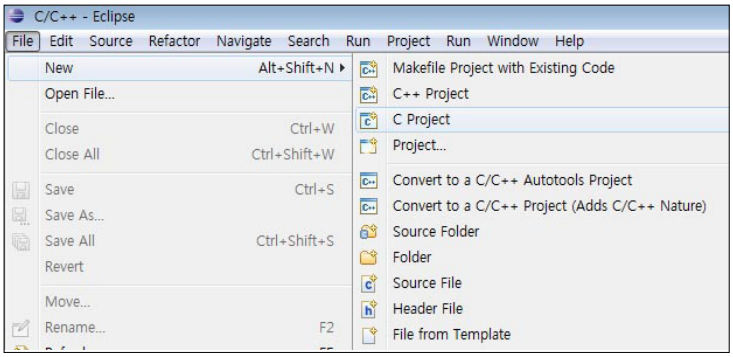
작업하고자 하는 폴더를 지정합니다. Browse 를 클릭하여 폴더를 지정하고 OK 버튼을 누릅니다.

03 이클립스 로딩 화면



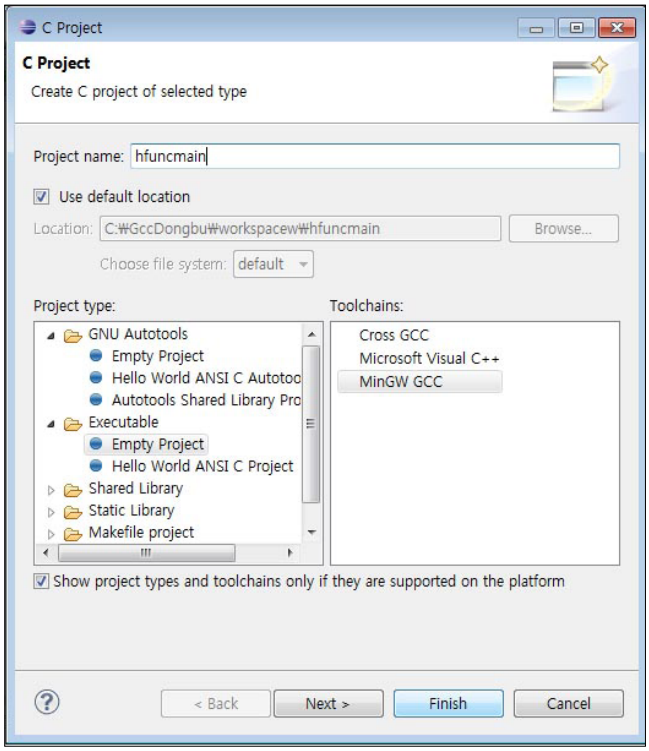
이클립스 로딩 화면입니다.

04 프로젝트 생성



이클립스 편집창 메뉴에서 File 〉 New 〉 C project 를 클릭합니다.

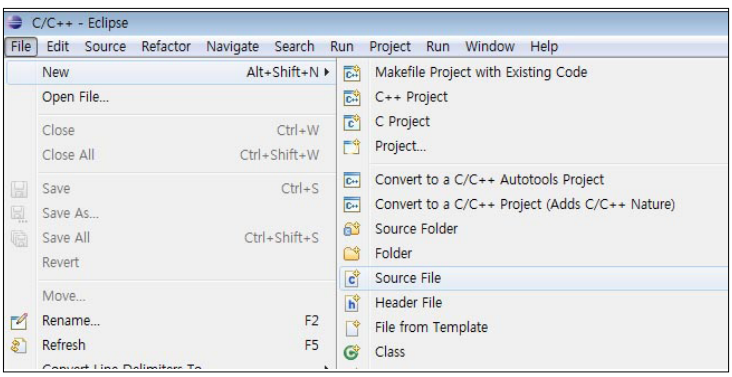
05 프로젝트 이름



Projec Name 을 hfuncmain 라고 입력하고, Project type 에서 Exectable 에서 Empty Project 를 선택합니다.

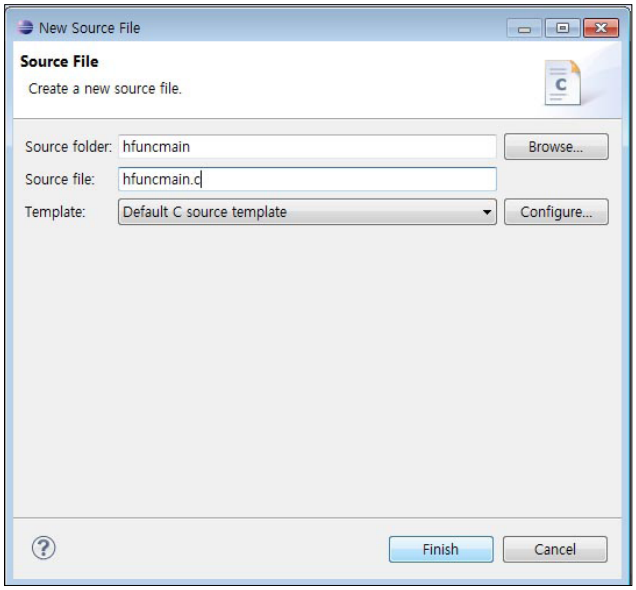
Toolchians 에서 MinGW GCC 를 선택하고 Finish 버튼을 누릅니다.

06 소스파일 추가



File 〉New 〉Source File 을 클릭합니다.

07 소스파일 이름

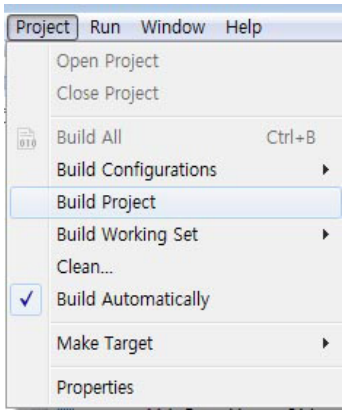


hfuncmain.c 라고 입력하고 Finish 버튼을 누릅니다.

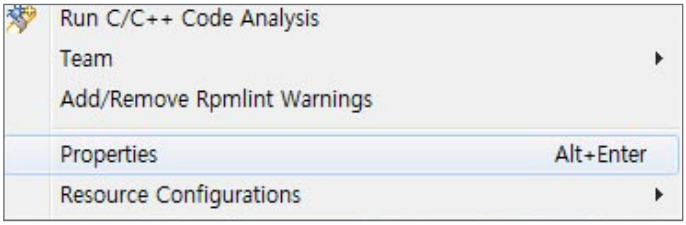
08 기본 제공 파일 추가

본 교재에게 C 프로그래밍을 하기 위해선 기본 제공되는 라이브러리 파일을 먼저 추가해주어야 올바른 컴파일을 할 수 있습니다. 필요한 파일은 drApi.dll, drApi.h, hovis.c, hovis.h 등 총 4가지 입니다.

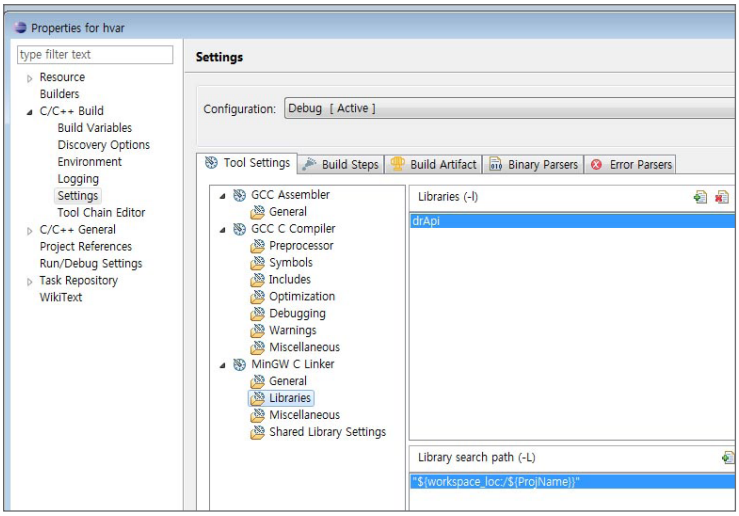
파일 추가 전에 먼저 Property 에서 라이브러리를 설정해주어야 합니다. 파일 생성을 한 후에 아무 코딩도 하지 말고, 바로 Build project 를 실행하세요. 그래야 라이브러리 추가가 가능합니다.



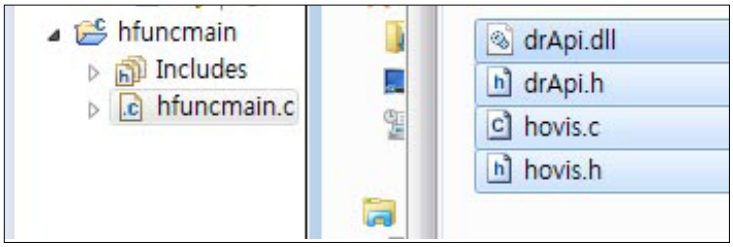
Build Project 를 클릭합니다.



좌측 프로젝트 이름에 커서를 놓고 오른쪽 마우스키를 누릅니다.
제일 하단에 Propertis 를 클릭합니다. Alt+Enter 도 동일합니다.

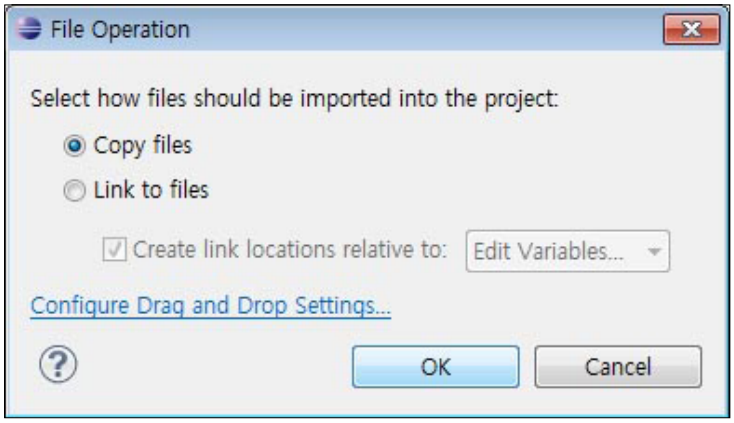


C/C++ Build 〉 Settings 를 클릭하고, 우측에 MinGW C Linker 〉 Libraties 를 클릭합니다. 우측에 Libraies 에 + 마크를 클릭하고 팝업창이 뜨면 drApi 라고 입력합니다. Library search path 에 + 마크를 클릭하고 팝업창이 뜨면 Workspace 를 클릭하고, Folder selection 에서 프로젝트 이름과 동일한 것을 클릭하고 OK 버튼을 누릅니다.



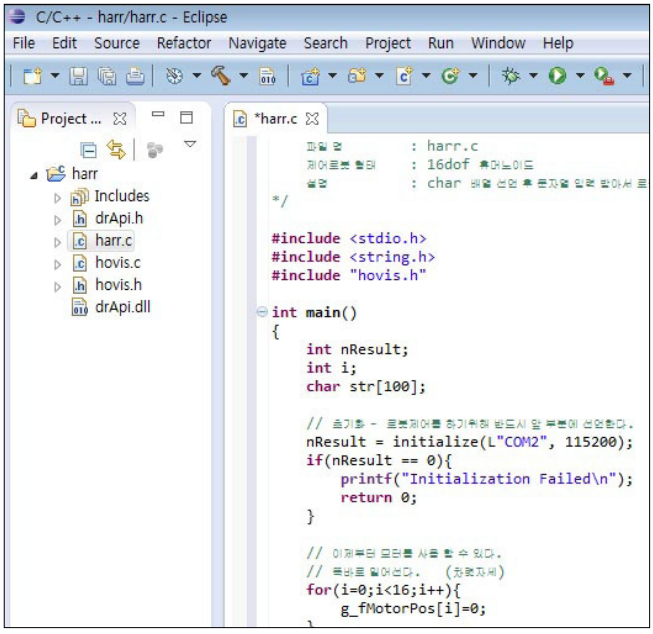
로봇을 동작시키기 위해선 위 네 개 파일을 프로젝트에 포함시켜야 합니다.
위 파일이 있는 파일탐색기 폴더에서 드래그 하는 방식으로 프로젝트에 포함시킵니다.
파일을 끌어다가 드래그해서 왼쪽 프로젝트에 추가합니다.

09 기본 파일 Import



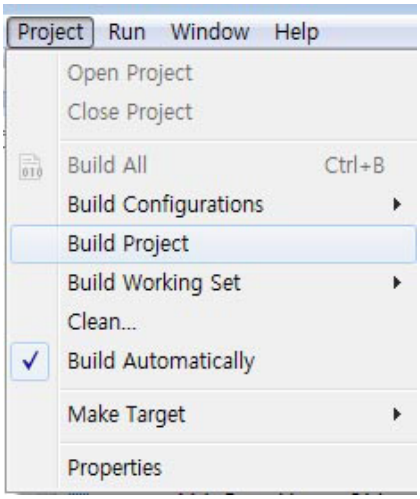
OK 버튼을 누릅니다.

10 소스 작성



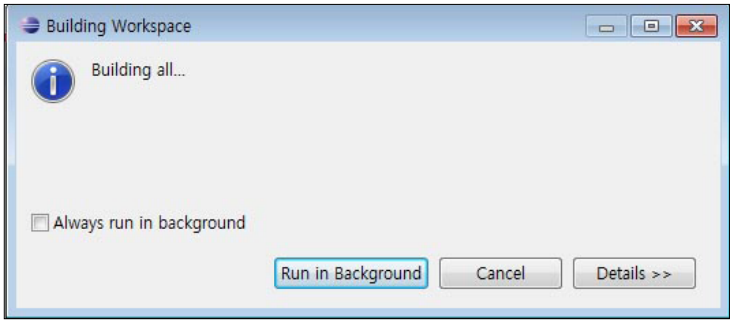
소스를 작성합니다.

11 빌드



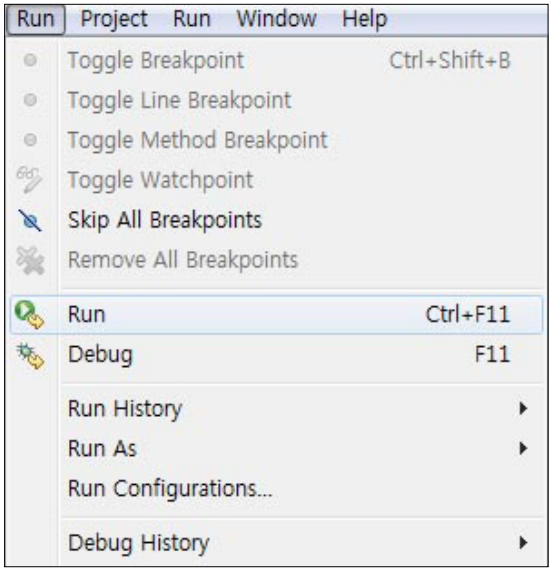
Projec 〉 Build Project 를 클릭합니다.

12 빌드중



빌드 팝업입니다. 그대로 두면 완료 후 사라집니다.

13 Run 실행



Run 〉 Run 을 클릭합니다. Ctrl+F11 단축키도 동일합니다.

14 실행화면

```
                // Off
                for(j = 0; j < 16; j++){
                    g_ucMotorRedLed[j] = 0;
                }
                run(0);
                dr_wait_delay(1000);

            }
        }
        else{
            delay(200);
        }
    }

    // 프로그램 종료
    terminate();
    return 0;
}
```

Problems Tasks Console Properties

```
hfuncmain.exe [C/C++ Application] C:₩GccDongbu₩workspacew
Input Remocon
```

15 로봇동작

PC 키보드에서 right 를 치면 오른손을 들고, left 를 치면 왼손을 듭니다.

4.2 함수호출

기능별로 구분해서 함수를 각각 만들어놓고, 메인함수 안에서 호출을 하면 그 함수의 기능을 사용할 수 있습니다.
함수 호출 과정을 살펴보면, 프로그램을 실행시키면 기본적으로 메인함수가 실행이 됩니다.
int add(int a, int b) –〉 int 형으로 반환하고, 매개변수를 2가지를 가지고 있는 함수를 의미합니다.

```
int add(int a, int b); –〉함수 원형 선언
    int main()
    {
            add(2,3);
            return result;
}
```

컴파일러의 특성상, 함수는 호출되기 전에 정의되어야 합니다.

아래 예제는 리모콘을 받는 상태가 되는 함수를 irreceive()로 정의, 이 함수를 호출하여 리모콘 값을 읽습니다. 리모콘 파워버튼을 누르면 앉았다 일어납니다. 그리고 모든 모터의 LED가 세 번 깜빡입니다.

hfunccall.c

```
#include 〈stdio.h〉
#include “hovis.h”

int irreceive()
{
        while (1) {
                read();

                if(g_nRemoconData != 0xFE)
                        break;
        }

        // 감지 후
        return g_nRemoconData;
}

int main()
{
```

```
int nResult;
int i, j;
int nData;
nResult = initialize(L"COM2", 115200);
if(nResult == 0){
        printf("Initialization Failed₩n");
        fflush(stdout);
        return 0;
}

for(i=0;i<16;i++){
        g_fMotorPos[i]=0;
}
g_fMotorPos[0] = -90;
g_fMotorPos[1] = 90;
g_fMotorPos[3] = -90;
g_fMotorPos[4] = 90;
run(1000);

delay(1000);

while(1){

printf("Input Remocon₩n");
fflush(stdout);
nData = irreceive();
printf("Value = %d₩n", nData);
fflush(stdout);
if (nData == 0) {
        g_fMotorPos[7] = 60;
        g_fMotorPos[8] = 120;
        g_fMotorPos[9] = 60;
        g_fMotorPos[12] = 60;
        g_fMotorPos[13] = 120;
        g_fMotorPos[14] = 60;
        run(1000);
        delay(1500);
        g_fMotorPos[7] = 0;
        g_fMotorPos[8] = 0;
        g_fMotorPos[9] = 0;
        g_fMotorPos[12] = 0;
        g_fMotorPos[13] = 0;
```

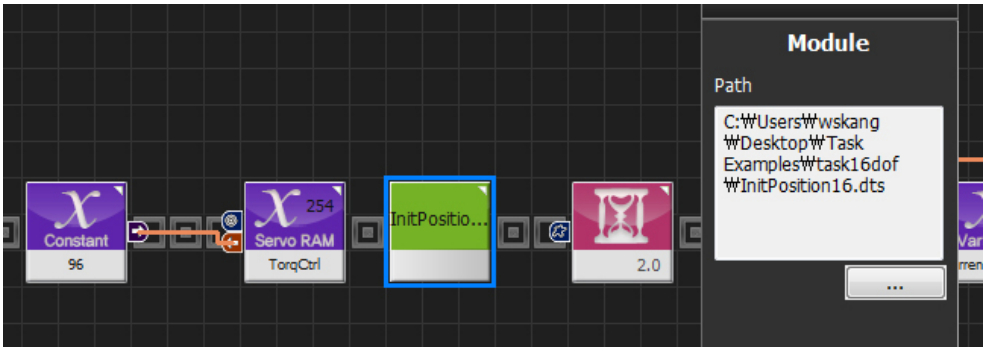
```
                    run(1000);
                    delay(1000);
                    for(i = 0; i < 3; i++) {
                        // On
                        for(j = 0; j < 16; j++){
                            g_ucMotorRedLed[j] = 1;
                        }
                        run(0);
                        dr_wait_delay(1000);

                        // Off
                        for(j = 0; j < 16; j++){
                            g_ucMotorRedLed[j] = 0;
                        }
                        run(0);
                        dr_wait_delay(1000);
                    }
                }
                else{
                    delay(200);
                }
            }

            terminate();
            return 0;
}
```

위 코드 중에 진하게 처리된 irrecieve 함수 정의와 호출 문장을 살펴봅니다.
irreceive() 함수를 정의하고, main 함수에서 nData = irreceive(); 방식으로 호출합니다.

문법요약

◆ 함수 호출 및 모듈화 프로그래밍

ex) Remocon16.dts 중 일부



C-like 보기

```
1  void InitPosition16()
2  {
3          jog( 512, 0, 254, 120 )
4          jog( 235, 0, 0, 120 )
5          jog( 235, 0, 1, 120 )
6          jog( 789, 0, 3, 120 )
7          jog( 789, 0, 4, 120 )
8  }
9  void main()
10 {
11         if( ( ( MPSU_ServoCnt == 16 ) && ( MPSU_ServoID_15 == 15 ) ) )
12         {
13                 melody( 1 )
14                 SERVO_TorqCtrl[254]=96
15                 InitPosition16()
16                 delay( 2000 )
17                 CurrentMotion=-1
18                 MotionAfterStop=-1
19                 RmcEnd=false
20                 MtnReady=false
21                 MtnStop=false
22                 Interruptable=true
23                 while( true )
24                 {
25                         if( ( ( MPSU_RmcLength >= 0 && MPSU_RmcD
26                         {
```


● 모듈화 프로그래밍

– 기능별로 파일을 나눠가며 프로그래밍하는 것을 말합니다.

– 유지 보수성이 좋아지고, 관리하기가 편해집니다.

예를 들어 200~300 라인은 괜찮지만, 500 라인~ 1000 라인 이상이면 구분도 어렵고, 유지보수도 어려워집니다.

● 파일의 분할 및 컴파일

– 파일을 나눌지라도 완전히 독립되는 것은 아닙니다.

– 파일이 나뉘어도 상호 참조가 발생할 수 있는데, 이는 전역 변수 및 전역 함수로 제한됩니다.

– 같은 코드를 두개의 파일로 분리, 분리된 파일은 하나의 프로젝트로 분류되어야 합니다.

코딩계획

```
//리모콘 수신하는 함수
// 결과값, for, irrecieve로 부터 받는 값을 저장하기 위한 변수 선언
//초기화
//모터사용
//무한반복
//리모콘 입력
//모터, LED 동작
```

프로그래밍 세부설계

```
/*
파일명              : hfunccall.c
제어로봇 형태       : 16dof 휴머노이드
설명                : 리모콘을 받는 상태가 되는 함수를 irreceive()로 정의, 이 함수를 호
출하여 리모콘 값을 읽는다. – 리모콘 파워버튼을 누르면 앉았다 일어난다. 그리고 모
든 모터의 LED가 세 번 깜빡임
*/

// 동부로봇의 api 를 사용하기 위해 반드시 이렇게 선언 해 두어야 한다.

// 리모컨을 수신하는 함수
int irreceive()
{
        // 리모컨 감지 대기
        // 탈출하기 전까지 무한 반복
                // 리모컨 입력값을 읽기 위해 통신 요청.
        // 리모컨 입력값이 0xFE가 아닌 경우(아무 입력 없을 때가 0xFE)
                        // while문을 탈출한다.

        // 감지 후
        // 리모컨 입력값을 리턴한다.

int main()
{
        // 결과값을 리턴받을 변수
        // for 문을 사용하기 위해 선언한 변수
        // irreceive 함수로부터 데이터를 받기 위한 변수

        // 초기화 – 로봇제어를 하기 위해 반드시 앞 부분에 선언한다.
        // 각 변수들을 초기화 한다.
```

```
// 열고 난 이후 이상이 있는 지 확인한다.
        // 이상이 있다면 에러 출력
        // 프로그램 종료

// 이제부터 모터를 사용 할 수 있다.
// 똑바로 일어선다.        (차렷자세)

        // 모든 모터(16개 모터)를 0 위치로 설정함

// 0 번 모터(오른쪽 어깨)
// 1 번 모터(오른쪽 윗팔)
// 3 번 모터(왼쪽 어깨)
// 4 번 모터(왼쪽 윗팔)
// 모터 동작(해당 자세를 1000ms 동안 동작)
// 동작 대기(1000ms)

// 무한 반복한다.
        // 리모컨을 입력하라는 메시지 출력

// 리모컨 값을 받는 함수를 실행해 결과 값을 nData에 저장.

        // 감지 후
        // 리모컨 입력 값을 출력한다.

        // Power 버튼이 눌렸다면
            // 앉기 동작
            // 7 번 모터(오른쪽 다리 앞뒤 방향)
            // 8 번 모터(오른쪽 다리 무릎)
            // 9 번 모터(오른쪽 발 앞뒤 방향)
            // 12 번 모터(왼쪽 다리 앞뒤 방향)
            // 13 번 모터(왼쪽 다리 무릎)
            // 13 번 모터(왼쪽 발 앞뒤 방향)
            // 모터 동작(해당 자세를 1000ms 동안 동작)
            // 동작 대기(1500ms)

            // 일어서기 동작
            // 7 번 모터(오른쪽 다리 앞뒤 방향)
            // 8 번 모터(오른쪽 다리 무릎)
            // 9 번 모터(오른쪽 발 앞뒤 방향)
            // 12 번 모터(왼쪽 다리 앞뒤 방향)
            // 13 번 모터(왼쪽 다리 무릎)
            // 13 번 모터(왼쪽 발 앞뒤 방향)
            // 모터 동작(해당 자세를 1000ms 동안 동작)
            // 동작 대기(1000ms)
```

```
                        // LED 동작
                        // 3번 동안 LED를 켰다가 끄기 위해 for문으로 반복
                                // On
                                // 0 ~ 15번 모터를 설정하기 위해 for문으로 반복
                                        // 모터의 빨간색 LED 를 켠다.

                                // 모터 동작(LED 동작)
                                // 대기(1000ms)

                                // Off
                                // 0 ~ 15번 모터를 설정하기 위해 for문으로 반복
                                        // 모터의 빨간색 LED 를 다시 끈다.
                                // 모터 동작(LED 동작)
                                // 대기(1000ms)

                // 그 외의 키가 눌린 경우
                        // 200ms 동안 대기(리모컨 값이 초기화 될 때까지)
        // 프로그램 종료
        // 제어 종료 함수를 실행한다.
```

프로그래밍 작성

```
/*
파일 명             : hfunccall.c
제어로봇 형태       : 16dof 휴머노이드
설명                : 리모콘을 받는 상태가 되는 함수를 irreceive()로 정의, 이 함수를 호
출하여 리모콘 값을 읽는다. – 리모콘 파워버튼을 누르면 앉았다 일어난다. 그리고 모
든 모터의 LED가 세 번 깜빡임

*/

#include <stdio.h>
#include "hovis.h"
        // 동부로봇의 api 를 사용하기 위해 반드시 이렇게 선언 해 두어야 한다.

// 리모컨을 수신하는 함수
int irreceive()
{
        // 리모컨 감지 대기
        while (1) {
                // 탈출하기 전까지 무한 반복
                read();             // 리모컨 입력값을 읽기 위해 통신 요청.
                if(g_nRemoconData != 0xFE)
        // 리모컨 입력값이 0xFE가 아닌 경우(아무 입력 없을 때가 0xFE)
                        break;  // while문을 탈출한다.
```

```
        }

        // 감지 후
        return g_nRemoconData; // 리모컨 입력값을 리턴한다.
}

int main()
{
        int nResult;                                    // 결과값을 리턴받을 변수
        int i, j;                       // for 문을 사용하기 위해 선언한 변수
        int nData;          // irreceive 함수로부터 데이터를 받기 위한 변수
        // 초기화 – 로봇제어를 하기위해 반드시 앞 부분에 선언한다.
        nResult = initialize(L"COM2", 115200);          // 각 변수들을 초기화 한다.
        if(nResult == 0){  // 열고 난 이후 이상이 있는 지 확인한다.
                printf("Initialization Failed₩n");      // 이상이 있다면 에러 출력
                fflush(stdout);
                return 0;                       // 프로그램 종료
        }

        // 이제부터 모터를 사용 할 수 있다.
        // 똑바로 일어선다.          (차렷자세)
        for(i=0;i<16;i++){
                g_fMotorPos[i]=0;
                // 모든 모터(16개 모터)를 0 위치로 설정함
        }
        g_fMotorPos[0] = –90;                   // 0 번 모터(오른쪽 어깨)
        g_fMotorPos[1] = 90;                    // 1 번 모터(오른쪽 윗팔)
        g_fMotorPos[3] = –90;                   // 3 번 모터(왼쪽 어깨)
        g_fMotorPos[4] = 90;                    // 4 번 모터(왼쪽 윗팔)
        run(1000);          // 모터 동작(해당 자세를 1000ms 동안 동작)
        delay(1000);                            // 동작 대기(1000ms)

        while(1){                       // 무한 반복한다.
                printf("Input Remocon₩n"); // 리모컨을 입력하라는 메시지 출력
                fflush(stdout);
                nData = irreceive(); // 리모컨 값을 받는 함수를 실행해 결과 값을 nData에 저장.
                // 감지 후
                printf("Value = %d₩n", nData);  // 리모컨 입력 값을 출력한다.
                fflush(stdout);
                if (nData == 0) {
        // Power 버튼이 눌렸다면
                        // 앉기 동작
```

```
                g_fMotorPos[7] = 60;
// 7 번 모터(오른쪽 다리 앞뒤 방향)
                g_fMotorPos[8] = 120;
// 8 번 모터(오른쪽 다리 무릎)
                g_fMotorPos[9] = 60;
// 9 번 모터(오른쪽 발 앞뒤 방향)
                g_fMotorPos[12] = 60;
// 12 번 모터(왼쪽 다리 앞뒤 방향)
                g_fMotorPos[13] = 120;
// 13 번 모터(왼쪽 다리 무릎)
                g_fMotorPos[14] = 60;
// 13 번 모터(왼쪽 발 앞뒤 방향)
                run(1000);
                // 모터 동작(해당 자세를 1000ms 동안 동작)
                delay(1500);
        // 동작 대기(1500ms)

                // 일어서기 동작
                g_fMotorPos[7] = 0;
        // 7 번 모터(오른쪽 다리 앞뒤 방향)
                g_fMotorPos[8] = 0;
        // 8 번 모터(오른쪽 다리 무릎)
                g_fMotorPos[9] = 0;
        // 9 번 모터(오른쪽 발 앞뒤 방향)
                g_fMotorPos[12] = 0;
// 12 번 모터(왼쪽 다리 앞뒤 방향)
                g_fMotorPos[13] = 0;
// 13 번 모터(왼쪽 다리 무릎)
                g_fMotorPos[14] = 0;
// 13 번 모터(왼쪽 발 앞뒤 방향)
                run(1000);
                // 모터 동작(해당 자세를 1000ms 동안 동작)
                delay(1000);
        // 동작 대기(1000ms)

                // LED 동작
```

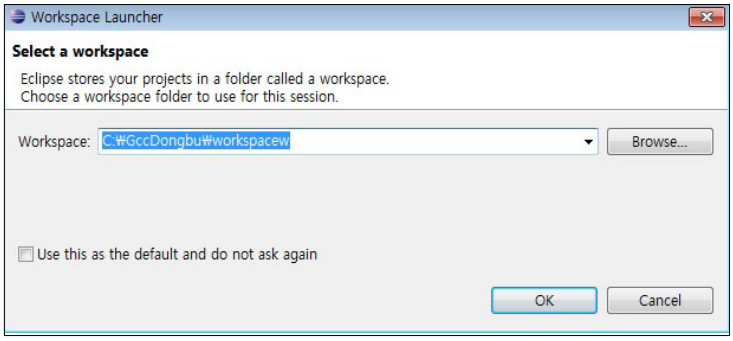
```
                    for(i = 0; i < 3; i++) {
// 3번 동안 LED를 켰다가 끄기 위해 for문으로 반복
                        // On
                        for(j = 0; j < 16; j++){
// 0 ~ 15번 모터를 설정하기 위해 for문으로 반복
                            g_ucMotorRedLed[j] = 1;
        // 모터의 빨간색 LED 를 켠다.
                        }
                        run(0);
                    // 모터 동작(LED 동작)
                        dr_wait_delay(1000);
        // 대기(1000ms)

                        // Off
                        for(j = 0; j < 16; j++){
// 0 ~ 15번 모터를 설정하기 위해 for문으로 반복
                            g_ucMotorRedLed[j] = 0;
        // 모터의 빨간색 LED 를 다시 끈다.
                        }
                        run(0);
                    // 모터 동작(LED 동작)
                        dr_wait_delay(1000);
        // 대기(1000ms)

                    }
                }
                else{
                    // 그 외의 키가 눌린 경우
                    delay(200);
                    // 200ms 동안 대기(리모컨 값이 초기화 될 때까지)
                }
        }

        // 프로그램 종료
        terminate();
            // 제어 종료 함수를 실행한다.
        return 0;
}
```

빌드 및 실행

01 이클립스 실행

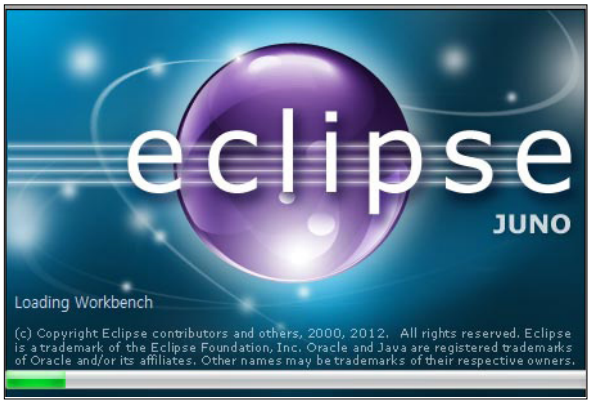
이클립스 실행 버튼을 누릅니다.

02 workspace



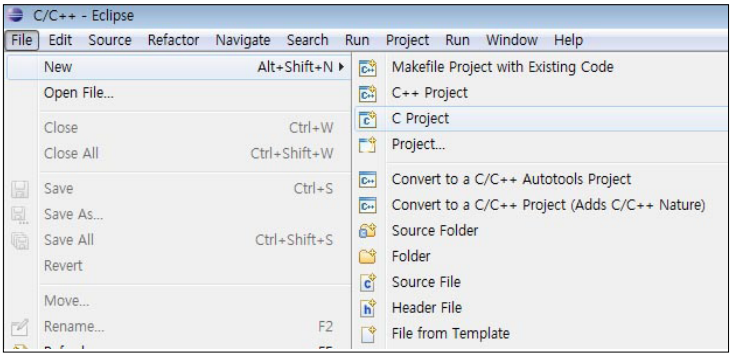
작업하고자 하는 폴더를 지정합니다. Browse 를 클릭하여 폴더를 지정하고 OK 버튼을 누릅니다.

03 이클립스 로딩 화면



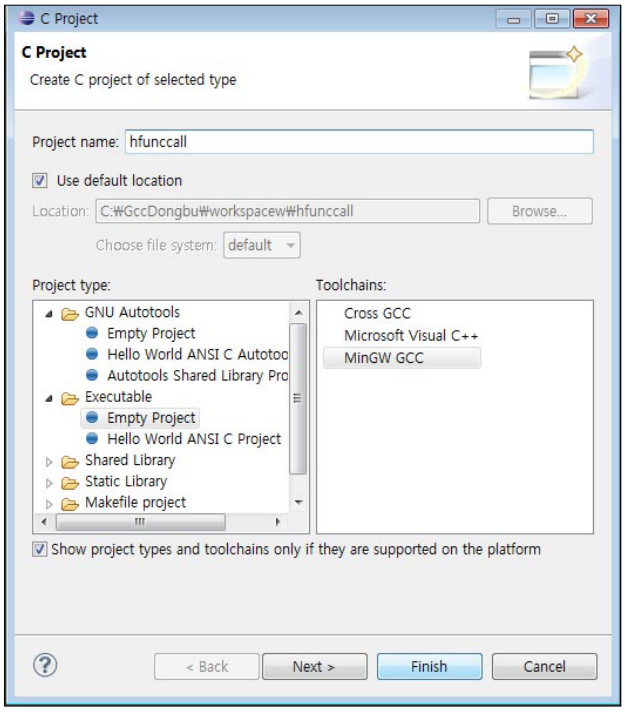
이클립스 로딩 화면입니다.

04 프로젝트 생성



이클립스 편집창 메뉴에서 File 〉 New 〉 C project 를 클릭합니다.

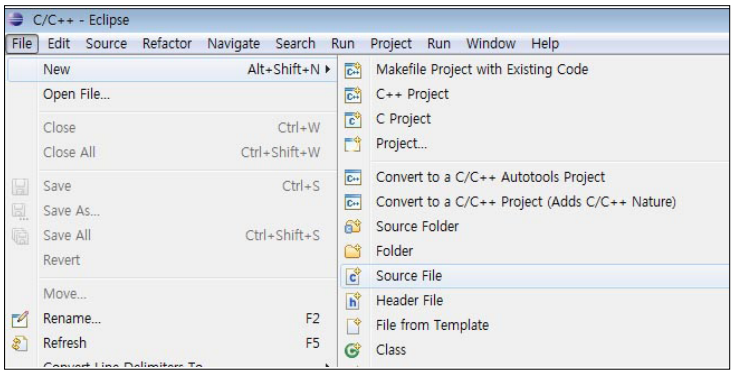
05 프로젝트 이름



Projec Name 을 hfunccall 라고 입력하고, Project type 에서 Exectable 에서 Empty Project 를 선택합니다.

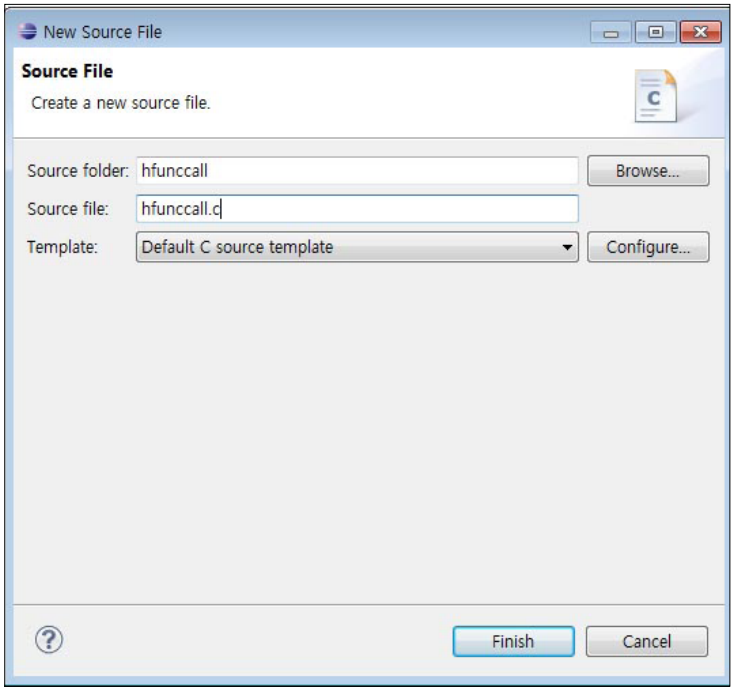
Toolchians 에서 MinGW GCC 를 선택하고 Finish 버튼을 누릅니다.

06 소스파일 추가



File 〉New 〉Source File 을 클릭합니다.

07 소스파일 이름

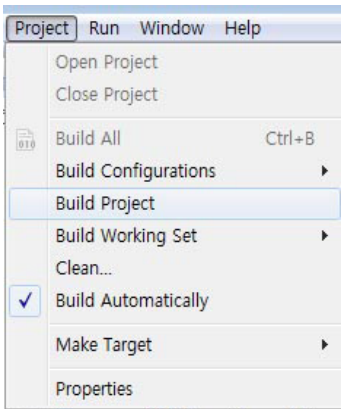


hfunccall.c 라고 입력하고 Finish 버튼을 누릅니다.

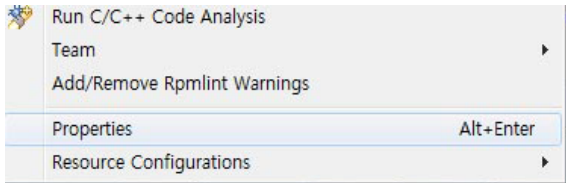
08 기본 제공 파일 추가

본 교재에게 C 프로그래밍을 하기 위해선 기본 제공되는 라이브러리 파일을 먼저 추가해주어야 올바른 컴파일을 할 수 있습니다. 필요한 파일은 drApi.dll, drApi.h, hovis.c, hovis.h 등 총 4가지 입니다.

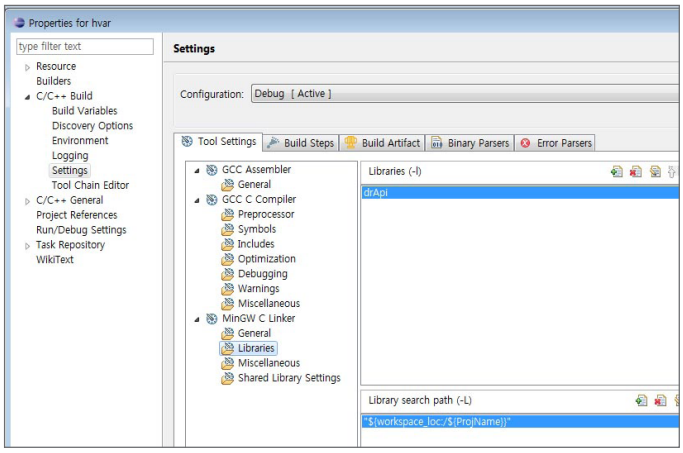
파일 추가 전에 먼저 Property 에서 라이브러리를 설정해주어야 합니다. 파일 생성을 한 후에 아무 코딩도 하지 말고, 바로 Build project 를 실행하세요. 그래야 라이브러리 추가가 가능합니다.



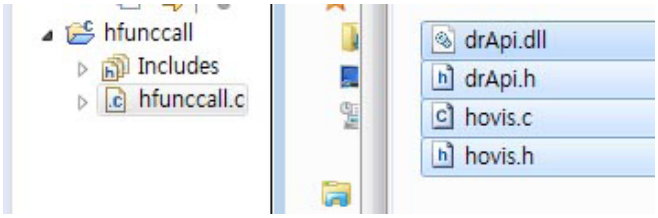
Build Project 를 클릭합니다.



좌측 프로젝트 이름에 커서를 놓고 오른쪽 마우스키를 누릅니다.
제일 하단에 Propertis 를 클릭합니다. Alt+Enter 도 동일합니다.

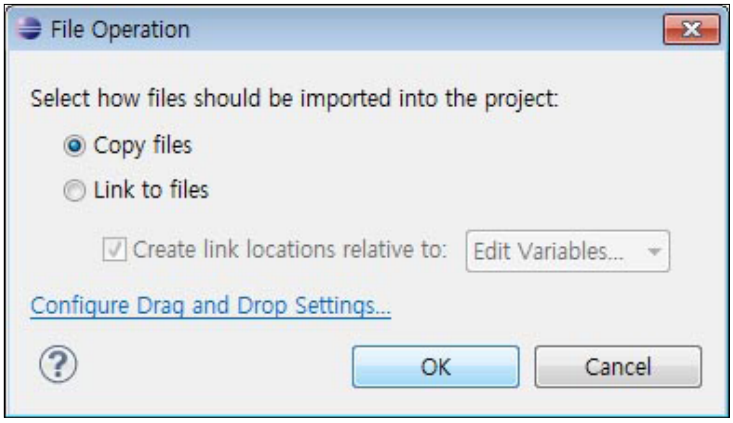


C/C++ Build 〉 Settings 를 클릭하고, 우측에 MinGW C Linker 〉 Libraries 를 클릭합니다. 우측에 Libraies 에 + 마크를 클릭하고 팝업창이 뜨면 drApi 라고 입력합니다. Library search path 에 + 마크를 클릭하고 팝업창이 뜨면 Workspace 를 클릭하고, Folder selection 에서 프로젝트 이름과 동일한 것을 클릭하고 OK 버튼을 누릅니다.



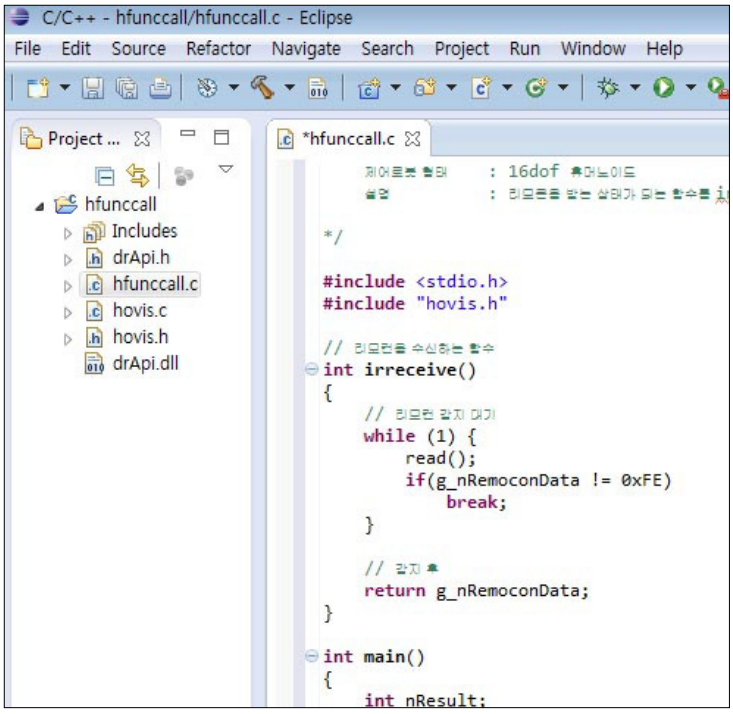
로봇을 동작시키기 위해선 위 네 개 파일을 프로젝트에 포함시켜야 합니다.
위 파일이 있는 파일탐색기 폴더에서 드래그 하는 방식으로 프로젝트에 포함시킵니다.
파일을 끌어다가 드래그해서 왼쪽 프로젝트에 추가합니다.

09 기본 파일 Import



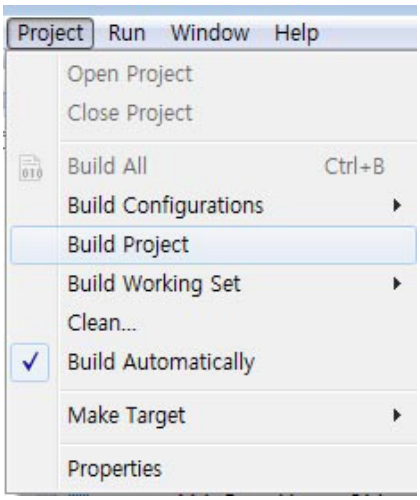
OK 버튼을 누릅니다.

10 소스 작성



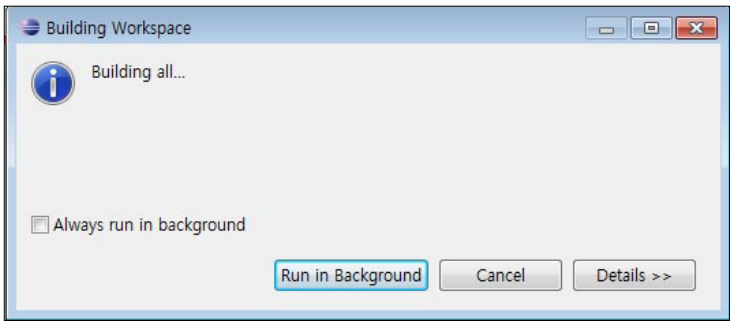
소스를 작성합니다.

11 빌드



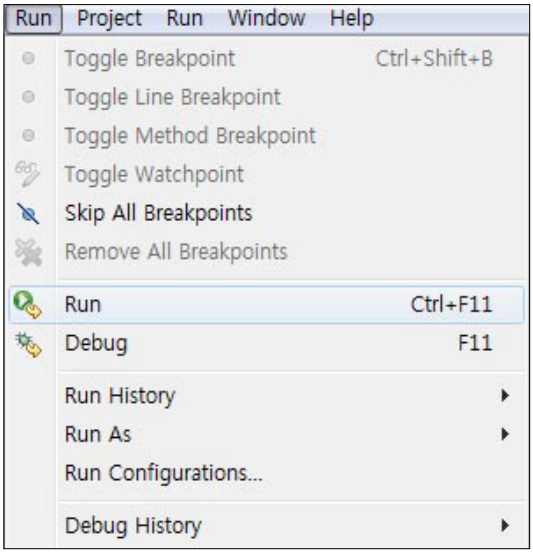
Projec 〉 Build Project 를 클릭합니다.

12 빌드중



빌드 팝업입니다. 그대로 두면 완료 후 사라집니다.

13 Run 실행



Run 〉Run 을 클릭합니다. Ctrl+F11 단축키도 동일합니다.

14 실행화면

```
                run(0);
                dr_wait_delay(1000);

                // Off
                for(j = 0; j < 16; j++){
                    g_ucMotorRedLed[j] = 0;
                }
                run(0);
                dr_wait_delay(1000);

            }
        }
        else{
            delay(200);
        }
    }

    // 프로그램 종료
    terminate();
    return 0;
}
```

15 로봇동작

리모콘 파워버튼을 누르면 앉았다 일어납니다. 그리고 모든 모터의 LED가 세 번 깜빡입니다.

4.3 변수범위

변수의 범위(scope)는 변수의 특성에 따라 분류될 수 있습니다.
지역변수 (local variable) : 중괄호 내에 선언되는 변수
전역변수 (Global variable) : 함수 내에 선언되지 않는 변수
정적변수 (Static variable) : 함수 내부, 외부 모두 선언 가능
레지스터 변수(Register variable) : 선언에 제한이 많이 따름

아래 예제는 차렷자세를 취한 후 소리가 들리는 쪽의 손을 들어올린 후 리모컨의 OK 버튼(15)를 입력받으면 다시 차렷자세를 취하는 프로그램입니다.

hfuncscope.c

```
#include <stdio.h>
#include "hovis.h"
int nSoundCount = 0;
int nSoundDirection = 0;

void soundreceive()
{
        while (1) {
                read();
                if(g_nSoundCount)
                        break;
        }
        nSoundCount = g_nSoundCount;
        nSoundDirection = g_nSoundDirection;
}

int irreceive()
{
        while (1) {
                read();
                if(g_nRemoconData != 0xFE)
                        break;
```

```
        }

        return g_nRemoconData;

}

int main()
{
        int nResult;

        int i;

        int nData;

        nResult = initialize(L"COM2", 115200);
        if(nResult == 0){
                printf("Initialization Failed₩n");
                fflush(stdout);
                return 0;
        }
        for(i=0;i<16;i++){
                g_fMotorPos[i]=0;
}
        g_fMotorPos[0] = -90;
        g_fMotorPos[1] = 90;
        g_fMotorPos[3] = -90;
        g_fMotorPos[4] = 90;
        run(1000);
        delay(1000);

        while(1){
                printf("Input Sound₩n");
                fflush(stdout);
              while (1) {
                soundreceive();
                if(nSoundCount)
                break;
        }

                printf("Direction = %d₩n", nSoundDirection);
                fflush(stdout);
                if (nSoundDirection == -2) {
                        g_fMotorPos[0] = -90;
                        g_fMotorPos[3] = 90;
                        run(1000);
                        delay(3000);
                }
                else if (nSoundDirection == 2) {
                g_fMotorPos[0] = 90;
                g_fMotorPos[3] = -90;
                run(1000);
                delay(3000);
```

```
                }
                else {
                        delay(1000);
                        continue;
                }

                do{
                        printf("Input Remocon₩n");
                         fflush(stdout);

                        nData = irreceive();

                        printf("Value = %d₩n", nData);
                         fflush(stdout);
                        delay(200);

                }while(nData != 15);
                g_fMotorPos[0] = -90;
                g_fMotorPos[1] = 90;
                g_fMotorPos[3] = -90;
                g_fMotorPos[4] = 90;
                run(1000);
                delay(3000);
        }
        terminate();
        return 0;
}
```

위 코드 중에 진하게 처리된 변수 선언 문장을 살펴봅니다.

int nSoundCount = 0; int nSoundDirection = 0; 두개는 전역변수로 이 프로그램 모든 곳에서 사용되고, int main() 안의 int nResult; int i; int nData; 는 메인함수내에 사용되는 지역변수 입니다.

코딩계획

```
//전역변수 선언
//소리수신함수
//리모콘수신함수

// 결과값, for, irrecieve 저장 변수
//초기화
//모터사용
//무한반복
//소리감지
//오른쪽 왼쪽 소리 감지별 로봇 동작
```

프로그래밍 세부설계

```
/*
파일명              : hfuncscope.c
제어로봇 형태       : 16dof 휴머노이드
설명                : 차렷자세를 취한 후 소리가 들리는 쪽의 손을 들어올린 후 리모컨
의 OK 버튼(15)를 입력받으면 다시 차렷자세를 취함
*/

// 동부로봇의 api 를 사용하기 위해 반드시 이렇게 선언 해 두어야 한다.

// 전역 변수 선언

// 소리를 수신하는 함수
{
        // 소리 감지 대기
        // 탈출하기 전까지 무한 반복
                // 소리 센서를 읽기 위해 통신 요청.
                // 소리 감지 횟수가 0이 아닌 경우
                        // while문을 탈출한다.

        // 위에 선언한 전역변수 nSoundCount에 값을 대입한다.
        // 위에 선언한 전역변수 nSoundDirection에 값을 대입한다.

// 리모콘을 수신하는 함수
int irreceive()
{
        // 리모컨 감지 대기
        // 탈출하기 전까지 무한 반복
                // 리모컨 입력값을 읽기 위해 통신 요청.
        // 리모컨 입력값이 0xFE가 아닌 경우(아무 입력 없을 때가 0xFE)
```

```
                        // while문을 탈출한다.

        // 감지 후
        // 리모컨 입력값을 리턴한다.

int main()
{
        // 결과값을 리턴받을 변수
        // for 문을 사용하기 위해 선언한 변수

        // irreceive 함수로부터 데이터를 받기 위한 변수
        // 초기화 – 로봇제어를 하기위해 반드시 앞 부분에 선언한다.
        // 각 변수들을 초기화 한다.
        // 열고 난 이후 이상이 있는 지 확인한다.
                // 이상이 있다면 에러 출력

                // 프로그램 종료
        // 이제부터 모터를 사용 할 수 있다.
        // 똑바로 일어선다.          (차렷자세)

                // 모든 모터(16개 모터)를 0 위치로 설정함

        // 0 번 모터(오른쪽 어깨)
        // 1 번 모터(오른쪽 윗팔)
        // 3 번 모터(왼쪽 어깨)
        // 4 번 모터(왼쪽 윗팔)
        // 모터 동작(해당 자세를 1000ms 동안 동작)
        // 동작 대기(1000ms)

        // 무한 반복한다.
                // 소리를 입력하라는 메시지 출력
                // 소리 감지 대기
                // 탈출하기 전까지 무한 반복
                        // 소리 센서를 읽는 함수 호출.

                            // 함수가 끝날 때 전역변수 nSoundCount와 nSoundDirection 값이 변한다.
                        // 소리 감지 횟수가 0이 아닌 경우
                                // while문을 탈출한다.
                // 감지 후
                // 소리 감지 방향을 출력한다.

                // 소리 감지 방향에 따라서 다른 동작을 수행
                // –2인 경우 : 왼쪽에서 소리가 난 경우
                        // 동작 – 왼팔들기
                        // 0 번 모터(오른쪽 어깨). 팔을 차렷자세로 하는 값을 넣는다.
                        // 3 번 모터(왼쪽 어깨). 팔을 위로 들어올리는 값을 넣는다.
```

```
                    // 모터 동작(해당 자세를 1000ms 동안 동작)
                    // 동작 대기(3000ms)

                // 2인 경우 : 오른쪽에서 소리가 난 경우
                    // 동작 – 오른팔들기
                    // 0 번 모터(오른쪽 어깨). 팔을 위로 들어올리는 값을 넣는다.
                    // 3 번 모터(왼쪽 어깨). 팔을 차렷자세로 하는 값을 넣는다.
                    // 모터 동작(해당 자세를 1000ms 동안 동작)
                    // 동작 대기(3000ms)

                    // 1000ms 동안 대기(소리 값이 초기화 될 때까지)
                    // while문의 처음으로 돌아간다.

                    // 리모컨을 입력하라는 메시지 출력
                    // 리모컨 값을 받는 함수를 실행해 결과 값을 nData에 저장.
                    // 감지 후
                    // 리모컨 입력 값을 출력한다.
                    // 200ms 동안 대기(리모컨 값이 초기화 될 때까지)
                // nData가 15(OK 버튼에 해당하는 값)가 아닌 동안 반복(15면 탈출)

                // 동작 – 준비자세 – 차렷
                // 0 번 모터(오른쪽 어깨)
                // 1 번 모터(오른쪽 윗팔)
                // 3 번 모터(왼쪽 어깨)
                // 4 번 모터(왼쪽 윗팔)
                // 모터 동작(해당 자세를 1000ms 동안 동작)
                // 동작 대기(3000ms)

        // 프로그램 종료
        // 제어 종료 함수를 실행한다.
```

프로그래밍 작성

```
/*
파일 명             : hfuncscope.c
제어로봇 형태       : 16dof 휴머노이드
설명                : 차렷자세를 취한 후 소리가 들리는 쪽의 손을 들어올린 후 리모컨
의 OK 버튼(15)를 입력받으면 다시 차렷자세를 취함
*/

#include 〈stdio.h〉
#include "hovis.h"
```

```
        // 동부로봇의 api 를 사용하기 위해 반드시 이렇게 선언 해 두어야 한다.

// 전역 변수 선언
int nSoundCount = 0;
int nSoundDirection = 0;

// 소리를 수신하는 함수
void soundreceive()
{
        // 소리 감지 대기
        while (1) {                     // 탈출하기 전까지 무한 반복
                read();                 // 소리 센서를 읽기 위해 통신 요청.
                if(g_nSoundCount)               // 소리 감지 횟수가 0이 아닌 경우
                        break;  // while문을 탈출한다.
        }

        nSoundCount = g_nSoundCount;
// 위에 선언한 전역변수 nSoundCount에 값을 대입한다.
        nSoundDirection = g_nSoundDirection;
// 위에 선언한 전역변수 nSoundDirection에 값을 대입한다.
}

// 리모콘을 수신하는 함수
int irreceive()
{
        // 리모컨 감지 대기
        while (1) {                             // 탈출하기 전까지 무한 반복
                read();                 // 리모컨 입력값을 읽기 위해 통신 요청
                if(g_nRemoconData != 0xFE)
        // 리모컨 입력값이 0xFE가 아닌 경우(아무 입력 없을 때가 0xFE)
                        break;                  // while문을 탈출한다.
        }
        // 감지 후
        return g_nRemoconData;  // 리모컨 입력값을 리턴한다.
}

int main()
{
        int nResult;                            // 결과값을 리턴받을 변수
        int i;                          // for 문을 사용하기 위해 선언한 변수
        int nData;              // irreceive 함수로부터 데이터를 받기 위한 변수
```

```
        // 초기화 – 로봇제어를 하기위해 반드시 앞 부분에 선언한다.
        nResult = initialize(L"COM2", 115200);// 각 변수들을 초기화.
        if(nResult == 0){  // 열고 난 이후 이상이 있는 지 확인한다.
                printf("Initialization Failed₩n");// 이상 있다면 에러 출력
                fflush(stdout);
                return 0;                                   // 프로그램 종료
        }

        // 이제부터 모터를 사용 할 수 있다.
        // 똑바로 일어선다.          (차렷자세)
        for(i=0;i<16;i++){
                g_fMotorPos[i]=0;// 모든 모터(16개 모터)를 0 위치로 설정
        }
        g_fMotorPos[0] = –90;                               // 0 번 모터(오른쪽 어깨)
        g_fMotorPos[1] = 90;                                // 1 번 모터(오른쪽 윗팔)
        g_fMotorPos[3] = –90;                               // 3 번 모터(왼쪽 어깨)
        g_fMotorPos[4] = 90;                                // 4 번 모터(왼쪽 윗팔)
        run(1000);          // 모터 동작(해당 자세를 1000ms 동안 동작)
        delay(1000);                                        // 동작 대기(1000ms)

        while(1){                                   // 무한 반복한다.
                printf("Input Sound₩n");// 소리 입력하라는 메시지 출력
                fflush(stdout);
                // 소리 감지 대기
                while (1) {                         // 탈출하기 전까지 무한 반복
                        soundreceive();    // 소리 센서를 읽는 함수 호출.

          // 함수가 끝날 때 전역변수 nSoundCount와 nSoundDirection 값이 변한다.
                        if(nSoundCount)// 소리 감지 횟수가 0이 아닌 경우
                                break;    // while문을 탈출한다.
                }
                // 감지 후
                printf("Direction = %d₩n", nSoundDirection);  // 소리 감지 방향을 출력한다.
                fflush(stdout);

                // 소리 감지 방향에 따라서 다른 동작을 수행
                if (nSoundDirection == –2) {
// –2인 경우 : 왼쪽에서 소리가 난 경우
                        // 동작 – 왼팔들기
                        g_fMotorPos[0] = –90;
        // 0 번 모터(오른쪽 어깨). 팔을 차렷자세로 하는 값을 넣는다.
                        g_fMotorPos[3] = 90;
        // 3 번 모터(왼쪽 어깨). 팔을 위로 들어올리는 값을 넣는다.
```

```
                run(1000);
                // 모터 동작(해당 자세를 1000ms 동안 동작)
                delay(3000);                    // 동작 대기(3000ms)
        }
        else if (nSoundDirection == 2) {// 2인 경우 : 오른쪽에서 소리가 난 경우
                // 동작 – 오른팔들기
                g_fMotorPos[0] = 90;
// 0 번 모터(오른쪽 어깨). 팔을 위로 들어올리는 값을 넣는다.
                g_fMotorPos[3] = –90;
// 3 번 모터(왼쪽 어깨). 팔을 차렷자세로 하는 값을 넣는다.
                run(1000);
                // 모터 동작(해당 자세를 1000ms 동안 동작)
                delay(3000);                    // 동작 대기(3000ms)
        }
        else {
                delay(1000);
        // 1000ms 동안 대기(소리 값이 초기화 될 때까지)
                continue;               // while문의 처음으로 돌아간다.
        }

        do{
                printf("Input Remocon₩n");
                fflush(stdout);
// 리모컨을 입력하라는 메시지 출력
                nData = irreceive();
// 리모컨 값을 받는 함수를 실행해 결과 값을 nData에 저장.

                // 감지 후
                printf("Value = %d₩n", nData);
                fflush(stdout);
// 리모컨 입력 값을 출력한다.
                delay(200);
                // 200ms 동안 대기(리모컨 값이 초기화 될 때까지)
        }while(nData != 15);
// nData가 15(OK 버튼에 해당하는 값)가 아닌 동안 반복(15면 탈출)

        // 동작 – 준비자세 – 차렷
        g_fMotorPos[0] = –90;           // 0 번 모터(오른쪽 어깨)
        g_fMotorPos[1] = 90;            // 1 번 모터(오른쪽 윗팔)
        g_fMotorPos[3] = –90;           // 3 번 모터(왼쪽 어깨)
        g_fMotorPos[4] = 90;            // 4 번 모터(왼쪽 윗팔)
```

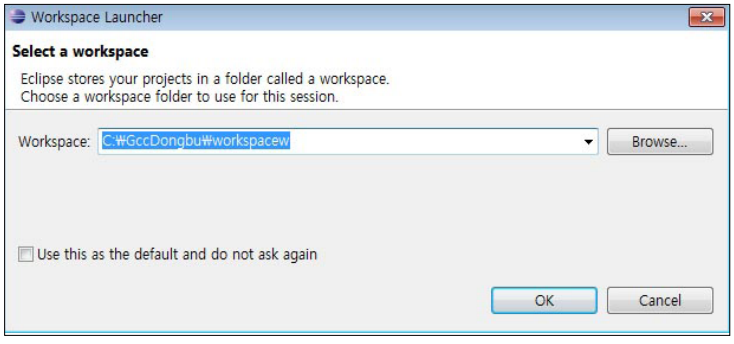
```
            run(1000);// 모터 동작(해당 자세를 1000ms 동안 동작)
            delay(3000);                    // 동작 대기(3000ms)
        }

        // 프로그램 종료
        terminate();                    // 제어 종료 함수를 실행한다.
        return 0;
}
```

빌드 및 실행

01 이클립스 실행

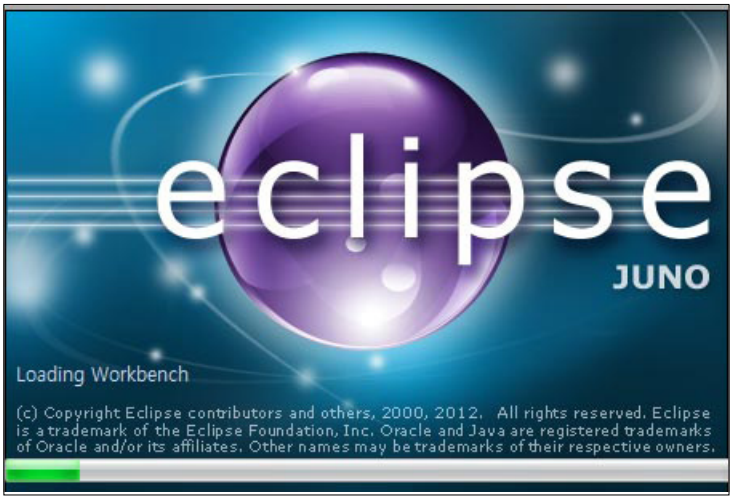
이클립스 실행 버튼을 누릅니다.

02 workspace



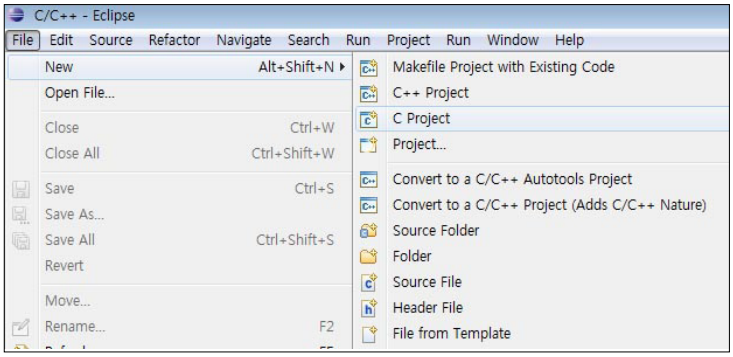
작업하고자 하는 폴더를 지정합니다. Browse 를 클릭하여 폴더를 지정하고 OK 버튼을 누릅니다.

03 이클립스 로딩 화면



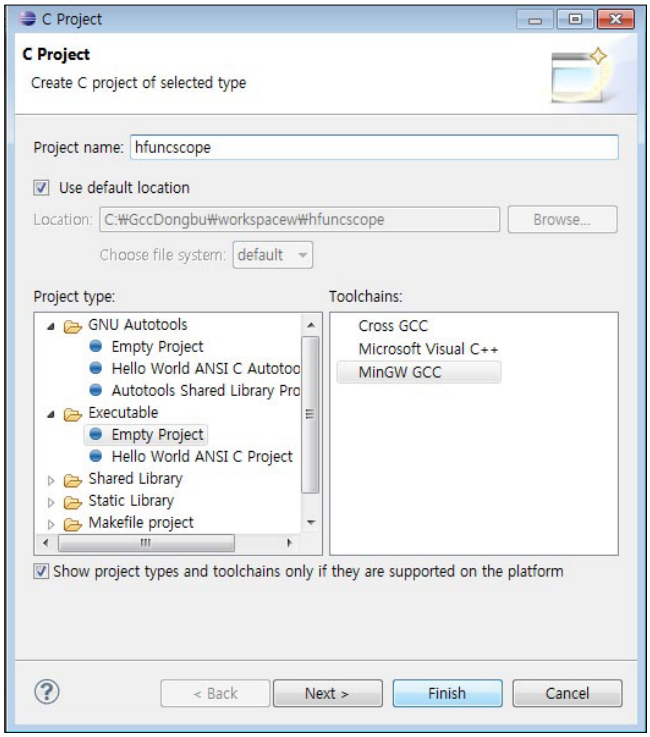
이클립스 로딩 화면입니다.

04 프로젝트 생성



이클립스 편집창 메뉴에서 File 〉 New 〉 C project 를 클릭합니다.

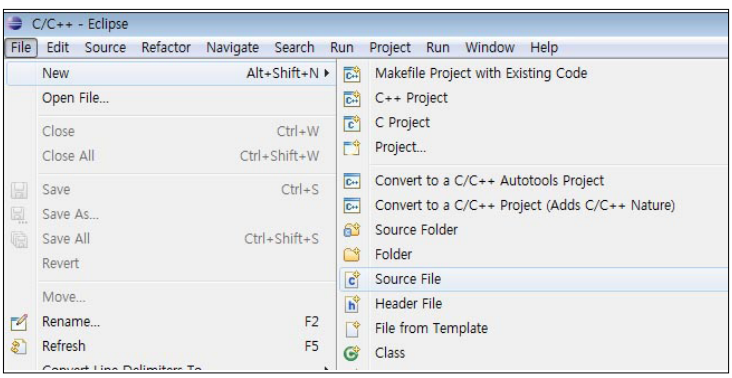
05 프로젝트 이름



Projec Name 을 hfuncscope 라고 입력하고, Project type 에서 Exectable 에서 Empty Project 를 선택합니다.

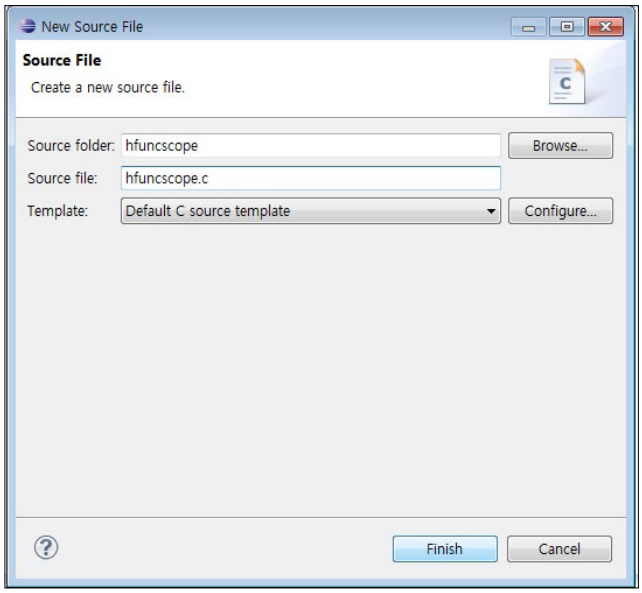
Toolchians 에서 MinGW GCC 를 선택하고 Finish 버튼을 누릅니다.

06 소스파일 추가



File〉New〉Source File 을 클릭합니다.

07 소스파일 이름

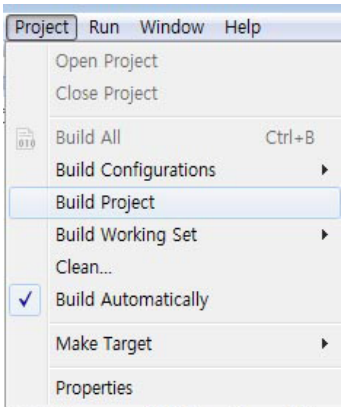


hfuncscope.c 라고 입력하고 Finish 버튼을 누릅니다.

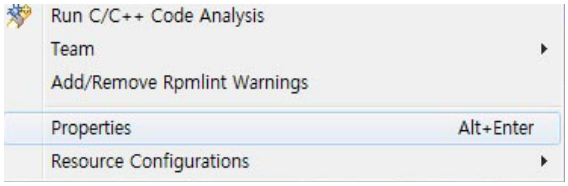
08 기본 제공 파일 추가

본 교재에게 C 프로그래밍을 하기 위해선 기본 제공되는 라이브러리 파일을 먼저 추가해주어야 올바른 컴파일을 할 수 있습니다. 필요한 파일은 drApi.dll, drApi.h, hovis.c, hovis.h 등 총 4가지 입니다.

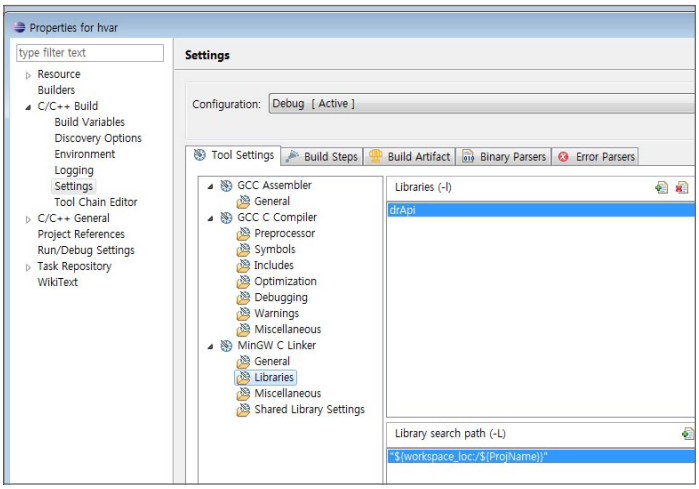
파일 추가 전에 먼저 Property 에서 라이브러리를 설정해주어야 합니다. 파일 생성을 한 후에 아무 코딩도 하지 말고, 바로 Build project 를 실행하세요. 그래야 라이브러리 추가가 가능합니다.



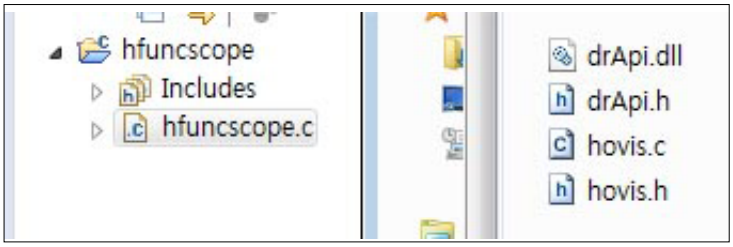
Build Project 를 클릭합니다.



좌측 프로젝트 이름에 커서를 놓고 오른쪽 마우스키를 누릅니다.
제일 하단에 Propertis 를 클릭합니다. Alt+Enter 도 동일합니다.

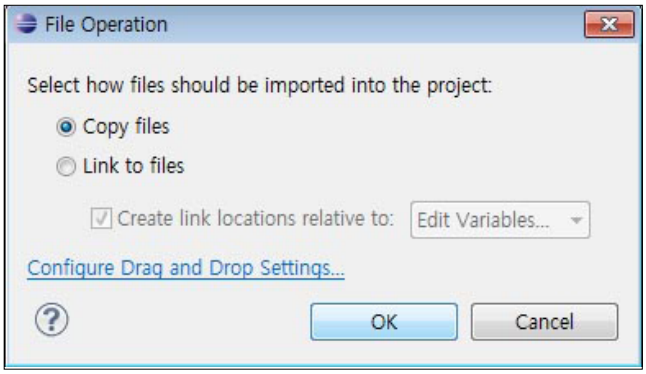


C/C++ Build 〉 Settings 를 클릭하고, 우측에 MinGW C Linker 〉 Libraties 를 클릭합니다. 우측에 Libraies 에 + 마크를 클릭하고 팝업창이 뜨면 drApi 라고 입력합니다. Library search path 에 + 마크를 클릭하고 팝업창이 뜨면 Workspace 를 클릭하고, Folder selection 에서 프로젝트 이름과 동일한 것을 클릭하고 OK 버튼을 누릅니다.



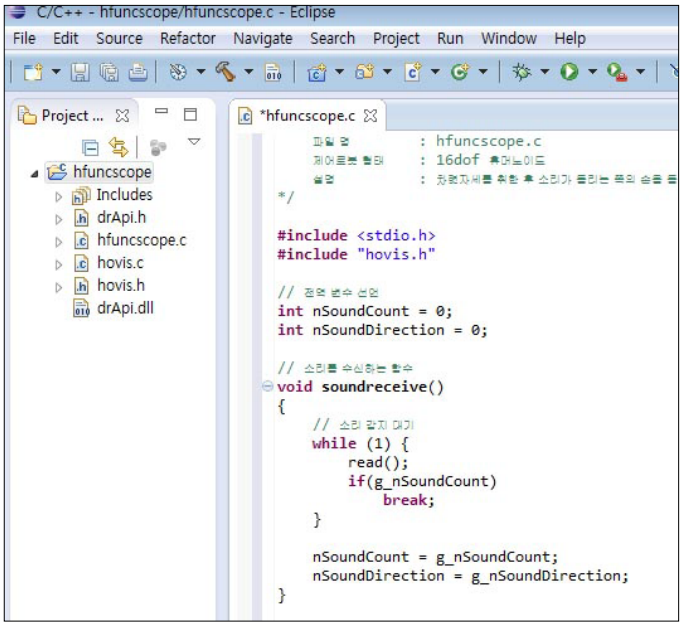
로봇을 동작시키기 위해선 위 네 개 파일을 프로젝트에 포함시켜야 합니다.
위 파일이 있는 파일탐색기 폴더에서 드래그 하는 방식으로 프로젝트에 포함시킵니다.
파일을 끌어다가 드래그해서 왼쪽 프로젝트에 추가합니다.

09 기본 파일 Import



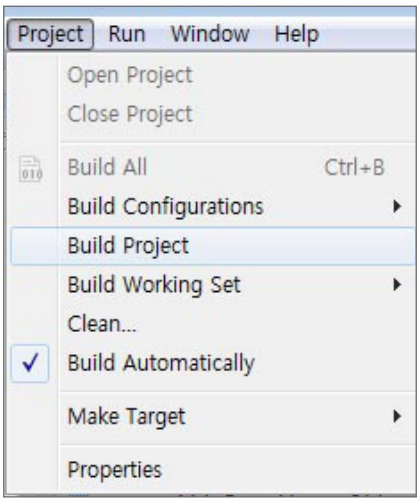
OK 버튼을 누릅니다.

10 소스 작성



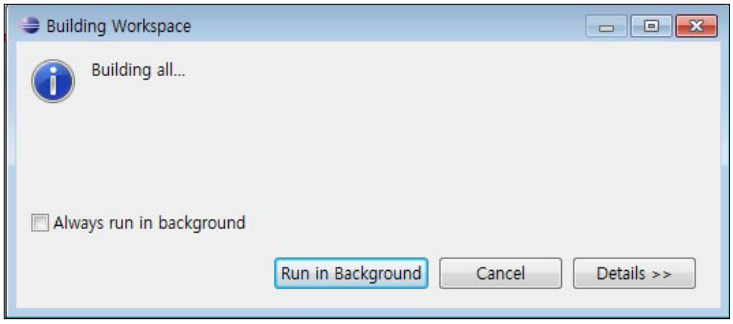
소스를 작성합니다.

11 빌드



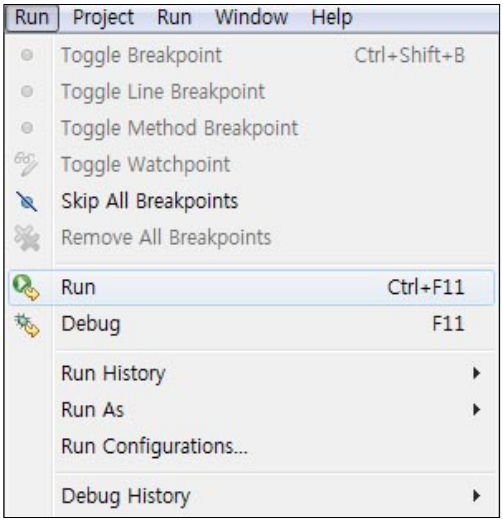
Projec 〉 Build Project 를 클릭합니다.

12 빌드중



빌드 팝업입니다. 그대로 두면 완료 후 사라집니다.

13 Run 실행



Run 〉 Run 을 클릭합니다. Ctrl+F11 단축키도 동일합니다.

14 실행화면

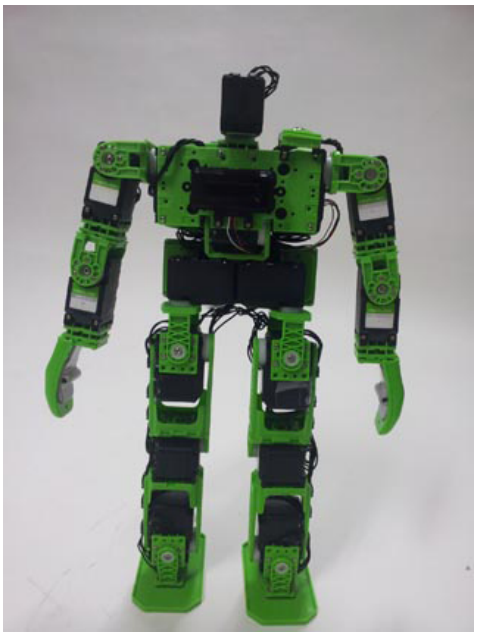
```
            // 값지 출
            printf("Value = %d\n", nData);
            fflush(stdout);
            delay(200);
        }while(nData != 15);

        // 동작 - 준비자세 - 차렷
        g_fMotorPos[0] = -90;
        g_fMotorPos[1] = 90;
        g_fMotorPos[3] = -90;
        g_fMotorPos[4] = 90;
        run(1000);
        delay(3000);
    }

    // 프로그램 종료
    terminate();
    return 0;
}
```

Problems Tasks Console Properties

hfuncscope.exe [C/C++ Application] C:₩GccDongbu₩workspace

```
Input Sound
Direction = -2
Input Remocon
```

15 로봇동작



차렷자세를 취한 후 소리가 들리는 쪽의 손을 들어올린 후 리모컨의 OK 버튼(15)를 입력받으면 다시 차렷자세를 취합니다.

05 1차원 배열

배열이란 둘 이상의 변수를 동시에 선언하는 효과를 지닙니다. 또한 많은 양의 데이터를 일괄적으로 처리해야 하는 경우에 유용합니다. 지역적 특성을 지닐 수도 있고, 전역적 특성을 지닐 수도 있습니다.
일반적인 변수는 데이터를 저장하기 위한 것인데, 데이터가 아닌 주소값을 저장하면 포인터라고 합니다.

5.1 1차원 배열

1차원 배열의 선언 및 초기화
비열 선언에 있어서 필요한 것 세가지는 아래와 같습니다.
– 배열 길이 : 배열을 구성하는 변수의 개수 (반드시 상수를 사용)
– 배열 요소 자료형 : 배열을 구성하는 변수의 자료형
– 배열 이름 : 배열에 접근할 때 사용되는 이름

예시)
Int arr [5]
int 형 데이터 5개를 저장할 수 메모리 공간을 할당하고, 이름을 arr 라고 붙여줍니다.

아래 예제는 char 배열 선언 후 문자열 입력 받아서 로봇이 동작시키는 프로그램입니다.

harr.c

```
#include <stdio.h>
#include “hovis.h”
#include “hovis.h”

int main()
{
        int nResult;

        int i, j;

        char str[100];

        nResult = initialize(L”COM2”, 115200);
        if(nResult == 0){
```

```
        printf("Initialization Failed₩n");
        fflush(stdout);
        return 0;

}

        for(i=0;i<16;i++){
                g_fMotorPos[i]=0;

        }
        g_fMotorPos[0] = -90;
        g_fMotorPos[1] = 90;
        g_fMotorPos[3] = -90;
        g_fMotorPos[4] = 90;
        run(1000);

        delay(1000);

        while(1){

                printf("Input Command₩n");
                fflush(stdout);
                scanf("%s", str);
                if(strcmp(str, "left")==0){
                g_fMotorPos[3] = 90;
                g_fMotorPos[0] = -90;
                run(1000);
                delay(1000);

        }
        else if(strcmp(str, "right")==0){
                        g_fMotorPos[0] = 90;
                        g_fMotorPos[3] = -90;
                        run(1000);
                        delay(1000);

                }
                else if(strcmp(str, "down")==0){

                        g_fMotorPos[0] = -90;
                        g_fMotorPos[3] = -90;
                        run(1000);
                        delay(1000);
                }
                else if(strcmp(str, "sit")==0){
```

```
                    g_fMotorPos[7] = 60;
                    g_fMotorPos[8] = 120;
                    g_fMotorPos[9] = 60;
                    g_fMotorPos[12] = 60;
                    g_fMotorPos[13] = 120;
                    g_fMotorPos[14] = 60;
                    run(1000);

                    delay(1000);
                }
                else if(strcmp(str, "stand")==0){

                    g_fMotorPos[7] = 0;

                    g_fMotorPos[8] = 0;

                    g_fMotorPos[9] = 0;

                    g_fMotorPos[12] = 0;
                    g_fMotorPos[13] = 0;
                    g_fMotorPos[14] = 0;
                    run(1000);

                    delay(1000);

                }
                else{

                    printf("unknown command.₩n");
                    fflush(stdout);
                }
        }
        terminate();

        return 0;
}
```

위 코드 중에 진하게 처리된 char 배열과 scanf 에서 문자열 입력 문장을 살펴봅니다.

```
    char str[100];        // 입력된 문자열이 저장될 배열
if(strcmp(str, "left")==0){   // 문자열이 left와 같을 경우
                          // 왼팔 들기 동작
```

문법요약

DR-Visual Logic 에는 C언어에서 다루는 배열과 포인터, 구조체는 다루지 않습니다. 그래픽 로직을 구성하기 위해서 복잡한 문법을 최소화 시키기 위함입니다.
따라서 이 파트에서는 Visual Logic 과 대응없이 배열,포인터, 구조체를 간략히 설명하겠습니다.

◆ 1차원 배열

※ 배열이란

- 둘 이상의 변수를 동시에 선언하는 효과를 지닙니다.
- 많은 양의 데이터를 일괄적으로 처리해야 하는 경우에 유용합니다.
- 지역적 특성을 지닐 수도 있고, 전역적 특성을 지닐 수도 있습니다.

※ 배열 선언

```
int  array  [10];
```

배열요소자료형 배열이름 배열길이

➔ Int 형 데이터 10개를 저장할 수 메모리 공간을 할당하고, 이름을 array 라고 붙여줍니다.

- 배열 길이 : 배열을 구성하는 변수의 개수 (반드시 상수를 사용)
- 배열 요소 자료형 : 배열을 구성하는 변수의 자료형
- 배열 이름 : 배열에 접근할 때 사용되는 이름

코딩계획

```
// 결과값, for, 문자열 저장 변수
//초기화
//모터사용
//무한반복
//left, right 문자 입력
//문자입력에 따라 로봇 동작
//로봇 앉기 일어나기 동작
```

프로그래밍 세부설계

```
/*
파일명              : harr.c
제어로봇 형태       : 16dof 휴머노이드
설명                : char 배열 선언 후 문자열 입력 받아서 로봇이 동작
*/

// 문자열 비교 등의 함수를 사용하려면 선언 해 두어야 한다.
// 동부로봇의 api 를 사용하기 위해 반드시 이렇게 선언 해 두어야 한다.

int main()
{
        // 결과값을 리턴받을 변수
        // for 문을 사용하기 위해 선언한 변수
        // 입력된 문자열이 저장될 배열
// 초기화 – 로봇제어를 하기 위해 반드시 앞 부분에 선언한다.
        // 각 변수들을 초기화 한다.
        // 열고 난 이후 이상이 있는 지 확인한다.
        // 이상이 있다면 에러 출력
        // 프로그램 종료

// 이제부터 모터를 사용 할 수 있다.
// 똑바로 일어선다.          (차렷자세)
        // 모든 모터(16개 모터)를 0 위치로 설정함
        // 0 번 모터(오른쪽 어깨)
        // 1 번 모터(오른쪽 윗팔)
        // 3 번 모터(왼쪽 어깨)
        // 4 번 모터(왼쪽 윗팔)
        // 모터 동작(해당 자세를 1000ms 동안 동작)
```

```
// 동작 대기(1000ms)

// 무한 반복한다.
        // 명령을 입력하라는 메시지 출력
        // 명령 문자열 입력 받기
// 문자열이 left와 같을 경우
        // 왼팔 들기 동작
// 3 번 모터(왼쪽 어깨). 팔을 위로 들어올리는 값을 넣는다.
// 0 번 모터(오른쪽 어깨). 팔을 차렷자세로 하는 값을 넣는다.
        // 모터 동작(해당 자세를 1000ms 동안 동작)
        // 동작 대기(!000ms)

// 문자열이 right와 같을 경우
        // 오른팔 들기 동작
// 0 번 모터(오른쪽 어깨). 팔을 위로 들어올리는 값을 넣는다.
// 3 번 모터(왼쪽 어깨). 팔을 차렷자세로 하는 값을 넣는다.
        // 모터 동작(해당 자세를 1000ms 동안 동작)
        // 동작 대기(!000ms)

        // 문자열이 down과 같을 경우
                // 차렷 자세 동작
// 0 번 모터(오른쪽 어깨). 팔을 차렷자세로 하는 값을 넣는다.
// 3 번 모터(왼쪽 어깨). 팔을 차렷자세로 하는 값을 넣는다.
                // 모터 동작(해당 자세를 1000ms 동안 동작)
                // 동작 대기(!000ms)

        // 문자열이 sit와 같을 경우
                // 앉기 동작
                // 7 번 모터(오른쪽 다리 앞뒤 방향)
                // 8 번 모터(오른쪽 다리 무릎)
                // 9 번 모터(오른쪽 발 앞뒤 방향)
                // 12 번 모터(왼쪽 다리 앞뒤 방향)
                // 13 번 모터(왼쪽 다리 무릎)
                // 13 번 모터(왼쪽 발 앞뒤 방향)
                // 모터 동작(해당 자세를 1000ms 동안 동작)
                // 동작 대기(1500ms)

        // 문자열이 stand와 같을 경우
                // 일어서기 동작
                // 7 번 모터(오른쪽 다리 앞뒤 방향)
                // 8 번 모터(오른쪽 다리 무릎)
                // 9 번 모터(오른쪽 발 앞뒤 방향)
```

```
                                // 12 번 모터(왼쪽 다리 앞뒤 방향)
                                // 13 번 모터(왼쪽 다리 무릎)
                                // 13 번 모터(왼쪽 발 앞뒤 방향)
                                // 모터 동작(해당 자세를 1000ms 동안 동작)
                                // 동작 대기(1000ms)

                        // 문자열이 맞는 명령이 없을 경우

                                // 에러 메시지 출력
        // 프로그램 종료
        // 제어 종료 함수를 실행한다.
```

프로그래밍 작성

```
/*
파일명              : harr.c
제어로봇 형태       : 16dof 휴머노이드
설명 : 리모콘을 받을 수 있는 상태가 되는 함수 정의 – 리모콘 파워버튼을 누르면 앉았다
일어난다. 그리고 모든 모터의 LED가 세 번 깜빡임
*/

#include <stdio.h>
#include <string.h>          // 문자열 비교 등의 함수를 사용하려면 선언 해 두어야 한다.
#include "hovis.h"  // 동부로봇의 api 를 사용하기 위해 반드시 이렇게 선언 해 두어야 한다.

int main()
{
        int nResult;                    // 결과값을 리턴받을 변수
        int i, j;               // for 문을 사용하기 위해 선언한 변수
        char str[100];                  // 입력된 문자열이 저장될 배열

        // 초기화 – 로봇제어를 하기위해 반드시 앞 부분에 선언한다.
        nResult = initialize(L"COM2", 115200);// 각 변수들을 초기화함
        if(nResult == 0){   // 열고 난 이후 이상이 있는 지 확인한다.
                printf("Initialization Failed₩n");          // 이상이 있다면 에러 출력
                fflush(stdout);
                return 0;                       // 프로그램 종료
        }

        // 이제부터 모터를 사용 할 수 있다.
        // 똑바로 일어선다.          (차렷자세)
        for(i=0;i<16;i++){
                g_fMotorPos[i]=0;
```

```
        // 모든 모터(16개 모터)를 0 위치로 설정함
}
g_fMotorPos[0] = –90;           // 0 번 모터(오른쪽 어깨)
g_fMotorPos[1] = 90;            // 1 번 모터(오른쪽 윗팔)
g_fMotorPos[3] = –90;           // 3 번 모터(왼쪽 어깨)
g_fMotorPos[4] = 90;            // 4 번 모터(왼쪽 윗팔)
run(1000);      // 모터 동작(해당 자세를 1000ms 동안 동작)
delay(1000);                    // 동작 대기(1000ms)

while(1){                       // 무한 반복한다.
        printf("Input Command₩n");
        fflush(stdout);
// 명령을 입력하라는 메시지 출력
        scanf("%s", str);       // 명령 문자열 입력 받기

        if(strcmp(str, "left")==0){// 문자열이 left와 같을 경우
                // 왼팔 들기 동작
                g_fMotorPos[3] = 90;
// 3 번 모터(왼쪽 어깨). 팔을 위로 들어올리는 값을 넣는다.
                g_fMotorPos[0] = –90;
// 0 번 모터(오른쪽 어깨). 팔을 차렷자세로 하는 값을 넣는다.
                run(1000);
                // 모터 동작(해당 자세를 1000ms 동안 동작)
                delay(1000);            // 동작 대기(!000ms)
        }
        else if(strcmp(str, "right")==0){       // 문자열이 right와 같을 경우
                // 오른팔 들기 동작
                g_fMotorPos[0] = 90;
// 0 번 모터(오른쪽 어깨). 팔을 위로 들어올리는 값을 넣는다.
                g_fMotorPos[3] = –90;
// 3 번 모터(왼쪽 어깨). 팔을 차렷자세로 하는 값을 넣는다.
                run(1000);
                // 모터 동작(해당 자세를 1000ms 동안 동작)
                delay(1000);            // 동작 대기(!000ms)
        }
        else if(strcmp(str, "down")==0){        // 문자열이 down과 같을 경우
                // 차렷 자세 동작
                g_fMotorPos[0] = –90;
```

```
        // 0 번 모터(오른쪽 어깨). 팔을 차렷자세로 하는 값을 넣는다.
                g_fMotorPos[3] = –90;
        // 3 번 모터(왼쪽 어깨). 팔을 차렷자세로 하는 값을 넣는다.
                run(1000);
                // 모터 동작(해당 자세를 1000ms 동안 동작)
                delay(1000);                    // 동작 대기(!000ms)
            }
            else if(strcmp(str, “sit”)==0){
// 문자열이 sit와 같을 경우
                // 앉기 동작
                g_fMotorPos[7] = 60;
        // 7 번 모터(오른쪽 다리 앞뒤 방향)
                g_fMotorPos[8] = 120;
        // 8 번 모터(오른쪽 다리 무릎)
                g_fMotorPos[9] = 60;
        // 9 번 모터(오른쪽 발 앞뒤 방향)
                g_fMotorPos[12] = 60;
        // 12 번 모터(왼쪽 다리 앞뒤 방향)
                g_fMotorPos[13] = 120;
        // 13 번 모터(왼쪽 다리 무릎)
                g_fMotorPos[14] = 60;
        // 13 번 모터(왼쪽 발 앞뒤 방향)
                run(1000);
                // 모터 동작(해당 자세를 1000ms 동안 동작)
                delay(1000);                    // 동작 대기(1500ms)
            }
            else if(strcmp(str, “stand”)==0){       // 문자열이 stand와 같을 경우
                // 일어서기 동작
                g_fMotorPos[7] = 0;         // 7 번 모터(오른쪽 다리 앞뒤 방향)
                g_fMotorPos[8] = 0;         // 8 번 모터(오른쪽 다리 무릎)
                g_fMotorPos[9] = 0;         // 9 번 모터(오른쪽 발 앞뒤 방향)
                g_fMotorPos[12] = 0;        // 12 번 모터(왼쪽 다리 앞뒤 방향)
                g_fMotorPos[13] = 0;        // 13 번 모터(왼쪽 다리 무릎)
```

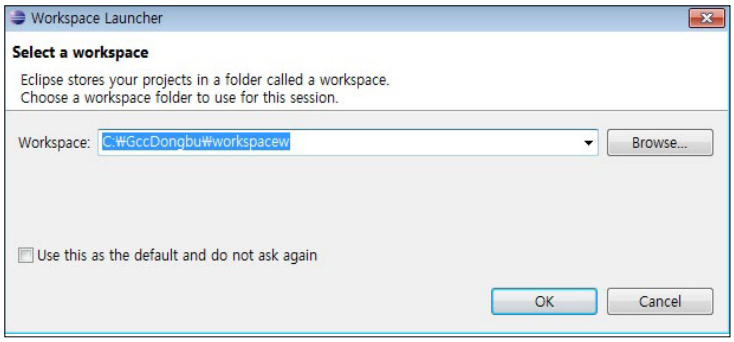
```
                    g_fMotorPos[14] = 0;  // 14번 모터(왼쪽 발 앞뒤 방향)
                    run(1000);
                    // 모터 동작(해당 자세를 1000ms 동안 동작)
                    delay(1000);
                   // 동작 대기(1000ms)
                }
                else{
                // 문자열이 맞는 명령이 없을 경우
                    printf("unknown command.₩n");  // 에러 메시지 출력
                    fflush(stdout);
                }
        }

        // 프로그램 종료
        terminate();
       // 제어 종료 함수를 실행한다.
        return 0;
}
```

빌드 및 실행

01 이클립스 실행

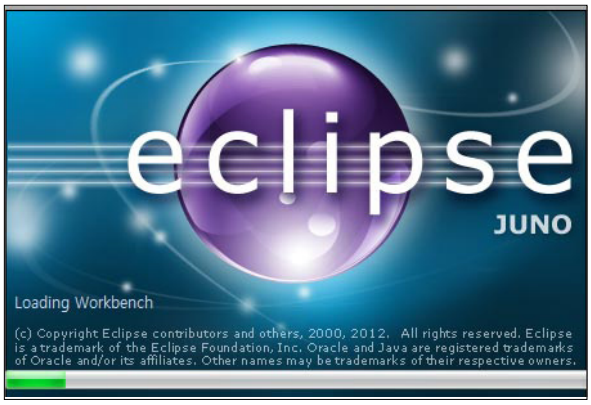
이클립스 실행 버튼을 누릅니다.

02 workspace



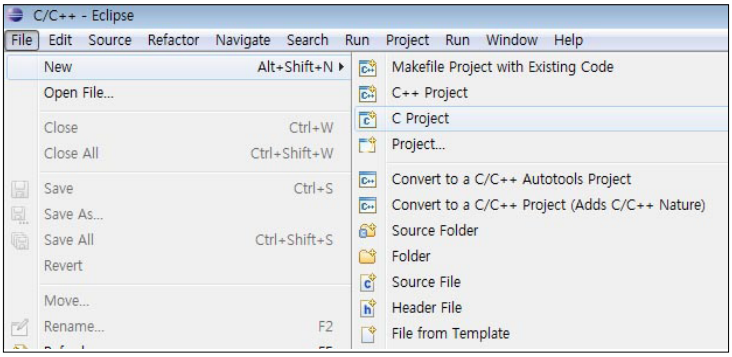
작업하고자 하는 폴더를 지정합니다. Browse 를 클릭하여 폴더를 지정하고 OK 버튼을 누릅니다.

03 이클립스 로딩 화면



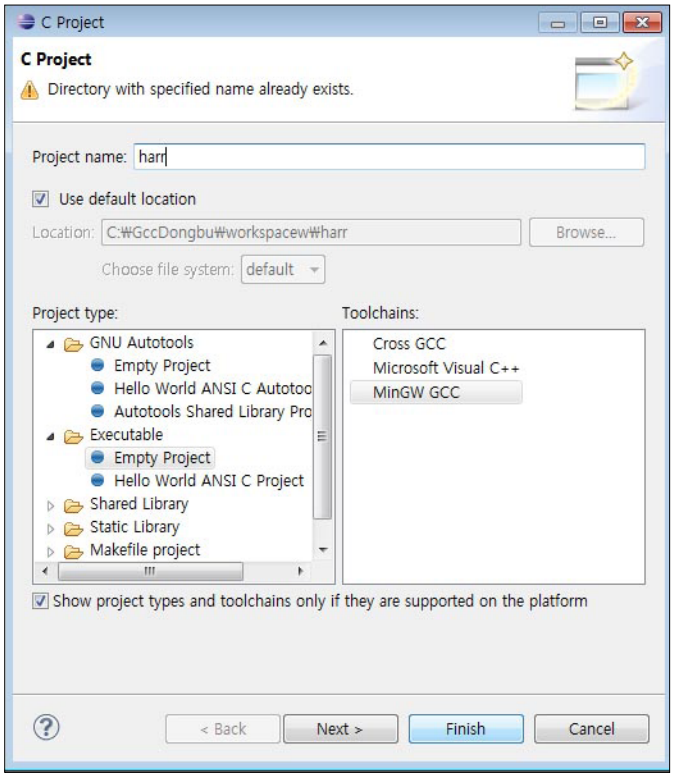
이클립스 로딩 화면입니다.

04 프로젝트 생성



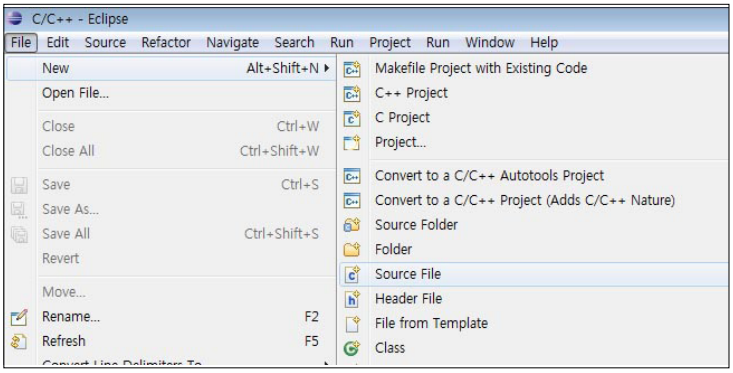
이클립스 편집창 메뉴에서 File 〉 New 〉 C project 를 클릭합니다.

05 프로젝트 이름



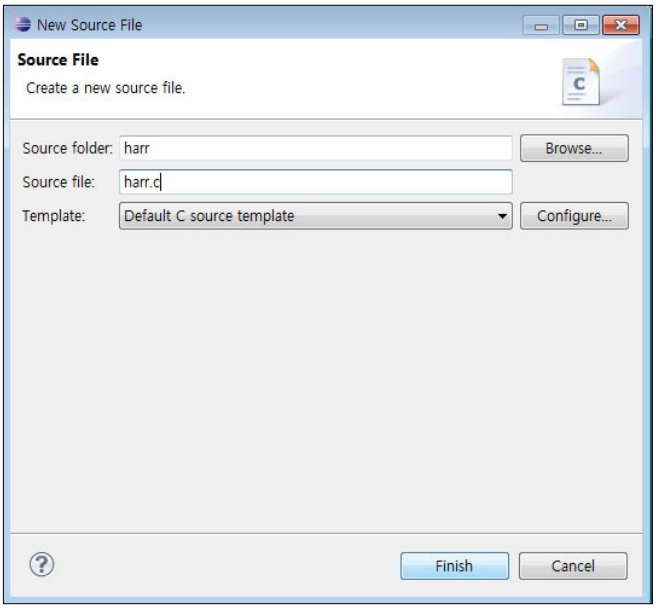
Projec Name 을 harr 라고 입력하고, Project type 에서 Exectable 에서 Empty Project 를 선택합니다. Toolchians 에서 MinGW GCC 를 선택하고 Finish 버튼을 누릅니다.

06 소스파일 추가



File 〉 New 〉 Source File 을 클릭합니다.

07 소스파일 이름

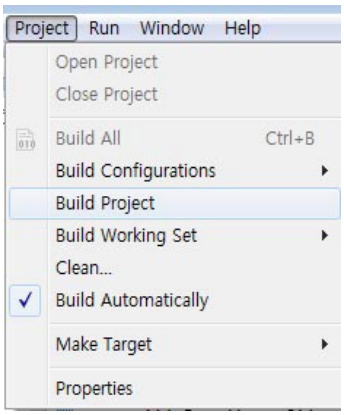


harr.c 라고 입력하고 Finish 버튼을 누릅니다.

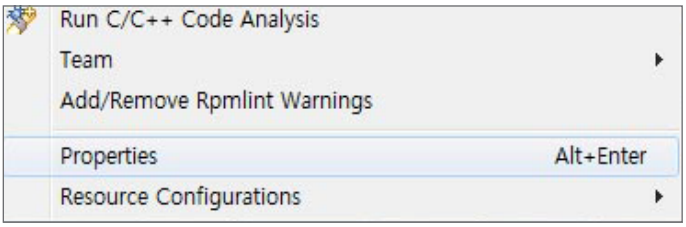
08 기본 제공 파일 추가

본 교재에게 C 프로그래밍을 하기 위해선 기본 제공되는 라이브러리 파일을 먼저 추가해주어야 올바른 컴파일을 할 수 있습니다. 필요한 파일은 drApi.dll, drApi.h, hovis.c, hovis.h 등 총 4가지 입니다.

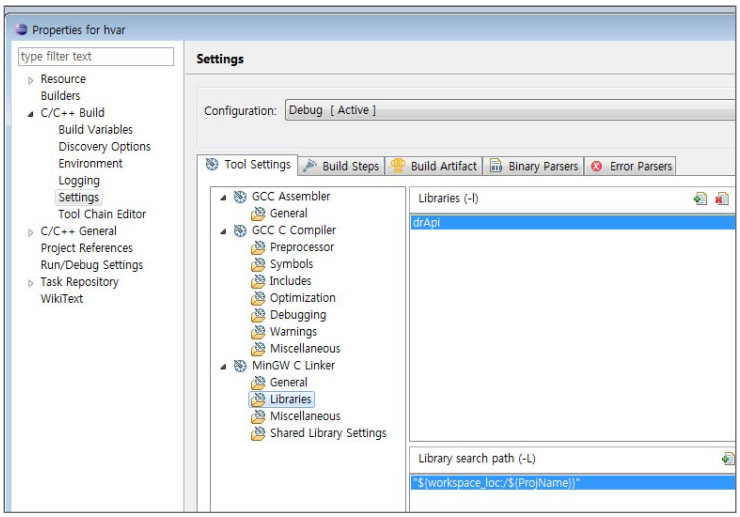
파일 추가 전에 먼저 Property 에서 라이브러리를 설정해주어야 합니다. 파일 생성을 한 후에 아무 코딩도 하지 말고, 바로 Build project 를 실행하세요. 그래야 라이브러리 추가가 가능합니다.



Build Project 를 클릭합니다.



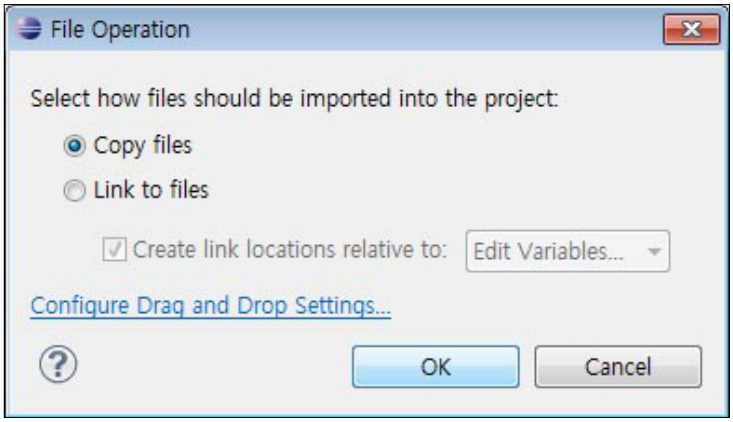
좌측 프로젝트 이름에 커서를 놓고 오른쪽 마우스키를 누릅니다.
제일 하단에 Propertis 를 클릭합니다. Alt+Enter 도 동일합니다.



C/C++ Build 〉Settings 를 클릭하고, 우측에 MinGW C Linker 〉Libraties 를 클릭합니다. 우측에 Libraies 에 + 마크를 클릭하고 팝업창이 뜨면 drApi 라고 입력합니다. Library search path 에 + 마크를 클릭하고 팝업창이 뜨면 Workspace 를 클릭하고, Folder selection 에서 프로젝트 이름과 동일한 것을 클릭하고 OK 버튼을 누릅니다.

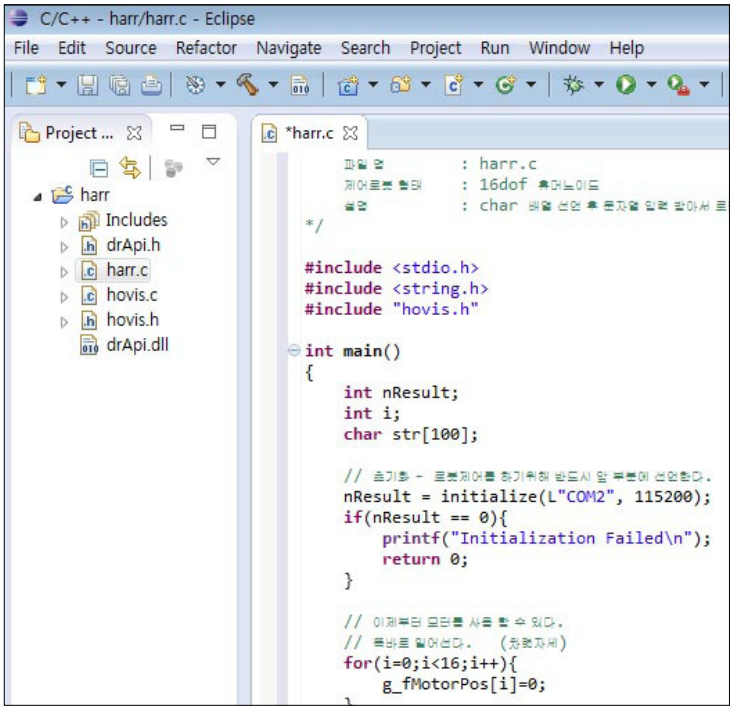
로봇을 동작시키기 위해선 위 네 개 파일을 프로젝트에 포함시켜야 합니다.
위 파일이 있는 파일탐색기 폴더에서 드래그 하는 방식으로 프로젝트에 포함시킵니다.
파일을 끌어다가 드래그해서 왼쪽 프로젝트에 추가합니다.

09 기본 파일 Import



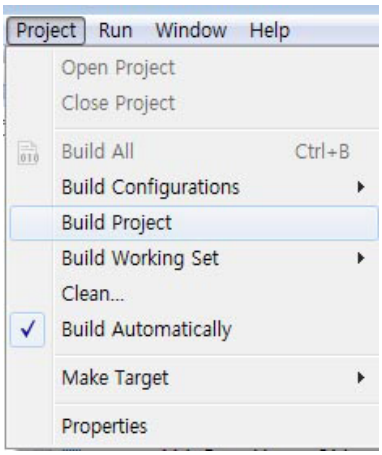
OK 버튼을 누릅니다.

10 소스 작성



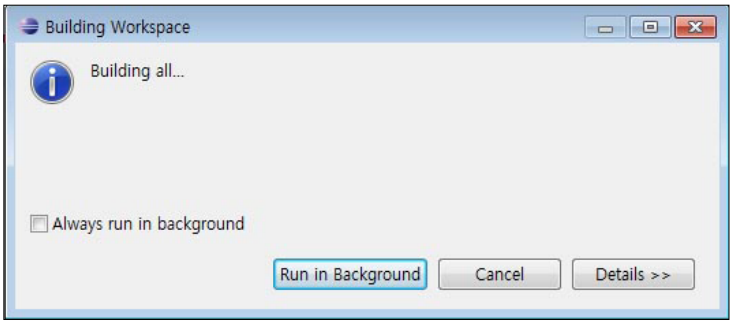
소스를 작성합니다.

11 빌드



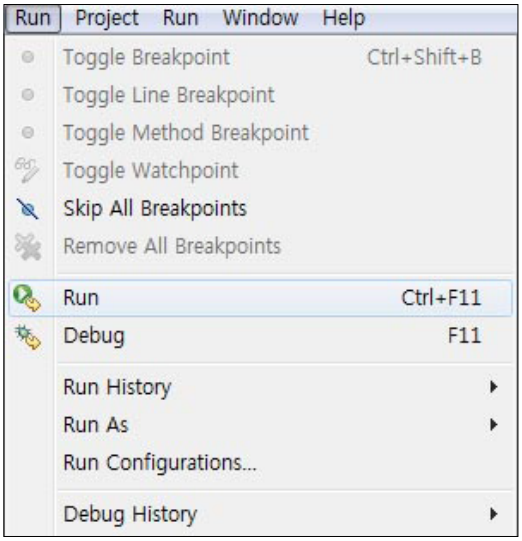
Projec 〉Build Project 를 클릭합니다.

12 빌드중



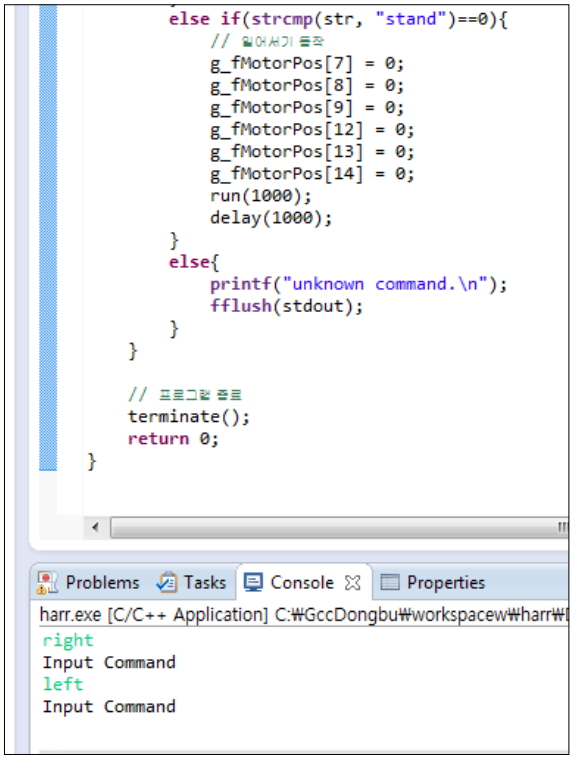
빌드 팝업입니다. 그대로 두면 완료 후 사라집니다.

13 Run 실행



Run 〉 Run 을 클릭합니다. Ctrl+F11 단축키도 동일합니다.

14 실행화면



15 로봇동작

PC 키보드에서 right 를 치면 오른손을 들고, left 를 치면 왼손을 듭니다.

5.2 다차원 배열

다차원 배열이란 무엇인가 2차원 이상의 배열을 의미합니다.
다차원 배열의 선언

- Int arr[10] 1차원배열
- Int arr[10][10] 10x10, 2차원배열
- Int arr[5][5][5] 5x5x5, 3차원배열

1차원은 선, 2차원은 면, 3차원은 직육면체를 의미합니다. 하지만, 다차원 배열의 실제 메모리 구성은 1차원 배열과 동일합니다. 다만 접근 방법을 2차원적으로 해석할 뿐입니다.

아래 예제는 float 2차원 배열 선언 후 자세 위치 값을 저장. 숫자 입력 받아서 자세를 동작시키는 프로그램입니다.

harrtwo.c

```
#include <stdio.h>
#include "hovis.h"

int main()
{

        int nResult;

        int i, j;

        float fPos[4][16];

        int nInputNum;

        nResult = initialize(L"COM2", 115200);
        if(nResult == 0){
                printf("Initialization Failed₩n");
                fflush(stdout);
                return 0;
        }

        for(i=0;i<16;i++){
                g_fMotorPos[i]=0;

        }
        g_fMotorPos[0] = -90;
        g_fMotorPos[1] = 90;
        g_fMotorPos[3] = -90;
        g_fMotorPos[4] = 90;
        run(1000);
```

```
        delay(1000);

        for(i = 0;i < 4;i++){
                for(j = 0;j < 16;j++){
                        fPos[i][j] = 0;

                }
        }
        for(i = 0;i < 4;i++){
        fPos[i][0] = -90;

        fPos[i][1] = 90;
        fPos[i][3] = -90;

        fPos[i][4] = 90;
}

        fPos[1][3] = 90;

        fPos[2][0] = 90;

        fPos[3][7] = 60;

        fPos[3][8] = 120;

        fPos[3][9] = 60;

        fPos[3][12] = 60;

        fPos[3][13] = 120;

        fPos[3][14] = 60;

        while(1){

                printf("Input Number₩n");
                fflush(stdout);
                scanf("%d", &nInputNum);
                if(nInputNum>=0 && nInputNum<=3){
                        for(i = 0;i < 16;i++){
                                g_fMotorPos[i] = fPos[nInputNum][i];
                        }
                        run(1000);
                        delay(1000);

                }
                else{

                        printf("unknown command.₩n");
                        fflush(stdout);
        }
        }
        terminate();

        return 0;
}
```

위 코드 중에 진하게 처리된 float fPos[4][16]; 문장을 살펴봅니다.

```
float fPos[4][16];  // 자세 위치값이 저장될 2차원 배열
        fPos[i][0] = –90;          // 0 번 모터(오른쪽 어깨)
        fPos[i][1] = 90;                  // 1 번 모터(오른쪽 윗팔)
```

문법요약

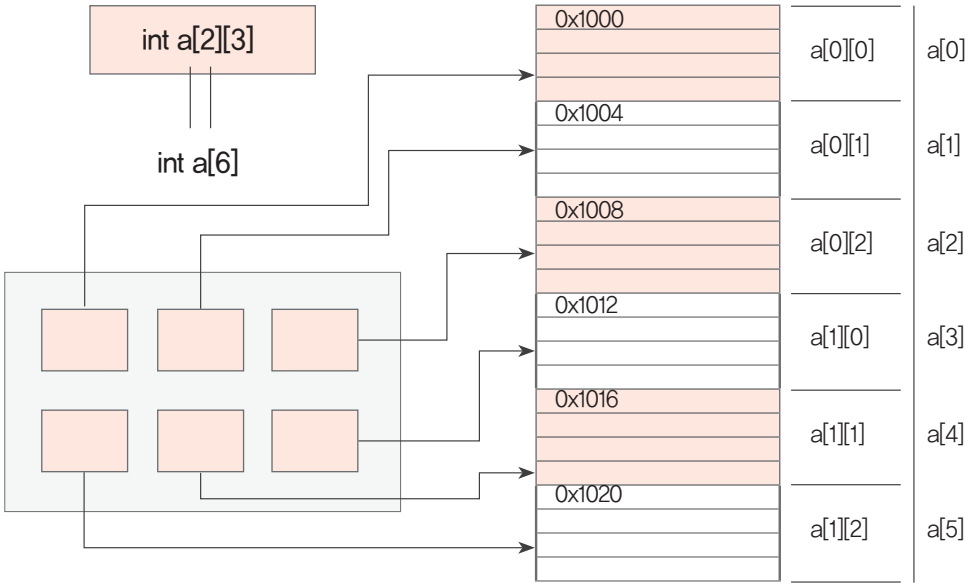
◆ 2차원 배열

※ 다차원 배열의 예

Int arr[100]	1차원배열
Int arr[10][10]	10x10, 2차원배열
Int arr[5][5][5]	5x5x5, 3차원배열

1차원은 선, 2차원은 면, 3차원은 직육면체입니다.
보통 다차원배열은 2차원 배열을 의미합니다.

◆ 다차원 배열의 실제 메모리 구성



※ 2차원 배열 ! 선언과 동시에 초기화

중괄호 안에 또 다른 중괄호 → 2차원 이상 초기화 할 때 사용합니다.
case 1 : 행 단위로 모든 요소들을 초기화
case 2 : 행 단위로 일부 요소들만 초기화

int somang[3][3]={{1,2,3},{4,5,6,},{7,8,9}};
첫번째 중괄호 → 첫번째 행 초기확, 두번째, 세번째
int somang[3][3]= {{1},{4,5},{7,8,9}};
행의 일부만을 초기화 → 첫번째 행 초기화 요소리스트는 1개뿐,
왼쪽부터 넣고, 나머지는 0으로 채워줍니다.

Case 3 : 내부 중괄호 없이 초기화
int somang[3][3]={1,2,3 ,4,5,6, 7};
처음부터 세개 채우고, 두번째 3개 채우고, 나머지 1개 채웁니다.

코딩계획

```
//결과값, 자세위치값,숫자 저장 변수
//초기화
//모터사용
//번호별 모터 동작
```

프로그래밍 세부설계

```
/*
파일명              : harrtwo.c
제어로봇 형태       : 16dof 휴머노이드
설명                : float 2차원 배열 선언 후 자세 위치 값을 저장. 숫자 입력 받아서 자세를 동작
*/

// 동부로봇의 api 를 사용하기 위해 반드시 이렇게 선언 해 두어야 한다.
// 리모컨을 수신하는 함수
int main()
{
        // 결과값을 리턴받을 변수
        // for 문을 사용하기 위해 선언한 변수
        // 자세 위치값이 저장될 2차원 배열
        // 입력된 숫자가 저장될 변수

        // 초기화 – 로봇제어를 하기 위해 반드시 앞 부분에 선언한다.
        // 각 변수들을 초기화 한다.
        // 열고 난 이후 이상이 있는 지 확인한다.
                // 이상이 있다면 에러 출력
                // 프로그램 종료

        // 이제부터 모터를 사용 할 수 있다.
```

```
// 똑바로 일어선다.          (차렷자세)

        // 모든 모터(16개 모터)를 0 위치로 설정함

// 0 번 모터(오른쪽 어깨)
// 1 번 모터(오른쪽 윗팔)
// 3 번 모터(왼쪽 어깨)
// 4 번 모터(왼쪽 윗팔)
// 모터 동작(해당 자세를 1000ms 동안 동작)
// 동작 대기(1000ms)

// 일단 모든 자세를 차렷 자세로 만든다.
// 4개의 자세에 대해서 반복
        // 16개의 모터에 대해서 반복
                // 0 값을 각 배열 원소에 넣는다.

// 4개의 자세에 대해서 반복
        // 0 번 모터(오른쪽 어깨)
        // 1 번 모터(오른쪽 윗팔)
        // 3 번 모터(왼쪽 어깨)
        // 4 번 모터(왼쪽 윗팔)

// 1번 자세를 왼팔 든 자세로 만든다.
// 3 번 모터(왼쪽 어깨)에 팔을 위로 들어올리는 값을 대입

// 2번 자세를 오른팔 든 자세로 만든다.
// 0 번 모터(오른쪽 어깨)에 팔을 위로 들어올리는 값을 대입

// 3번 자세를 앉은 자세로 만든다.
// 7 번 모터(오른쪽 다리 앞뒤 방향)
// 8 번 모터(오른쪽 다리 무릎)
// 9 번 모터(오른쪽 발 앞뒤 방향)
// 12 번 모터(왼쪽 다리 앞뒤 방향)
// 13 번 모터(왼쪽 다리 무릎)
// 13 번 모터(왼쪽 발 앞뒤 방향)

// 무한 반복한다.
        // 명령을 입력하라는 메시지 출력
        // 명령 숫자 입력 받기
        // 0~3 사이의 숫자가 입력된 경우
                // 16개의 모터에 대해서 반복
                // nInputNum 번 자세를 모터 값에 대입한다.
                }
                // 모터 동작(해당 자세를 1000ms 동안 동작)
                // 동작 대기(1000ms)
```

```
                    // 문자열이 맞는 명령이 없을 경우
                        // 에러 메시지 출력

            // 프로그램 종료
            // 제어 종료 함수를 실행한다.
```

프로그래밍 작성

```
/*
파일 명              : harrtwo.c
제어로봇 형태        : 16dof 휴머노이드
설명                 : float 2차원 배열 선언 후 자세 위치 값을 저장. 숫자 입력 받아서 자세를 동작
*/

#include <stdio.h>
#include "hovis.h" // 동부로봇의 api 를 사용하기 위해 반드시 이렇게 선언 해 두어야 한다.

int main()
{
        int nResult;                    // 결과값을 리턴받을 변수
        int i, j;                       // for 문을 사용하기 위해 선언한 변수
        float fPos[4][16];   // 자세 위치값이 저장될 2차원 배열
        int nInputNum;                  // 입력된 숫자가 저장될 변수

        // 초기화 – 로봇제어를 하기 위해 반드시 앞 부분에 선언한다.
        nResult = initialize(L"COM2", 115200);// 각 변수들을 초기화함
        if(nResult == 0){     // 열고 난 이후 이상이 있는 지 확인한다.
                printf("Initialization Failed₩n");          // 이상이 있다면 에러 출력
                fflush(stdout);
                return 0;               // 프로그램 종료
        }

        // 이제부터 모터를 사용 할 수 있다.
        // 똑바로 일어선다.             (차렷자세)
        for(i=0;i<16;i++){
                g_fMotorPos[i]=0;// 모든 모터(16개 모터)를 0 위치로 설정
        }
        g_fMotorPos[0] = –90;                    // 0 번 모터(오른쪽 어깨)
        g_fMotorPos[1] = 90;                    // 1 번 모터(오른쪽 윗팔)
        g_fMotorPos[3] = –90;                    // 3 번 모터(왼쪽 어깨)
```

```
        g_fMotorPos[4] = 90;                    // 4 번 모터(왼쪽 윗팔)
        run(1000);          // 모터 동작(해당 자세를 1000ms 동안 동작)
        delay(1000);                            // 동작 대기(1000ms)

        // 일단 모든 자세를 차렷 자세로 만든다.
        for(i = 0;i < 4;i++){           // 4개의 자세에 대해서 반복
                for(j = 0;j < 16;j++){          // 16개의 모터에 대해서 반복
                        fPos[i][j] = 0;         // 0 값을 각 배열 원소에 넣는다.
                }
        }
        for(i = 0;i < 4;i++){           // 4개의 자세에 대해서 반복
                fPos[i][0] = -90;   // 0 번 모터(오른쪽 어깨)
                fPos[i][1] = 90;                // 1 번 모터(오른쪽 윗팔)
                fPos[i][3] = -90;   // 3 번 모터(왼쪽 어깨)
                fPos[i][4] = 90;    // 4 번 모터(왼쪽 윗팔)
        }

        // 1번 자세를 왼팔 든 자세로 만든다.
        fPos[1][3] = 90;
        // 3 번 모터(왼쪽 어깨)에 팔을 위로 들어올리는 값을 대입

        // 2번 자세를 오른팔 든 자세로 만든다.
        fPos[2][0] = 90;
// 0 번 모터(오른쪽 어깨)에 팔을 위로 들어올리는 값을 대입
        // 3번 자세를 앉은 자세로 만든다.
        fPos[3][7] = 60;    // 7 번 모터(오른쪽 다리 앞뒤 방향)
        fPos[3][8] = 120;   // 8 번 모터(오른쪽 다리 무릎)
        fPos[3][9] = 60;    // 9 번 모터(오른쪽 발 앞뒤 방향)
        fPos[3][12] = 60;   // 12 번 모터(왼쪽 다리 앞뒤 방향)
        fPos[3][13] = 120;  // 13 번 모터(왼쪽 다리 무릎)
        fPos[3][14] = 60;   // 13 번 모터(왼쪽 발 앞뒤 방향)

        while(1){               // 무한 반복한다.
                printf("Input Number₩n");   // 명령을 입력하라는 메시지 출력
                fflush(stdout);
                scanf("%d", &nInputNum);  // 명령 숫자 입력 받기

                if(nInputNum>=0 && nInputNum<=3){ // 0~3 사이의 숫자가 입력된 경우
                        for(i = 0;i < 16;i++){// 16개의 모터에 대해서 반복
                                g_fMotorPos[i] = fPos[nInputNum][i];
// nInputNum 번 자세를 모터 값에 대입한다.
                        }
                        run(1000);
```

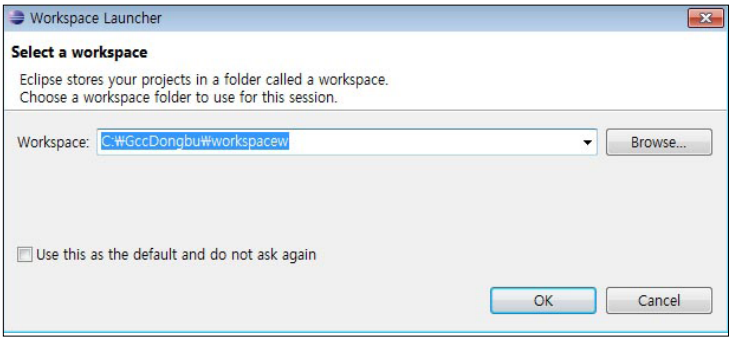
```
                    // 모터 동작(해당 자세를 1000ms 동안 동작)
                    delay(1000);                    // 동작 대기(1000ms)
                }
                else{                               // 문자열이 맞는 명령이 없을 경우
                    printf("unknown command.₩n"); // 에러 메시지 출력
                    fflush(stdout);
                }
        }

        // 프로그램 종료
        terminate();                                // 제어 종료 함수를 실행한다.
        return 0;
}
```

빌드 및 실행

01 이클립스 실행

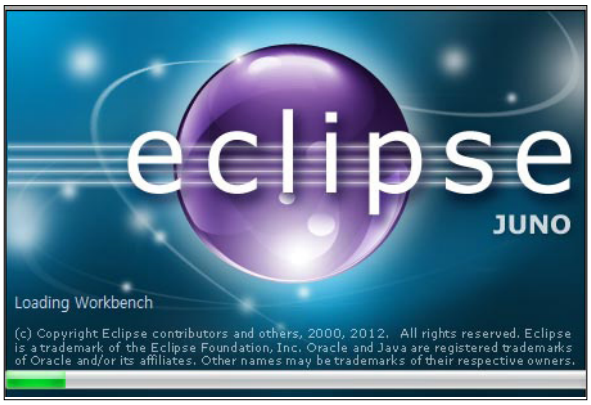
이클립스 실행 버튼을 누릅니다.

02 workspace



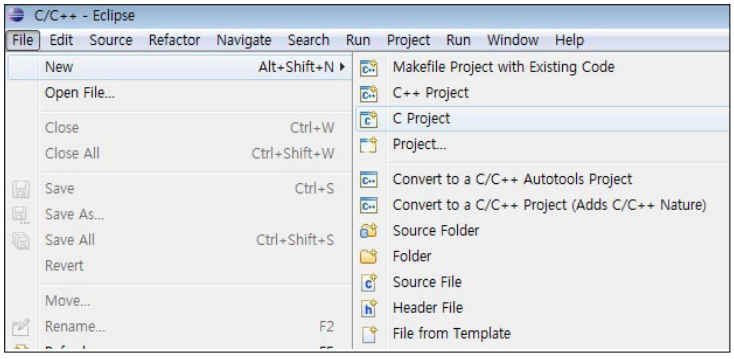
작업하고자 하는 폴더를 지정합니다. Browse 를 클릭하여 폴더를 지정하고 OK 버튼을 누릅니다.

03 이클립스 로딩 화면



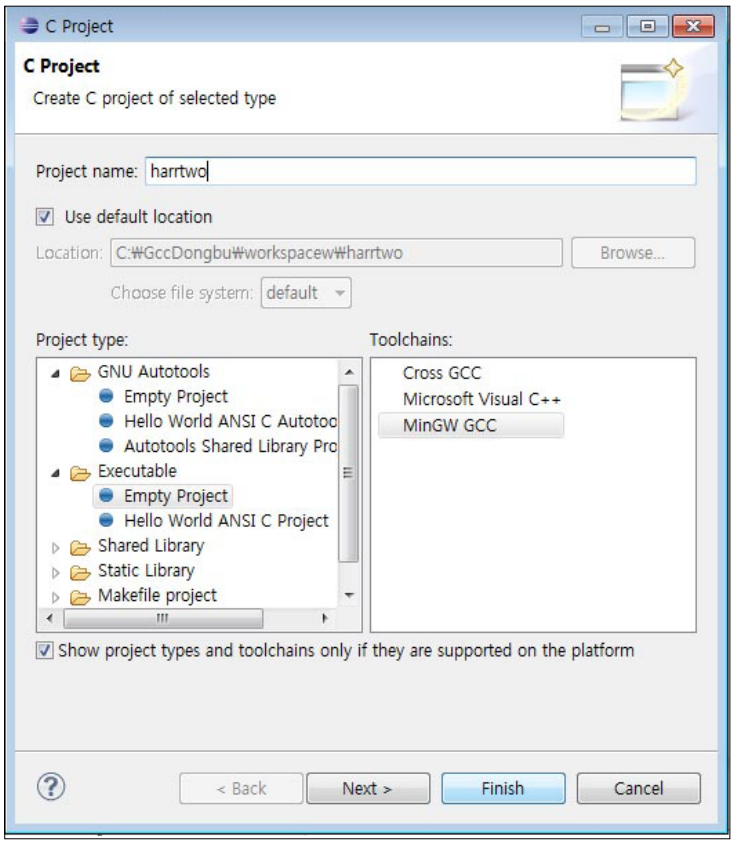
이클립스 로딩 화면입니다.

04 프로젝트 생성



이클립스 편집창 메뉴에서 File 〉 New 〉 C project 를 클릭합니다.

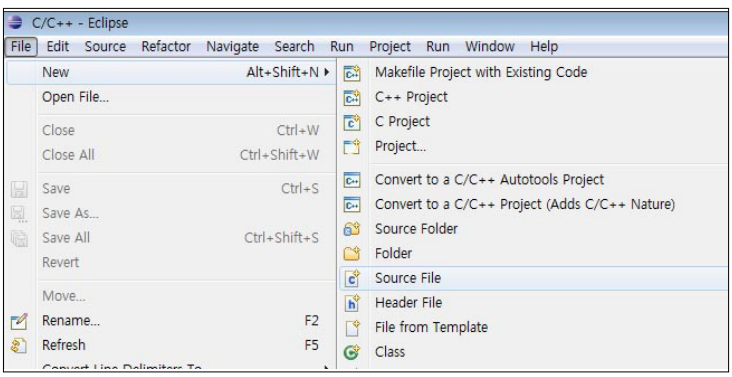
05 프로젝트 이름



Projec Name 을 harrtwo 라고 입력하고, Project type 에서 Exectable 에서 Empty Project 를 선택합니다.

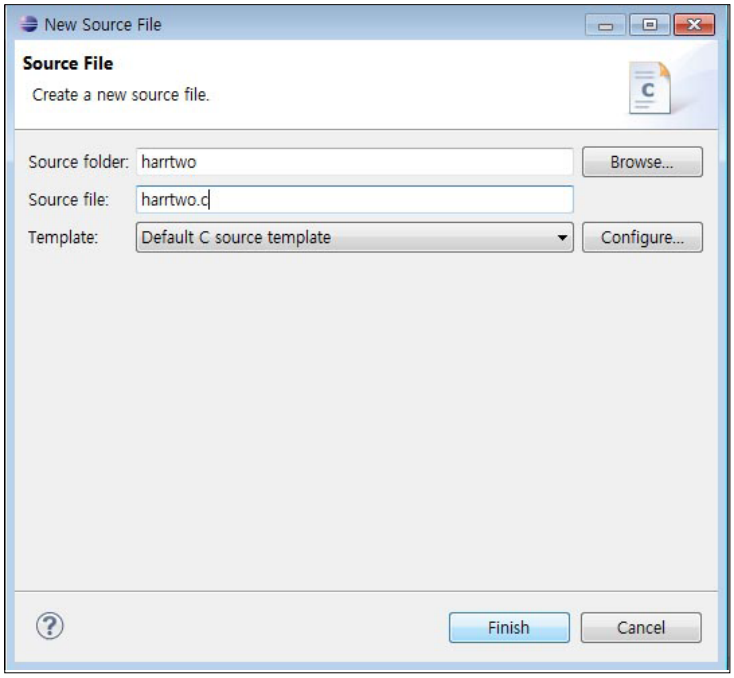
Toolchians 에서 MinGW GCC 를 선택하고 Finish 버튼을 누릅니다.

06 소스파일 추가



File 〉New 〉Source File 을 클릭합니다.

07 소스파일 이름

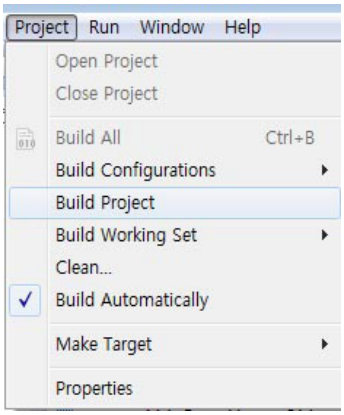


harrtwo.c 라고 입력하고 Finish 버튼을 누릅니다.

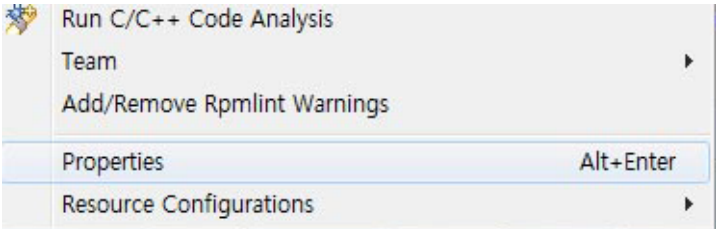
08 기본 제공 파일 추가

본 교재에게 C 프로그래밍을 하기 위해선 기본 제공되는 라이브러리 파일을 먼저 추가해주어야 올바른 컴파일을 할 수 있습니다. 필요한 파일은 drApi.dll, drApi.h, hovis.c, hovis.h 등 총 4가지 입니다.

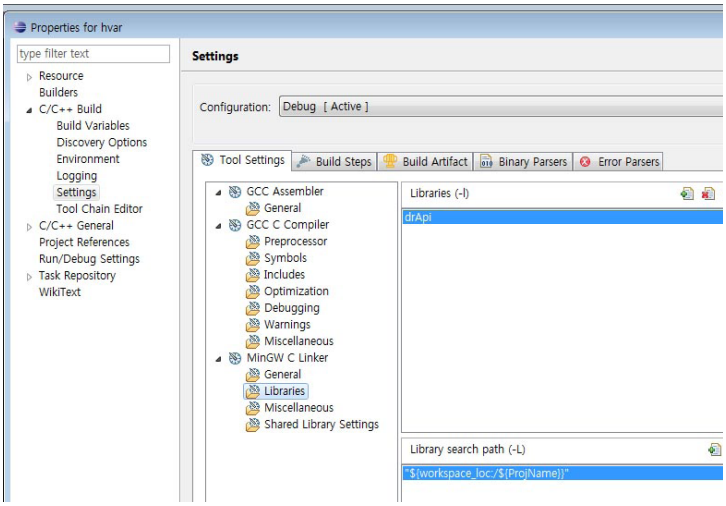
파일 추가 전에 먼저 Property 에서 라이브러리를 설정해주어야 합니다. 파일 생성을 한 후에 아무 코딩도 하지 말고, 바로 Build project 를 실행하세요. 그래야 라이브러리 추가가 가능합니다.



Build Project 를 클릭합니다.



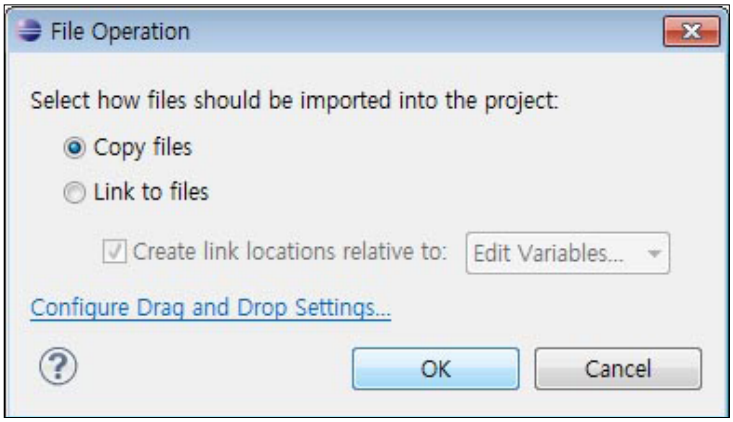
좌측 프로젝트 이름에 커서를 놓고 오른쪽 마우스키를 누릅니다.
제일 하단에 Propertis 를 클릭합니다. Alt+Enter 도 동일합니다.



C/C++ Build 〉 Settings 를 클릭하고, 우측에 MinGW C Linker 〉 Libraties 를 클릭합니다. 우측에 Libraies 에 + 마크를 클릭하고 팝업창이 뜨면 drApi 라고 입력합니다. Library search path 에 + 마크를 클릭하고 팝업창이 뜨면 Workspace 를 클릭하고, Folder selection 에서 프로젝트 이름과 동일한 것을 클릭하고 OK 버튼을 누릅니다.

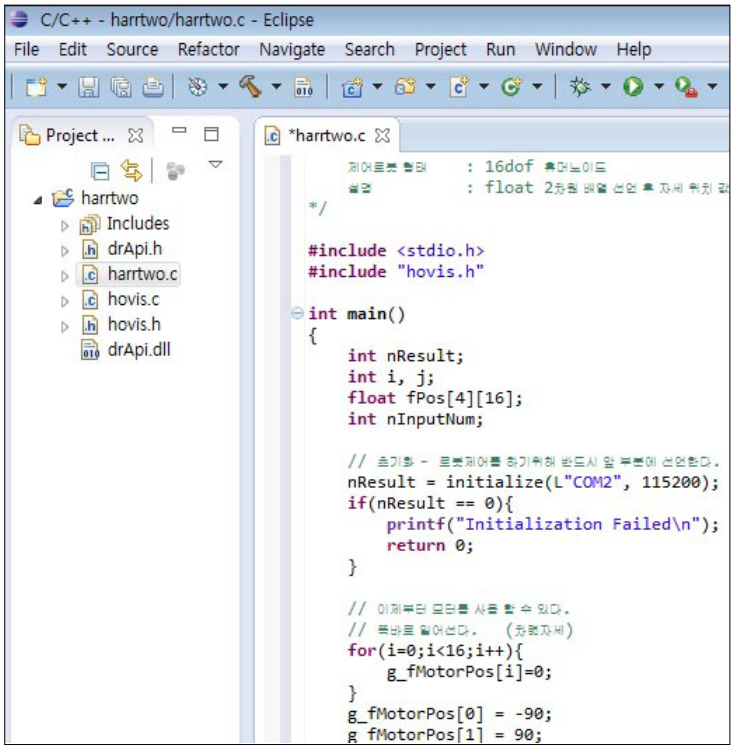
로봇을 동작시키기 위해선 위 네 개 파일을 프로젝트에 포함시켜야 합니다.
위 파일이 있는 파일탐색기 폴더에서 드래그 하는 방식으로 프로젝트에 포함시킵니다.
파일을 끌어다가 드래그해서 왼쪽 프로젝트에 추가합니다.

09 기본 파일 Import



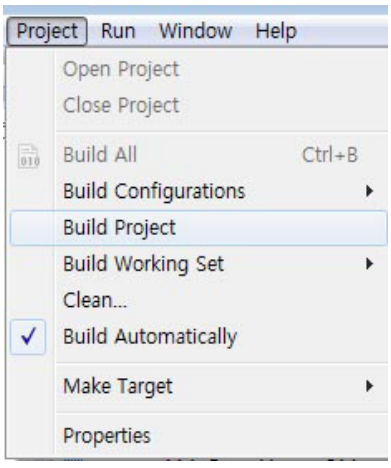
OK 버튼을 누릅니다.

10 소스 작성



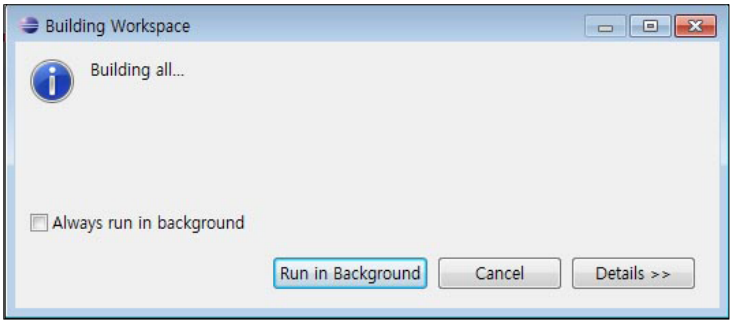
소스를 작성합니다.

11 빌드



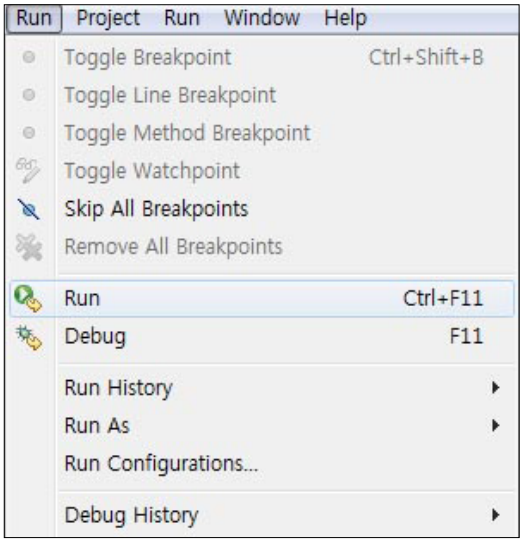
Projec 〉Build Project 를 클릭합니다.

12 빌드중



빌드 팝업입니다. 그대로 두면 완료 후 사라집니다.

13 Run 실행



Run 〉 Run 을 클릭합니다. Ctrl+F11 단축키도 동일합니다.

14 실행화면

```
        if(nInputNum>=0 && nInputNum<=3){
            for(i = 0;i < 16;i++){
                g_fMotorPos[i] = fPos[nInputNum][i];
            }
            run(1000);
            delay(1000);
        }
        else{
            printf("unknown command.\n");
            fflush(stdout);
        }
    }

    // 프로그램 종료
    terminate();
    return 0;
}
```

Problems Tasks Console Properties

```
harrtwo.exe [C/C++ Application] C:₩GccDongbu₩workspacew₩harrtwo₩De
1
Input Number
2
Input Number
```

15 로봇동작

입력받은 숫자에 따라 로봇이 다른 자세를 취합니다.

5.3 포인터

포인터라고 하면 변수를 의미한다. 포인터변수와 같은 개념입니다. 변수는 데이터를 저장하기 위한 것이지만, 데이터가 아닌 주소값을 저장하면 포인터 입니다.

```
Char c='d'   //2진 데이터로 변환해서 들어감
Int n=10;    //
Double d=5.29;       //

메모리 구조 표현 방식으로 이해하면 아래와 같습니다.
Ox1000  —〉 d
Ox1001 ~ Ox1004 —〉 7  ==〉 n의 주소값은 시작번지 Ox1001 입니다.
n 은 int 형 데이터이기 때문입니다.
Ox1005 ~ Ox100c —〉 3.14  ==〉 d의 주소값은 시작번지  Ox1005 입니다.

int *a; //  a는 포인터임, int형 변수 상수를 가르킬수 있는 포인터입니다.
char *b; // b는 포인터임, char 형 변수 상수를 가르킬 수 있는 포인터입니다.
double *c; //c는 포인터임, double 형 변수 상수를 가르킬 수 있는 포인터입니다.
```

아래 예제는 int 변수와 int 포인터 변수 선언 후 int 변수 값을 변경해가며 LED 제어하는 프로그램입니다.

hpointer.c

```
#include 〈stdio.h〉
#include "hovis.h"

int main()
{
        int nResult;
        int i, j;
        int nLed;
        int *pnLed;

        nResult = initialize(L"COM2", 115200);
        if(nResult == 0){
                printf("Initialization Failed₩n");
                fflush(stdout);
        return 0;

}
```

```
        for(i=0;i<16;i++){
                g_fMotorPos[i]=0;

        }
        g_fMotorPos[0] = –90;
        g_fMotorPos[1] = 90;
        g_fMotorPos[3] = –90;
        g_fMotorPos[4] = 90;
        run(1000);

        delay(1000);

        pnLed = &nLed;

        nLed = 1;

        printf("pnLed = %d, nLed = %d₩n", pnLed, nLed);
        fflush(stdout);
        printf("*pnLed = %d, &nLed = %d₩n", *pnLed, &nLed);
        fflush(stdout);
        for(i = 0;i < 16;i++){
                g_ucMotorGreenLed[i] = nLed;
        }
        run(0);

        delay(1000);

        *pnLed = 0;

        printf("pnLed = %d, nLed = %d₩n", pnLed, nLed);
        fflush(stdout);
        printf("*pnLed = %d, &nLed = %d₩n", *pnLed, &nLed);
        fflush(stdout);

        for(i = 0;i < 16;i++){
                g_ucMotorGreenLed[i] = nLed;
        }
        run(0);

        delay(1000);

        nLed = 1;

        printf("pnLed = %d, nLed = %d₩n", pnLed, nLed);
        fflush(stdout);
        printf("*pnLed = %d, &nLed = %d₩n", *pnLed, &nLed);
        fflush(stdout);
        for(i = 0;i < 16;i++){
                g_ucMotorGreenLed[i] = *pnLed;
        }
        run(0);

        delay(1000);

        terminate();

        return 0;
}
```

위 코드 중에 진하게 처리된 int *pnLed; 포인터를 살펴봅니다.

```
int nLed;          // 모터의 LED 값이 저장될 변수
int *pnLed;        // nLed의 주소가 저장될 포인터 변수
// nLed 값을 변경 후 nLed값으로 LED 제어
pnLed = &nLed; // pnLed 값을 nLed의 주소 값으로 대입
nLed = 1;          // nLed에 1 대입
printf("pnLed = %d, nLed = %d₩n", pnLed, nLed);
fflush(stdout);
// pnLed 값과 nLed 값을 출력
printf("*pnLed = %d, &nLed = %d₩n", *pnLed, &nLed);
fflush(stdout);
// pnLed가 가리키는 값과 nLed의 주소 값을 출력
```

문법요약

◆ 포인터

※ 포인터란 무엇인가?

– 포인터와 포인터 변수

포인터라고하면 변수를 의미합니다. 포인터변수와 같은 개념입니다.

– 메모리의 주소값을 저장하기 위한 변수를 말합니다. 일반적인 변수는 데이터를 저장하기 위한 것이지만, 데이터가 아닌 주소값을 저장하면 포인터입니다.

– "포인터"를 흔히 "포인터 변수"라고 합니다.

포인터도 경우에 따라 상수가 될 수도 있습니다. 하지만 상수는 매우 적습니다. 그 상수의 절반

은 상수화 변수입니다.

– 주소값과 포인터는 다릅니다.

◆ 메모리 구조 표현 방식

```
int main(void)
{
    char a='c';
    int n=4;
    double d=5.16
```

0x1000	a='c'
0x1001	
0x1002	
0x1003	n=7
0x1004	
0x1005	
0x1006	
0x1007	
0x1008	d=5.16
0x1009	
0x100a	
0x100b	
0x100c	

※ 포인터란 주소값을 저장하기 위한, 변수입니다.

8비트 → 1바이트 주소값

16비트 → 2바이트 주소값

32비트 → 4바이트 주소값.

역사적으로 포인터의 주소값은 변해왔습니다. 오늘날은 4바이트로 표현됩니다. → 32비트기반은 4바이트 입니다.

문제자체가 2차원적인 경우가 많습니다.

내가 해결해야할 문제는 2차원인데, 실제 구조는 1차원적이기는 하지만 2차원적으로 하는 습관을 가져야 합니다.

코딩계획

```
// 결과값, for, 모터 led 값,주소  저장 변수
//초기화
//모터사용
//LED 사용
```

프로그래밍 세부설계

```
/*
파일명              : hpointer.c
제어로봇 형태       : 16dof 휴머노이드
설명                : int 변수와 int 포인터 변수 선언 후 int 변수 값을 변경해가며 LED 제어
*/

// 동부로봇의 api 를 사용하기 위해 반드시 이렇게 선언 해 두어야 한다.

int main()
{
        // 결과값을 리턴받을 변수
        // for 문을 사용하기 위해 선언한 변수
```

```
// 모터의 LED 값이 저장될 변수
// nLed의 주소가 저장될 포인터 변수

// 초기화 – 로봇제어를 하기 위해 반드시 앞 부분에 선언한다.
// 각 변수들을 초기화 한다.
// 열고 난 이후 이상이 있는 지 확인한다.
        // 이상이 있다면 에러 출력
        // 프로그램 종료

// 이제부터 모터를 사용 할 수 있다.
// 똑바로 일어선다.        (차렷자세)

        // 모든 모터(16개 모터)를 0 위치로 설정함
// 0 번 모터(오른쪽 어깨)
// 1 번 모터(오른쪽 윗팔)
// 3 번 모터(왼쪽 어깨)
// 4 번 모터(왼쪽 윗팔)
// 모터 동작(해당 자세를 1000ms 동안 동작)
// 동작 대기(1000ms)

// nLed 값을 변경 후 nLed값으로 LED 제어
// pnLed 값을 nLed의 주소 값으로 대입
// nLed에 1 대입

// pnLed 값과 nLed 값을 출력
// pnLed가 가리키는 값과 nLed의 주소 값을 출력

// 16개의 모터에 대해서 반복
        // nLed 값을 초록색 LED 배열 원소에 넣는다.

// 동작 실행(LED만이므로 0ms로 해도 됨)
// 동작 대기(1000ms)

// *pnLed 값을 변경 후 nLed값으로 LED 제어
// pnLed가 가리키는 값(nLed)에 0 대입

// pnLed 값과 nLed 값을 출력
// pnLed가 가리키는 값과 nLed의 주소 값을 출력

// 16개의 모터에 대해서 반복
        // nLed 값을 초록색 LED 배열 원소에 넣는다.

// 동작 실행(LED만이므로 0ms로 해도 됨)
// 동작 대기(1000ms)

// nLed 값을 변경 후 *pnLed값으로 LED 제어
// nLed에 1 대입
```

```
        // pnLed 값과 nLed 값을 출력
        // pnLed가 가리키는 값과 nLed의 주소 값을 출력

        // 16개의 모터에 대해서 반복
                // pnLed가 가리키는 곳의 값을 초록색 LED 배열 원소에 넣는다.

        // 동작 실행(LED만이므로 0ms로 해도 됨)
        // 동작 대기(1000ms)

        // 프로그램 종료
        // 제어 종료 함수를 실행한다.
```

프로그래밍 세부설계

```
/*
파일 명             : hpointer.c
제어로봇 형태       : 16dof 휴머노이드
설명                : int 변수와 int 포인터 변수 선언 후 int 변수 값을 변경해가며 LED 제어
*/

#include <stdio.h>
#include "hovis.h"
// 동부로봇의 api 를 사용하기 위해 반드시 이렇게 선언 해 두어야 한다.

int main()
{
        int nResult;                    // 결과값을 리턴받을 변수
        int i, j;                       // for 문을 사용하기 위해 선언한 변수
        int nLed;                       // 모터의 LED 값이 저장될 변수
        int *pnLed;                     // nLed의 주소가 저장될 포인터 변수
        // 초기화 – 로봇제어를 하기위해 반드시 앞 부분에 선언한다.
        nResult = initialize(L"COM2", 115200);      // 각 변수들을 초기화 한다.
        if(nResult == 0){  // 열고 난 이후 이상이 있는 지 확인한다.
                printf("Initialization Failed₩n");      // 이상이 있다면 에러 출력
                fflush(stdout);
                return 0;// 프로그램 종료
        }
        // 이제부터 모터를 사용 할 수 있다.
        // 똑바로 일어선다.         (차렷자세)
        for(i=0;i<16;i++){
                g_fMotorPos[i]=0;// 모든 모터(16개 모터)를 0 위치로 설정
}
```

```
g_fMotorPos[0] = –90;                    // 0 번 모터(오른쪽 어깨)
g_fMotorPos[1] = 90;                     // 1 번 모터(오른쪽 윗팔)
g_fMotorPos[3] = –90;                    // 3 번 모터(왼쪽 어깨)
g_fMotorPos[4] = 90;                     // 4 번 모터(왼쪽 윗팔)
run(1000);          // 모터 동작(해당 자세를 1000ms 동안 동작)
delay(1000);                             // 동작 대기(1000ms)

// nLed 값을 변경 후 nLed값으로 LED 제어
pnLed = &nLed; // pnLed 값을 nLed의 주소 값으로 대입
nLed = 1;                                // nLed에 1 대입

printf("pnLed = %d, nLed = %d₩n", pnLed, nLed);
fflush(stdout);
// pnLed 값과 nLed 값을 출력
printf("*pnLed = %d, &nLed = %d₩n", *pnLed, &nLed);
fflush(stdout);
// pnLed가 가리키는 값과 nLed의 주소 값을 출력
for(i = 0;i < 16;i++){                   // 16개의 모터에 대해서 반복
        g_ucMotorGreenLed[i] = nLed;
// nLed 값을 초록색 LED 배열 원소에 넣는다.
}
run(0);                          // 동작 실행(LED만이므로 0ms로 해도 됨)
delay(1000);                             // 동작 대기(1000ms)

// *pnLed 값을 변경 후 nLed값으로 LED 제어
*pnLed = 0;                      // pnLed가 가리키는 값(nLed)에 0 대입

printf("pnLed = %d, nLed = %d₩n", pnLed, nLed);
fflush(stdout);
// pnLed 값과 nLed 값을 출력
printf("*pnLed = %d, &nLed = %d₩n", *pnLed, &nLed);
fflush(stdout);
// pnLed가 가리키는 값과 nLed의 주소 값을 출력

for(i = 0;i < 16;i++){                   // 16개의 모터에 대해서 반복
        g_ucMotorGreenLed[i] = nLed;
// nLed 값을 초록색 LED 배열 원소에 넣는다.
}
run(0);                          // 동작 실행(LED만이므로 0ms로 해도 됨)
delay(1000);                             // 동작 대기(1000ms)
// nLed 값을 변경 후 *pnLed값으로 LED 제어
nLed = 1;                                // nLed에 1 대입
printf("pnLed = %d, nLed = %d₩n", pnLed, nLed);
fflush(stdout);
// pnLed 값과 nLed 값을 출력
```

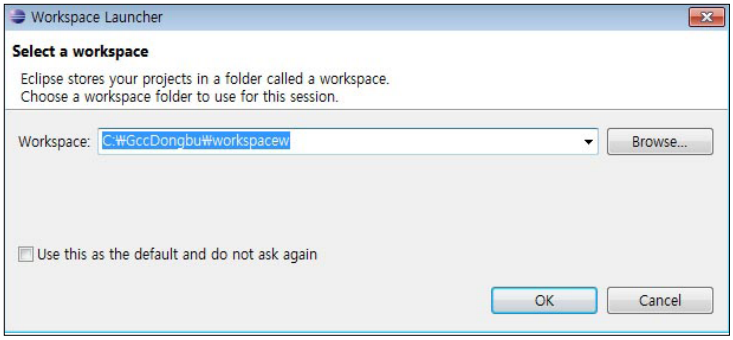
```
        printf("*pnLed = %d, &nLed = %d₩n", *pnLed, &nLed);
        fflush(stdout);
        // pnLed가 가리키는 값과 nLed의 주소 값을 출력
        for(i = 0;i < 16;i++){                    // 16개의 모터에 대해서 반복
                g_ucMotorGreenLed[i] = *pnLed;
// pnLed가 가리키는 곳의 값을 초록색 LED 배열 원소에 넣는다.
        }
        run(0);                          // 동작 실행(LED만이므로 0ms로 해도 됨)
        delay(1000);                              // 동작 대기(1000ms)

        // 프로그램 종료
        terminate();                              // 제어 종료 함수를 실행한다.
        return 0;
}
```

빌드 및 실행

01 이클립스 실행

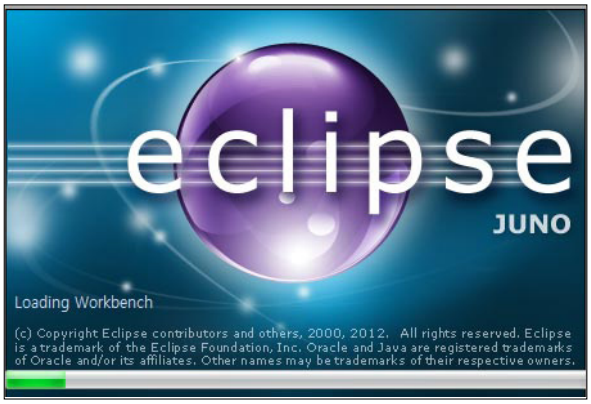
이클립스 실행 버튼을 누릅니다.

02 workspace



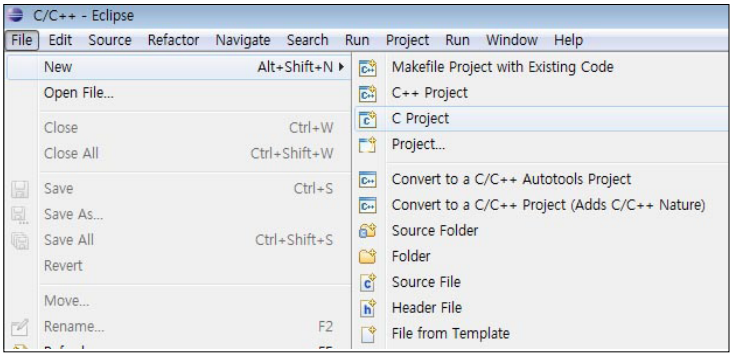
작업하고자 하는 폴더를 지정합니다. Browse 를 클릭하여 폴더를 지정하고 OK 버튼을 누릅니다.

03 이클립스 로딩 화면



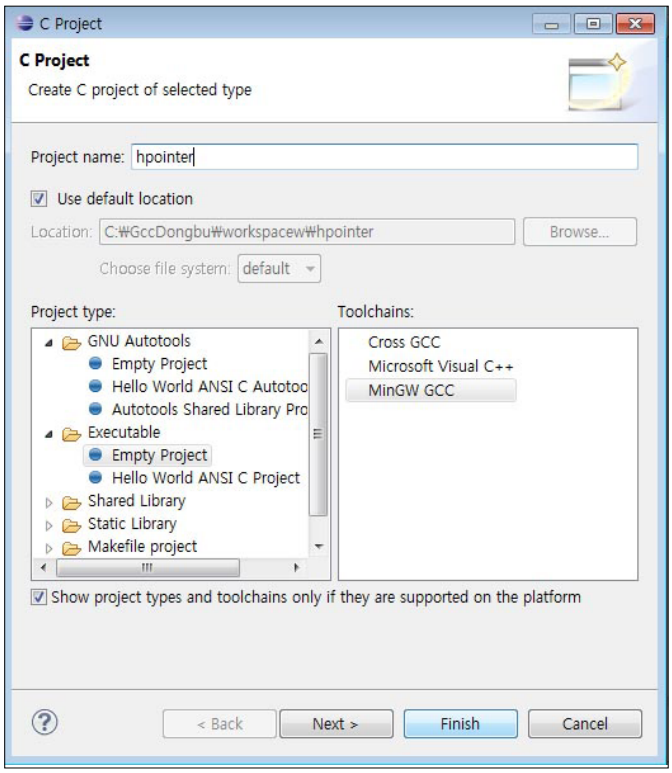
이클립스 로딩 화면입니다.

04 프로젝트 생성



이클립스 편집창 메뉴에서 File 〉 New 〉 C project 를 클릭합니다.

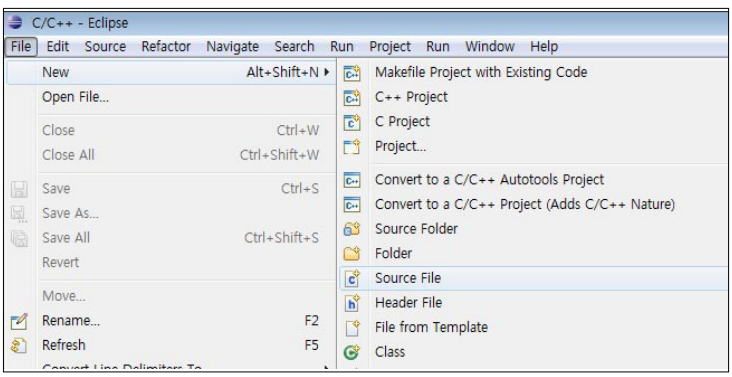
05 프로젝트 이름



Projec Name 을 hpointer 라고 입력하고, Project type 에서 Exectable 에서 Empty Project 를 선택합니다.

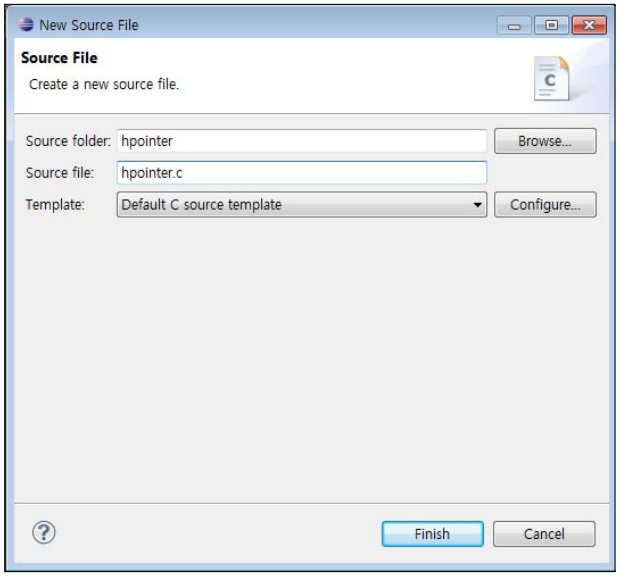
Toolchians 에서 MinGW GCC 를 선택하고 Finish 버튼을 누릅니다.

06 소스파일 추가



File〉New〉Source File 을 클릭합니다.

07 소스파일 이름

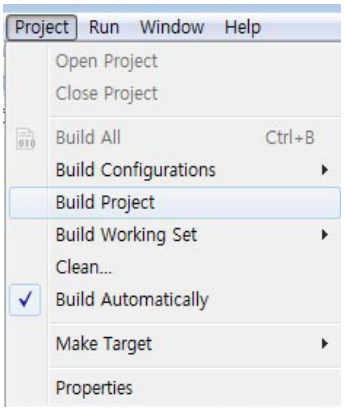


hpointer.c 라고 입력하고 Finish 버튼을 누릅니다.

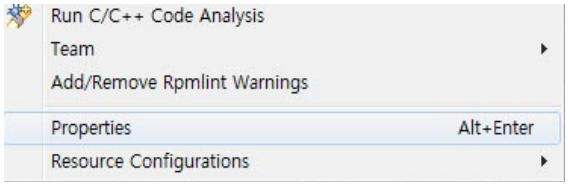
08 기본 제공 파일 추가

본 교재에게 C 프로그래밍을 하기 위해선 기본 제공되는 라이브러리 파일을 먼저 추가해주어야 올바른 컴파일을 할 수 있습니다. 필요한 파일은 drApi.dll, drApi.h, hovis.c, hovis.h 등 총 4가지 입니다.

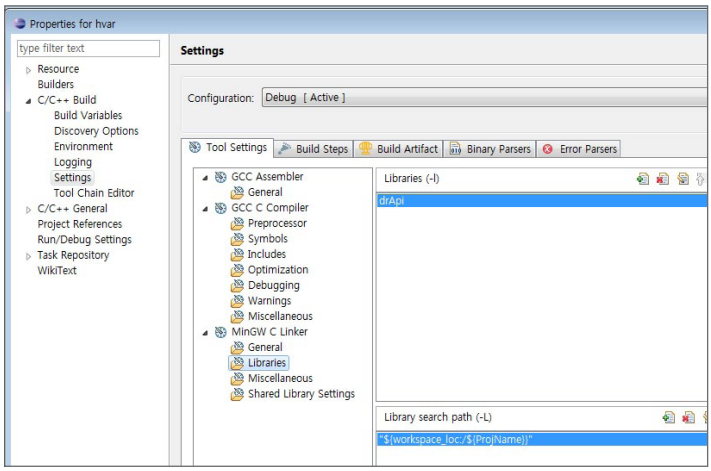
파일 추가 전에 먼저 Property 에서 라이브러리를 설정해주어야 합니다. 파일 생성을 한 후에 아무 코딩도 하지 말고, 바로 Build project 를 실행하세요. 그래야 라이브러리 추가가 가능합니다.



Build Project 를 클릭합니다.



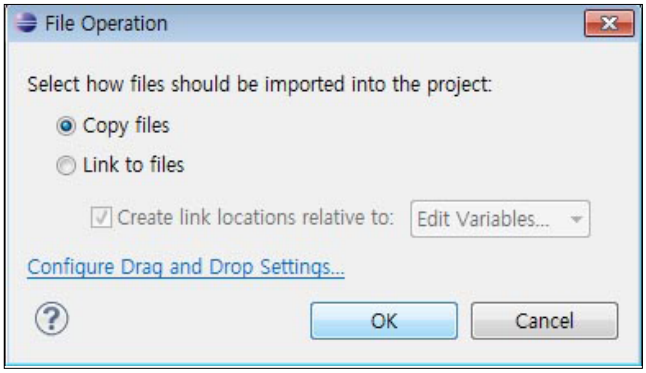
좌측 프로젝트 이름에 커서를 놓고 오른쪽 마우스키를 누릅니다.
제일 하단에 Propertis 를 클릭합니다. Alt+Enter 도 동일합니다.



C/C++ Build 〉 Settings 를 클릭하고, 우측에 MinGW C Linker 〉 Libraties 를 클릭합니다. 우측에 Libraies 에 + 마크를 클릭하고 팝업창이 뜨면 drApi 라고 입력합니다. Library search path 에 + 마크를 클릭하고 팝업창이 뜨면 Workspace 를 클릭하고, Folder selection 에서 프로젝트 이름과 동일한 것을 클릭하고 OK 버튼을 누릅니다.

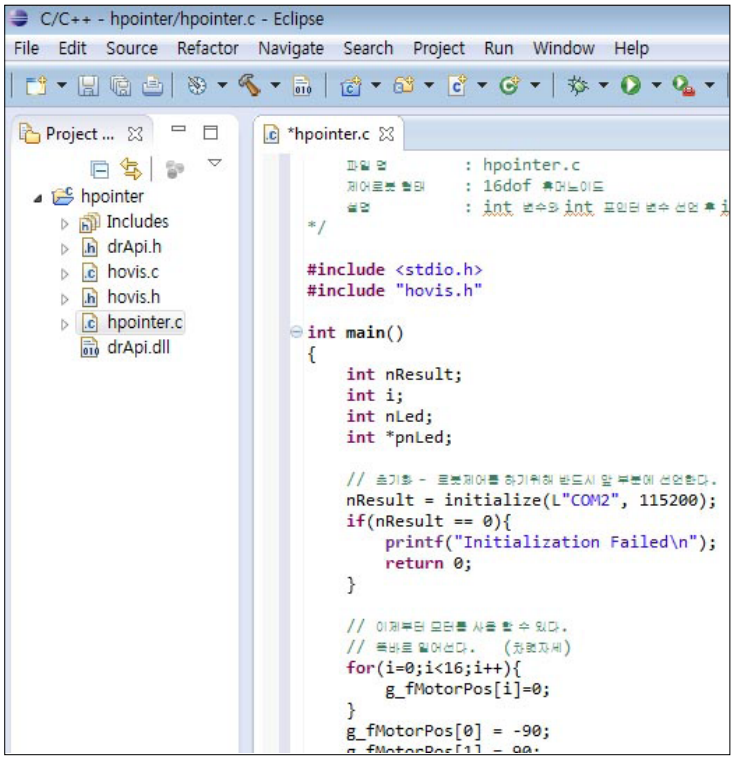
로봇을 동작시키기 위해선 위 네 개 파일을 프로젝트에 포함시켜야 합니다.
위 파일이 있는 파일탐색기 폴더에서 드래그 하는 방식으로 프로젝트에 포함시킵니다.
파일을 끌어다가 드래그해서 왼쪽 프로젝트에 추가합니다.

09 기본 파일 Import



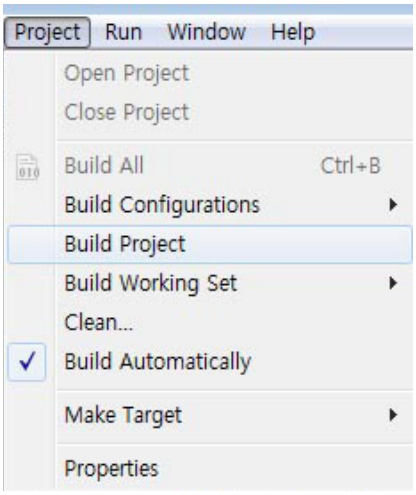
OK 버튼을 누릅니다.

10 소스 작성



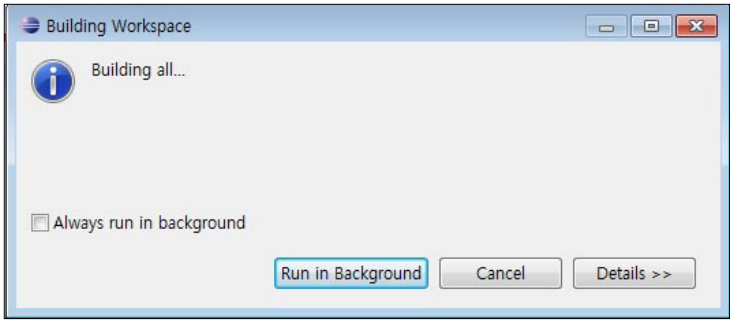
소스를 작성합니다.

11 빌드



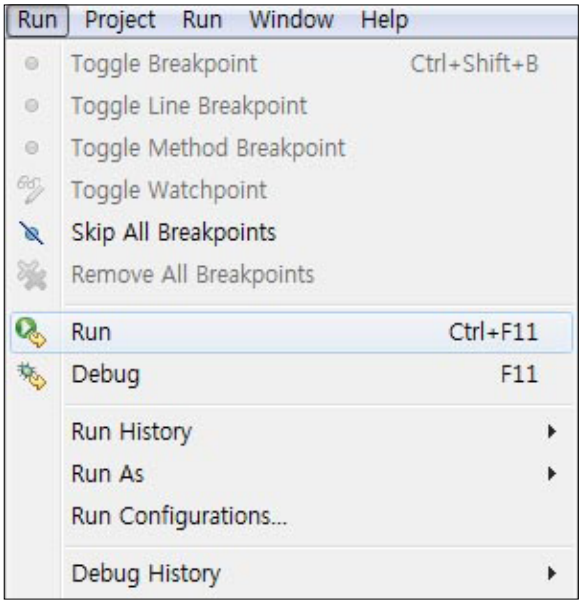
Projec 〉Build Project 를 클릭합니다.

12 빌드중



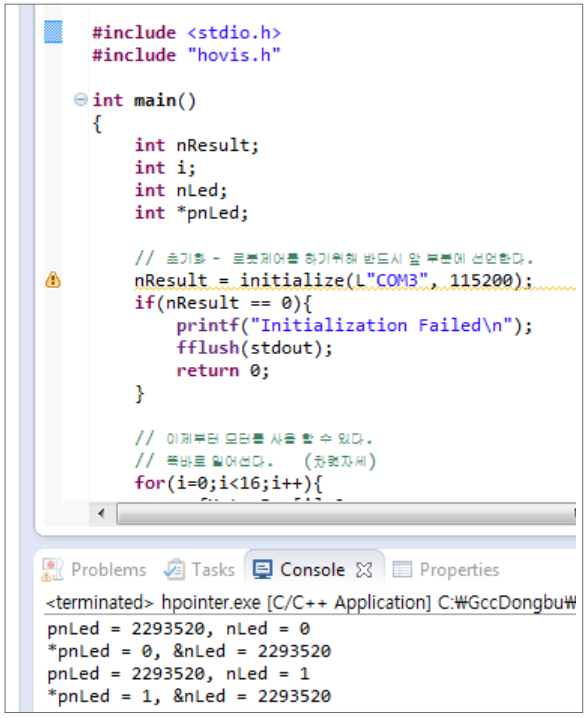
빌드 팝업입니다. 그대로 두면 완료 후 사라집니다.

13 Run 실행



Run 〉Run 을 클릭합니다. Ctrl+F11 단축키도 동일합니다.

14 실행화면



15 로봇동작

int 변수 값을 변경하면 LED 가 제어됩니다.

5.4 포인터와 배열

포인터 = 주소값 + 자료형 입니다. 따라서 배열이름도 포인터입니다.
배열 이름은 포인터 상수이기도 합니다.
배열은 생성과 동시에 배열요소의 첫 번째 요소를 가리키게 됩니다. 따라서 배열이름은 상수일 수밖에 없습니다. 배열 이름을 포인터처럼, 포인터를 배열 이름처럼 활용하는 것이 가능합니다.

아래 예제는 배열의 이름과 배열의 각 원소의 주소가 어떤 관계인지 모니터 출력을 통해 알아봅니다.. 또한 포인터 변수를 선언하고 전역 변수 배열의 주소를 포인터 변수에 넣어서 주소를 서로 비교하고, 선언한 포인터 변수를 통해 로봇을 제어합니다.

hparr.c

```
#include <stdio.h>
#include "hovis.h"

int main()
{
        int nResult;

        int i;

        unsigned char *pucLed;
        float *pfPos;

        nResult = initialize(L"COM2", 115200);
        if(nResult == 0){
        printf("Initialization Failed₩n");
        fflush(stdout);
                return 0;

        }
        for(i=0;i<16;i++){
                g_fMotorPos[i]=0;
}
        g_fMotorPos[0] = -90;
        g_fMotorPos[1] = 90;
        g_fMotorPos[3] = -90;
        g_fMotorPos[4] = 90;
        run(1000);

        delay(1000);
```

```
        printf("&g_ucMotorGreenLed[0] = %d₩n", &g_ucMotorGreenLed[0]);
        fflush(stdout);
        printf("&g_ucMotorGreenLed[1] = %d₩n", &g_ucMotorGreenLed[1]);
        fflush(stdout);
        printf("g_ucMotorGreenLed = %d₩n₩n", g_ucMotorGreenLed);
        fflush(stdout);
        printf("&g_fMotorPos[0] = %d₩n", &g_fMotorPos[0]);
        fflush(stdout);
        printf("&g_fMotorPos[1] = %d₩n", &g_fMotorPos[1]);
        fflush(stdout);
        printf("g_fMotorPos = %d₩n₩n", g_fMotorPos);
        fflush(stdout);
        pucLed = g_ucMotorGreenLed;
        pfPos = g_fMotorPos;

        printf("&pucLed[0] = %d₩n", &pucLed[0]);
        fflush(stdout);
        printf("&pucLed[1] = %d₩n", &pucLed[1]);
        fflush(stdout);
        printf("pucLed = %d₩n", pucLed);
        fflush(stdout);
        printf("pucLed+1 = %d₩n₩n", pucLed+1);
        fflush(stdout);
        printf("&pfPos[0] = %d₩n", &pfPos[0]);
        fflush(stdout);
        printf("&pfPos[1] = %d₩n", &pfPos[1]);
        fflush(stdout);
        printf("pfPos = %d₩n", pfPos);
        fflush(stdout);
        printf("pfPos+1 = %d₩n₩n", pfPos+1);
        fflush(stdout);
        for(i = 0;i < 16;i++){
                pucLed[i] = 1;
        }

        pfPos[0] = 90;

        pfPos[3] = 90;

        run(1000);

        delay(1000);

        terminate();
        return 0;
}
```

위 코드 중에 진하게 처리된 포인터와 배열 문장을 살펴봅니다.

```
unsigned char *pucLed;// Led 배열의 주소가 저장될 포인터 변수
float *pfPos;         // 모터 위치 배열의 주소가 저장될 포인터 변수
```

문법요약

◆ 포인터와 배열의 관계

– 배열 이름은 첫 번째 요소의 주소값을 나타냅니다.
Int a[5]={0,1,2,3,4}
Ox1000
a[0] : a라는 배열의 0번째 요소, &a[0] : 0번째 인덱스에 해당하는 주소값을 반환 →
Ox1000
Ox1004
Ox1008
….
a 라는것은 배열 첫번째 요소의 주소를 나타냅니다.

◆ 포인터 연산과 배열

※ arr이 포인터이거나 배열이름인 경우

```
arr[i] == *(arr+i)
```

*ptr++ // *ptr의 값을 구하고 ptr을 1증가시킵니다.
(*ptr)++ // 먼저 *ptr의 값을 구한 후 *ptr의 값을 1 증가시킵니다.
*++ptr // ptr을 1증가시킨 후 *ptr을 구합니다.
++*ptr // *ptr의 값을 1증가 시킨 후 , 그 값을 결과로 취합니다.

코딩계획

```
// 결과값, for, LED 배열 주소, 모터위치 배열 주소 저장 변수
//초기화
//모터사용
//배열이름을 가리키는 주소
//포인터변수에 배열이름 넣기
```

프로그래밍 세부설계

```
/*
파일명              : hparr.c
제어로봇 형태       : 16dof 휴머노이드
설명                : 배열의 이름과 배열의 각 원소의 주소가 어떤 관계인지 모니터 출력을
통해 알아본다. 또한 포인터 변수를 선언하고 전역 변수 배열의 주소를 포인터 변수에 넣
어서 주소를 서로 비교하고, 선언한 포인터 변수를 통해 로봇을 제어한다.
*/

// 동부로봇의 api 를 사용하기 위해 반드시 이렇게 선언 해 두어야 한다.

int main()
{
        // 결과값을 리턴받을 변수
        // for 문을 사용하기 위해 선언한 변수
        // Led 배열의 주소가 저장될 포인터 변수
        // 모터 위치 배열의 주소가 저장될 포인터 변수

        // 초기화 – 로봇제어를 하기 위해 반드시 앞 부분에 선언한다.
        // 각 변수들을 초기화 한다.
        // 열고 난 이후 이상이 있는 지 확인한다.
                // 이상이 있다면 에러 출력
                // 프로그램 종료
        }

        // 이제부터 모터를 사용 할 수 있다.
        // 똑바로 일어선다.          (차렷자세)

                // 모든 모터(16개 모터)를 0 위치로 설정함
        }
        // 0 번 모터(오른쪽 어깨)
```

```
// 1 번 모터(오른쪽 윗팔)
// 3 번 모터(왼쪽 어깨)
// 4 번 모터(왼쪽 윗팔)
// 모터 동작(해당 자세를 1000ms 동안 동작)
// 동작 대기(1000ms)

// 배열 이름이 가리키는 주소는 배열의 첫 번째 원소의 주소다.
// 배열의 종류가 unsigned char(1바이트) 이기 때문에
// g_ucMotorGreenLed[0]와 g_ucMotorGreenLed[1]의 주소는 1 차이가 난다.
//g_ucMotorGreenLed의 0번 원소의 주소 출력
//g_ucMotorGreenLed의 1번 원소의 주소 출력
//g_ucMotorGreenLed라는 배열 이름이 가리키는 주소 출력

// 마찬가지로 배열 이름이 가리키는 주소는 배열의 첫 번째 원소의 주소다.
// 배열의 종류가 float(4바이트) 이기 때문에
// g_fMotorPos[0]와 g_fMotorPos[1]의 주소는 1 차이가 난다.
//g_fMotorPos의 0번 원소의 주소 출력
//g_fMotorPos의 1번 원소의 주소 출력
//g_fMotorPos라는 배열 이름이 가리키는 주소 출력

// 사용자가 선언한 포인터 변수에 배열의 이름을 집어 넣는다.
// 배열의 이름이 곧 포인터이기 때문에 가능하다.
// g_ucMotorGreenLed 배열의 주소를 pucLed에 대입
// g_fMotorPos 배열의 주소를 pfPos에 대입

// pucLed는 포인터 변수이지만 배열처럼 [0], [1]를 붙여서 사용할 수 있다.
// arr[i] == *(arr+i) 라는 공식이 성립하기 때문이다.
// 마찬가지로 &pucLed[0] == pucLed, &pucLed[1] == pucLed+1임을 확인할 수 있다.
// pucLed[0]의 주소 출력
// pucLed[1]의 주소 출력
// pucLed가 가리키는 주소 출력

// pucLed에 1을 더한 주소 출력
// 마찬가지로 pfPos도 [0], [1]를 붙여서 사용할 수 있다.
// &pfPos[0] == pfPos, &pfPos[1] == pfPos+1임을 확인할 수 있다.

// pfPos[0]의 주소 출력
// pfPos[1]의 주소 출력
// pfPos가 가리키는 주소 출력
// pfPos에 1을 더한 주소 출력
```

```
// pucLed의 [0]부터 [15]까지 1을 집어 넣는다.
// pucLed == g_ucMotorGreenLed이기 때문에 효과가 같다.
// 16개의 모터에 대해서 반복
        // 1 값을 pucLed의 배열 원소에 넣는다.

// pfPos의 [0]과 [3]에 90을 집어 넣는다.
// pfPos == g_fMotorPos이기 때문에 효과가 같다.
// 0번 모터에 팔을 들게 하는 위치 값을 넣는다.

// 3번 모터에 팔을 들게 하는 위치 값을 넣는다.
// 모터 동작(해당 자세를 1000ms 동안 동작)
// 동작 대기(1000ms)

// 프로그램 종료
// 제어 종료 함수를 실행한다.
```

프로그래밍 작성

```
/*
파일 명              : hparr.c
제어로봇 형태        : 16dof 휴머노이드
설명                 : 배열의 이름과 배열의 각 원소의 주소가 어떤 관계인지 모니터 출력을
통해 알아본다. 또한 포인터 변수를 선언하고 전역 변수 배열의 주소를 포인터 변수에 넣
어서 주소를 서로 비교하고, 선언한 포인터 변수를 통해 로봇을 제어한다.
*/

#include 〈stdio.h〉
#include "hovis.h" // 동부로봇의 api 를 사용하기 위해 반드시 이렇게 선언 해 두어야 한다.

int main()
{
        int nResult;                    // 결과값을 리턴받을 변수
        int i;                          // for 문을 사용하기 위해 선언한 변수
        unsigned char *pucLed;// Led 배열의 주소가 저장될 포인터 변수
        float *pfPos;       // 모터 위치 배열의 주소가 저장될 포인터 변수

        // 초기화 – 로봇제어를 하기위해 반드시 앞 부분에 선언한다.
        nResult = initialize(L"COM2", 115200);=   // 각 변수들을 초기화 한다.
```

```
if(nResult == 0){   // 열고 난 이후 이상이 있는 지 확인한다.
        printf("Initialization Failed₩n");          // 이상이 있다면 에러 출력
        fflush(stdout);
        return 0;                          // 프로그램 종료
}

// 이제부터 모터를 사용 할 수 있다.
// 똑바로 일어선다.          (차렷자세)
for(i=0;i<16;i++){
        g_fMotorPos[i]=0;
// 모든 모터(16개 모터)를 0 위치로 설정함
}
g_fMotorPos[0] = -90;                              // 0 번 모터(오른쪽 어깨)
g_fMotorPos[1] = 90;                               // 1 번 모터(오른쪽 윗팔)
g_fMotorPos[3] = -90;                              // 3 번 모터(왼쪽 어깨)
g_fMotorPos[4] = 90;                               // 4 번 모터(왼쪽 윗팔)
run(1000);          // 모터 동작(해당 자세를 1000ms 동안 동작)
delay(1000);                                       // 동작 대기(1000ms)

// 배열 이름이 가리키는 주소는 배열의 첫 번째 원소의 주소다.
// 배열의 종류가 unsigned char(1바이트) 이기 때문에
// g_ucMotorGreenLed[0]와 g_ucMotorGreenLed[1]의 주소는 1 차이가 난다.
printf("&g_ucMotorGreenLed[0] = %d₩n", &g_ucMotorGreenLed[0]);
fflush(stdout);
//g_ucMotorGreenLed의 0번 원소의 주소 출력
printf("&g_ucMotorGreenLed[1] = %d₩n", &g_ucMotorGreenLed[1]);
fflush(stdout);
//g_ucMotorGreenLed의 1번 원소의 주소 출력
printf("g_ucMotorGreenLed = %d₩n₩n", g_ucMotorGreenLed);
fflush(stdout);
//g_ucMotorGreenLed라는 배열 이름이 가리키는 주소 출력

// 마찬가지로 배열 이름이 가리키는 주소는 배열의 첫 번째 원소의 주소다.
// 배열의 종류가 float(4바이트) 이기 때문에
// g_fMotorPos[0]와 g_fMotorPos[1]의 주소는 1 차이가 난다.
printf("&g_fMotorPos[0] = %d₩n", &g_fMotorPos[0]);
fflush(stdout);
//g_fMotorPos의 0번 원소의 주소 출력
printf("&g_fMotorPos[1] = %d₩n", &g_fMotorPos[1]);
fflush(stdout);
//g_fMotorPos의 1번 원소의 주소 출력
printf("g_fMotorPos = %d₩n₩n", g_fMotorPos);
fflush(stdout);
//g_fMotorPos라는 배열 이름이 가리키는 주소 출력
```

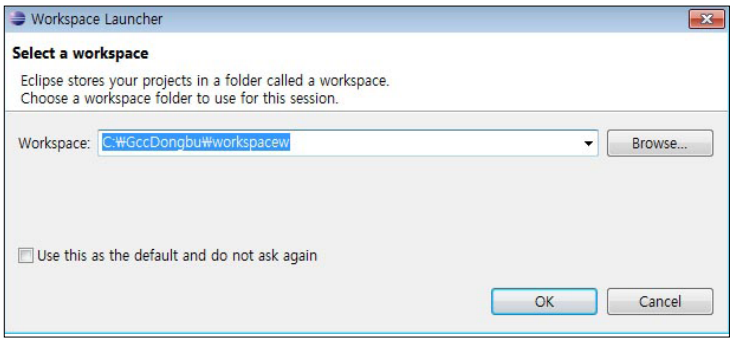
```
        // 사용자가 선언한 포인터 변수에 배열의 이름을 집어 넣는다.
        // 배열의 이름이 곧 포인터이기 때문에 가능하다.
        pucLed = g_ucMotorGreenLed;
        // g_ucMotorGreenLed 배열의 주소를 pucLed에 대입
        pfPos = g_fMotorPos;
        // g_fMotorPos 배열의 주소를 pfPos에 대입

        // pucLed는 포인터 변수이지만 배열처럼 [0], [1]를 붙여서 사용할 수 있다.
        // arr[i] == *(arr+i) 라는 공식이 성립하기 때문이다.
        // 마찬가지로 &pucLed[0] == pucLed, &pucLed[1] == pucLed+1임을 확인할 수 있다.
        printf("&pucLed[0] = %d₩n", &pucLed[0]);   // pucLed[0]의 주소 출력
        fflush(stdout);
        printf("&pucLed[1] = %d₩n", &pucLed[1]);   // pucLed[1]의 주소 출력
        fflush(stdout);
        printf("pucLed = %d₩n", pucLed);           // pucLed가 가리키는 주소 출력
        fflush(stdout);
        printf("pucLed+1 = %d₩n₩n", pucLed+1);     // pucLed에 1을 더한 주소 출력

        // 마찬가지로 pfPos도 [0], [1]를 붙여서 사용할 수 있다.
        // &pfPos[0] == pfPos, &pfPos[1] == pfPos+1임을 확인할 수 있다.
        printf("&pfPos[0] = %d₩n", &pfPos[0]);     // pfPos[0]의 주소 출력
        fflush(stdout);
        printf("&pfPos[1] = %d₩n", &pfPos[1]);     // pfPos[1]의 주소 출력
        fflush(stdout);
        printf("pfPos = %d₩n", pfPos);             // pfPos가 가리키는 주소 출력
        fflush(stdout);
        printf("pfPos+1 = %d₩n₩n", pfPos+1);       // pfPos에 1을 더한 주소 출력
        fflush(stdout);

        // pucLed의 [0]부터 [15]까지 1을 집어 넣는다.
        // pucLed == g_ucMotorGreenLed이기 때문에 효과가 같다.
        for(i = 0;i < 16;i++){                     // 16개의 모터에 대해서 반복
                pucLed[i] = 1;      // 1 값을 pucLed의 배열 원소에 넣는다.
        }

        // pfPos의 [0]과 [3]에 90을 집어 넣는다.
        // pfPos == g_fMotorPos이기 때문에 효과가 같다.
        pfPos[0] = 90;     // 0번 모터에 팔을 들게 하는 위치 값을 넣는다.
        pfPos[3] = 90;     // 3번 모터에 팔을 들게 하는 위치 값을 넣는다.

        run(1000);         // 모터 동작(해당 자세를 1000ms 동안 동작)
        delay(1000);                               // 동작 대기(1000ms)

        // 프로그램 종료
        terminate();                               // 제어 종료 함수를 실행한다.
        return 0;
}
```

빌드 및 실행

01 이클립스 실행

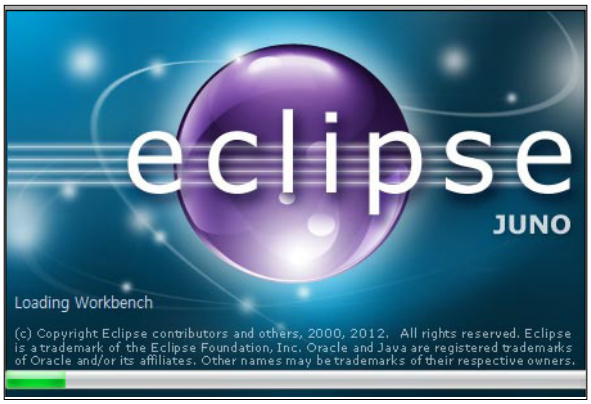
이클립스 실행 버튼을 누릅니다.

02 workspace



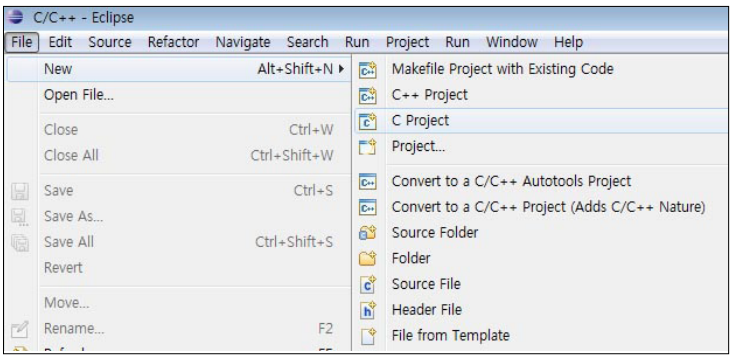
작업하고자 하는 폴더를 지정합니다. Browse 를 클릭하여 폴더를 지정하고 OK 버튼을 누릅니다.

03 이클립스 로딩 화면



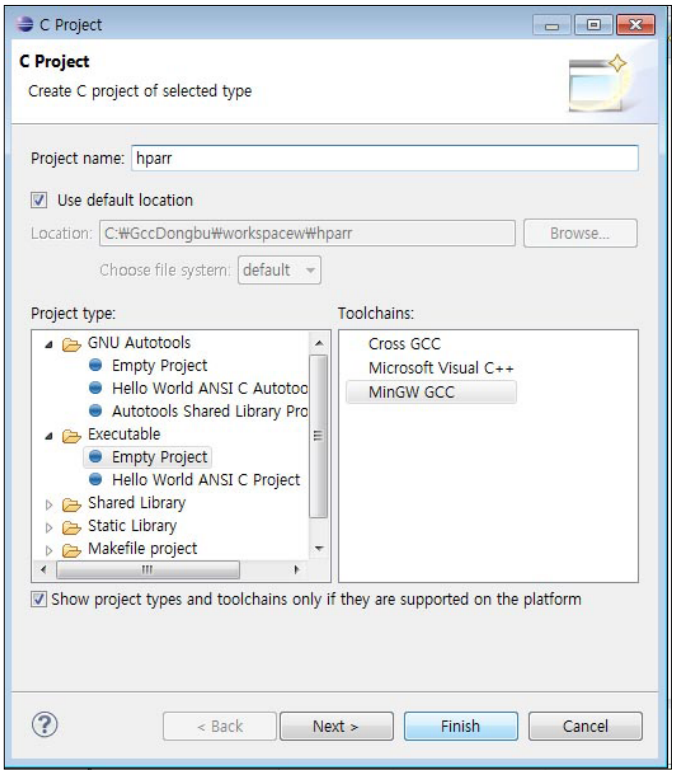
이클립스 로딩 화면입니다.

04 프로젝트 생성



이클립스 편집창 메뉴에서 File 〉 New 〉 C project 를 클릭합니다.

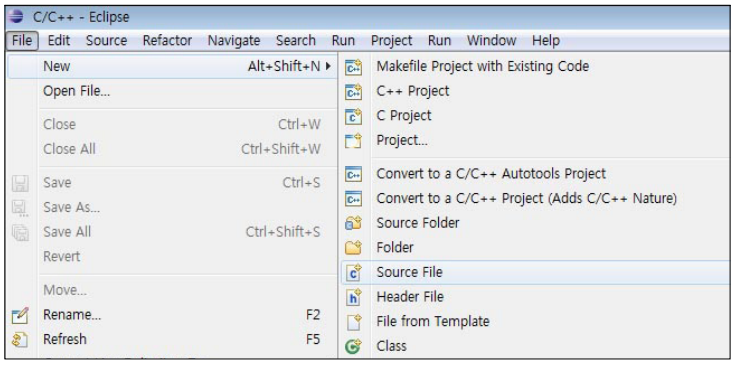
05 프로젝트 이름



Projec Name 을 hparr 라고 입력하고, Project type 에서 Exectable 에서 Empty Project 를 선택합니다.

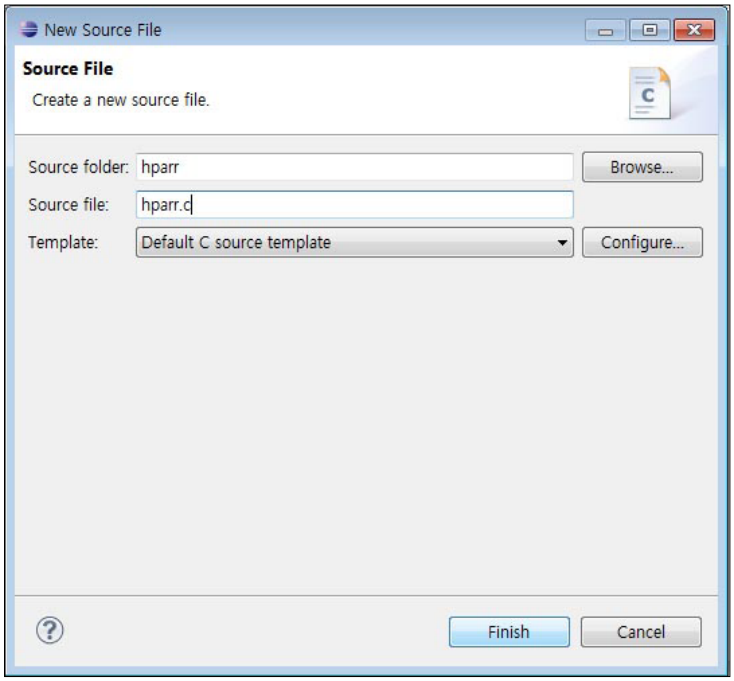
Toolchians 에서 MinGW GCC 를 선택하고 Finish 버튼을 누릅니다.

06 소스파일 추가



File 〉New 〉Source File 을 클릭합니다.

07 소스파일 이름

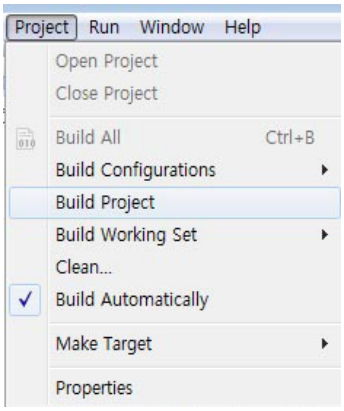


hparr.c 라고 입력하고 Finish 버튼을 누릅니다.

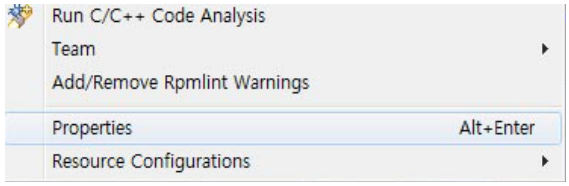
08 기본 제공 파일 추가

본 교재에게 C 프로그래밍을 하기 위해선 기본 제공되는 라이브러리 파일을 먼저 추가해주어야 올바른 컴파일을 할 수 있습니다. 필요한 파일은 drApi.dll, drApi.h, hovis.c, hovis.h 등 총 4가지 입니다.

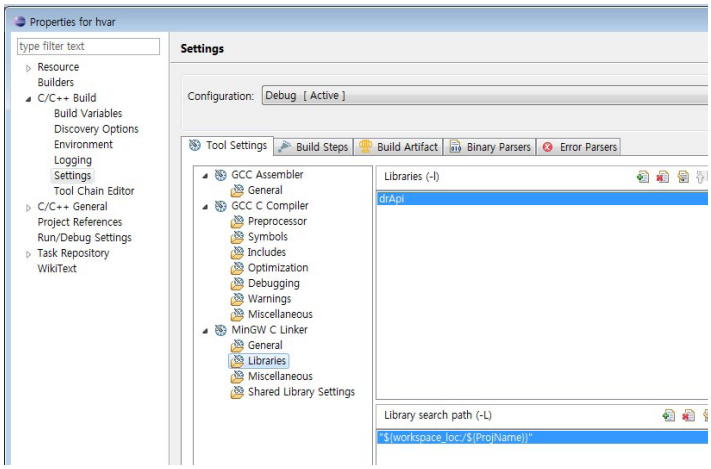
파일 추가 전에 먼저 Property 에서 라이브러리를 설정해주어야 합니다. 파일 생성을 한 후에 아무 코딩도 하지 말고, 바로 Build project 를 실행하세요. 그래야 라이브러리 추가가 가능합니다.



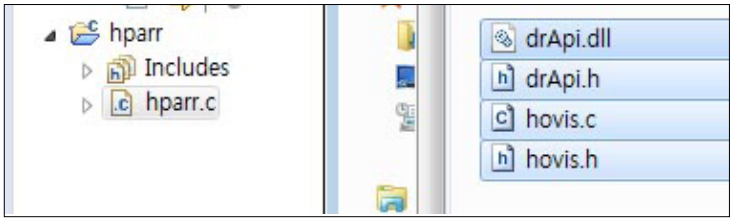
Build Project 를 클릭합니다.



좌측 프로젝트 이름에 커서를 놓고 오른쪽 마우스키를 누릅니다.
제일 하단에 Propertis 를 클릭합니다. Alt+Enter 도 동일합니다.

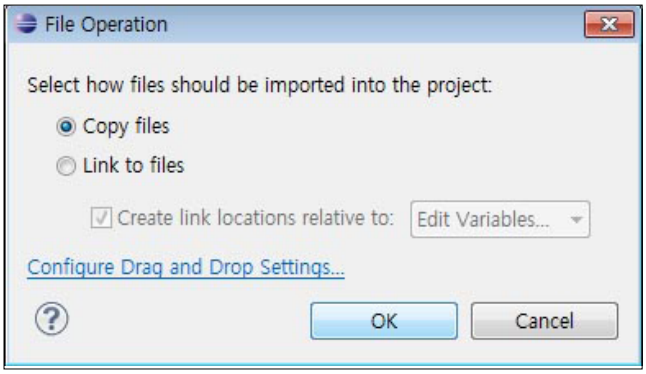


C/C++ Build 〉 Settings 를 클릭하고, 우측에 MinGW C Linker 〉 Libraties 를 클릭합니다. 우측에 Libraies 에 + 마크를 클릭하고 팝업창이 뜨면 drApi 라고 입력합니다. Library search path 에 + 마크를 클릭하고 팝업창이 뜨면 Workspace 를 클릭하고, Folder selection 에서 프로젝트 이름과 동일한 것을 클릭하고 OK 버튼을 누릅니다.



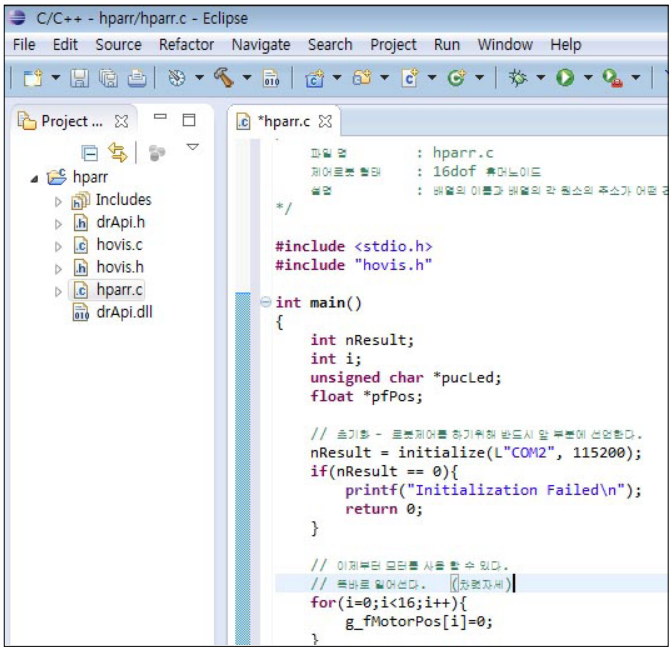
로봇을 동작시키기 위해선 위 네 개 파일을 프로젝트에 포함시켜야 합니다.
위 파일이 있는 파일탐색기 폴더에서 드래그 하는 방식으로 프로젝트에 포함시킵니다.
파일을 끌어다가 드래그해서 왼쪽 프로젝트에 추가합니다.

09 기본 파일 Import



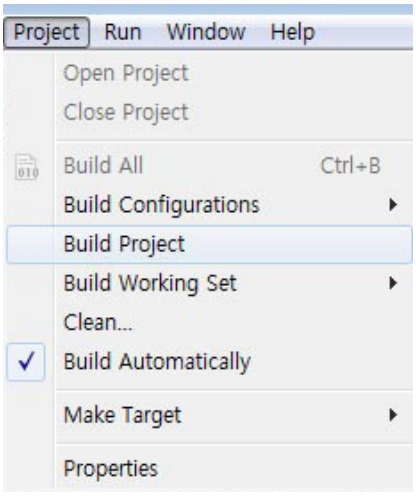
OK 버튼을 누릅니다.

10 소스 작성



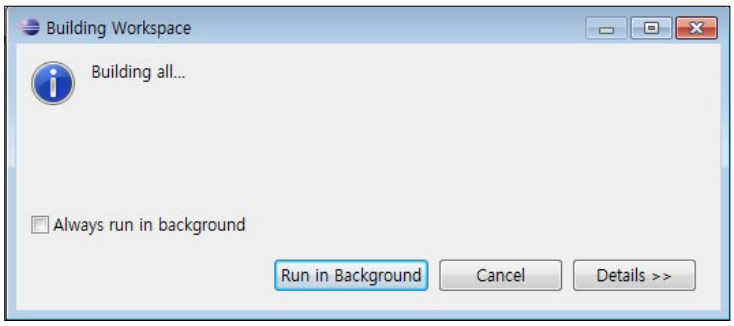
소스를 작성합니다.

11 빌드



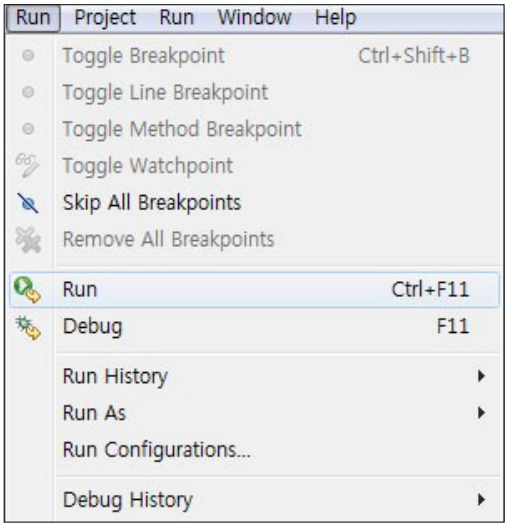
Projec 〉 Build Project 를 클릭합니다.

12 빌드중



빌드 팝업입니다. 그대로 두면 완료 후 사라집니다.

13 Run 실행



Run 〉 Run 을 클릭합니다. Ctrl+F11 단축키도 동일합니다.

14 실행화면

```
        // pucLed의 [0]부터 [15]까지 1을 집어 넣는다.
        // pucLed == g_ucMotorGreenLed이기 때문에 효과가 같다.
        for(i = 0;i < 16;i++){
            pucLed[i] = 1;
        }

        // pfPos의 [0]과 [3]에 90을 집어 넣는다.
        // pfPos == g_fMotorPos이기 때문에 효과가 같다.
        pfPos[0] = 90;
        pfPos[3] = 90;

        run(1000);
        delay(1000);

        // 프로그램 종료
        terminate();
        return 0;
    }
```

Problems | Tasks | Console | Properties

```
<terminated> hparr.exe [C/C++ Application] C:₩GccDongbu₩workspac
&pfPos[1] = 4206596
pfPos = 4206592
pfPos+1 = 4206596
```

15 로봇동작

포인터 변수를 선언하고 전역 변수 배열의 주소를 포인터 변수에 넣어서 주소를 서로 비교하고, 선언한 포인터 변수를 통해 로봇을 제어합니다.

5.5 포인터 함수

함수 포인터는 주소+자료형으로서 함수가 위치하는 주소값을 나타냅니다.
void 형 포인터는 주소값 저장이 제한되지 않는 포인터입니다. 예를 들면 Int* a 제한된 주소값, char* b 제한된 주소값을 가집니다. 하지만 vVoid* c 는 무엇이든지 담을 수 있는 상자와 같이 아무런 제한이 없으며, int char, 심지어 int 형 char 형 주소값을 모두 담을 수 있습니다.

아래 예제는 call_by_value, callby_reference 를 이해하기 위한 함수 호출 예제로서, 오른팔 든 상태로 call_by_value 실행 시 왼팔 들기를 의도했으나 그대로 오른팔을 들고, call_by_reference 실행 시 왼팔을 드는 예제입니다.

hpfunc.c

```
#include <stdio.h>
#include "hovis.h"

void motion_call_by_value(int nMode, float fRight, float fLeft)
{
        if (nMode == 0) {

                fRight = 90;

                fLeft = -90;

        }
        else {
                fRight = -90;

                fLeft = 90;
        }
        return;
}
void motion_call_by_reference(int nMode, float *pfRight, float *pfLeft)
{
        if (nMode == 0) {
                *pfRight = 90;
                *pfLeft = -90;
        else {
                *pfRight = -90;
        *pfLeft = 90;
}
```

```
        return;
}

int main()
{
        int nResult;

        int i;

        float afValue[2];
        nResult = initialize(L"COM2", 115200);
        if(nResult == 0){
                printf("Initialization Failed₩n");
                fflush(stdout);
                return 0;

}

        for(i=0;i<16;i++){
                g_fMotorPos[i]=0;

}
        g_fMotorPos[0] = -90;
        g_fMotorPos[1] = 90;
        g_fMotorPos[3] = -90;
        g_fMotorPos[4] = 90;
        run(1000);

        delay(1000);

        printf("Right hand up₩n");
        afValue[0] = 90;
        afValue[1] = -90;

        g_fMotorPos[0] = afValue[0];
        g_fMotorPos[3] = afValue[1];
        run(1000);
        dr_wait_delay(3000);
        // call by value
        printf("Left hand up : Call by value₩n");
        fflush(stdout);
        motion_call_by_value(1, afValue[0], afValue[1]);

        g_fMotorPos[0] = afValue[0];
        g_fMotorPos[3] = afValue[1];
        run(1000);
```

```
        dr_wait_delay(3000);

        // call by reference
        printf("Left hand up : Call by reference₩n");
        fflush(stdout);
        motion_call_by_reference(1, &afValue[0], &afValue[1]);

        g_fMotorPos[0] = afValue[0];
        g_fMotorPos[3] = afValue[1];
        run(1000);

        dr_wait_delay(3000);

        terminate();
        return 0;
}
```

위 코드 중에 진하게 처리된 call by value, referece 문장을 살펴봅니다.

// call by value

motion_call_by_value(1, afValue[0], afValue[1]); // 왼팔 들기 모션을 위한 모터 값을 받기 위해 함수를 호출한다.-->// 동작 – 왼팔 들기 모션(실제로는 오른팔을 든 채로 있다)

// call by reference

motion_call_by_reference(1, &afValue[0], &afValue[1]); // 왼팔 들기 모션을 위한 모터 값을 call by reference로 받아온다. --> 동작 – 왼팔 들기 모션(이제 실제로 왼팔을 든다)

문법요약

◆ 함수포인터

함수의 이름은 함수의 위치를 가리키는 포인터입니다.
함수이름의 포인터 타입을 결정짓는 요소는 리턴형과 전달인자입니다.

```
int (*fun)(int);  // int형 매개변수 1개 받습니다.
                  // fun은 포인터 이름입니다.
                  // 리턴형은 int형입니다.
```

◆ void 포인터

어떤 형의 주소값 이라도 저장할 수 있는 포인터 입니다.

예) char c='a';
int n=10;
void* vp;
vp = &c;
vp = &n;

포인터 연산이나 값을 변경, 참조하는 것 등을 할 수 없습니다.

◆ 포인터 함수

※ Call-By-Value와 Call-By-Reference

값에 의한 참조에 의한

※ Call-By-Value

- 값의 복사에 의한 함수의 호출입니다.
- 가장 일반적인 함수 호출 형태입니다.

int main(void)
add(val1, val2)
➔ int add(int a, int b)
add 라는 함수에서 main 함수의 들어가서 val1 을 조작할 수 없습니다.
왜냐하면, 복사한 것이기 때문에, add에서만 사용이 가능합니다.

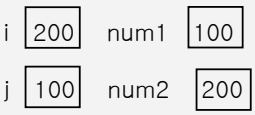
※ Call-By-Reference

- 참조(참조를 가능테 하는 주소값)를 인자로 전달하는 형태의 함수 호출입니다.
- 예를 들어 변수 val의 주소값이 Ox10 라면,adder(&val)는 val 의 주소값을 전달합니다.

. pVal =Ox10 입니다.
adder 라는 함수는 주소값을 알고 있고, (*pAvl)++; 가 가리키는 메모리 공간에 가서 1을 증가시켜라, 라는 의미입니다.

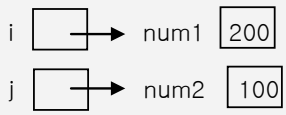
※ 값에 의한 호출 예제

```
#include <stdio.h>
void swap(int i, int j);

int main(void)
{
        int num1, num2;
        num1=100;
        num2=200;
        swap(num1, num2);
        printf(%d %d, num1, num2);
        fflush(stdout);

        return 0;
}

void swap(int i, int j)
{
        int temp;
        temp=i;  i=j;  j=temp;
}
```



※ 참조에 의한 호출 예제

```
#include <stdio.h>
void swap(int *i, int *j);

int main(void)
{
        int num1, num2;
        num1=100;
        num2=200;
        swap(&num1, &num2);
        printf(%d %d, num1, num2);
        fflush(stdout);
        return 0;
}

void swap(int *i, int *j)
{
        int temp;
        temp=*i; *i=*j; *j=temp;
}
```



◆ 함수의 매개변수

※ 함수 매개변수의 종류

- 실 매개변수

  호출 함수 쪽에서 전달하는 실제 변수입니다.

- 형식 매개 변수

  피호출함수 쪽에서 전달받는 변수입니다.

※ 함수 매개변수의 특징

- 실 매개변수와 형식매개변수의 개수는 일치합니다.
- 실 매개변수명과 형식매개변수의 이름은 같아도 상관없습니다.

※ 매개변수 전달

– 배열이 함수에 대한 인수로 사용될 때 배열의 주소만이 매개변수에게 전달 됩니다.

→ 매개변수가 포인터 형으로 선언되어야 합니다.

```
#include <stdio.h>
void a1(int num[5]), f2(int num[]), f3(int *num);

int main(void)
{
        int count[5]={1,2,3,4,5};
        a1(count);
        a2(count);
        a3(count);
        return 0;
}
```

→ 아래 세 가지 모든 경우 하나의 포인터 매개변수 사용

```
void a1(int num[5]) /*배열의 사용*/
{
        int i;
        for(i=0; i<5;i++)    printf("%d", num[i]);
                             fflush(stdout);
}
void a2(int num[])  /*언사이즈드 배열 사용*/
{
        int i;
        for(i=0; i<5;i++)    printf("%d", num[i]);
                             fflush(stdout);
}
void a3(int *num)  /*포인터를 사용*/
{
        int i;
        for(i=0; i<5;i++)   printf("%d", num[i]);
                            fflush(stdout);
}
```

◆ 재귀함수

※ 재귀함수의 기본적 이해

– 자기 자신을 다시 호출하는 형태의 함수입니다.
– 재귀함수 프로그램을 잘 이해하기 위해서는 컴퓨터 Process 를 잘 이해해야합니다.
– exe 파일 만들려면, 컴파일된 바이너리가 필요합니다.
– 프로그램이 시작되면 메모리 Code area 에 함수등 모든 것이 올라갑니다.
– 메인함수가 실행되면, 순차적으로 함수 진행됩니다. 이 코드를 가져다가 하나씩 패치를 시작하고, CPU 가 메인함수
코드를 가져다가 순차적으로 실행하는 것입니다.
– 리커시브 함수 호출 → CPU 가 리커시브 가져다가 놓고 실행합니다.
– 코드 영역에서 다시 리커시브 함수 호출하여, 가져다가 놓고 실행합니다. →계속 반복함
– 리커시브 함수는 종료되지 않고 계속 실행되고, 메모리에 계속 쌓입니다. → 문제가됨
(Stack overflow 라고 함 : 재귀적 함수가 계속쌓여서 꽉 차는 것)

※ 탈출 조건의 필요성

무한 재귀 호출을 피하기 위함입니다.
재귀함수를 공부하는 이유는, 자료구조(데이터를 표현하는 방식) 나 알고리즘(문제해결)을 공부하게 되면, 재귀함수가 매우 유용합니다. 문제해결을 고민하는데 있어서, 재귀적으로 해결하는게 너무나 많습니다.

```
if(n==1)
return;   → 1, 함수를 빠져나오는 것, 2, 값을 반환하는 것, 두가지 기능이 있습니다.
      (리턴타입이 void 라고 해도, 함수를 빠져나오는 용도로 쓰입니다.)
recursive(n-1);
```

재귀함수 할 때는 중요한 것은 탈출 조건을 달아뒀냐가 매우 중요합니다.

※ 재귀(recursive) 호출 : 순환(recursion)

: 어떤 프로시저(procedure)가 자기 자신을 호출 하는 것입니다.

```c
#include <stdio.h>

int facto1(int n);

int main(void)
{
        printf("5 factorial : %d", facto1(5));
        return 0;
}

int facto1(int n)
{
        if(n==0)
                return 1;
        else
                return n * facto1(n-1);
}
```

```c
#include <stdio.h>

int facto2(int n);

int main(void)
{
        printf("5 factorial : %d", facto2(5));
        return 0;
}

int facto2(int n)
{
        int i, result;
        result=1;
        if(n==0)
                return 0;
        for(i=n; i>0; i--)
                result=result *i;
        return result;
}
```

코딩계획

```
// 모션 번호에 해당하는 관절값 저장 변수
//call by value
//call by reference

//결과값, for, 모터값 받아올  저장 변수
//초기화
//모터사용
//call by value
//call by reference
```

프로그래밍 세부설계

```
/*
파일명              : hpfunc.c
제어로봇 형태       : 16dof 휴머노이드
설명                : call_by_value, callby_reference 를 이해하기 위한 함수 호출 예제,
오른팔 든 상태로 call_by_value 실행시 왼팔 들기를 의도했으나 그대로 오른팔을 들고,
call_by_reference 실행 시 왼팔을 드는 예제
*/

// 동부로봇의 api 를 사용하기 위해 반드시 이렇게 선언 해 두어야 한다.

// 모션 번호에 해당하는 관절 값을 fRight, fLeft에 저장하는 함수.
// nMode가 0번일 땐 오른팔 드는 모션, 1번일 땐 왼팔 드는 모션
// 그러나 call by value 이기 때문에 main 함수에서는 값이 바뀌지 않는다.

        // 오른팔을 드는 모션 일때
                // 오른팔을 드는 자세
                // 왼팔을 내리는 자세

        // 왼팔을 드는 모션 일때
                // 오른팔을 내리는 자세
                // 왼팔을 드는 자세

        // 함수 종료

// 모션 번호에 해당하는 관절 값을 pfRight, pfLeft에 저장하는 함수.
// nMode가 0번일 땐 오른팔 드는 모션, 1번일 땐 왼팔 드는 모션
// call by reference 이기 때문에 main 함수에서도 값이 바뀐다.

        // 오른팔을 드는 모션 일때
                // 오른팔을 드는 자세
                // 왼팔을 내리는 자세

                // 왼팔을 드는 모션 일때
                // 오른팔을 내리는 자세
                // 왼팔을 드는 자세

        // 함수 종료

int main()
{
        // 결과값을 리턴받을 변수
        // for 문을 사용하기 위해 선언한 변수
        // 모터 값을 받아 올 변수
        // 초기화 – 로봇제어를 하기 위해 반드시 앞 부분에 선언한다.
        // 각 변수들을 초기화 한다.
```

```
// 열고 난 이후 이상이 있는 지 확인한다.
        // 이상이 있다면 에러 출력
        // 프로그램 종료

// 이제부터 모터를 사용 할 수 있다.
// 똑바로 일어선다.          (차렷자세)

        // 모든 모터(16개 모터)를 0 위치로 설정함

// 0 번 모터(오른쪽 어깨)
// 1 번 모터(오른쪽 윗팔)
// 3 번 모터(왼쪽 어깨)
// 4 번 모터(왼쪽 윗팔)
// 모터 동작(해당 자세를 1000ms 동안 동작)
// 동작 대기(1000ms)

// 오른팔 들기
// 메시지 출력

// 동작 – 오른팔 들기 모션
// 0 번 모터(오른쪽 어깨)에 오른팔 모터 값 대입
// 3 번 모터(왼쪽 어깨)에 왼팔 모터 값 대입
// 모터 동작(해당 자세를 1000ms 동안 동작)
// 3초간 대기한다.

// call by value
// 메시지 출력
// 왼팔 들기 모션을 위한 모터 값을 받기 위해 함수를 호출한다.

// 동작 – 왼팔 들기          모션(실제로는 오른팔을 든채로 있다)
// 0 번 모터(오른쪽 어깨)에 오른팔 모터 값 대입
// 3 번 모터(왼쪽 어깨)에 왼팔 모터 값 대입
// 모터 동작(해당 자세를 1000ms 동안 동작)
// 3초간 대기한다.

// call by reference
// 메시지 출력
// 왼팔 들기 모션을 위한 모터 값을 call by reference로 받아온다.
// 동작 – 왼팔 들기          모션(이제 실제로 왼팔을 든다)
// 0 번 모터(오른쪽 어깨)에 오른팔 모터 값 대입
// 3 번 모터(왼쪽 어깨)에 왼팔 모터 값 대입
// 모터 동작(해당 자세를 1000ms 동안 동작)
// 3초간 대기한다.

// 프로그램 종료
// 제어 종료 함수를 실행한다.
```

프로그래밍 작성

```
/*
파일 명              : hpfunc.c
제어로봇 형태        : 16dof 휴머노이드
설명                 : call_by_value, callby_reference 를 이해하기 위한 함수 호출 예제, 오
른팔 든 상태로 call_by_value 실행 시 왼팔 들기를 의도 했으나 그대로 오른팔을 들고,
call_by_reference 실행 시 왼팔을 드는 예제
*/

#include 〈stdio.h〉
#include "hovis.h"  // 동부로봇의 api 를 사용하기 위해 반드시 이렇게 선언 해 두어야 한다.

// 모션 번호에 해당하는 관절 값을 fRight, fLeft에 저장하는 함수.
// nMode가 0번일 땐 오른팔 드는 모션, 1번일 땐 왼팔 드는 모션
// 그러나 call by value 이기 때문에 main 함수에서는 값이 바뀌지 않는다.
void motion_call_by_value(int nMode, float fRight, float fLeft)
{
        if (nMode == 0) {                   // 오른팔을 드는 모션일 때
                fRight = 90;                        // 오른팔을 드는 자세
                fLeft = -90;                        // 왼팔을 내리는 자세
        }
        else {                                      // 왼팔을 드는 모션일 때
                fRight = -90;                       // 오른팔을 내리는 자세
                fLeft = 90;                         // 왼팔을 드는 자세
        }
        return;                                     // 함수 종료
}

// 모션 번호에 해당하는 관절 값을 pfRight, pfLeft에 저장하는 함수.
// nMode가 0번일 땐 오른팔 드는 모션, 1번일 땐 왼팔 드는 모션
// call by reference 이기 때문에 main 함수에서도 값이 바뀐다.
void motion_call_by_reference(int nMode, float *pfRight, float *pfLeft)
{
        if (nMode == 0) {                   // 오른팔을 드는 모션일 때
                *pfRight = 90;                      // 오른팔을 드는 자세
                *pfLeft = -90;                      // 왼팔을 내리는 자세
        }
```

```
        else {                                  // 왼팔을 드는 모션일 때
                *pfRight = -90;    // 오른팔을 내리는 자세
                *pfLeft = 90;                   // 왼팔을 드는 자세
        }
        return;                                 // 함수 종료
}

int main()
{
        int nResult;                            // 결과값을 리턴받을 변수
        int i;                          // for 문을 사용하기 위해 선언한 변수
        float afValue[2];               // 모터 값을 받아 올 변수

        // 초기화 - 로봇제어를 하기 위해 반드시 앞 부분에 선언한다.
        nResult = initialize(L"COM2", 115200);          // 각 변수들을 초기화 한다.
        if(nResult == 0){   // 열고 난 이후 이상이 있는 지 확인한다.
                printf("Initialization Failed₩n");      // 이상이 있다면 에러 출력
                fflush(stdout);
                return 0;               // 프로그램 종료
        }

        // 이제부터 모터를 사용 할 수 있다.
        // 똑바로 일어선다.             (차렷자세)
        for(i=0;i<16;i++){
                g_fMotorPos[i]=0;
        // 모든 모터(16개 모터)를 0 위치로 설정함
        }
        g_fMotorPos[0] = -90;                           // 0 번 모터(오른쪽 어깨)
        g_fMotorPos[1] = 90;                            // 1 번 모터(오른쪽 윗팔)
        g_fMotorPos[3] = -90;                           // 3 번 모터(왼쪽 어깨)
        g_fMotorPos[4] = 90;                            // 4 번 모터(왼쪽 윗팔)
        run(1000);          // 모터 동작(해당 자세를 1000ms 동안 동작)
        delay(1000);                                    // 동작 대기(1000ms)

        // 오른팔 들기
        printf("Right hand up₩n");              // 메시지 출력
        fflush(stdout);
        afValue[0] = 90;
        afValue[1] = -90;               // 동작 - 오른팔 들기 모션
```

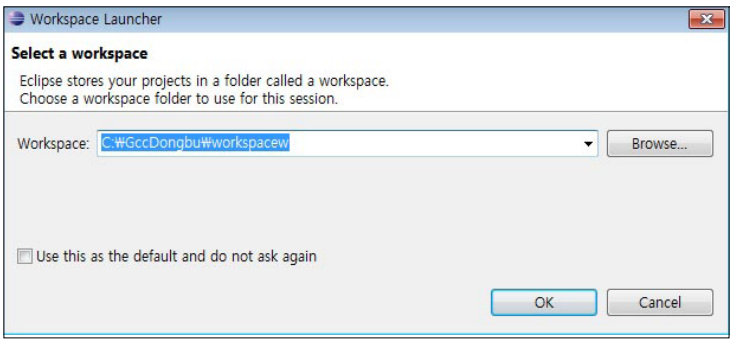
```
        g_fMotorPos[0] = afValue[0];
// 0 번 모터(오른쪽 어깨)에 오른팔 모터 값 대입
        g_fMotorPos[3] = afValue[1];
// 3 번 모터(왼쪽 어깨)에 왼팔 모터 값 대입
        run(1000);          // 모터 동작(해당 자세를 1000ms 동안 동작)
        dr_wait_delay(3000);                    // 3초간 대기한다.

        // call by value
        printf("Left hand up : Call by value₩n");       // 메시지 출력
        fflush(stdout);
        motion_call_by_value(1, afValue[0], afValue[1]);
// 왼팔 들기 모션을 위한 모터 값을 받기 위해 함수를 호출한다.

        // 동작 – 왼팔 들기          모션(실제로는 오른팔을 든채로 있다)
        g_fMotorPos[0] = afValue[0];
// 0 번 모터(오른쪽 어깨)에 오른팔 모터 값 대입
        g_fMotorPos[3] = afValue[1];
// 3 번 모터(왼쪽 어깨)에 왼팔 모터 값 대입
        run(1000);          // 모터 동작(해당 자세를 1000ms 동안 동작)
        dr_wait_delay(3000);                    // 3초간 대기한다.

        // call by reference
        printf("Left hand up : Call by reference₩n");// 메시지 출력
        fflush(stdout);
        motion_call_by_reference(1, &afValue[0], &afValue[1]);
// 왼팔 들기 모션을 위한 모터 값을 call by reference로 받아온다.
        // 동작 – 왼팔 들기          모션(이제 실제로 왼팔을 든다)
        g_fMotorPos[0] = afValue[0];
        // 0 번 모터(오른쪽 어깨)에 오른팔 모터 값 대입
        g_fMotorPos[3] = afValue[1];
        // 3 번 모터(왼쪽 어깨)에 왼팔 모터 값 대입
        run(1000);          // 모터 동작(해당 자세를 1000ms 동안 동작)
        dr_wait_delay(3000);                    // 3초간 대기한다.

        // 프로그램 종료
        terminate();                    // 제어 종료 함수를 실행한다.
        return 0;
}
```

빌드 및 실행

01 이클립스 실행

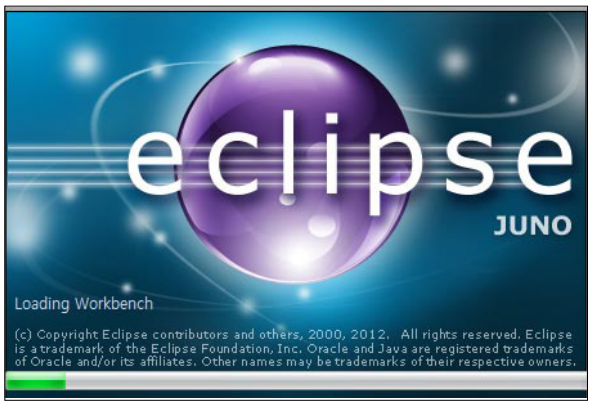
이클립스 실행 버튼을 누릅니다.

02 workspace



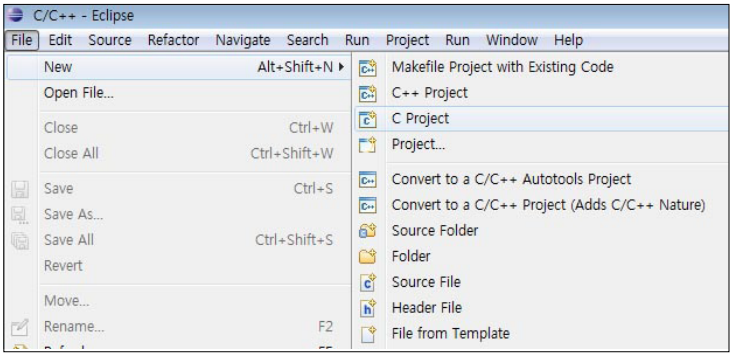
작업하고자 하는 폴더를 지정합니다. Browse 를 클릭하여 폴더를 지정하고 OK 버튼을 누릅니다.

03 이클립스 로딩 화면



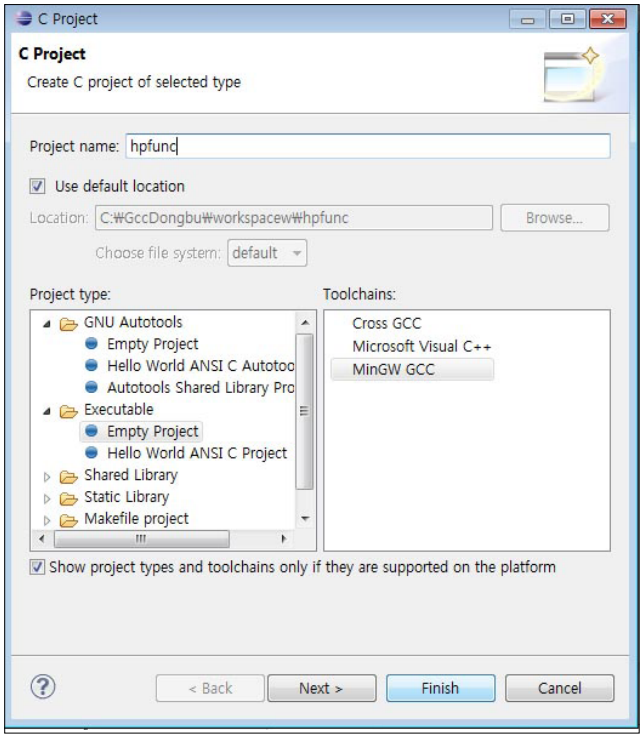
이클립스 로딩 화면입니다.

04 프로젝트 생성



이클립스 편집창 메뉴에서 File 〉 New 〉 C project 를 클릭합니다.

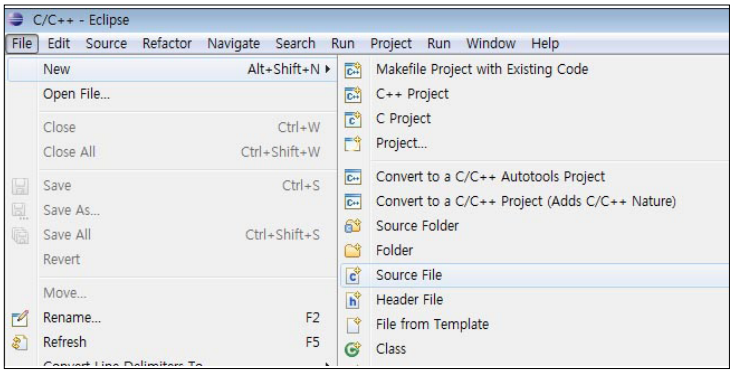
05 프로젝트 이름



Projec Name 을 hpfunc 라고 입력하고, Project type 에서 Exectable 에서 Empty Project 를 선택합니다.

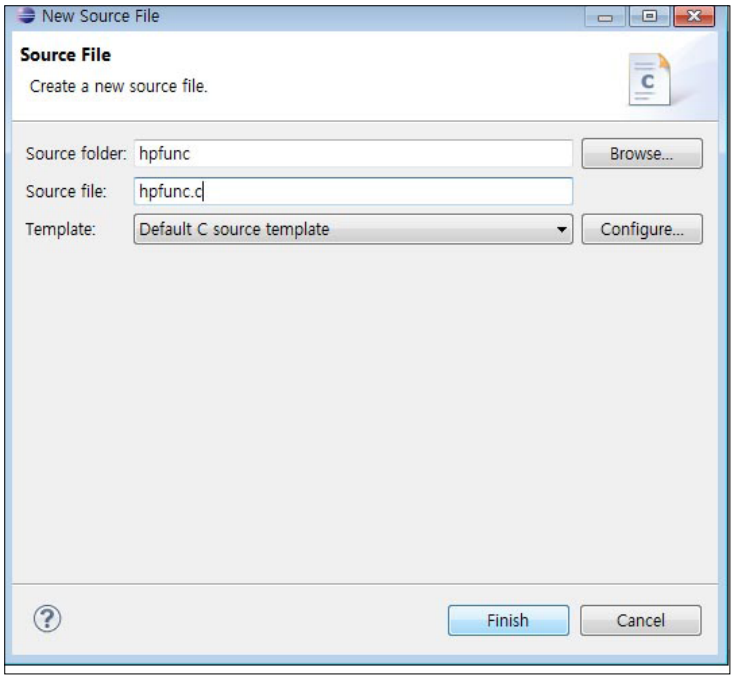
Toolchians 에서 MinGW GCC 를 선택하고 Finish 버튼을 누릅니다.

06 소스파일 추가



File 〉 New 〉 Source File 을 클릭합니다.

07 소스파일 이름

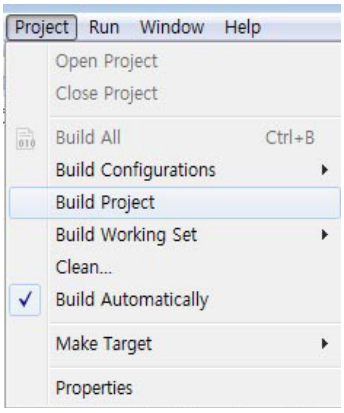


hpfunc.c 라고 입력하고 Finish 버튼을 누릅니다.

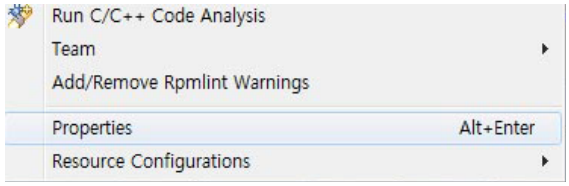
08 기본 제공 파일 추가

본 교재에게 C 프로그래밍을 하기 위해선 기본 제공되는 라이브러리 파일을 먼저 추가해주어야 올바른 컴파일을 할 수 있습니다. 필요한 파일은 drApi.dll, drApi.h, hovis.c, hovis.h 등 총 4가지 입니다.

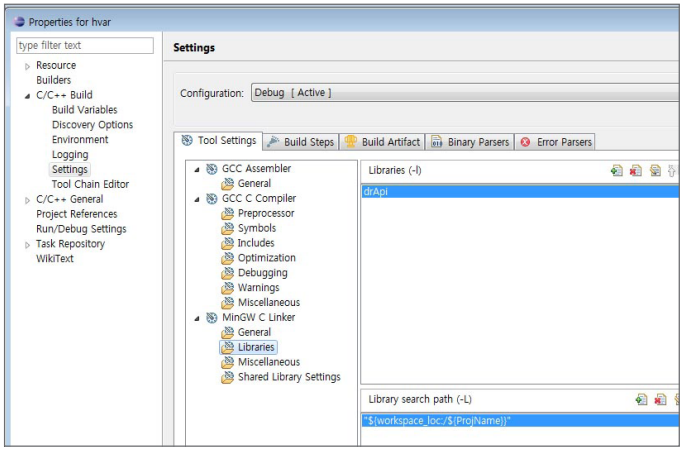
파일 추가 전에 먼저 Property 에서 라이브러리를 설정해주어야 합니다. 파일 생성을 한 후에 아무 코딩도 하지 말고, 바로 Build project 를 실행하세요. 그래야 라이브러리 추가가 가능합니다.



Build Project 를 클릭합니다.



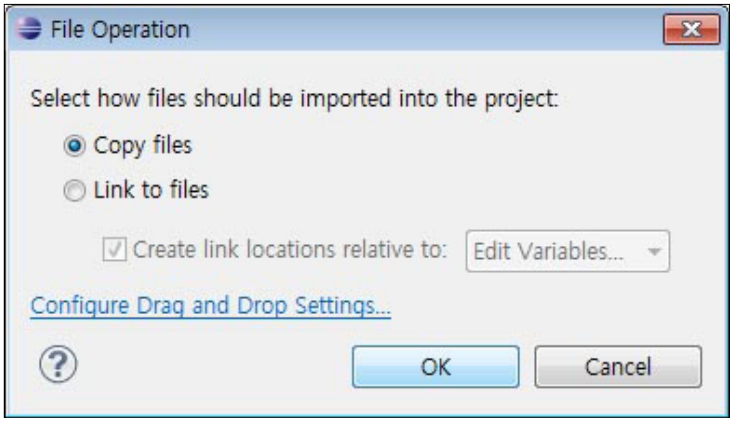
좌측 프로젝트 이름에 커서를 놓고 오른쪽 마우스키를 누릅니다.
제일 하단에 Propertis 를 클릭합니다. Alt+Enter 도 동일합니다.



C/C++ Build 〉 Settings 를 클릭하고, 우측에 MinGW C Linker 〉 Libraries 를 클릭합니다. 우측에 Libraies 에 + 마크를 클릭하고 팝업창이 뜨면 drApi 라고 입력합니다. Library search path 에 + 마크를 클릭하고 팝업창이 뜨면 Workspace 를 클릭하고, Folder selection 에서 프로젝트 이름과 동일한 것을 클릭하고 OK 버튼을 누릅니다.

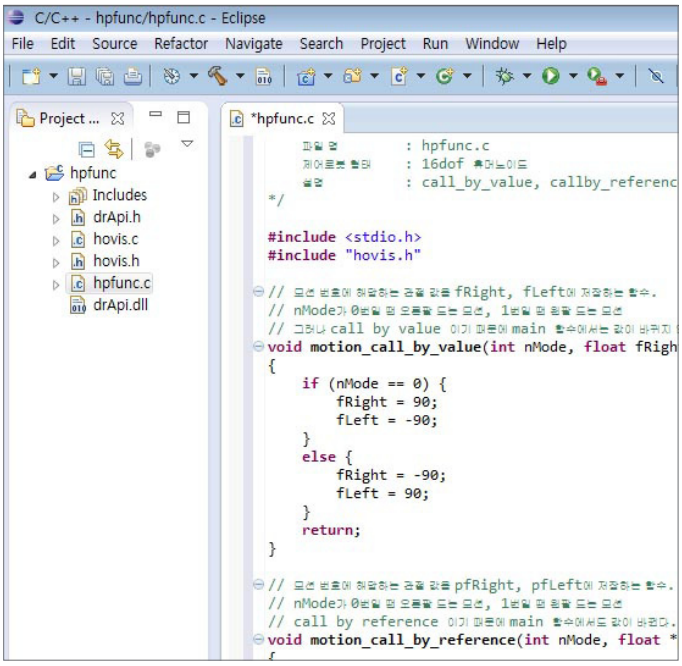
로봇을 동작시키기 위해선 위 네 개 파일을 프로젝트에 포함시켜야 합니다.
위 파일이 있는 파일탐색기 폴더에서 드래그 하는 방식으로 프로젝트에 포함시킵니다.
파일을 끌어다가 드래그해서 왼쪽 프로젝트에 추가합니다.

09 기본 파일 Import



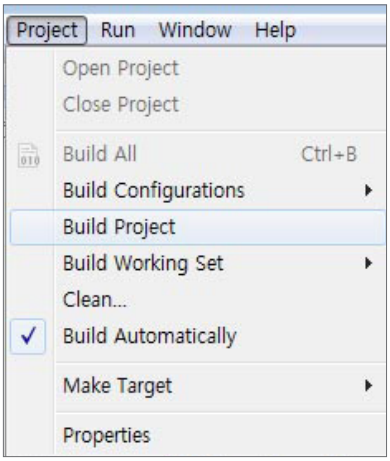
OK 버튼을 누릅니다.

10 소스 작성



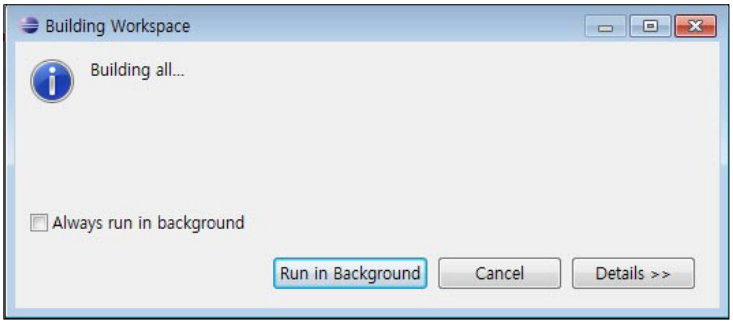
소스를 작성합니다.

11 빌드



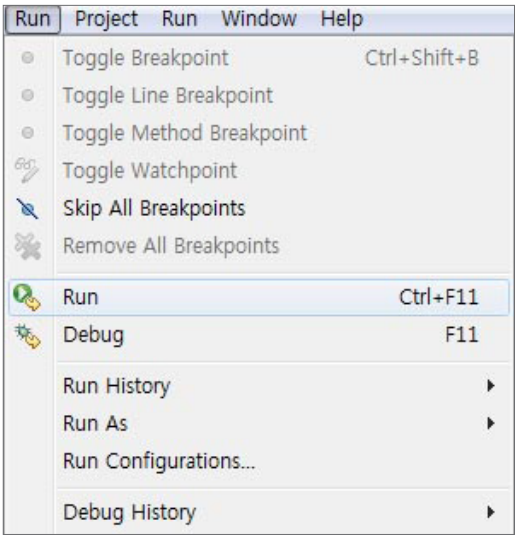
Projec 〉 Build Project 를 클릭합니다.

12 빌드중



빌드 팝업입니다. 그대로 두면 완료 후 사라집니다.

13 Run 실행



Run 〉 Run 을 클릭합니다. Ctrl+F11 단축키도 동일합니다.

14 실행화면

```
    // call by reference
    printf("Left hand up : Call by reference\n");
    fflush(stdout);
    motion_call_by_reference(1, &afValue[0], &afValue[1]);

    // 동작 - 왼팔 듦기   모션(이제 실제로 왼팔을 듭니다)
    g_fMotorPos[0] = afValue[0];
    g_fMotorPos[3] = afValue[1];
    run(1000);
    dr_wait_delay(3000);

    // 프로그램 종료
    terminate();                                        //
    return 0;
}
```

Problems Tasks Console Properties

```
<terminated> hpfunc.exe [C/C++ Application] C:₩GccDongbu₩workspacew₩hpf
Right hand up
Left hand up : Call by value
Left hand up : Call by reference
```

15 로봇동작

오른팔 든 상태로 call_by_value 실행 시 왼팔 들기를 의도했으나 그대로 오른팔을 들고, call_by_reference 실행 시 왼팔을 듭니다.

06 구조체정의

6.1 구조체정의

구조체는 하나(보통 둘) 이상의 기본 자료형을 기반으로 사용자 정의 자료형을 만들 수 있는 문법 요소를 가리킵니다.

```
struct point   // point 라는 이름의 구조체 선언
{
lnt x;       // 구조체 멤버 int x
lnt y;        // 구조체 멤버 int y.
};
```

아래 예제는 왼쪽 어깨 모터와 오른쪽 어깨 모터를 한 개씩 정의해서 묶어놓고 입력값에 따라 각도 조절하는 프로그램입니다.

hstruct.c

```
#include <stdio.h>
#include <stdlib.h>

#include "hovis.h"

struct SMotor_t {a
        float fLeftShoulder;
        float fRightShoulder;
};

int main()
{
        int nResult;

        int i;
        struct SMotor_t SMotor;

        nResult = initialize(L"COM2", 115200);
        if(nResult == 0){
                printf("Initialization Failed₩n");
                fflush(stdout);
                return 0;
        }
```

```
        for(i=0;i<16;i++){
                g_fMotorPos[i]=0;

}
        g_fMotorPos[0] = -90;
        g_fMotorPos[1] = 90;
        g_fMotorPos[3] = -90;
        g_fMotorPos[4] = 90;
        run(1000);

        delay(1000);

        while(1){
                printf("Input Left Shoulder Angle₩n");
                fflush(stdout);
                scanf("%f", &SMotor.fLeftShoulder);
                printf("Input Right Shoulder Angle₩n");
                fflush(stdout);
                scanf("%f", &SMotor.fRightShoulder);
                g_fMotorPos[3] = SMotor.fLeftShoulder;
                g_fMotorPos[0] = SMotor.fRightShoulder;
                run(1000);

                delay(1000);

        }

        terminate();

        return 0;
}
```

위 코드 중에 진하게 처리된 struct 구조문을 살펴봅니다.

```
struct SMotor_t {
        float fLeftShoulder;
        float fRightShoulder;
};
struct SMotor_t SMotor; // 왼쪽, 오른쪽 어깨 각도가 저장될 구조체 변수
scanf("%f", &SMotor.fLeftShoulder);  // 왼쪽 어깨 각도 값 입력 받기
```

문법요약

◆ 구조체의 정의

– 하나(보통 둘) 이상의 기본 자료형을 기반으로 사용자 정의 자료형을 만들 수 있는 문법 요소입니다.

```
struct point   // point 라는 이름의 구조체 선언
{
Int x;       // 구조체 멤버 int x
Int y;        // 구조체 멤버 int y.
};
```

구조체 선언과 변수 선언을 동시에!!

```
struct 구조체명 {
        구조체 멤버선언;
        구조체 멤버선언;
              :
} 구조체 변수명1, 구조체변수명2…;
```

구조체 맴버

구조체 선언과 변수 선언을 따로!!

```
struct 구조체명 {
        구조체 멤버선언;
        구조체 멤버선언;
              :
};
struct 구조체명 구조체변수명1, 구조체변수명2…;
```

◆ 구조체의 특징

구조체 변수는 함수의 전달인자로 사용가능합니다.
구조체는 기본자료형과 같이 배열을 선언하여 사용할 수 있습니다.

```
예) struct phone list[10];  // 구조체 배열의 선언
    list[3].age; // list배열의 네번째배열요소의
                 // age멤버 참조
```

◆ 구조체 포인터

• 구조체는 마치 일반변수와 같이 주소 연산이 가능하고 구조체 변수 전체를 가리키는 포인터를 가질 수 있다.

```
예) 구조체 포인터 변수의 선언과 초기화
    struct score *sp = &a;

예) 구조체 포인터를 이용한 멤버 참조
    (*sp).kor;
```

◆ 간접 멤버 참조 연산자

• 간접 멤버 참조 연산자 : (–〉)

```
예) 간접 멤버 참조 연산자 사용법
    (*sp).kor  == sp–〉kor

예) lp[i].name          // 배열표현
    == (*(lp+i)).name;      // 포인터표현
    == (lp+i)–〉name;      //  간접멤버 참조 연산자를 이용한 표현
```

◆ 자기참조 구조체

● 개념

구조체 멤버 중에서 자기 자신을 가리키는 포인터를 가지고 있는 구조체

```
예) struct list{
            int I;
            struct list *ptr;
    } s1, s2, s3;

    s1.prt=&s2;
    s2.prt=&s3;
    s3.prt=&s1;
```

◆ 공용체

구조체 선언과 변수 선언을 동시에!!

```
union 공용체명 {
        공용체 멤버선언;
        공용체 멤버선언;
              :
} 공용체 변수명1, 공용체변수명2…;
```

구조체 선언과 변수 선언을 따로!!

```
union 공용체명 {
        공용체 멤버선언;
        공용체 멤버선언;
              :
};
union 공용체명 공용체변수명1, 공용체변수명2…;
```

typedef

구조체, 공용체, 열거형과 같은 응용자료형을 사용할 때 길어지는 선언문을 간결하게 작성할 수 있도록 자료형의 이름을 재정의한다.

```
예) struct student{
      int num;
      double grade;
    };
    typeof struct student Student;
```

열거형

- 개념
  - 기억공간에 저장될 데이터의 집합을 정의한다.
  - 열거형 멤버들은 정수형 상수로 취급 당한다.
  - 프로그램 가독성을 높일 수 있다.
  - 특정 정수값에 의미를 부여할 수 있다.

```
예)  enum color {RED=1, GREEN=3, BLUE=5};
       enum color {sun=5, mon, tue, wed=10}
```

코딩계획

```
//결과값, for, 왼쪽 오른쪽 어깨각도 구조체 저장 변수
//초기화
//모터사용
//어깨각도 조절
```

프로그래밍 세부설계

```
/*
파일명              : hstruct.c
제어로봇 형태       : 16dof 휴머노이드
설명                : 왼쪽 어깨 모터와 오른쪽 어깨 모터를 한 개 씩 정의해서 묶어놓고 입
력값에 따라 각도 조절
*/

// malloc, free를 사용하기 위해서는 선언 해 두어야 한다.
// 동부로봇의 api 를 사용하기 위해 반드시 이렇게 선언 해 두어야 한다.

int main()
{
        // 결과값을 리턴받을 변수
        // for 문을 사용하기 위해 선언한 변수
        // 왼쪽, 오른쪽 어깨 각도가 저장될 구조체 변수

        // 초기화 – 로봇제어를 하기 위해 반드시 앞 부분에 선언한다.
        // 각 변수들을 초기화 한다.
        // 열고 난 이후 이상이 있는 지 확인한다.
        // 이상이 있다면 에러 출력
        // 프로그램 종료

        // 이제부터 모터를 사용 할 수 있다.
        // 똑바로 일어선다.          (차렷자세)

                // 모든 모터(16개 모터)를 0 위치로 설정함
        // 0 번 모터(오른쪽 어깨)
        // 1 번 모터(오른쪽 윗팔)
        // 3 번 모터(왼쪽 어깨)
        // 4 번 모터(왼쪽 윗팔)
        // 모터 동작(해당 자세를 1000ms 동안 동작)
        // 동작 대기(1000ms)
```

```
// 무한 반복한다.
        // 왼쪽 어깨 각도 값을 입력하라는 메시지 출력
        // 왼쪽 어깨 각도 값 입력 받기
        // 오른쪽 어깨 각도 값을 입력하라는 메시지 출력
        // 오른쪽 어깨 각도 값 입력 받기

        // 3 번 모터(왼쪽 어깨)에 입력 받은 각도 값을 넣는다.
        // 0 번 모터(오른쪽 어깨)에 입력 받은 각도 값을 넣는다.
        // 모터 동작(해당 자세를 1000ms 동안 동작)
        // 동작 대기(1000ms)

// 프로그램 종료
// 제어 종료 함수를 실행한다.
```

프로그래밍 작성

```
/*
파일명              : hstruct.c
제어로봇 형태       : 16dof 휴머노이드
설명                : 왼쪽 어깨 모터와 오른쪽 어깨 모터를 한 개 씩 정의해서 묶어놓고 입
력값에 따라 각도 조절
*/

#include <stdio.h>
#include <stdlib.h>   // malloc, free를 사용하기 위해서는 선언 해 두어야 한다.
#include "hovis.h"    // 동부로봇의 api 를 사용하기위해 반드시 이렇게 선언 해 두어야 한다.

struct SMotor_t {
        float fLeftShoulder;
        float fRightShoulder;
};

int main()
{
        int nResult;                    // 결과값을 리턴받을 변수
        int i;                          // for 문을 사용하기 위해 선언한 변수
        struct SMotor_t SMotor;
                        // 왼쪽, 오른쪽 어깨 각도가 저장될 구조체 변수

        // 초기화 – 로봇제어를 하기위해 반드시 앞 부분에 선언한다.
        nResult = initialize(L"COM2", 115200);// 각 변수들을 초기화함
        if(nResult == 0){   // 열고 난 이후 이상이 있는 지 확인한다.
                printf("Initialization Failed₩n");
                 fflush(stdout);
                        // 이상이 있다면 에러 출력
                return 0; // 프로그램 종료
        }

        // 이제부터 모터를 사용 할 수 있다.
        // 똑바로 일어선다.          (차렷자세)
        for(i=0;i<16;i++){
                g_fMotorPos[i]=0;
        // 모든 모터(16개 모터)를 0 위치로 설정함
        }
```

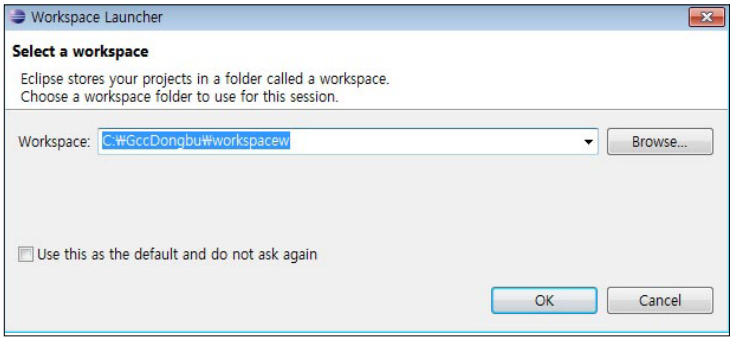
```
        g_fMotorPos[0] = –90;                    // 0 번 모터(오른쪽 어깨)
        g_fMotorPos[1] = 90;                     // 1 번 모터(오른쪽 윗팔)
        g_fMotorPos[3] = –90;                    // 3 번 모터(왼쪽 어깨)
        g_fMotorPos[4] = 90;                     // 4 번 모터(왼쪽 윗팔)
        run(1000);          // 모터 동작(해당 자세를 1000ms 동안 동작)
        delay(1000);                             // 동작 대기(1000ms)

        while(1){                                // 무한 반복한다.
                printf("Input Left Shoulder Angle₩n");
                 fflush(stdout);
// 왼쪽 어깨 각도 값을 입력하라는 메시지 출력
                scanf("%f", &SMotor.fLeftShoulder);
// 왼쪽 어깨 각도 값 입력 받기
                printf("Input Right Shoulder Angle₩n");
// 오른쪽 어깨 각도 값을 입력하라는 메시지 출력
                scanf("%f", &SMotor.fRightShoulder);
// 오른쪽 어깨 각도 값 입력 받기
                g_fMotorPos[3] = SMotor.fLeftShoulder;
// 3 번 모터(왼쪽 어깨)에 입력 받은 각도 값을 넣는다.
                g_fMotorPos[0] = SMotor.fRightShoulder;
// 0 번 모터(오른쪽 어깨)에 입력 받은 각도 값을 넣는다.
                run(1000);  // 모터 동작(해당 자세를 1000ms 동안 동작)
                delay(1000);                     // 동작 대기(1000ms)
        }

        // 프로그램 종료
        terminate();                             // 제어 종료 함수를 실행한다.
        return 0;
}
```

빌드 및 실행

01 이클립스 실행

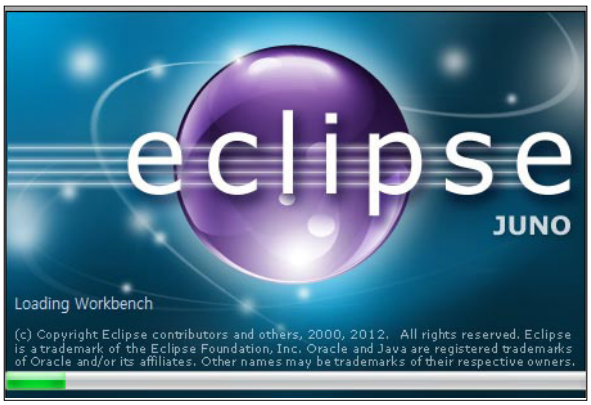
이클립스 실행 버튼을 누릅니다.

02 workspace



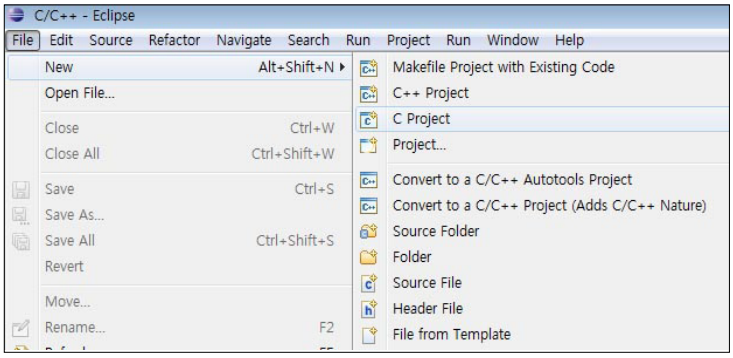
작업하고자 하는 폴더를 지정합니다. Browse 를 클릭하여 폴더를 지정하고 OK 버튼을 누릅니다.

03 이클립스 로딩 화면



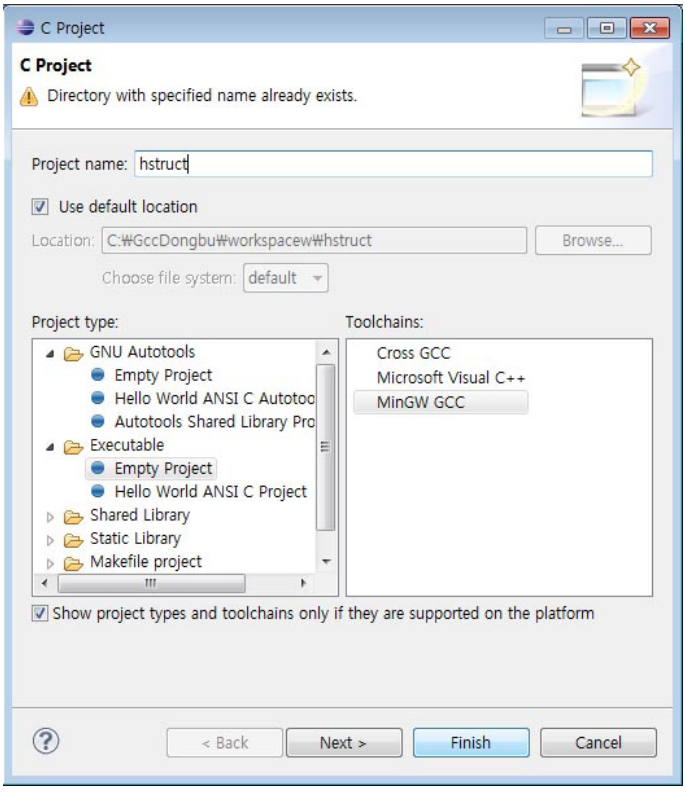
이클립스 로딩 화면입니다.

04 프로젝트 생성



이클립스 편집창 메뉴에서 File 〉 New 〉 C project 를 클릭합니다.

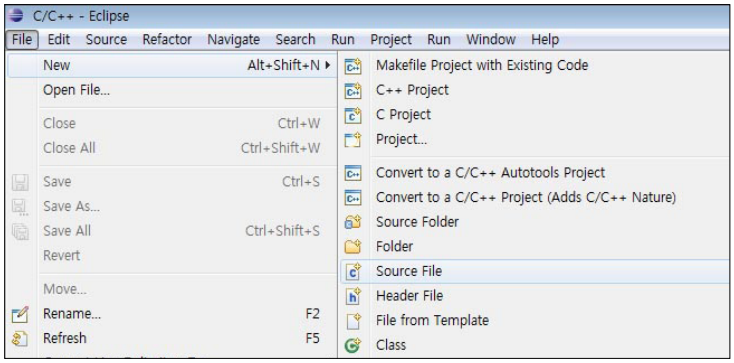
05 프로젝트 이름



Projec Name 을 hstruct 라고 입력하고, Project type 에서 Exectable 에서 Empty Project 를 선택합니다.

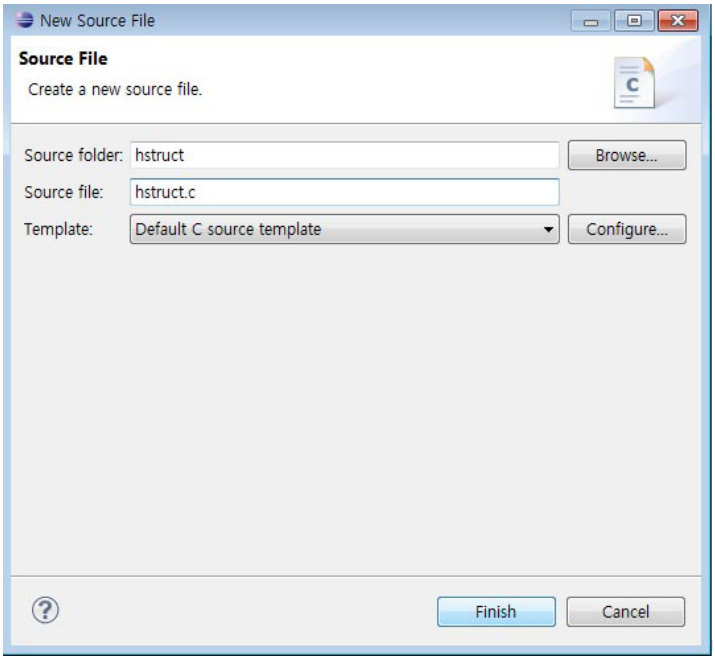
Toolchians 에서 MinGW GCC 를 선택하고 Finish 버튼을 누릅니다.

06 소스파일 추가



File 〉 New 〉 Source File 을 클릭합니다.

07 소스파일 이름

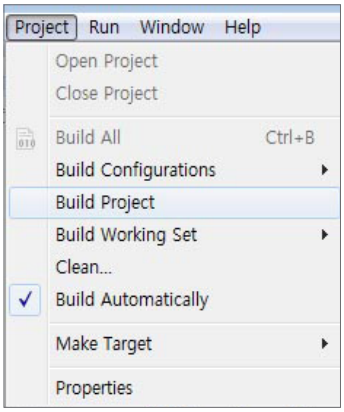


hstruct.c 라고 입력하고 Finish 버튼을 누릅니다.

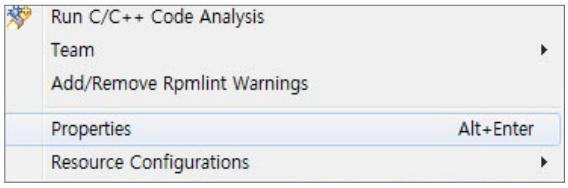
08 기본 제공 파일 추가

본 교재에게 C 프로그래밍을 하기 위해선 기본 제공되는 라이브러리 파일을 먼저 추가해주어야 올바른 컴파일을 할 수 있습니다. 필요한 파일은 drApi.dll, drApi.h, hovis.c, hovis.h 등 총 4가지 입니다.

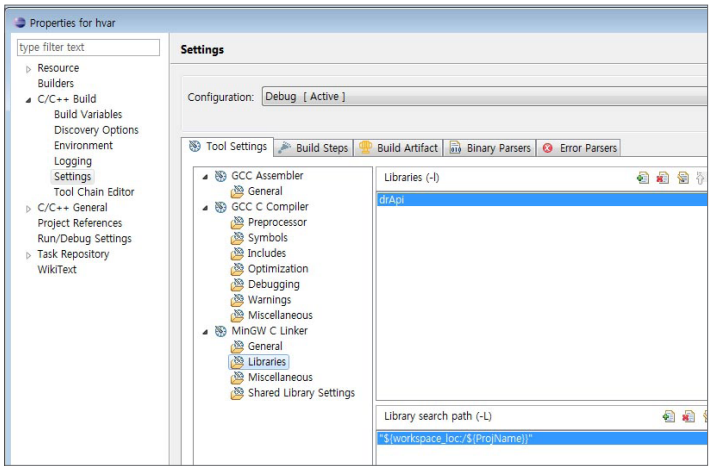
파일 추가 전에 먼저 Property 에서 라이브러리를 설정해주어야 합니다. 파일 생성을 한 후에 아무 코딩도 하지 말고, 바로 Build project 를 실행하세요. 그래야 라이브러리 추가가 가능합니다.



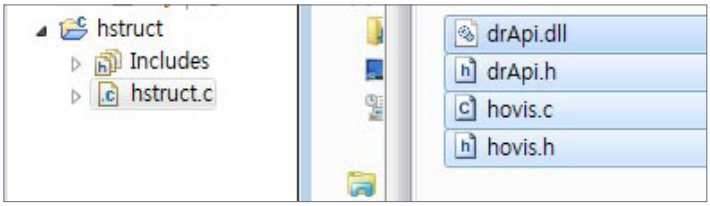
Build Project 를 클릭합니다.



좌측 프로젝트 이름에 커서를 놓고 오른쪽 마우스키를 누릅니다.
제일 하단에 Propertis 를 클릭합니다. Alt+Enter 도 동일합니다.

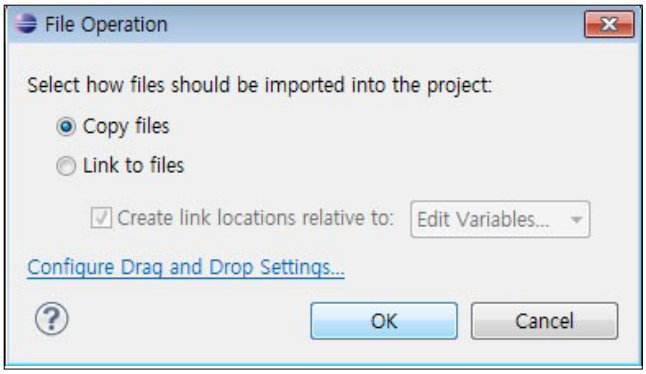


C/C++ Build 〉 Settings 를 클릭하고, 우측에 MinGW C Linker 〉 Libraties 를 클릭합니다. 우측에 Libraies 에 + 마크를 클릭하고 팝업창이 뜨면 drApi 라고 입력합니다. Library search path 에 + 마크를 클릭하고 팝업창이 뜨면 Workspace 를 클릭하고, Folder selection 에서 프로젝트 이름과 동일한 것을 클릭하고 OK 버튼을 누릅니다.



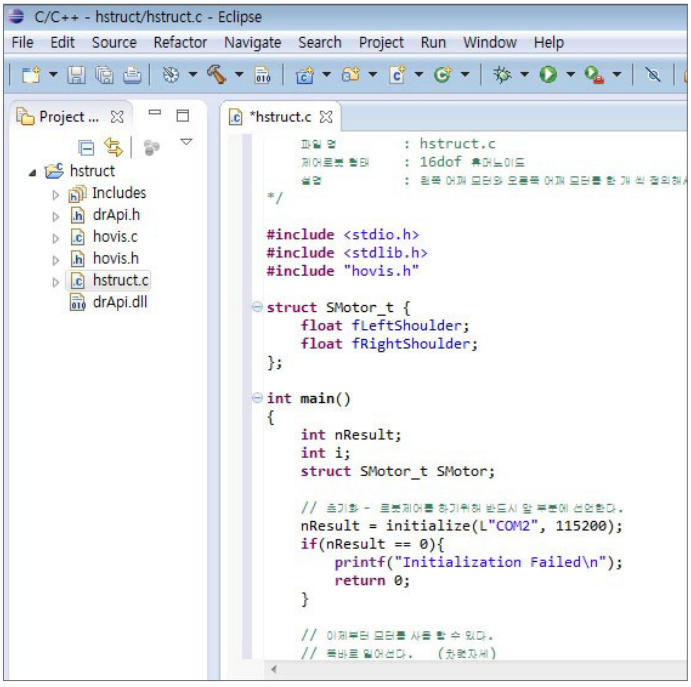
로봇을 동작시키기 위해선 위 네 개 파일을 프로젝트에 포함시켜야 합니다.
위 파일이 있는 파일탐색기 폴더에서 드래그 하는 방식으로 프로젝트에 포함시킵니다.
파일을 끌어다가 드래그해서 왼쪽 프로젝트에 추가합니다.

09 기본 파일 Import



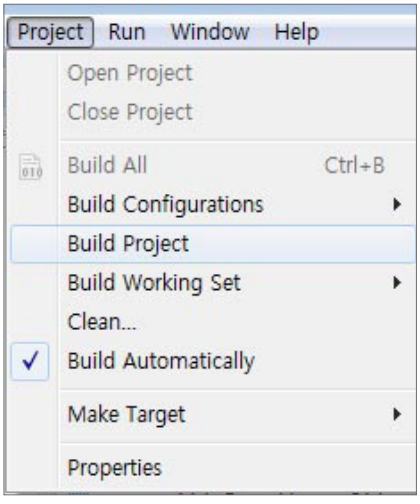
OK 버튼을 누릅니다.

10 소스 작성



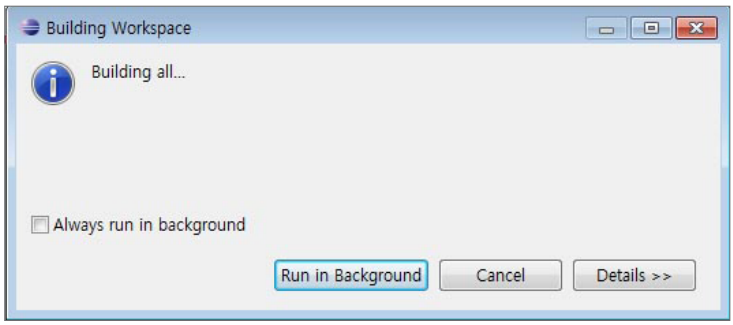
소스를 작성합니다.

11 빌드



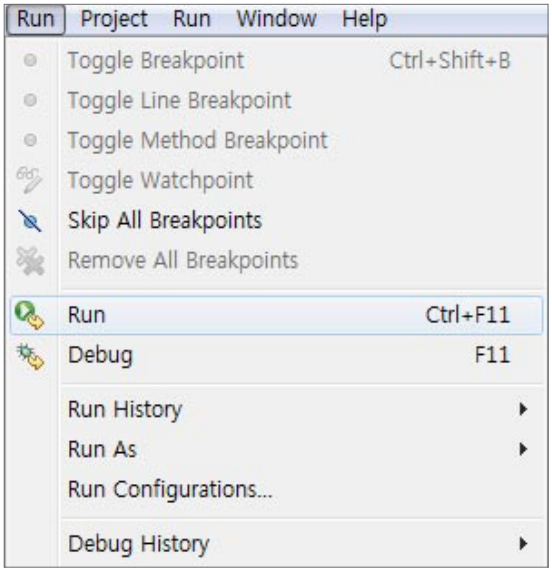
Projec 〉 Build All 을 클릭합니다. 단축키 Ctrl+B 를 눌러도 결과는 마찬가지 입니다.

12 빌드중



빌드 팝업입니다. 그대로 두면 완료 후 사라집니다.

13 Run 실행



Run 〉 Run 을 클릭합니다. Ctrl+F11 단축키도 동일합니다.

14 실행화면

```
        while(1){
            printf("Input Left Shoulder Angle\n");
            fflush(stdout);
            scanf("%f", &SMotor.fLeftShoulder);
            printf("Input Right Shoulder Angle\n");
            fflush(stdout);
            scanf("%f", &SMotor.fRightShoulder);

            g_fMotorPos[3] = SMotor.fLeftShoulder;
            g_fMotorPos[0] = SMotor.fRightShoulder;
            run(1000);
            delay(1000);
        }

        // 프로그램 종료
        terminate();
        return 0;
    }
```

Problems Tasks Console Properties

hstruct.exe [C/C++ Application] C:₩GccDongbu₩workspacew₩hstruct₩D

```
40
Input Right Shoulder Angle
50
Input Left Shoulder Angle
```

15 로봇동작

왼쪽 어깨 모터와 오른쪽 어깨 모터를 한 개 씩 정의해서 묶어놓고 입력값에 따라 각도 조절합니다.

07 메모리동적할당

7.1 메모리 동적할당

데이터 영역 : 전역변수, Static 변수 저장하기 위한 장소입니다.
스택 영역 : 지역변수, 매개 변수를 저장하기 위한 장소입니다.
힙 : Run time 에 실행하는 시간에 할당하기 위한 메모리 공간임 함수가 malloc 이라는 함수입니다.
메모리 동적 할당이란 런 타임에 메모리 공간의 크기를 결정지어서 할당합니다.
(힙 영역에 할당)

아래 예제는 리모컨 1~9 키를 받으면 그 값만큼 메모리를 할당하고 서보 LED가 색을 다르게 해서 깜빡이는 프로그램입니다.

hmallocfree.c

```
#include 〈stdio.h〉
#include 〈stdlib.h〉
#include “hovis.h”
int main()
{

        int nResult;
        int i, j;
        int *pnLed;

        nResult = initialize(L”COM2”, 115200);
        if(nResult == 0){
        printf(“Initialization Failed₩n”);
        fflush(stdout);
        return 0;
}

        for(i=0;i〈16;i++){
                g_fMotorPos[i]=0;
        }
```

```
g_fMotorPos[0] = –90;
g_fMotorPos[1] = 90;
g_fMotorPos[3] = –90;
g_fMotorPos[4] = 90;
run(1000);

delay(1000);

while(1){
        printf("Input Remocon₩n");
        fflush(stdout);
        while (1) {
                read();
                if(g_nRemoconData != 0xFE)
                        break;
        }

        printf("Value = %d₩n", g_nRemoconData);
        fflush(stdout);

        if(g_nRemoconData >= 1 && g_nRemoconData <= 9){

            pnLed = (int*)malloc(g_nRemoconData*sizeof(int));
            for(i = 0;i < g_nRemoconData;i++)
                pnLed[i] = i%3;

            for(i = 0;i < g_nRemoconData;i++){
                if(pnLed[i] == 0){
                        for(j = 0;j < 16; j++){

                                g_ucMotorGreenLed[j] = 1;

                                g_ucMotorBlueLed[j] = 0;

                                g_ucMotorRedLed[j] = 0;
                        }
                }
                else if(pnLed[i] == 1){

                        for(j = 0;j < 16; j++){

                                g_ucMotorGreenLed[j] = 0;

                                g_ucMotorBlueLed[j] = 1;

                                g_ucMotorRedLed[j] = 0;
```

```
                        }
                    }
                    else if(pnLed[i] == 2){
                        for(j = 0;j < 16; j++){
                            g_ucMotorGreenLed[j] = 0;
                            g_ucMotorBlueLed[j] = 0;
                            g_ucMotorRedLed[j] = 1;
                        }
                    }
                    run(0);
                    delay(1000);
                }
                        for(j = 0;j < 16; j++){
                            g_ucMotorGreenLed[j] = 0;
                            g_ucMotorBlueLed[j] = 0;
                            g_ucMotorRedLed[j] = 0;
                    }
                    run(0);
                    free(pnLed);
                }
                else{
                    delay(200);
                }
        }
        terminate();
        return 0;
}
```

위 코드 중에 진하게 처리된 malloc free 문장을 살펴봅니다.

```
#include <stdlib.h>          // malloc, free를 사용하기 위해서는 선언 해 두어야 한다.
//메모리 동적 할당
pnLed = (int*)malloc(g_nRemoconData*sizeof(int));    // 눌린 키의 숫자만큼 메모리 동적 할당
//동적 할당 메모리 해제
free(pnLed);
```

문법요약

◆ 메모리 동적할당

※ 스택, 힙 그리고 데이터 영역

– 프로그램의 실행을 위해 기본적으로 할당하는 메모리 공간입니다.

– 컴파일 타임에 함수에서 요구하는 스택의 크기 결정되어야 합니다.

– OS 가 관리해주는 메모리 구조이고, C++, JAVA 등도 실행되면, 아래와 같이 메모리 관리됨

※ 데이터 영역 : 전역변수, Static 변수 저장하기 위한 장소입니다.

※ 스택 영역 : 지역변수, 매개 변수를 저장하기 위한 장소입니다.

– 컴파일 타임에 함수에서 요구하는 스택의 크기 결정되어야 합니다.

– 반드시 컴파일 타임에 결정됩니다.

● malloc함수

heap영역에 메모리를 할당합니다.

```
void* malloc(size_t size)
```

● free함수

heap영역에 할당된 메모리를 해제합니다

```
void free(void* ptr)
```

코딩계획

```
// malloc, free를 사용하기 위해서는 선언
//결과값, for, 메모리 동적할당 포인터 저장 변수
//초기화
//모터사용
//리모콘 입력, 출력
//LED 실행 끄기
```

프로그래밍 세부설계

```
/*
파일명              : hmalloc.c
제어로봇 형태       : 16dof 휴머노이드
설명                : 리모컨 1~9 키를 받으면 그 값만큼 메모리를 할당하고 서보 LED를
색을 다르게 해서 깜빡인다.
*/

// malloc, free를 사용하기 위해서는 선언 해 두어야 한다.
// 동부로봇의 api 를 사용하기 위해 반드시 이렇게 선언 해 두어야 한다.

int main()
{
        // 결과값을 리턴받을 변수
        // for 문을 사용하기 위해 선언한 변수
        // 메모리를 동적 할당할 정수 포인터

       // 초기화 – 로봇제어를 하기 위해 반드시 앞 부분에 선언한다.
        // 각 변수들을 초기화 한다.
        // 열고 난 이후 이상이 있는 지 확인한다.
                // 이상이 있다면 에러 출력
                // 프로그램 종료

        // 이제부터 모터를 사용 할 수 있다.
        // 똑바로 일어선다.        (차렷자세)
                // 모든 모터(16개 모터)를 0 위치로 설정함

        // 0 번 모터(오른쪽 어깨)
        // 1 번 모터(오른쪽 윗팔)
        // 3 번 모터(왼쪽 어깨)
        // 4 번 모터(왼쪽 윗팔)
        // 모터 동작(해당 자세를 1000ms 동안 동작)
        // 동작 대기(1000ms)
```

```
// 무한 반복한다.
        // 리모컨을 입력하라는 메시지 출력
        // 리모컨 감지 대기
        // 탈출하기 전까지 무한 반복
                // 리모컨 입력값을 읽기 위해 통신 요청.
                // 리모컨 입력값이 0xFE가 아닌 경우(아무 입력 없을 때가 0xFE)
                        // while문을 탈출한다.

        // 감지 후
        // 리모컨 입력 값을 출력한다.

        // 1~9번 키 중 하나가 눌린 경우
                //메모리 동적 할당
                // 눌린 키의 숫자만큼 메모리 동적 할당

                //동적 할당한 메모리에 값 대입
                // 눌린 키의 숫자만큼 for문으로 반복
                        // pnLED[i]에 0, 1, 2, 0, 1, 2...로 값 대입

                //값에 따라서 LED 실행
                // 눌린 키의 숫자만큼 for문으로 반복
                        // pnLED[i] 값이 0이라면
                        // 모든 모터(0~15번)에 대해서 반복
                                        // 초록 색 LED를 켜고
                                        // 파란 색 LED를 끄고
                                        // 빨간 색 LED를 끈다.

                        // pnLED[i] 값이 1이라면
                        // 모든 모터(0~15번)에 대해서 반복
                                        // 초록 색 LED를 끄고
                                        // 파란 색 LED를 켜고
                                        // 빨간 색 LED를 끈다.

                        // pnLED[i] 값이 2라면
                        // 모든 모터(0~15번)에 대해서 반복
                                        // 초록 색 LED를 끄고
                                        // 파란 색 LED를 끄고
                                        // 빨간 색 LED를 켠다.

                        // 동작 실행(LED만이므로 0ms로 해도 됨)
                        // 동작 대기(1000ms)
                        //끝난 후 LED 모두 끄기
                // 모든 모터(0~15번)에 대해서 반복
                        // 초록 색 LED를 끄고
```

```
                                    // 파란 색 LED를 끄고
                                    // 빨간 색 LED를 끈다.

                                // 동작 실행(LED만이므로 0ms로 해도 됨)

                                //동적 할당 메모리 해제

                            // 그 외의 키가 눌린 경우
                                // 200ms 동안 대기(리모컨 값이 초기화 될 때까지)

// 프로그램 종료
// 제어 종료 함수를 실행한다.
```

프로그래밍 작성

```
/*
파일명              : hmalloc.c
제어로봇 형태       : 16dof 휴머노이드
설명                : 리모컨 1~9 키를 받으면 그 값만큼 메모리를 할당하고 서보 LED를
색을 달리 해서 깜빡인다.
*/

#include <stdio.h>
#include <stdlib.h>
// malloc, free를 사용하기 위해서는 선언 해 두어야 한다.
#include "hovis.h"
// 동부로봇의 api 를 사용하기위해 반드시 이렇게 선언 해 두어야 한다.

int main()
{
        int nResult;                                    // 결과값을 리턴받을 변수
        int i;                          // for 문을 사용하기 위해 선언한 변수
        int *pnLed;                     // 메모리를 동적 할당할 정수 포인터
        // 초기화 – 로봇제어를 하기위해 반드시 앞 부분에 선언한다.
        nResult = initialize(L"COM2", 115200);      // 각 변수들을 초기화 한다.
        if(nResult == 0){ // 열고 난 이후 이상이 있는 지 확인한다.
                printf("Initialization Failed₩n");      // 이상이 있다면 에러 출력
                fflush(stdout);
                return 0;                       // 프로그램 종료
        }

        // 이제부터 모터를 사용 할 수 있다.
        // 똑바로 일어선다.         (차렷자세)
        for(i=0;i<16;i++){
                g_fMotorPos[i]=0;
                // 모든 모터(16개 모터)를 0 위치로 설정함
        }
        g_fMotorPos[0] = -90;                           // 0 번 모터(오른쪽 어깨)
        g_fMotorPos[1] = 90;                            // 1 번 모터(오른쪽 윗팔)
        g_fMotorPos[3] = -90;                           // 3 번 모터(왼쪽 어깨)
        g_fMotorPos[4] = 90;                            // 4 번 모터(왼쪽 윗팔)
        run(1000);          // 모터 동작(해당 자세를 1000ms 동안 동작)
```

```
        delay(1000);                                    // 동작 대기(1000ms)

        while(1){                                       // 무한 반복한다.
                printf("Input Remocon₩n"); // 리모컨을 입력하라는 메시지 출력
                fflush(stdout);
                // 리모컨 감지 대기
                while (1) {                             // 탈출하기 전까지 무한 반복
                        read();   // 리모컨 입력값을 읽기 위해 통신 요청.
                        if(g_nRemoconData != 0xFE)
        // 리모컨 입력값이 0xFE가 아닌 경우(아무 입력 없을 때가 0xFE)
                                break;                  // while문을 탈출한다.
                }

                // 감지 후
                printf("Value = %d₩n", g_nRemoconData);  // 리모컨 입력 값을 출력한다.
                fflush(stdout);
                if(g_nRemoconData 〉= 1 && g_nRemoconData 〈= 9){
                // 1~9번 키 중 하나가 눌린 경우
                        //메모리 동적 할당
                        pnLed = (int*)malloc(g_RemoconData*sizeof(int));
                // 눌린 키의 숫자만큼 메모리 동적 할당

                //동적 할당한 메모리에 값 대입
for(i = 0;i 〈 g_nRemoconData;i++)// 눌린 키의 숫자만큼 for문으로 반복
pnLed[i] = i%3;   // pnLED[i]에 0, 1, 2, 0, 1, 2...로 값 대입

                        //값에 따라서 LED 실행
for(i = 0;i 〈 g_nRemoconData;i++){// 눌린 키의 숫자만큼 for문으로 반복
if(pnLed[i] == 0){// pnLED[i] 값이 0이라면
for(j = 0;j 〈 16; j++){         // 모든 모터(0~15번)에 대해서 반복
                        g_ucMotorGreenLed[j] = 1;// 초록 색 LED를 켜고
                        g_ucMotorBlueLed[j] = 0;// 파란 색 LED를 끄고
                        g_ucMotorRedLed[j] = 0;// 빨간 색 LED를 끈다.
                                }
                        }
                        else if(pnLed[i] == 1){       // pnLED[i] 값이 1이라면
        for(j = 0;j 〈 16; j++){         // 모든 모터(0~15번)에 대해서 반복
                        g_ucMotorGreenLed[j] = 0;// 초록 색 LED를 끄고
                        g_ucMotorBlueLed[j] = 1;  // 파란 색 LED를 켜고
                        g_ucMotorRedLed[j] = 0;// 빨간 색 LED를 끈다.
                        }
```

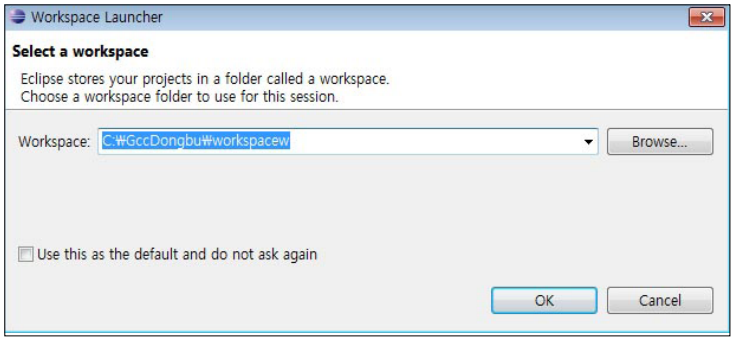
```
                    }
                else if(pnLed[i] == 2){      // pnLED[i] 값이 2라면
                for(j = 0;j < 16; j++){ // 모든 모터(0~15번)에 대해서 반복
                g_ucMotorGreenLed[j] = 0;// 초록 색 LED를 끄고
                g_ucMotorBlueLed[j] = 0; // 파란 색 LED를 끄고
                g_ucMotorRedLed[j] = 1;// 빨간 색 LED를 켠다.
                        }
                    }
                run(0);   // 동작 실행(LED만이므로 0ms로 해도 됨)
                delay(1000);                // 동작 대기(1000ms)
                }

                //끝난 후 LED 모두 끄기
                for(j = 0;j < 16; j++){ // 모든 모터(0~15번)에 대해서 반복
                g_ucMotorGreenLed[j] = 0;// 초록 색 LED를 끄고
                g_ucMotorBlueLed[j] = 0; // 파란 색 LED를 끄고
                g_ucMotorRedLed[j] = 0; // 빨간 색 LED를 끈다.
                }
                run(0);   // 동작 실행(LED만이므로 0ms로 해도 됨)

                //동적 할당 메모리 해제
                free(pnLed);
            }
            else{                       // 그 외의 키가 눌린 경우
            delay(200);// 200ms 동안 대기(리모컨 값이 초기화 될 때까지)
            }
        }
        // 프로그램 종료
        terminate();                    // 제어 종료 함수를 실행한다.
        return 0;
}
```

빌드 및 실행

01 이클립스 실행

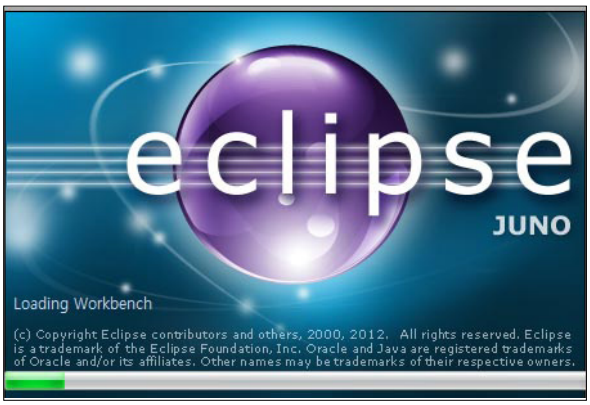
이클립스 실행 버튼을 누릅니다.

02 workspace



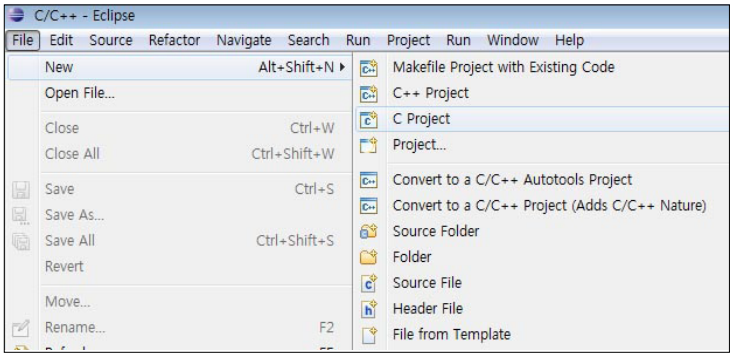
작업하고자 하는 폴더를 지정합니다. Browse 를 클릭하여 폴더를 지정하고 OK 버튼을 누릅니다.

03 이클립스 로딩 화면



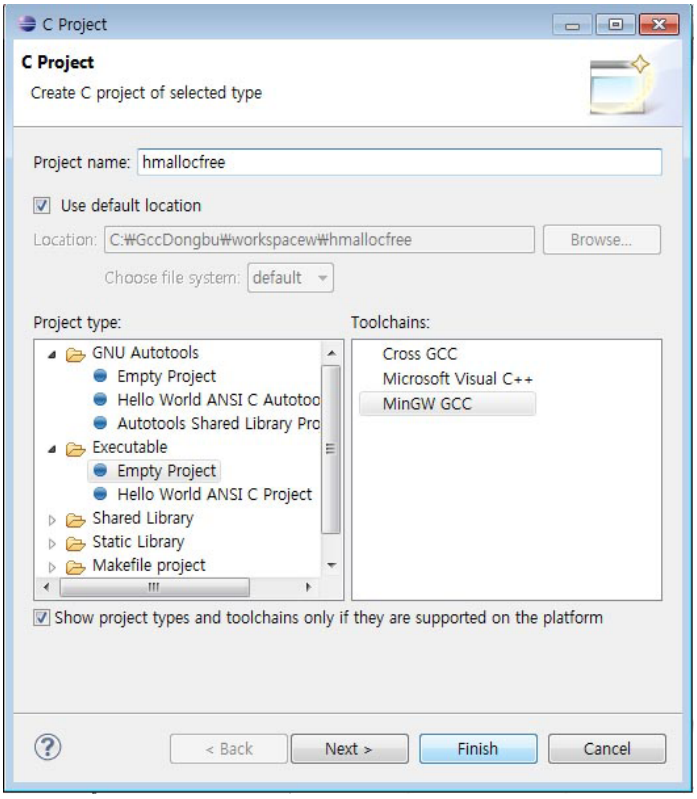
이클립스 로딩 화면입니다.

04 프로젝트 생성



이클립스 편집창 메뉴에서 File 〉 New 〉 C project 를 클릭합니다.

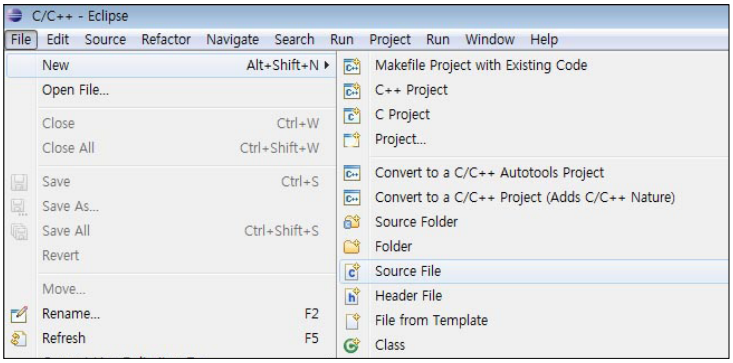
05 프로젝트 이름



Projec Name 을 hmallocfree 라고 입력하고, Project type 에서 Exectable 에서 Empty Project 를 선택합니다.

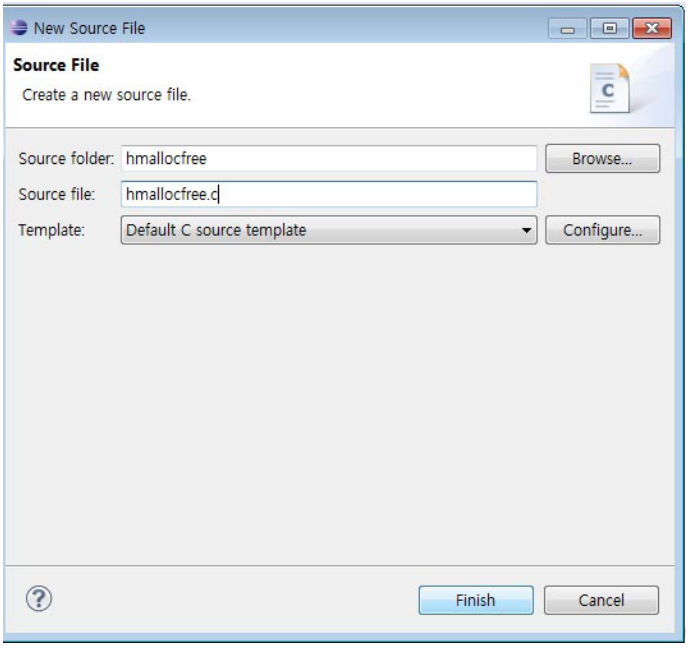
Toolchians 에서 MinGW GCC 를 선택하고 Finish 버튼을 누릅니다.

06 소스파일 추가



File 〉 New 〉 Source File 을 클릭합니다.

07 소스파일 이름

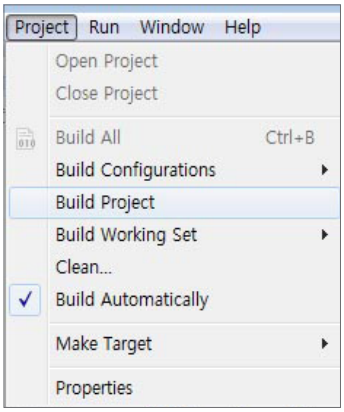


hmallocfree.c 라고 입력하고 Finish 버튼을 누릅니다.

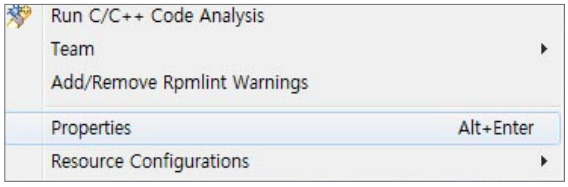
08 기본 제공 파일 추가

본 교재에게 C 프로그래밍을 하기 위해선 기본 제공되는 라이브러리 파일을 먼저 추가해주어야 올바른 컴파일을 할 수 있습니다. 필요한 파일은 drApi.dll, drApi.h, hovis.c, hovis.h 등 총 4가지 입니다.

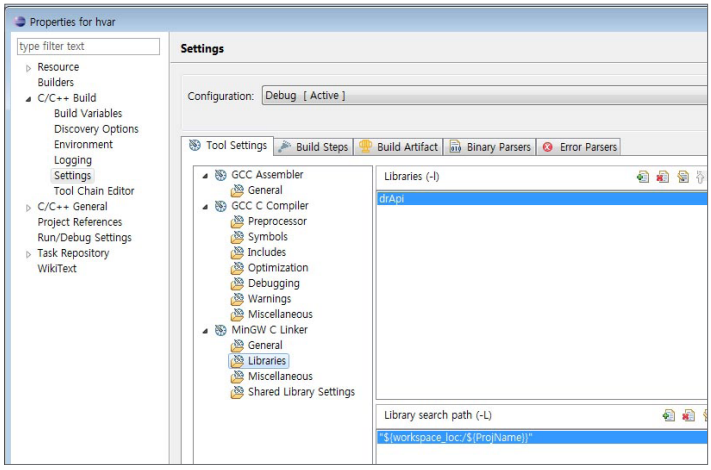
파일 추가 전에 먼저 Property 에서 라이브러리를 설정해주어야 합니다. 파일 생성을 한 후에 아무 코딩도 하지 말고, 바로 Build project 를 실행하세요. 그래야 라이브러리 추가가 가능합니다.



Build Project 를 클릭합니다.



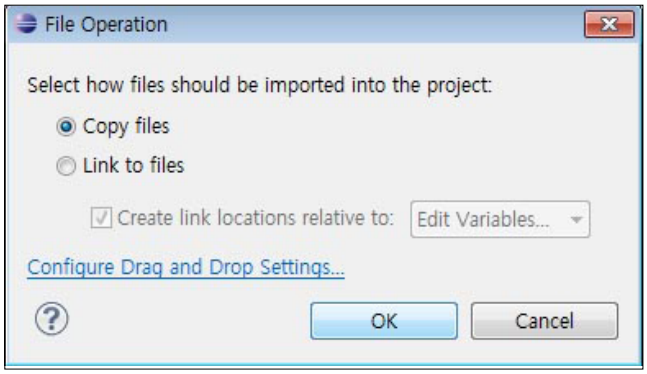
좌측 프로젝트 이름에 커서를 놓고 오른쪽 마우스키를 누릅니다.
제일 하단에 Propertis 를 클릭합니다. Alt+Enter 도 동일합니다.



C/C++ Build 〉 Settings 를 클릭하고, 우측에 MinGW C Linker 〉 Libraties 를 클릭합니다. 우측에 Libraies 에 + 마크를 클릭하고 팝업창이 뜨면 drApi 라고 입력합니다. Library search path 에 + 마크를 클릭하고 팝업창이 뜨면 Workspace 를 클릭하고, Folder selection 에서 프로젝트 이름과 동일한 것을 클릭하고 OK 버튼을 누릅니다.

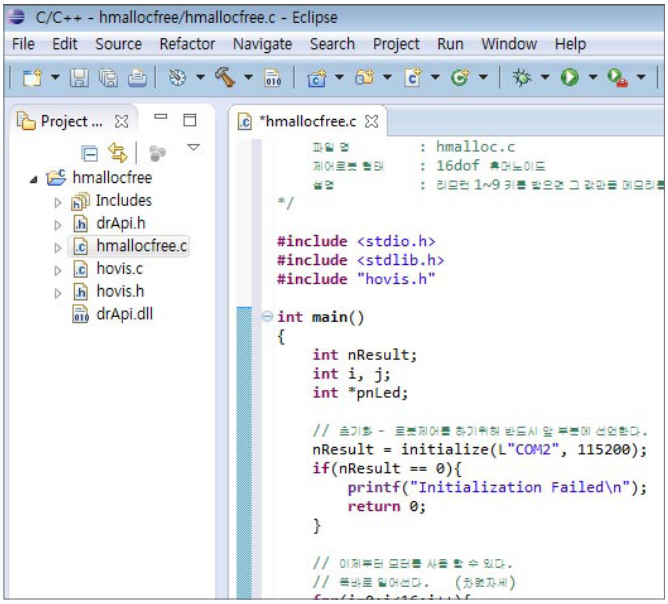
로봇을 동작시키기 위해선 위 네 개 파일을 프로젝트에 포함시켜야 합니다.
위 파일이 있는 파일탐색기 폴더에서 드래그 하는 방식으로 프로젝트에 포함시킵니다.
파일을 끌어다가 드래그해서 왼쪽 프로젝트에 추가합니다.

09 기본 파일 Import



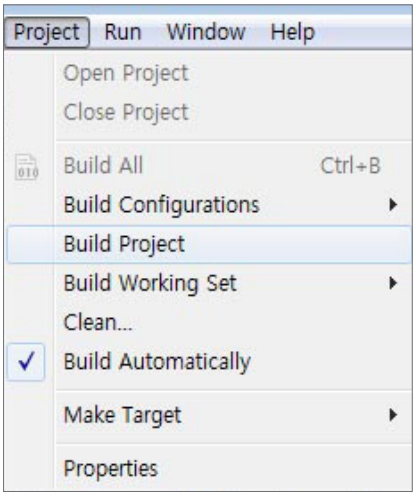
OK 버튼을 누릅니다.

10 소스 작성



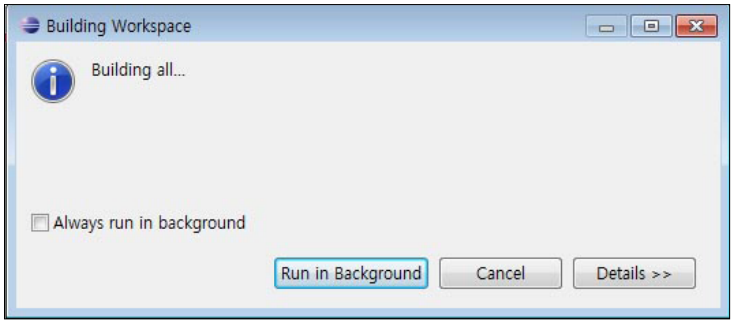
소스를 작성합니다.

11 빌드



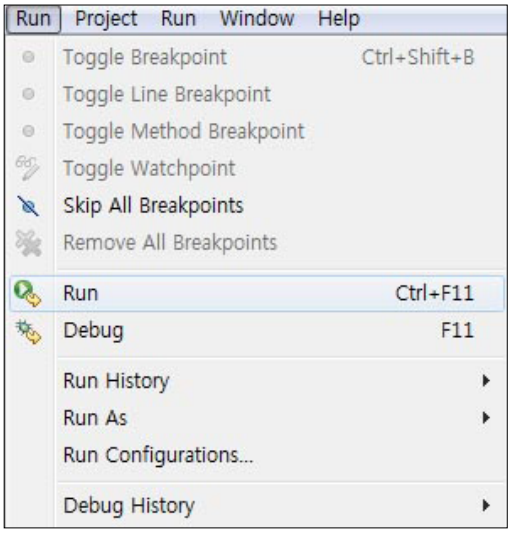
Projec 〉Build Project 를 클릭합니다.

12 빌드중



빌드 팝업입니다. 그대로 두면 완료 후 사라집니다.

13 Run 실행



Run 〉 Run 을 클릭합니다. Ctrl+F11 단축키도 동일합니다.

14 실행화면

```
            //끝난 후 LED 모두 끄기
            for(j = 0;j < 16; j++){
                g_ucMotorGreenLed[j] = 0;
                g_ucMotorBlueLed[j] = 0;
                g_ucMotorRedLed[j] = 0;
            }
            run(0);

            //동적 할당 메모리 해제
            free(pnLed);
        }
        else{
            delay(200);
        }
    }

    // 프로그램 종료
    terminate();
    return 0;
}
```

Problems Tasks Console Properties

hmallocfree.exe [C/C++ Application] C:₩GccDongbu₩workspacew₩

Input Remocon

15 로봇동작

리모컨 1~9 키를 받으면 그 값만큼 메모리를 할당하고 서보 LED가 색을 다르게 해서 깜빡입니다.

08 매크로와 전처리기

8.1 매크로와 전처리기

define 으로 시작하는 전처리기 지시자는 컴파일러에 의해 처리되는 것이 아니고, 전처리기에게 단순 치환 작업을 요청할 때 사용되는 지시자입니다.

#define PI 3.1415

#deifne 은 전처리기 지시자, PI 는 매크로, 3.14는 데체 리스트입니다.

아래 예제는 #define으로 선언한 NUMBER, MELODY에 따라, 모션을 실행하고 MELODY 번의 멜로디를 2번 실행하는 걸 NUMBER 번 반복하는 프로그램입니다.

hdefine.c

```
#include <stdio.h>
#include <stdlib.h>
#include "hovis.h"

#define NUMBER                  3

#define MELODY                  1

int main()
{
        int nResult;
        int i, j;
        int *pnLed;
        nResult = initialize(L"COM2", 115200);
if(nResult == 0){
printf("Initialization Failed₩n");
fflush(stdout);
return 0;

}
        for(i=0;i<16;i++){
                g_fMotorPos[i]=0;
        }
```

```
        g_fMotorPos[0] = -90;
        g_fMotorPos[1] = 90;
        g_fMotorPos[3] = -90;
        g_fMotorPos[4] = 90;
        run(1000);
        delay(1000);

        for (i = 0; i < NUMBER; i++)
    {

        motion(0, 0);
        motion_wait();
        for (j = 0; j < 2; j++)
        {
            g_nDrcMelody = MELODY;
            run(0);
            dr_wait_delay(300);
        }
    }

    terminate();
    return 0;
}
```

위 코드 중에 진하게 처리된 NUMBER, MELODY 문장을 살펴봅니다.

```
#define NUMBER   3     // main 함수에서 사용할 모션 반복 횟수를 미리 정의합니다.
#define MELODY   1     // main 함수에서 사용할 멜로디 번호를 미리 정의합니다.
//총 NUMBER번 반복
for (i = 0; i < NUMBER; i++)
g_nDrcMelody = MELODY; // Buzz 1 을 실행하도록 g_nDrcMelody 값을 설정합니다.
```

문법요약

'매크로'란 복잡한 상수 문자 등을 매크로 이름이라는 간단한 기호로 정의한 뒤, 문자대신 사용하는 것을 말합니다.
#define 매크로이름 치환내용
typedef 는 사용자가 원하는 것으로 데이터 타입을 다시 정의할 때 사용합니다.

```
typedef int INT
INT a,b,c
기타 선행처리 지시자
#iddef ~ #else ~#endif
#ifndef ~ #else ~#endif
#if ~#elif ~#endif
```

코딩계획

```
// #define을 사용한 전처리문 정의.
// main 함수의 숫자 대신 #define문을 바꾸어 주면 된다.
// main 함수에서 사용할 모션 반복 횟수를 미리 정의한다.
// main 함수에서 사용할 멜로디 번호를 미리 정의한다.
//결과값, for, 메모리 동적할당 정수 포인터 저장 변수
//초기화
//총 NUMBER 반복
//권투모션, 멜로디
```

프로그래밍 세부설계

```
/*
파일명              : hdefine.c
제어로봇 형태       : 16dof 휴머노이드
설명                : #define으로 선언한 NUMBER, MELODY에 따라, 모션을 실행하고
MELODY 번의 멜로디를 2번 실행하는 걸 NUMBER 번 반복한다.
*/

// malloc, free를 사용하기 위해서는 선언 해 두어야 한다.
// 동부로봇의 api 를 사용하기 위해 반드시 이렇게 선언 해 두어야 한다.

// #define을 사용한 전처리문 정의.
// 이렇게 정의해 놓으면 반복 횟수나 멜로디 번호를 바꾸고 싶을 때
// main 함수의 숫자 대신 #define문을 바꾸어 주면 된다.
// main 함수에서 사용할 모션 반복 횟수를 미리 정의한다.
// main 함수에서 사용할 멜로디 번호를 미리 정의한다.

int main()
{
        // 결과값을 리턴받을 변수
        // for 문을 사용하기 위해 선언한 변수
        // 메모리를 동적 할당할 정수 포인터

        // 초기화 – 로봇제어를 하기 위해 반드시 앞 부분에 선언한다.
        // 각 변수들을 초기화 한다.
        // 열고 난 이후 이상이 있는 지 확인한다.
                // 이상이 있다면 에러 출력
                // 프로그램 종료

        // 이제부터 모터를 사용 할 수 있다.
        // 똑바로 일어선다.        (차렷자세)

                // 모든 모터(16개 모터)를 0 위치로 설정함
        // 0 번 모터(오른쪽 어깨)
        // 1 번 모터(오른쪽 윗팔)
        // 3 번 모터(왼쪽 어깨)
        // 4 번 모터(왼쪽 윗팔)
        // 모터 동작(해당 자세를 1000ms 동안 동작)
        // 동작 대기(1000ms)

        //총 NUMBER번 반복
        // 권투모션의 실행
```

```
                // 모션 0번을 실행 (두 번째 파라미터는 준비 자세만 실행하는 모드인지 여부이므로 0)

                // 모션이 종료될 때까지 대기

                // 멜로디 출력 2회

                // Buzz 1 을 실행하도록 g_nDrcMelody 값을 설정한다.
                        // 로봇을 실제로 동작 시킨다.
                        // 멜로디가 울릴 동안 기다린다.

// 프로그램 종료
// 제어 종료 함수를 실행한다.
```

프로그래밍 작성

```
/*
파일명              : hdefine.c
제어로봇 형태       : 16dof 휴머노이드
설명                : #define으로 선언한 NUMBER, MELODY에 따라, 모션을 실행하고
MELODY 번의 멜로디를 2번 실행하는 걸 NUMBER 번 반복한다.
*/

#include <stdio.h>
#include <stdlib.h>
// malloc, free를 사용하기 위해서는 선언 해 두어야 한다.
#include "hovis.h"
// 동부로봇의 api 를 사용하기위해 반드시 이렇게 선언 해 두어야 한다.

// #define을 사용한 전처리문 정의.
// 이렇게 정의해 놓으면 반복 횟수나 멜로디 번호를 바꾸고 싶을 때
// main 함수의 숫자 대신 #define문을 바꾸어 주면 된다.
#define NUMBER                  3
        // main 함수에서 사용할 모션 반복 횟수를 미리 정의한다.
#define MELODY                  1
        // main 함수에서 사용할 멜로디 번호를 미리 정의한다.

int main()
{
        int nResult;                                    // 결과값을 리턴받을 변수
        int i;                          // for 문을 사용하기 위해 선언한 변수
        int *pnLed;                     // 메모리를 동적 할당할 정수 포인터

        // 초기화 – 로봇제어를 하기 위해 반드시 앞 부분에 선언한다.
        nResult = initialize(L"COM2", 115200);          // 각 변수들을 초기화 한다.
        if(nResult == 0){  // 열고 난 이후 이상이 있는 지 확인한다.
                printf("Initialization Failed₩n");      // 이상이 있다면 에러 출력
                fflush(stdout);
                return 0;                       // 프로그램 종료

        }
        // 이제부터 모터를 사용 할 수 있다.
        // 똑바로 일어선다.         (차렷자세)
        for(i=0;i<16;i++){
                g_fMotorPos[i]=0;// 모든 모터(16개 모터)를 0 위치로 설정
```

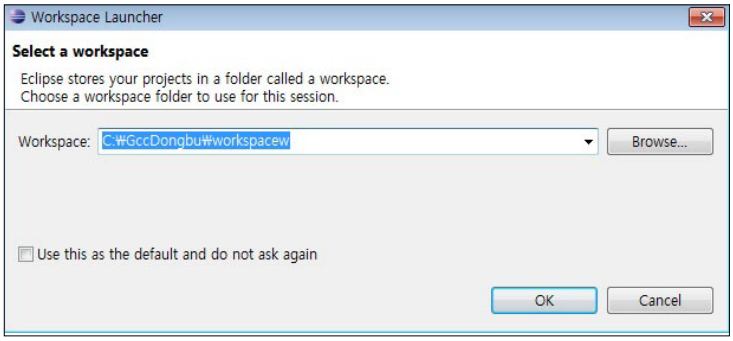
```
        }
        g_fMotorPos[0] = -90;                    // 0 번 모터(오른쪽 어깨)
        g_fMotorPos[1] = 90;                     // 1 번 모터(오른쪽 윗팔)
        g_fMotorPos[3] = -90;                    // 3 번 모터(왼쪽 어깨)
        g_fMotorPos[4] = 90;                     // 4 번 모터(왼쪽 윗팔)
        run(1000);          // 모터 동작(해당 자세를 1000ms 동안 동작)
        delay(1000);                             // 동작 대기(1000ms)

        //총 NUMBER번 반복
        for (i = 0; i < NUMBER; i++)
        {
                // 권투모션의 실행
                motion(0, 0);
                // 모션 0번을 실행(두 번째 파라미터는 준비 자세만 실행하는 모드
인지 여부이므로 0)
                motion_wait();              // 모션이 종료될 때까지 대기

                // 멜로디 출력 2회
                for (j = 0; j < 2; j++)
                {
                        g_nDrcMelody = MELODY;
                // Buzz 1 을 실행하도록 g_nDrcMelody 값을 설정한다.
                        run(0);             // 로봇을 실제로 동작 시킨다.
                        dr_wait_delay(300);
                // 멜로디가 울릴 동안 기다린다.
                }
        }
        // 프로그램 종료

        terminate();                             // 제어 종료 함수를 실행한다.
        return 0;
}
```

빌드 및 실행

01 이클립스 실행

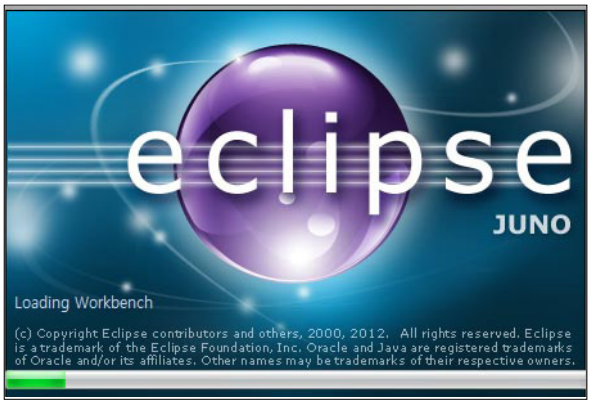
이클립스 실행 버튼을 누릅니다.

02 workspace



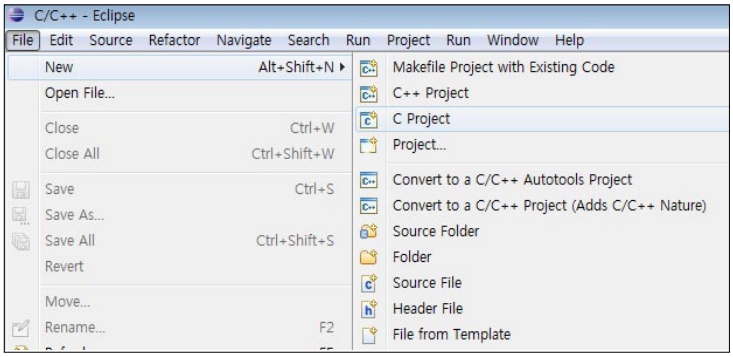
작업하고자 하는 폴더를 지정합니다. Browse 를 클릭하여 폴더를 지정하고 OK 버튼을 누릅니다.

03 이클립스 로딩 화면



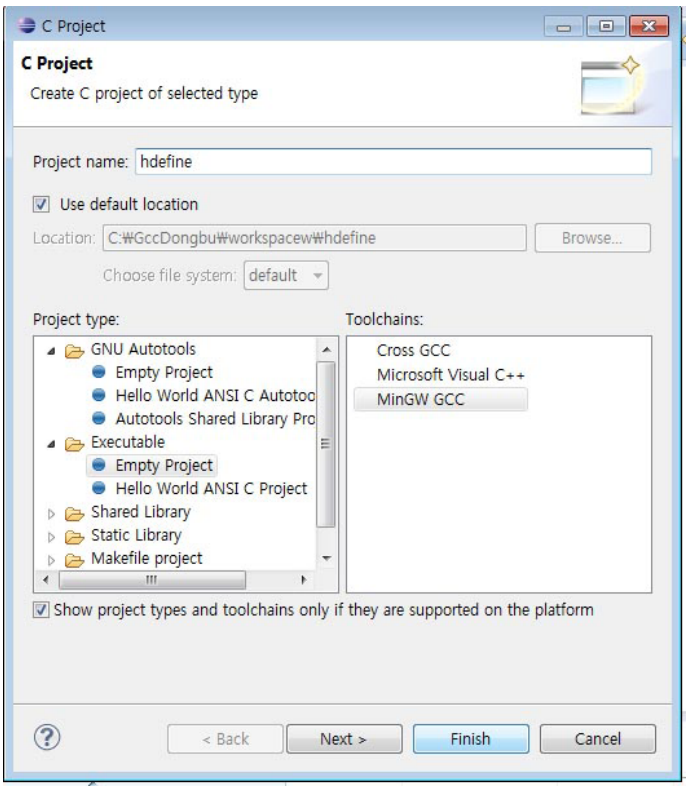
이클립스 로딩 화면입니다.

04 프로젝트 생성



이클립스 편집창 메뉴에서 File 〉 New 〉 C project 를 클릭합니다.

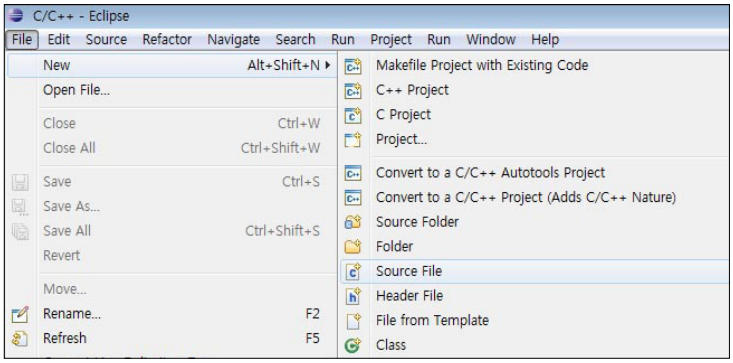
05 프로젝트 이름



Projec Name 을 hdefine 라고 입력하고, Project type 에서 Exectable 에서 Empty Project 를 선택합니다.

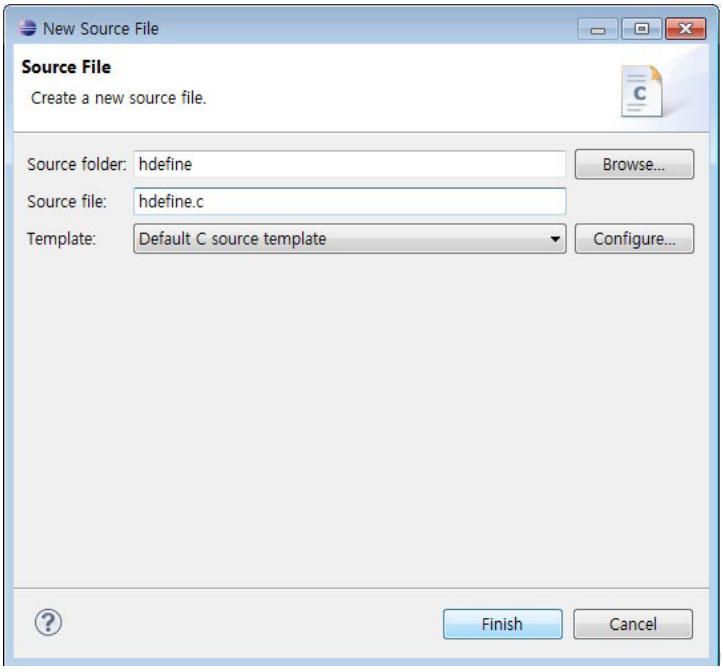
Toolchians 에서 MinGW GCC 를 선택하고 Finish 버튼을 누릅니다.

06 소스파일 추가



File〉New〉Source File 을 클릭합니다.

07 소스파일 이름

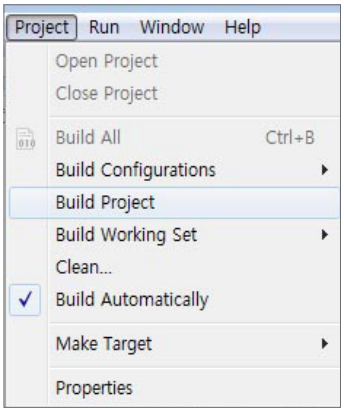


hdefine.c 라고 입력하고 Finish 버튼을 누릅니다.

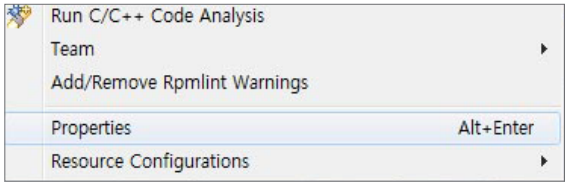
08 기본 제공 파일 추가

본 교재에게 C 프로그래밍을 하기 위해선 기본 제공되는 라이브러리 파일을 먼저 추가해주어야 올바른 컴파일을 할 수 있습니다. 필요한 파일은 drApi.dll, drApi.h, hovis.c, hovis.h 등 총 4가지 입니다.

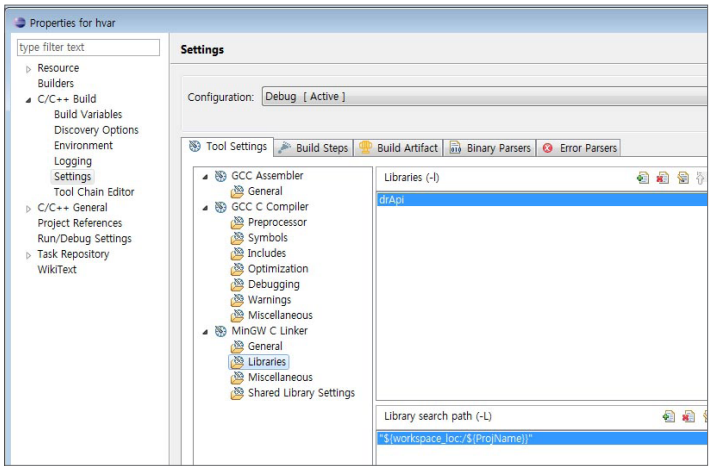
파일 추가 전에 먼저 Property 에서 라이브러리를 설정해주어야 합니다. 파일 생성을 한 후에 아무 코딩도 하지 말고, 바로 Build project 를 실행하세요. 그래야 라이브러리 추가가 가능합니다.



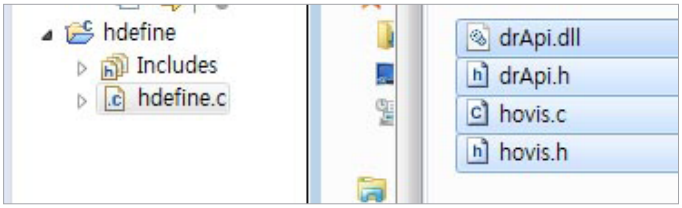
Build Project 를 클릭합니다.



좌측 프로젝트 이름에 커서를 놓고 오른쪽 마우스키를 누릅니다.
제일 하단에 Propertis 를 클릭합니다. Alt+Enter 도 동일합니다.

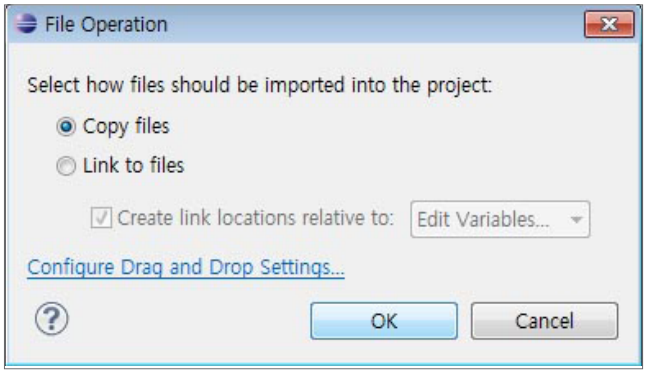


C/C++ Build 〉 Settings 를 클릭하고, 우측에 MinGW C Linker 〉 Libraties 를 클릭합니다. 우측에 Libraies 에 + 마크를 클릭하고 팝업창이 뜨면 drApi 라고 입력합니다. Library search path 에 + 마크를 클릭하고 팝업창이 뜨면 Workspace 를 클릭하고, Folder selection 에서 프로젝트 이름과 동일한 것을 클릭하고 OK 버튼을 누릅니다.



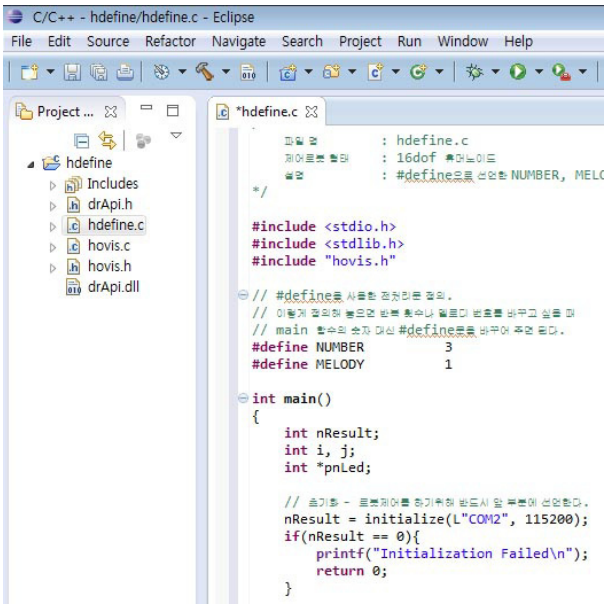
로봇을 동작시키기 위해선 위 네 개 파일을 프로젝트에 포함시켜야 합니다.
위 파일이 있는 파일탐색기 폴더에서 드래그 하는 방식으로 프로젝트에 포함시킵니다.
파일을 끌어다가 드래그해서 왼쪽 프로젝트에 추가합니다.

09 기본 파일 Import



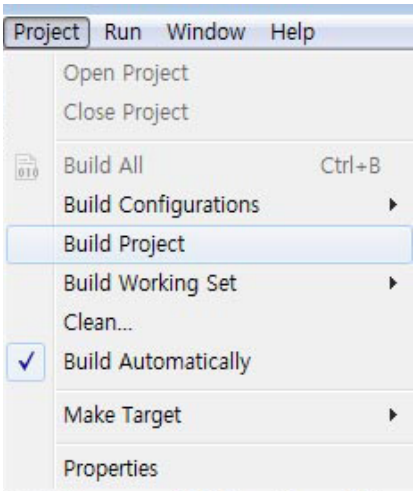
OK 버튼을 누릅니다.

10 소스 작성



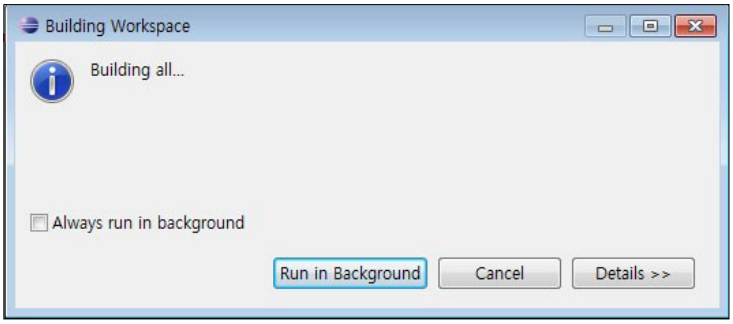
소스를 작성합니다.

11 빌드



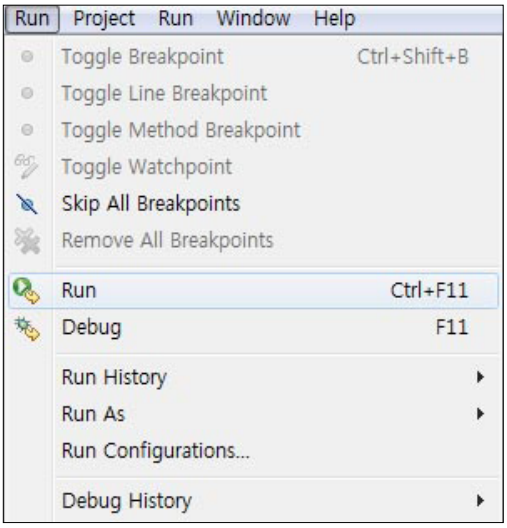
Projec 〉 Build Project 를 클릭합니다.

12 빌드중



빌드 팝업입니다. 그대로 두면 완료 후 사라집니다.

13 Run 실행



Run 〉 Run 을 클릭합니다. Ctrl+F11 단축키도 동일합니다.

14 실행화면

```
    //총 NUMBER번 반복
    for (i = 0; i < NUMBER; i++)
    {
        // 원투모션의 실행
        motion(0, 0);
        motion_wait();

        // 멜로디 총곡 2회
        for (j = 0; j < 2; j++)
        {
            g_nDrcMelody = MELODY;
            run(0);
            dr_wait_delay(300);
        }
    }

    // 프로그램 종료
    terminate();
    return 0;
}
```

Problems Tasks Console Properties

<terminated> hdefine.exe [C/C++ Application] C:₩GccDongbu₩

15 로봇동작

#define으로 선언한 NUMBER, MELODY에 따라, 모션을 실행하고 MELODY 번의 멜로디를 2번 실행하는 걸 NUMBER 번 반복합니다.

부록.1 C 언어 문법 요약

◆ 로봇에서 C 언어를 배운다는 것

로봇은 움직이는 컴퓨터라고 보면 됩니다. 그렇다면 로봇은 두뇌 역할을 하는 CPU를 가지고 있을 것입니다. 그 CPU 를 MCU 라고 부르는데, MCU 는 성능에 따라서 8bit, 16bit, 32bit 로 나눠집니다. 그 중 로봇 컨트롤러로 가장 많이 쓰이는 MCU가 8bit ATmel 사의 ATmega 128 입니다.
동부로봇 휴머노이드에 쓰이는 컨트롤러인 DRC 는 ATmega128 칩을 사용한 제어기입니다.
우리가 흔히 언어,즉 프로그래밍 랭귀지를 가장 처음 배울때 C 언어부터 익힙니다. 그렇다면, 로봇이 C 언어 입문 교재로 사용될 수 있나에 대한 의문점이 생깁니다. 현실은 그리 순탄치 않다가 정답입니다. 우선 로봇에 대한 메카니즘을 익히고, C 언어로 풀어가야하기 때문에, 가장 많이 쓰이는 Visual C++ 에 비해, 상대적으로 어렵다고 볼 수 있습니다.
하지만, 로봇으로 C 언어를 익힌다는 것은 PC 어플리케이션 개발에 사용되는 MFC 보다는 좀더 난이도 있고, 고급스러운 프로그래밍을 할 수 있다는 것을 의미하고, 로봇으로 인해 흥미를 유발할 수 있다는 점에서 더 큰 차별성을 둘 수 있습니다.
바로 Firmware 를 가르칠 수 있기 때문입니다.
하드웨어 설계가 끝나면 Firmware 프로그래밍을 해야합니다. Firmware 없이 어플리케이션을 만들 수 없고, 다른 응용 프로그램을 만들 수가 없습니다. 따라서 Firmware 엔지니어는 일반 어플리케이션 개발자보다 대우가 좋은게 현실입니다. 좀 더 난이도 있는 개발을 할 수 있기 때문입니다.
실무에서는 펌웨어와 어플리케이션 개발자는 엄격히 구분되어있습니다. 신입사원 때 한번 정하면 평생 직업이 될 가능성이 높습니다. 따라서 신중한 선택이 필요하며 동부로봇과 함께하는 C언어 교육은 좋은 지침이 될 것입니다.
로봇에서 C 언어를 배운다는 것은 하드웨어를 알아야 하고, Firmware 를 익혀야하므로 좀더 많은 엔지니어적인 지식을 필요로 하는게 사실입니다. C 언어가 어려운게 아니라, 하드웨어와 펌웨어가 더 어렵다고도 볼 수 있습니다.
따라서, 로봇으로 익히는 C 언어는 좀더 포괄적인 문제가 내포합니다.
동부로봇은 추후에 로봇을 이용한 입문자용 C 언어 책을 편찬할 예정입니다. 가장 쉽고, Visual C++ 보다 더 실용적이고, 효과적인 책이 될 것입니다.
본 C언어 문법은 요약형으로 진행할 예정이며, 예시는 필요에 따라 Firmware 와 Visual C++ 모두 사용 될 것입니다. 그리고, C 언어 자체는 다른 C언어 입문서를 보는게 더 낫다는 판단으로 여기에서 다루는 C 문법은 도표형태나 종합적인 데이타를 다루고, 유저는 그것을 참고하는 방식으로만 쓰일 수 있게 할 예정입니다.
또한 DR-Visual Logic 의 예시와 함께 어떻게 C 문법과 연계되는지도 같이 설명할 예정입니다.

부록 1.1 C 언어 훑어보기

◆ C 언어 예시

※ 동부로봇 AVR 예제

```
#include <avr/io.h>  //(1) 전처리기
#define BTN_MODE      0b00000001
int main(void)  // (2) 함수 - 기능정의
{
        unsigned char ucButton;   // (3) 변수 - 메모리 관리 - 포인터
        PORTB &= ~0b10000000;

        while(1) //(4) 반복
        {
                ucButton = (~PINA) & BTN_MASK;
                if(ucButton & BTN_OK){          //(5)조건분기
                        PORTB ^= 0b10000000;
                        _delay_ms(1);
                }
        }
        return 1;
}
```

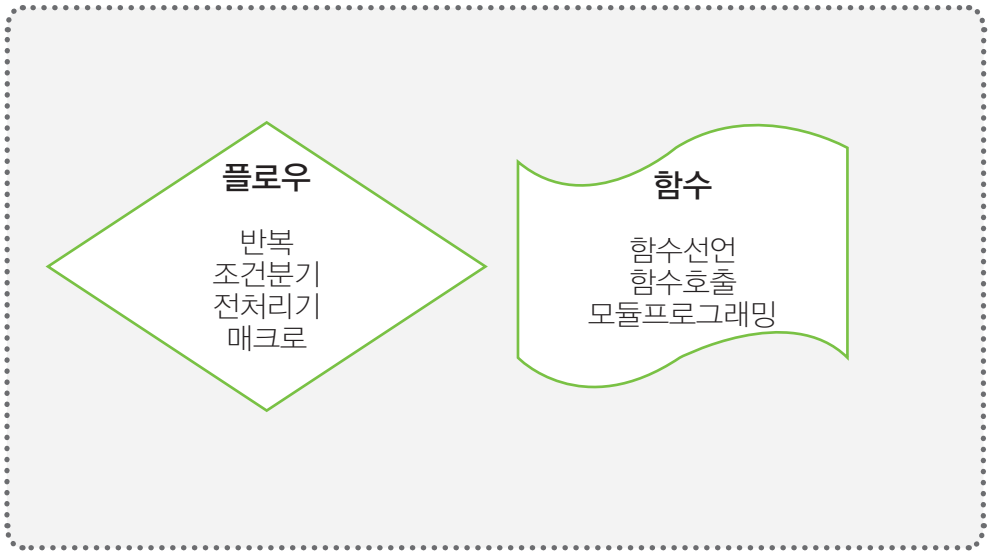
위 예제는 Dongbu Robot AVR 소스 예제중에 일부를 발취한 것으로, C 프로그래밍 문법이 어떻게 구성되는지 간략히 요약되어있습니다.

C언어는 크게 두가지 요소를 가집니다.
바로 변수와 함수입니다.

(3) 변수는 데이터를 저장하고자 하는 목적으로 메모리를 관리하고, 그 메모리는 각각 주소가 존재하는데, 그 주소를 저장하는 것을 포인터라고 부릅니다. 포인터 또한 변수의 일종입니다. 흔히 C 언어하면 포인터에 대한 난해성이 많은 언어로 규정짓기도 합니다.

(2) 함수는 동작시키는 기능을 말하는 것으로서, 프로그래밍은 함수들의 결합이라고 말할 수 있습니다.
프로그래밍을 하는 이유는 컴퓨터가 인간이 할 수 있는 계산 능력을 훨씬 뛰어넘기 때문입니다. 그것을 가장 잘 활용하는 것이 바로, 제어와 분기, 반복입니다.
(4) 반복과 (5) 조건 분기는 C 언어뿐 아니라, 모든 프로그래밍 언어의 문법이 거의 동일하다고 보면 됩니다. 이 두가지 요소는 왜 프로그래밍을 하면 편리해지는지를 보여주는 가장 좋은 사례입니다.

◆ C 언어 요약



C 프로그래밍 언어는 프로그램의 흐름을 제어하는 플로우 와 메모리에 데이터를 관리하는 메모리 파트, 그리고 기능을 부여하는 함수로 구성됩니다.

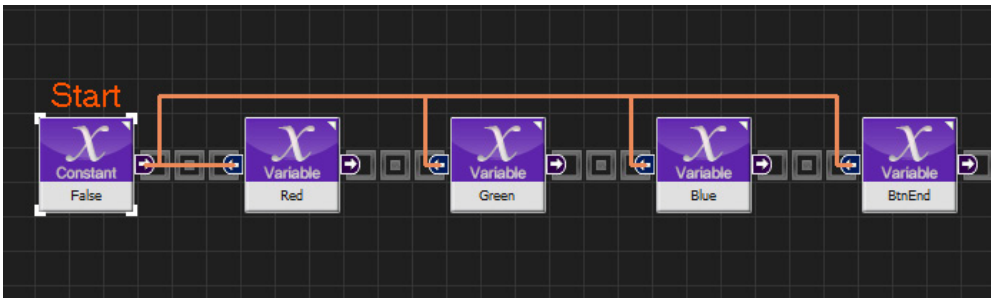
프로그램 작성 및 실행순서는 아래와 같습니다.

1) 프로그램 작성 합니다.
2) 컴파일 합니다.
3) 링크 : 연결합니다.
4) 실행파일이 생성됩니다.

부록 1.2 변수

◆ 변수

ex) ledbutton.dts 중 일부



C-like 보기

```
1  void main()
2  {
3          Red=false
4          Green=false
5          Blue=false
6          BtnEnd=false
7          while( true )
8          {
9                  if( ( ( MPSU_ButtonStat == 0x04 ) && ( !BtnEnd ) ) )
10                 {
11                         Red=( !Red )
12                         BtnEnd=true
13                 }
14                 else
15                 {
16                 }
```

※ 변수란 데이터를 저장할 수 있는 메모리 공간에 붙여진 이름을 의미합니다.

변경이 가능합니다. 단, 상수는 변경이 불가능합니다.

※ 다향한 형태(자료형) 의 변수는 메모리 공간 쓰임새에 따라서 아래와 같이 간략히 나뉩니다.

정수형 : char, int, long 3

실수형 : float, double 3.1

※ 변수의 선언 및 대입 예시입니다.

Int main(void)

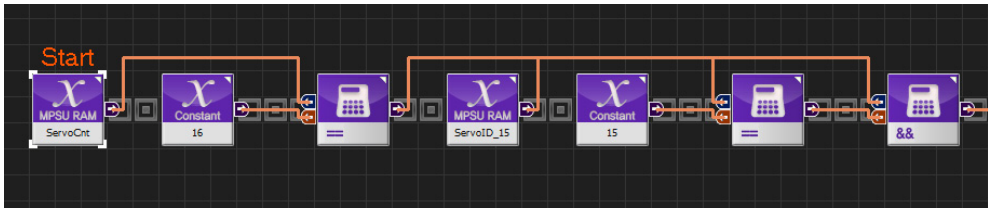
{

Int val; //int형 변수 val 의 선언

// → int 형 변수 val 이름 지어주고, 메모리 공간 할당

Val = 20; //변수 val 에 20을 저장

ex) ledbutton.dts 중 일부



C-like 보기

```
 9  void main()
10  {
11      if( ( ( MPSU_ServoCnt == 16 ) && ( MPSU_ServoID_15 == 15 ) ) )
12      {
13          melody( 1 )
14          SERVO_TorqCtrl[254]=96
15          InitPosition16()
16          delay( 2000 )
17          CurrentMotion=-1
18          MotionAfterStop=-1
19          RmcEnd=false
20          MtnReady=false
```

◆ 리터럴 상수의 기본 자료형

상수도 메모리 공간에 저장되기 위해서 자료형이 결정됩니다.

Char c = 'A' / 문자 상수 → 문자를 쓸 경우 char 형으로 인식하고, 메모리에 넣는다는 의미입니다.

Int I = 5 //정수상수

Double d = 3.15 // 실수상수

왼쪽은 변수, 오른쪽은 상수, 둘은 독립적인 관계입니다.

◆ 심볼릭(symbolic) 상수

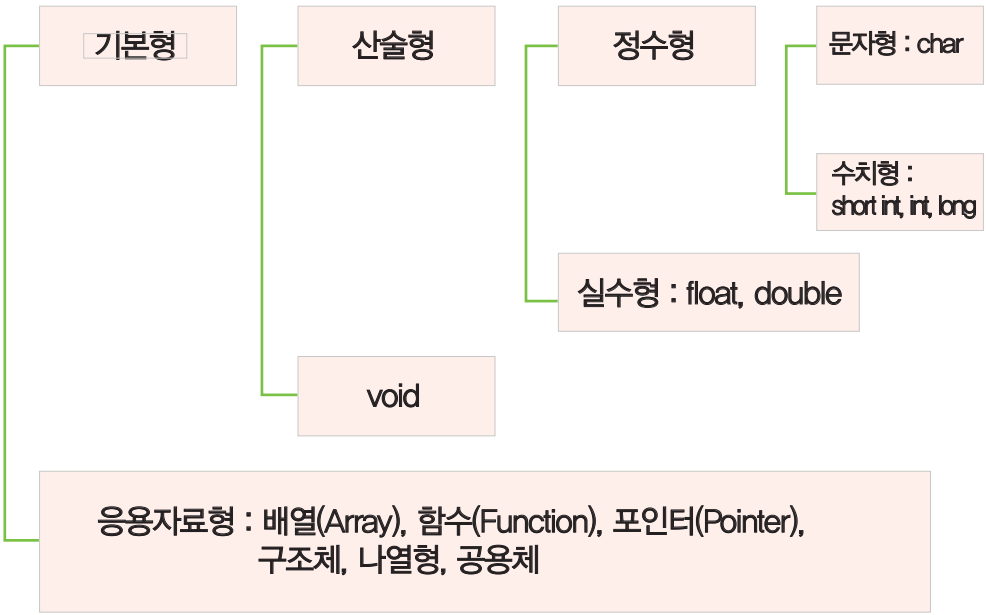
이름을 지니는 상수입니다. 심볼릭 상수를 정의하는 방법은 아래와 같습니다.

const 키워드 를 통한 변수의 상수화
const int MAX = 100 // 변수가 상수가 됨
MAX = 102 // 허용 안됨, MAX 는 상수이기 때문에 상수 변수 선언시 대문자로 써주는 게 관례입니다.

◆ 접미사에 따른 다양한 상수의 표현

접미사	자료형	사용 예
u or U	unsigned int	304U
l or L	long	304L
ul or UL	unsigned long	304UL
f or F	float	3.15F
l or L	long double	3.15L

◆ 자료형 종류



◆ 자료형 범위

구분	자료형	범위	byte
문자형	char	–128~127	1
	unsigned char	0~255	
정수형	int	–2147483648 ~ 2147483637	4
	unsigned int	0 ~ 4294967295	
	short	–32768 ~ 32767	2
	unsigned short	0 ~ 65535	
	long	–2147483648 ~ 2147483637	4
	unsigned long	0 ~ 4294967295	
실수형	float	±3.4 x 10–38 ~ ±3.4 x 1038	4
	double	±1.7 x 10–308 ~ ±1.7 x 10308	8

◆ 확장문자열

표기	의미	기능
₩n	New line	줄바꿈을 한다.
₩t	Tab	다음 탭 위치로 옮긴다.
₩b	Back space	현재 줄의 첫번째 칸으로 간다
₩r	Carriage return	한 칸 뒤로 옮긴다.
₩a	Alert	경보음을 낸다
₩”	” 표시 출력	큰 따옴표를 출력
₩’	’ 표시 출력	작은 따옴표를 출력

◆ printf 변환 문자열

구분	변환문자열	출력형태
정수형	%d	부호 있는 10진수
	%u	부호 없는 10진수
	%o	부호 없는 8진수
	%x, %X	부호 없는 16진수
실수형	%lf, %f	부호 있는 소수점(double, float)
	%le, %e	부호 있는 지수형(double, float)
문자형	%c	하나의 문자
문자열	%s	문자열

◆ printf 변환 문자열 옵션

변경자	의 미	
숫자	출력 자릿수 확보	
	%10d	화면에 10자리를 확보 오른쪽 정렬
	%-10d	화면에 10자리를 확보 왼쪽 정렬
	%10.2lf	화면에 10자리를 확보, 소수점 이하 2자리 까지 출력
h	Short 형임을 의미 예) %hu	
L , l	Long형임을 의미 예) %ld	

◆ Scanf 변환 문자열

구분	변환문자열	출력형태
정수형	%d, %u	10진수
	%o	8진수
	%x	16진수
실수형	%lf, %f	소수점 또는 지수형
	%le, %e	부호 있는 지수형(double, float)
문자형	%c	하나의 문자 (char 형 변수)
문자열	%s	문자열 (char 배열)

◆ C 언어의 키워드 와 설명 1

1.asm : 인라인 어셈블리 코드를 나타내는 키워드
2.auto : 기본적인 변수의 저장방식을 나타내는 키워드
3.break : for,while,switch,do...while문을 조건없이 마치는 명령
4.case : switch문 내에서 사용되는 명령
5.char : 가장 간단한 데이터형
6.const : 변수가 변경되지 않도록 방지하는 데이터 지정자
7.continue : for,while,switch,do...while문을 다음 반복동작으로 진행시키는 명령
8.default : case문에 일치하지 않는 경우를 처리하기 위해 switch문에서 사용되는 명령
9.do : while문과 함께 사용되는 순환명령.순환문은 최소한 한번 실행됨.
10.double : 배정도 부동 소수형값을 저장할 수 있는 데이터형
11.else : if문이 FALSE로 평가될 때 실행되는 선택적인 문장을 나타내는 명령
12.enum :변수가 특정값만을 받아들이도록 해주는 데이터형
13.extern : 변수가 프로그램의 다른 부분에서 선언된다는 것을 알려주는 데이터 지정자
14.float : 부동 소수형 숫자값을 저장하기 위해 사용되는 데이터형
15.for : 초기화,증가,조건 부분을 가지는 순환명령

◆ C 언어의 키워드 와 설명 2

16.goto :　　정의되어 있는 레이블로 이동시키는 명령
17.if : TRUE/FALSE의 결과에 따라 프로그램의 제어를 변경하는데 사용되는 명령
18.int : 정수형 값을 저장하는 데 사용되는 데이터형
19.long : int형보다 큰 정수형 값을 저장하는 데 사용되는 데이터형
20.register : 가능하다면 변수를 레지스터에 저장하도록 지정하는 저장형태 지정자
21.return : 현재의 함수를 마치고 호출한함수로 프로그램의 제어를 돌려주는 명령.
함수에서 값을 돌려주기 위해서 사용됨.
22.short : 정수형 값을 저장하는 데 사용되는 데이터형. 자주 사용되지는 않지만
대부분의 컴퓨터에서 int형과 동일한 크기를 가짐.
23.signed : 변수가 양수와 음수값을 모두 저장할 수 있다는 것을 지정하기 위해서
사용되는 지정자
24.sizeof : 항목의 크기를 바이트 단위로 알려주는 연산자
25.static :컴파일러가 변수의 값을 보존해야 한다는 것을 지정하는데 사용되는 지정자
26.struct :C에서 어떤 데이터형의 변수를 함께 결합시키는 데 사용되는 키워드
27.switch :여러 가지 조건을 통해서 프로그램의 흐름을 변경하는 데 사용되는 명령.
case문과 함께 사용됨.
28.typedef : 이미 존재하는 변수와 함수의 형태를 새로운 이름으로 변경하는 데
사용되는 지정자
29.union :여러 개의 변수가 동일한 메모리 영역을 공유하도록 해주는 데 사용되는 키워드
30.unsigned : 변수가 양수값만으르 저장할 수 있다는 것을 지정하는 데 사용되는 지정자.
31.void : 함수가 어떤 값을 돌려주지 않거나, 또는 사용되는 포인터가 범용
포인터이거나 모든 데이터형을 지적할 수 있다는 갓을 지정하는 데 사용되는 키워드
32.volatile : 변수가 변경될 수 없다는 것을 지정하는 지정자.
33.while : 지정된 조건이 TRUE로 평가되는 한 계속해서 포함된 문장을 실행하는 순환문

◆ 문자형

문자 표현을 위하여 ASCII 코드라는 것이 등장합니다.

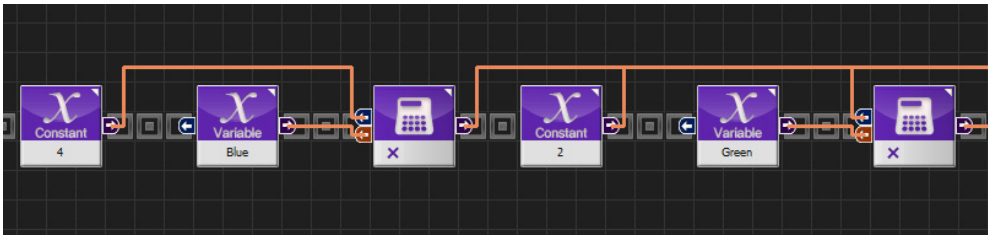
미국 표준 협회 (ANSI) 에 의해 정의, 컴퓨터를 통해서 문자를 표현하기 위한 표준으로서 컴퓨터는 문자를 표현하지 못하기때문에, 문자와 숫자의 연결 관계를 정의하였습니다. 문자 A 는 숫자 65, 문자 B 는 숫자 66 으로 매칭시킵니다.
문자, 컴퓨터는 2진수밖에 받지 못하기 때문에, 컴퓨터와 사람사이에 약속을 하기 시작하였습니다. 사람은 문자 표현 원하고, 문자와 숫자 사이에 매핑 관계를 유지하기를 원합니다. 에를 들어 A=65, B = 55 라고 약속, 사람이 A 입력, 컴퓨터는 65라고 저장합니다.

ANSI 에서 정의되었고, 숫자와 문자 매핑은 아래와 같습니다.

10진	16진	문자	10진	16진	문자	10진	16진	문자	10진	16진	문자	10진	16진	문자
0	0	Null	47	2F	/	68	44	D	89	59	Y	110	6E	n
7	7	Bell	48	30	0	69	45	E	90	5A	Z	111	6F	o
8	8	BS	49	31	1	70	46	F	91	5B	[	112	70	p
9	9	Tab	50	32	2	71	47	G	92	5C	₩	113	71	q
10	A	LF	51	33	3	72	48	H	93	5D	]	114	72	r
13	D	CR	52	34	4	73	49	I	94	5E	^	115	73	s
32	20	공백	53	35	5	74	4A	J	95	5F	_	116	74	t
33	21	!	54	36	6	75	4B	K	96	60	`	117	75	u
34	22	"	55	37	7	76	4C	L	97	61	a	118	76	v
35	23	#	56	38	8	77	4D	M	98	62	b	119	77	w
36	24	$	57	39	9	78	4E	N	99	63	c	120	78	x
37	25	%	58	3A	:	79	4F	O	100	64	d	121	79	y
38	26	&	59	3B	;	80	50	P	101	65	e	122	7A	z
39	27	'	60	3C	〈	81	51	Q	102	66	f	123	7B	{
40	28	(	61	3D	=	82	52	R	103	67	g	124	7C	|
41	29	)	62	3E	〉	83	53	S	104	68	h	125	7D	}
42	2A	*	63	3F	?	84	54	T	105	69	i	126	7E	~
43	2B	+	64	40	@	85	55	U	106	6A	j	127	7F	Del
44	2C	,	65	41	A	86	56	V	107	6B	k			
45	2D	–	66	42	B	87	57	W	108	6C	l			
46	2E	.	67	43	C	88	58	X	109	6D	m			

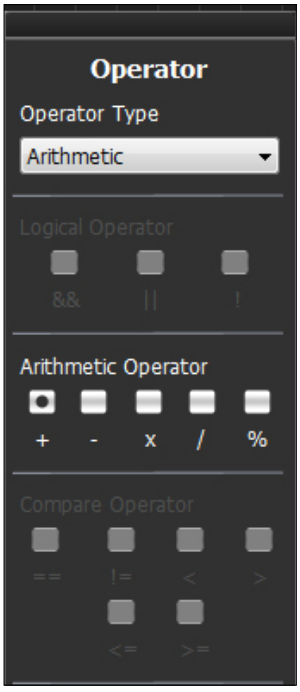
◆ 연산자

ex) ledbutton.dts 중 일부



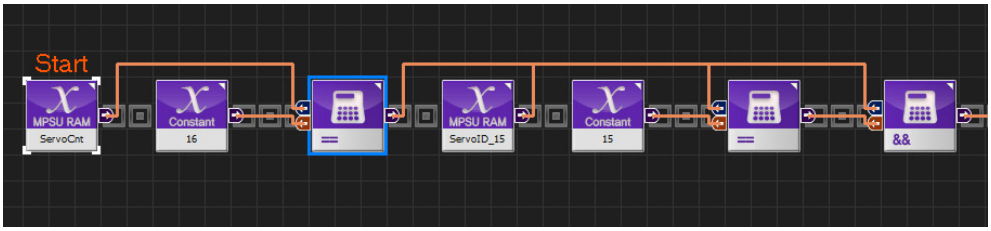
C-like 보기

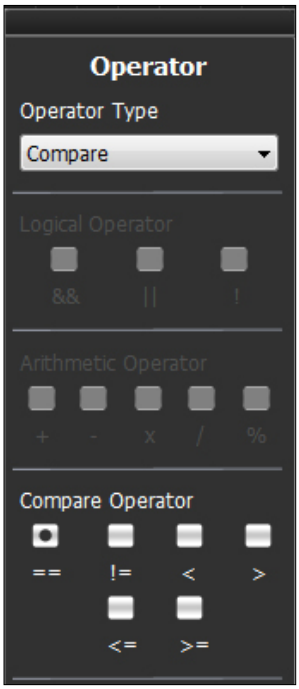
```
33  led( ( ( ( 4 * Blue ) + ( 2 * Green ) ) + Red ) )
34  if( ( ( MPSU_ButtonStat == 0x00 ) && BtnEnd ) )
35  {
36          BtnEnd=false
37  }
38  else
39  {
40  }
```

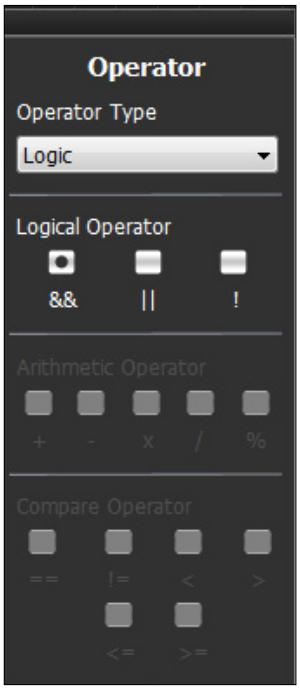


기능별 종류	연산자
산술 연산자	+ – * / %
부호 연산자	+ –
대입 연산자	= 복합 대입 연산자
관계 연산자	== != 〈= 〈 〉= 〉
증감 연산자	++ ––
포인터 연산자	* & []
구조체 연산자	. –〉
논리 연산자	|| && !
비트 연산자	| & ~ 〉〉 〈〈
삼항 조건 연산자	? :
쉼표 연산자	,
sizeof 연산자	sizeof
캐스트 연산자	(type) type()
괄호 연산자	()
C++ 연산자	new delete :: .* –〉*

ex) Remocon16.dts 중 일부







C-like 보기

```
 9  void main()
10  {
11          if( ( ( MPSU_ServoCnt == 16 ) && ( MPSU_ServoID_15 == 15 ) ) )
12          {
13                  melody( 1 )
14                  SERVO_TorqCtrl[254]=96
15                  InitPosition16()
16                  delay( 2000 )
```

◆ 연산자 우선순위

순위	명칭	연산자 종류	결합방향
1	1차 연산자	() {} . –>	→
2	단항 연산자	+ – ! ~ (type) sizeof ++ –– & *	←
3	승제	* / %	→
4	가감	+ –	
5	Shift 연산자	<< >>	
6	관계연산자	< > <= >=	
7	등가 연산자	== !=	
8	비트 곱	&	
9	비트 차	^	
10	비트 합	|	
11	논리곱	&&	
12	논리합	||	
13	조건 연산자	? :	→
14	대입연산자	= += –= *= /= %= <<= >>=&= |=	←
15	순차연산자	,	→

◆ 기억클래스

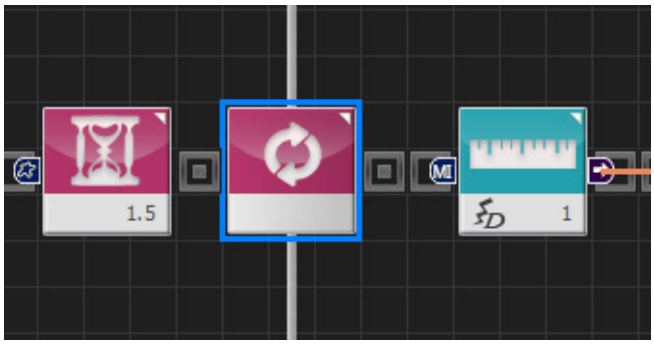
◆ 메모리 영역

지정자		유효범위	생존기간	메모리 위치	자동초기화
auto		선언된 블록 내	블록 종료 시	Stack	X
extern		한 모듈 전체 프로그램	프로그램 종료 시	(비)초기화 영역	O
static	내부 static	선언된 블록 내	프로그램 종료 시	(비)초기화 영역	O
	외부 static	선언된 모듈 내			
register			블록 종료 시	CPU 내 레지스터	X

부록 1.3 반복과 분기

◆ 반복

ex) digital.dts 중 일부



C-like 보기

```
1  void main()
2  {
3          SERVO_TorqCtrl[254]=96
4          motionready( 0 )
5          delay( 1500 )
6          while( true )
7          {
8                  if( ( MPSU_ADCType1 == 2 && MPSU_ADCVal1 == 1 ) )
```

반복문이란 ?

※ 반복문의 기능

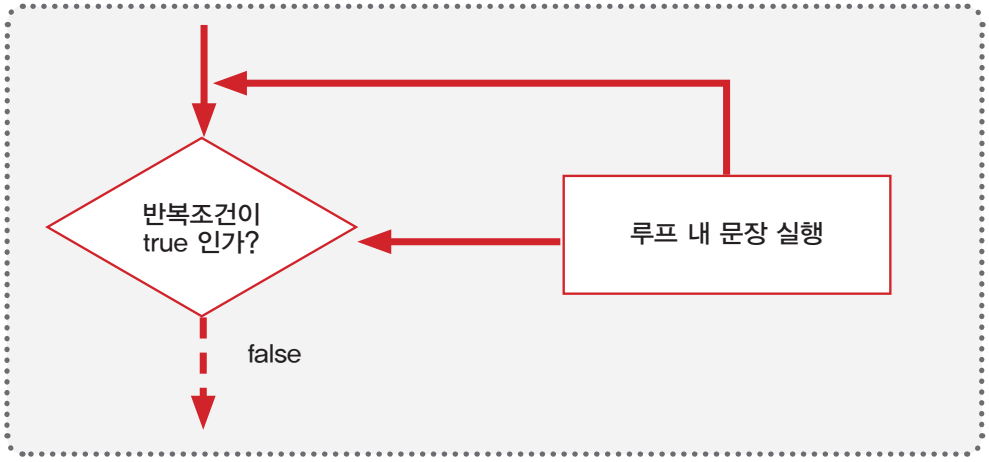
– 특정 영역을 특정 조건이 만족하는 동안에 반복 실행하기 위한 문장입니다.

※ 세가지 형태의 반복문

– while 문에 의한 반복
– do~while 문에 의한 반복
– for 문에 의한 반복

한가지만 다 이해해도 모두 잘 이해할 수 있습니다.

while 문의 순서도

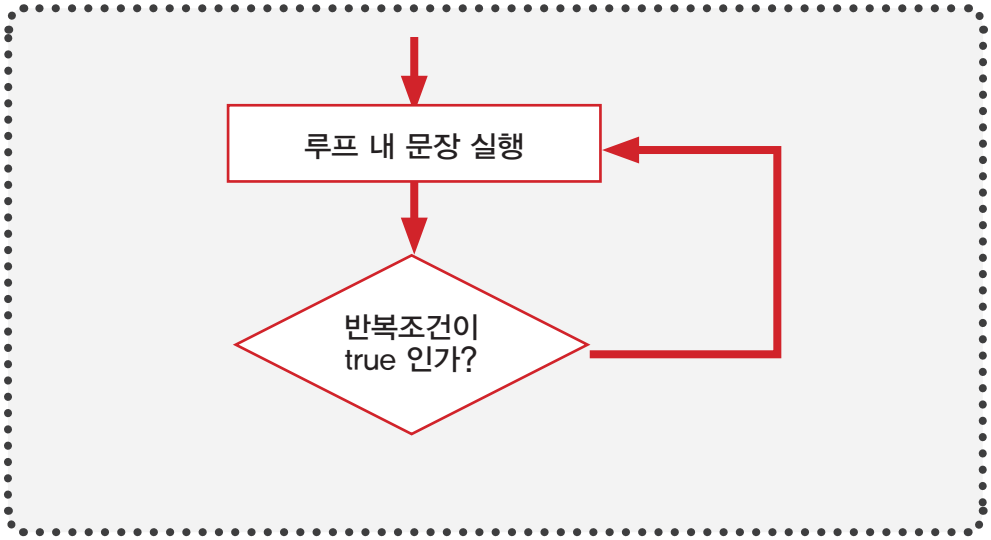


일반형식

```
While(조건식) {
        반복한 문장;
}
```

● 선조건 비교 순환문입니다. (조건이 참이 아닐 경우 한번도 실행하지 못할 수 있습니다.)
● 조건식에 0(zero)가 아닌 값이 오면 C는 참으로 인식하여 무한 loop에 빠지므로 반드시 탈출문을 반복 블록 안에 포함시켜야 합니다.

do~while 문의 순서도

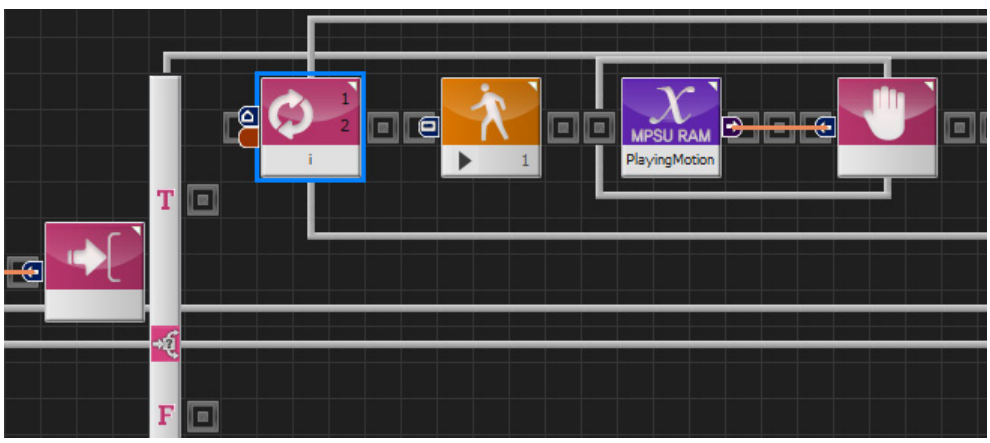


일반형식

```
do {
        반복한 문장;
} while(조건식);
```

● 후조건 비교 순환문이다. (조건이 참이 아닐 경우라도 한번은 실행한다.)

ex) digital.dts 중 일부



C-like 보기

```
if( ( MPSU_ADCType1 == 2 && MPSU_ADCVal1 == 0 ) )
{
        for( i = 1 ~ 2 )
        {
                motion( 1 )
                waituhile( MPSU_PlayingMotion )
```

for 문

일반형식

```
for (초기식 ; 조건식 ; 증감식) {
        반복할 문장;
}
```

● 초기식
– Loop제어변수를 초기화 한다.(처음 한번만 수행)

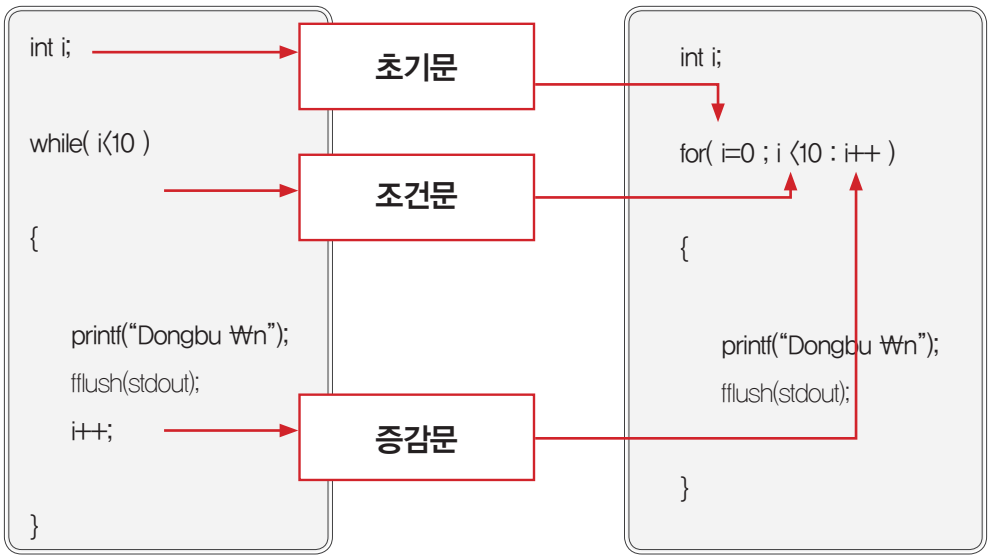
● 조건식
– Loop제어변수의 범위를 검사하여 반복여부를 결정한다.

● 증감식
– Loop제어변수를 증가 또는 감소한다.

다중 for 문

```
for (초기식 ; 조건식 ; 증감식) {
        for (초기식 ; 조건식 ; 증감식) {
                반복할 문장 1;
        }
        반복할 문장 2;
}
```

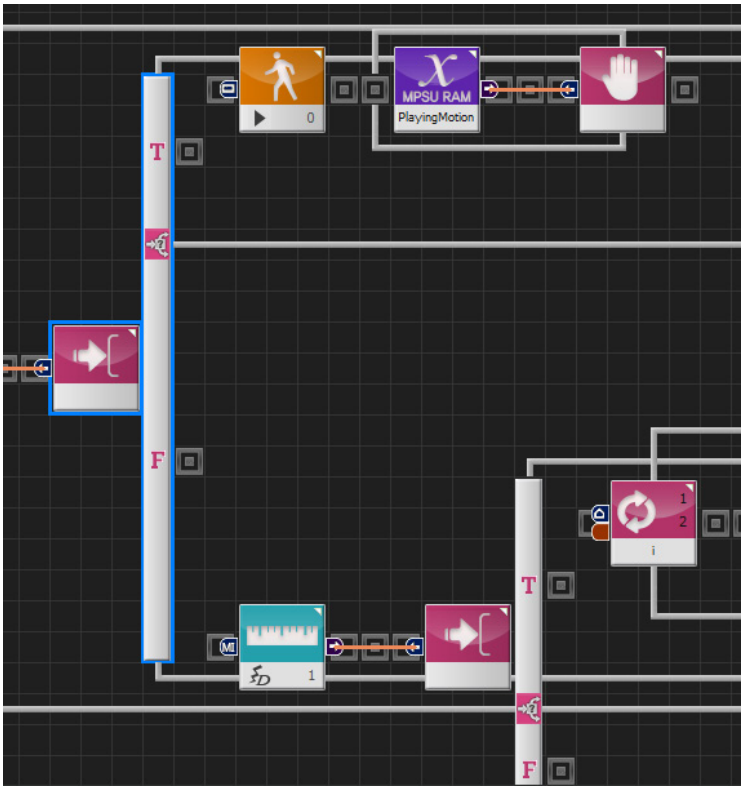
for 와 while 문 비교




```
for(i=0;i<n+1;i++)
        total+=i;

 while 문으로 바꾸면

i=0;
while(i<n+1)
{
        total+=i;
        i++;
}
```

◆ 조건분기

ex) digital.dts 중 일부



C-like 보기

```
8   if( ( MPSU_ADCType1 == 2 && MPSU_ADCVal1 == 1 ) )
9   {
10          motion( 0 )
11          waitwhile( MPSU_PlayingMotion )
12  }
13  else
14  {
15          if( ( MPSU_ADCType1 == 2 && MPSU_ADCVal1 == 0 ) )
16          {
```

◆ 조건 분기문

조건분기문은 if, if else, switch case 등 세가지 형태이지만, 기능은 동일하기 때문에 visual logic 에서는 if else 로보 표현했습니다.

◆ if

if문에 의한 조건적 실행 : 조건이 만족되는 경우에 한해서 실행합니다.

if(실행의조건) → true 라고 가정하면
{ – → 실행하고자 하는 문장이 한 문장일 때 중괄호 생략이 가능합니다.
실행하고자 하는 내용
}

```
int main(void)
{
  int val;
  printf("정수를 하나 입력하세요")
  fflush(stdout);
  scanf(%d, &val);

  if(val<0) //val <0 이 true이면…
  {
    printf("입력값은 0보다 작다"
    fflush(stdout);
  } → 문장이 둘 이상이면 반드시 넣어줘야함
  if(val>0)
    printf("입력값은 0보다 크다"
    fflush(stdout);
  if(val==0)
    printf("입력값은 0이다."
    fflush(stdout);

  return 0;
}
```

만약 1 을 입력하면, 두번째에서 True 이다, 따라서 세번째 조건 검사는 할 필요가 없습니다.
→ 이런 문제점을 해결하기 위해 else 를 넣습니다.

◆ if~else

앞조건이 만족하는데도, 그 아래 문자의 조건을 검사하게 됩니다. . 불필요한 비교연산을 많이 하게되고, 그것을 해결하기위해서 if else 를 씁니다.

```
if(조건)
{
        조건 만족시 실행 일명 “이거”
}
else
{
        조건 불만족시 실행 일명”저거”
}
```

◆ if, else if, else

if(조건 A) → case1: 조건 A가 만족 “요거”실행후 마지막 else 까지도 완전히 건너뜁니다.

```
{
        조건 A 만족시 실행 일명 ”요거”
}
```

else if(조건 B) → case2: 조건 A가 만족 하지 않았다. 따라서 조건 B 가 만족되는지 확인합니다.

```
        조건B는 만족!”이거”실행후 마지막 else 까지도 완전히 건너뜁니다.
{
조건 B만족시 실행 일명 “이거”
}
else if(조건 C)
        조건 C 만족시 실행 일명 ”그거”
}
```

else → case3: 조건 A,B,C가 모두 만족되지 않는다. Else 문 안에 있는 “저거”실행합니다.

```
{
        조건 ABC 불만족시 실행 일명 “저거”
}
```

예를 들어 조건 B가 만족된다면, A는 건너뛰고, B 를 실행한후 나머지는 다 건너띄어 버립니다.
만약 조건 C 가 만족한다면, else 문장은 보지도 않고 건너뜀뜁니다.
→ 불필요한 연산을 극복할 수 있습니다.

◆ switch ~ case

int 형 char 형 인자를 한 개 전달할 수 있음

switch(n)
Case1: → n 이 1인 경우
Break;
Case2:
Break;
Case3:
Break;
default:
n=2 인 경우, case1 은 아니므로 건너뜁니다.
case2는 맞으니까 실행합니다. break 가 있으면 break 문은 switch 를 빠져나가라는 의미입니다.
만약 break 가 없으면, 2부터 그 아래는 모두 실행시킵니다.

◆ 조건분기 전체 요약

if, if~else, if~else if~else, switch~case

```
if(조건)
{
  조건 만족시 실행"
}
```

```
if(조건)
{
  조건 만족시 실행"
}
else
{
  조건 불만족시 실행
}
```

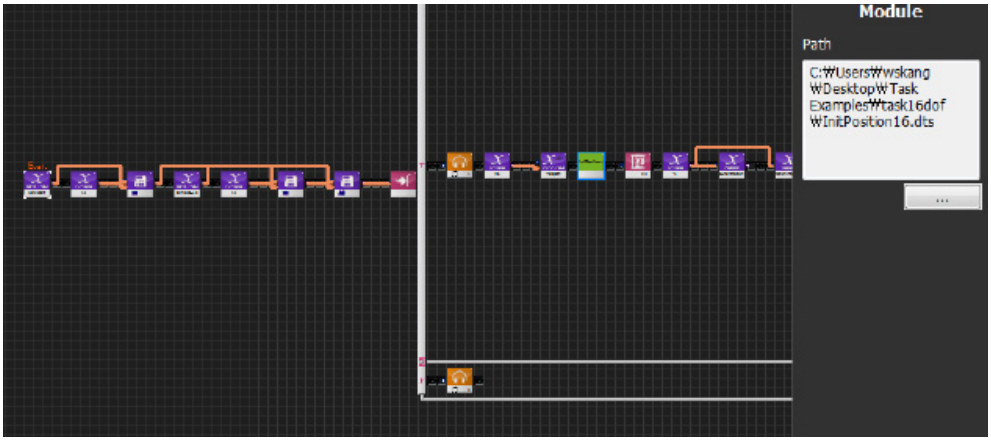
```
if(조건 A)
{
    조건 A 만족시 실행
}
else if(조건 B)
{
    조건 B만족시 실행
}
else if(조건 C)
{
    조건 C 만족시 실행
}
else
{
    조건 ABC 불만족시 실행
}
```

```
switch(n)

   case1:  → n 이 1인 경우
          Break;
   case2:
          Break;
   case3:
      Break;

default:
```

부록1.4 함수

◆ 메인함수

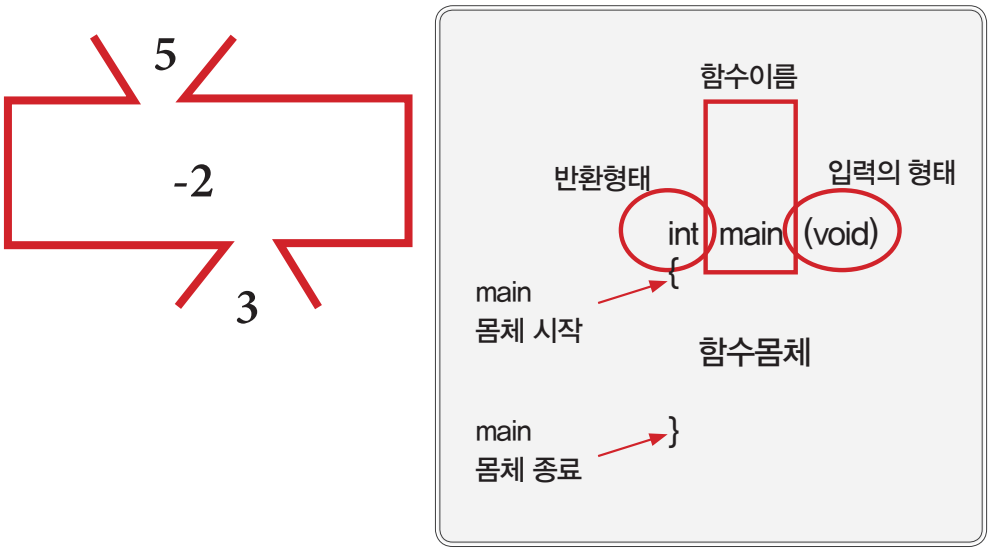
ex) Remocon16.dts 중 일부



C-like 보기

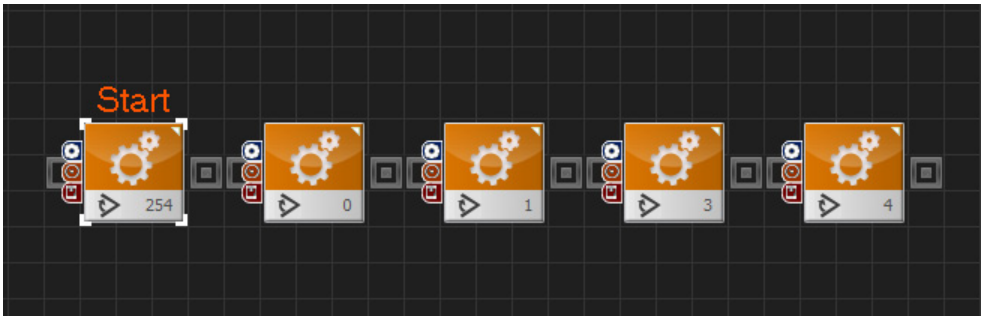
```
1   void InitPosition16()
2   {
3           jog( 512, 0, 254, 120 )
4           jog( 235, 0, 0, 120 )
5           jog( 235, 0, 1, 120 )
6           jog( 789, 0, 3, 120 )
7           jog( 789, 0, 4, 120 )
8   }
9   void main()
10  {
11          if( ( ( MPSU_ServoCnt == 16 ) && ( MPSU_ServoID_15 == 15 ) ) )
12          {
13                  melody( 1 )
14                  SERVO_TorqCtrl[254]=96
15                  InitPosition16()
16                  delay( 2000 )
17                  CurrentMotion=-1
18                  MotionAfterStop=-1
19                  RmcEnd=false
20                  MtnReady=false
21                  MtnStop=false
22                  Interruptable=true
23                  while( true )
```

※ C라는 언어로 구현된 프로그램은 함수로 시작해서, 함수로 끝난다고 해도 과언이 아닙니다. 함수라는 것은 C 언어를 이해하는데, 가장 중요한 일을 합니다. 포인터 가 어려운건 사실이지만, 함수가 훨씬 더 중요하다고 할 수 있습니다. 포인터는 공부하기가 어려운것이지 활용하는 것은 어렵지 않습니다.



첫째, 함수는 이름을 지닙니다. → main
들째, 입력의 형태가 있습니다. → (void)
셋째, 출력의 형태(반환형, 리턴형 이라는 말을 더 많이 함)가 있습니다. → int
넷째, 함수의 몸체 : 함수의 기능을 의미합니다.
A,B,C 함수를 많이 만들어놓고, 특정한 순서에 맞게 호출하는 것이 프로그래밍입니다.

ex)InitPosition16.dts 중 일부



C-like 보기

```
1  void main()
2  {
3          jog( 512, 0, 254, 120 )
4          jog( 235, 0, 0, 120 )
5          jog( 235, 0, 1, 120 )
6          jog( 789, 0, 3, 120 )
7          jog( 789, 0, 4, 120 )
8  }
```

◆ 함수선언

```
(1)  (2)    (3)
int minus (int i, int j)
{
    int result = i-j;
(4) return result;
}
```

(1) 반환 형
(2) 함수 이름
(3) 매개 변수
(4) 값의 반환

※ 4가지 형태의 함수

입력과 출력은 있을 수도 있고, 없을 수도 있습니다. 중요한 것은 기능입니다.
전달 인자 있음, 반환 값 있음
전달 인자 있음, 반환 값 없음
전달 인자 없음, 반환 값 있음
전달 인자 없음, 반환 값 없음

변수의 범위(scope)

※ 변수의 특성에 따른 분류

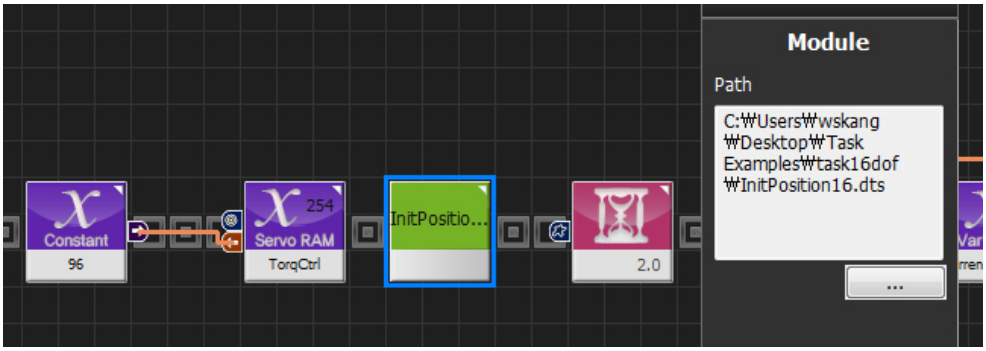
지역변수 (local variable) : 중괄호 내에 선언되는 변수
전역변수 (Global variable) : 함수 내에 선언되지 않는 변수
정적변수 (Static variable) : 함수 내부, 외부 모두 선언 가능
레지스터 변수(Register variable) : 선언에 제한이 많이 따름

static 변수

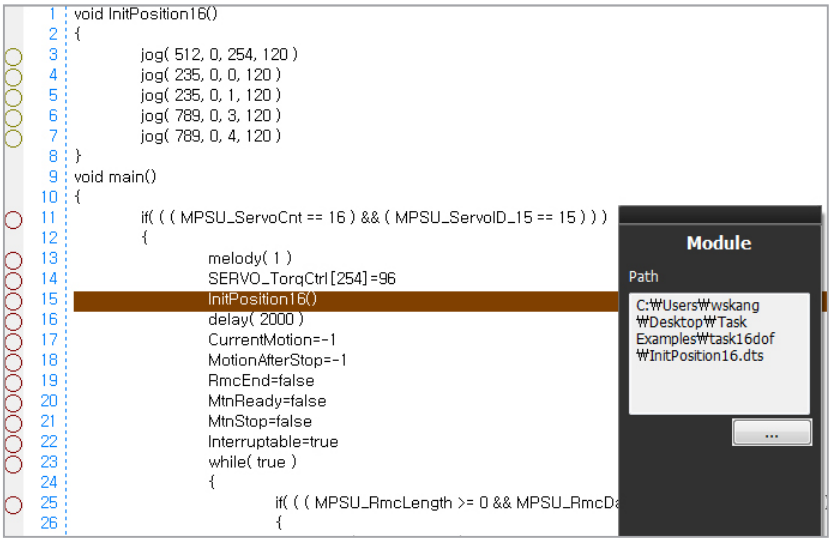
함수 내부 및 외부에 선언 가능합니다.
전역변수, 지역변수 앞에 static 키워드를 붙일 수 있습니다. → 파일 범위를 갖습니다.
↔ extern 한번만 초기화된다됩니다.: 전역변수의 특징
전역변수는 프로그램이 시작하면 올라갔다가, 종료되어야 내려갑니다.
지역변수는 함수가 호출될때마다 초기화됩니다.
함수 내부에서 선언될 경우 함수 내에서만 접근이 가능합니다. : 지역변수의 특징

◆ 함수 호출 및 모듈화 프로그래밍

ex) Remocon16.dts 중 일부



C-like 보기



● 모듈화 프로그래밍

– 기능별로 파일을 나눠가며 프로그래밍하는 것을 말합니다.

– 유지 보수성이 좋아지고, 관리하기가 편해집니다.

예를 들어 200~300 라인은 괜찮지만, 500 라인~ 1000 라인 이상이면 구분도 어렵고, 유지보수도 어려워집니다.

● 파일의 분할 및 컴파일

– 파일을 나눌지라도 완전히 독립되는 것은 아닙니다.

– 파일이 나뉘어도 상호 참조가 발생할 수 있는데, 이는 전역 변수 및 전역 함수로 제한됩니다.

부록1.5 배열, 포인터

DR-Visual Logic 에는 C언어에서 다루는 배열과 포인터, 구조체는 다루지 않습니다. 그래픽 로직을 구성하기 위해서 복잡한 문법을 최소화 시키기 위함입니다.
따라서 이 파트에서는 Visual Logic 과 대응없이 배열,포인터, 구조체를 간략히 설명하겠습니다.

◆ 1차원 배열

※ 배열이란

- 둘 이상의 변수를 동시에 선언하는 효과를 지닙니다.
- 많은 양의 데이터를 일괄적으로 처리해야 하는 경우에 유용합니다.
- 지역적 특성을 지닐 수도 있고, 전역적 특성을 지닐 수도 있습니다.

※ 배열 선언

```
int  array  [10];
```

배열요소자료형 배열이름 배열길이

➔ Int 형 데이터 10개를 저장할 수 메모리 공간을 할당하고, 이름을 array 라고 붙여줍니다.

- 배열 길이 : 배열을 구성하는 변수의 개수 (반드시 상수를 사용)
- 배열 요소 자료형 : 배열을 구성하는 변수의 자료형
- 배열 이름 : 배열에 접근할 때 사용되는 이름

◆ 2차원 배열

※ 다차원 배열의 예

Int arr[100]	1차원배열
Int arr[10][10]	10x10, 2차원배열
Int arr[5][5][5]	5x5x5, 3차원배열

1차원은 선, 2차원은 면, 3차원은 직육면체입니다.
보통 다차원배열은 2차원 배열을 의미합니다.

※ 다차원 배열의 실제 메모리 구성

1차원 배열과 동일하나, 접근 방법을 2차원적으로 해석할 뿐입니다.

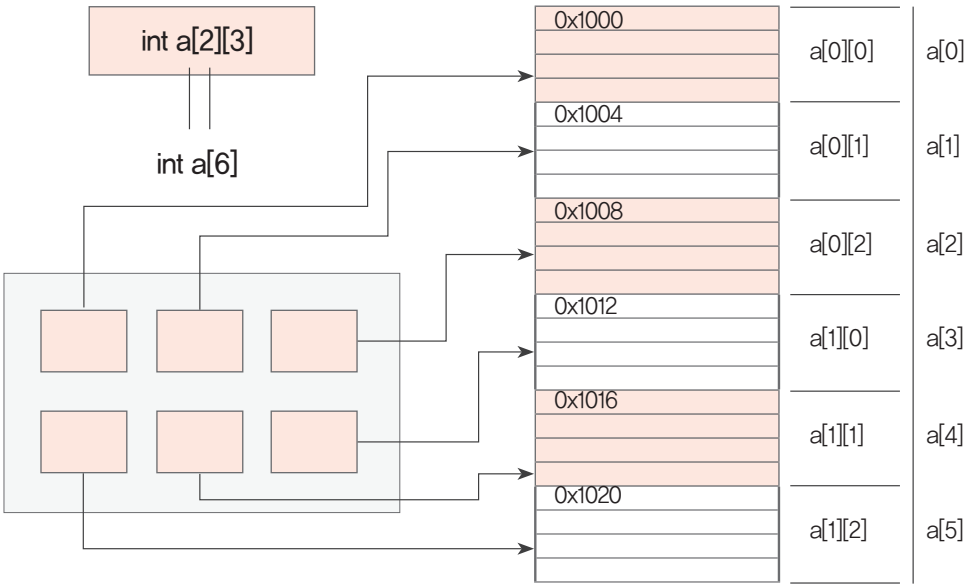
Int a[2][3]; → 2차원 접근 a[0][0]. 세로가 2, 가로가 3

Int a[6]; → 1차원 접근 a[0]~a[5] → 반드시 2차원적으로 해석하기 바랍니다.

문제자체가 2차원적인 경우가 많습니다.

내가 해결해야할 문제는 2차원인데, 실제 구조는 1차원적이기는 하지만 2차원적으로 하는 습관을 가져야 합니다.

◆ 다차원 배열의 실제 메모리 구성



※ 2차원 배열 ! 선언과 동시에 초기화

중괄호 안에 또 다른 중괄호 → 2차원 이상 초기화 할 때 사용합니다.

case 1 : 행 단위로 모든 요소들을 초기화

case 2 : 행 단위로 일부 요소들만 초기화

int somang[3][3]={{1,2,3},{4,5,6,},{7,8,9}};

첫번째 중괄호 → 첫번째 행 초기확, 두번째, 세번째

int somang[3][3]= {{1},{4,5},{7,8,9}};

행의 일부만을 초기화 → 첫번째 행 초기화 요소리스트는 1개뿐, 왼쪽부터 넣고, 나머지는 0으로 채워줍니다.

Case 3 : 내부 중괄호 없이 초기화

int somang[3][3]={1,2,3 ,4,5,6, 7};

처음부터 세개 채우고, 두번째 3개 채우고, 나머지 1개 채웁니다.

◆ 포인터

※ 포인터란 무엇인가?

– 포인터와 포인터 변수

포인터라고하면 변수를 의미합니다. 포인터변수와 같은 개념입니다.

– 메모리의 주소값을 저장하기 위한 변수를 말합니다. 일반적인 변수는 데이터를 저장하기 위한 것이지만, 데이터가 아닌 주소값을 저장하면 포인터입니다.

– "포인터"를 흔히 "포인터 변수"라고 합니다.

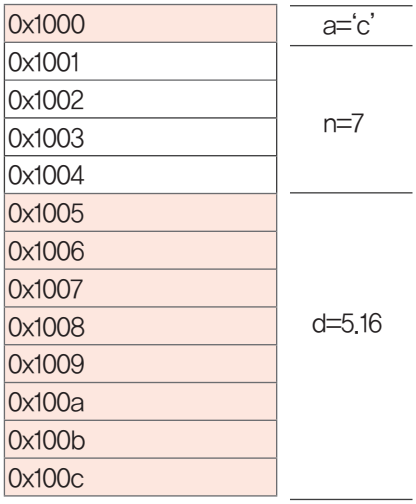
포인터도 경우에 따라 상수가 될 수도 있습니다. 하지만 상수는 매우 적습니다. 그 상수의 절반

은 상수화 변수입니다.

– 주소값과 포인터는 다릅니다.

◆ 메모리 구조 표현 방식

```
int main(void)
{
    char a='c';
    int n=4;
    double d=5.16
```



※ 포인터란 주소값을 저장하기 위한, 변수입니다.

8비트 → 1바이트 주소값

16비트 → 2바이트 주소값

32비트 → 4바이트 주소값.

역사적으로 포인터의 주소값은 변해왔습니다. 오늘날은 4바이트로 표현됩니다. → 32비트기반은 4바이트 입니다.

문제자체가 2차원적인 경우가 많습니다.

내가 해결해야할 문제는 2차원인데, 실제 구조는 1차원적이기는 하지만 2차원적으로 하는 습관을 가져야 합니다.

◆ 포인터와 배열의 관계

– 배열 이름은 첫 번째 요소의 주소값을 나타냅니다.
Int a[5]={0,1,2,3,4}
Ox1000
a[0] : a라는 배열의 0번째 요소, &a[0] : 0번째 인덱스에 해당하는 주소값을 반환 →
Ox1000
Ox1004
Ox1008
….
a 라는것은 배열 첫번째 요소의 주소를 나타냅니다.

◆ 포인터 연산과 배열

※ arr이 포인터이거나 배열이름인 경우

```
arr[i] == *(arr+i)
```

*ptr++ // *ptr의 값을 구하고 ptr을 1증가시킵니다.
(*ptr)++ // 먼저 *ptr의 값을 구한 후 *ptr의 값을 1 증가시킵니다.
*++ptr // ptr을 1증가시킨 후 *ptr을 구합니다.
++*ptr // *ptr의 값을 1증가 시킨 후 , 그 값을 결과로 취합니다.

◆ 함수포인터

함수의 이름은 함수의 위치를 가리키는 포인터입니다.
함수이름의 포인터 타입을 결정짓는 요소는 리턴형과 전달인자입니다.

```
int (*fun)(int);  // int형 매개변수 1개 받습니다.
                  // fun은 포인터 이름입니다.
                  // 리턴형은 int형입니다.
```

◆ void 포인터

어떤 형의 주소값 이라도 저장할 수 있는 포인터 입니다.

예) char c='a';
int n=10;
void* vp;
vp = &c;
vp = &n;

포인터 연산이나 값을 변경, 참조하는 것 등을 할 수 없습니다.

◆ 포인터 함수

※ Call-By-Value와 Call-By-Reference

값에 의한 참조에 의한

※ Call-By-Value

– 값의 복사에 의한 함수의 호출입니다.
– 가장 일반적인 함수 호출 형태입니다.

int main(void)
add(val1, val2)
➔ int add(int a, int b)

add 라는 함수에서 main 함수의 들어가서 val1 을 조작할 수 없습니다.
왜냐하면, 복사한 것이기 때문에, add에서만 사용이 가능합니다.

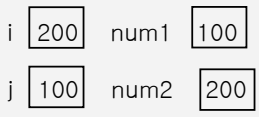
※ Call-By-Reference

– 참조(참조를 가능테 하는 주소값)를 인자로 전달하는 형태의 함수 호출입니다.
– 예를 들어 변수 val의 주소값이 Ox10 라면,adder(&val)는 val 의 주소값을 전달합니다.
. pVal =Ox10 입니다.

adder 라는 함수는 주소값을 알고 있고, (*pAvl)++; 가 가리키는 메모리 공간에 가서 1을 증가시켜라, 라는 의미입니다.

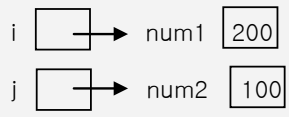
※ 값에 의한 호출 예제

```
#include <stdio.h>
void swap(int i, int j);

int main(void)
{
        int num1, num2;
        num1=100;
        num2=200;
        swap(num1, num2);
        printf(%d %d, num1, num2);
        fflush(stdout);
        return 0;
}

void swap(int i, int j)
{
        int temp;
        temp=i;  i=j;  j=temp;
}
```



※ 참조에 의한 호출 예제

```
#include <stdio.h>
void swap(int *i, int *j);

int main(void)
{
        int num1, num2;
        num1=100;
        num2=200;
        swap(&num1, &num2);
        printf(%d %d, num1, num2);
        fflush(stdout);
        return 0;
}

void swap(int *i, int *j)
{
        int temp;
        temp=*i; *i=*j; *j=temp;
}
```



◆ 함수의 매개변수

※ 함수 매개변수의 종류

- 실 매개변수

  호출 함수 쪽에서 전달하는 실제 변수입니다.

- 형식 매개 변수

  피호출함수 쪽에서 전달받는 변수입니다.

※ 함수 매개변수의 특징

- 실 매개변수와 형식매개변수의 개수는 일치합니다.
- 실 매개변수명과 형식매개변수의 이름은 같아도 상관없습니다.

※ 매개변수 전달

– 배열이 함수에 대한 인수로 사용될 때 배열의 주소만이 매개변수에게 전달 됩니다.

→ 매개변수가 포인터 형으로 선언되어야 합니다.

```
#include <stdio.h>
void a1(int num[5]), f2(int num[]), f3(int *num);

int main(void)
{
        int count[5]={1,2,3,4,5};
        a1(count);
        a2(count);
        a3(count);
        return 0;
}
```

→ 아래 세 가지 모든 경우 하나의 포인터 매개변수 사용

```
void a1(int num[5]) /*배열의 사용*/
{
        int i;
        for(i=0; i<5;i++)    printf("%d", num[i]);
                             fflush(stdout);
}
void a2(int num[])  /*언사이즈드 배열 사용*/
{
        int i;
        for(i=0; i<5;i++)    printf("%d", num[i]);
                             fflush(stdout);
}
void a3(int *num)  /*포인터를 사용*/
{
        int i;
        for(i=0; i<5;i++)   printf("%d", num[i]);
                            fflush(stdout);
}
```

◆ 재귀함수

※ 재귀함수의 기본적 이해

– 자기 자신을 다시 호출하는 형태의 함수입니다.
– 재귀함수 프로그램을 잘 이해하기 위해서는 컴퓨터 Process 를 잘 이해해야합니다.
– exe 파일 만들려면, 컴파일된 바이너리가 필요합니다.
– 프로그램이 시작되면 메모리 Code area 에 함수등 모든 것이 올라갑니다.
– 메인함수가 실행되면, 순차적으로 함수 진행됩니다. 이 코드를 가져다가 하나씩 패치를 시작하고, CPU 가 메인함수
코드를 가져다가 순차적으로 실행하는 것입니다.
– 리커시브 함수 호출 → CPU 가 리커시브 가져다가 놓고 실행합니다.
– 코드 영역에서 다시 리커시브 함수 호출하여, 가져다가 놓고 실행합니다. →계속 반복함
– 리커시브 함수는 종료되지 않고 계속 실행되고, 메모리에 계속 쌓입니다. → 문제가됨
(Stack overflow 라고 함 : 재귀적 함수가 계속쌓여서 꽉 차는 것)

※ 탈출 조건의 필요성

무한 재귀 호출을 피하기 위함입니다.

재귀함수를 공부하는 이유는, 자료구조(데이터를 표현하는 방식) 나 알고리즘(문제해결)을 공부하게 되면, 재귀함수가 매우 유용합니다. 문제해결을 고민하는데 있어서, 재귀적으로 해결하는게 너무나 많습니다.

```
if(n==1)
return;   → 1, 함수를 빠져나오는 것, 2, 값을 반환하는 것, 두가지 기능이 있습니다.
      (리턴타입이 void 라고 해도, 함수를 빠져나오는 용도로 쓰입니다.)
recursive(n-1);
```

재귀함수 할 때는 중요한 것은 탈출 조건을 달아뒀냐가 매우 중요합니다.

※ 재귀(recursive) 호출 : 순환(recursion)

: 어떤 프로시저(procedure)가 자기 자신을 호출 하는 것입니다.

```
#include <stdio.h>

int facto1(int n);

int main(void)
{
        printf("5 factorial : %d", facto1(5));
        fflush(stdout);
        return 0;
}

int facto1(int n)
{
        if(n==0)
                return 1;
        else
                return n * facto1(n-1);
}
```

```
#include <stdio.h>

int facto2(int n);

int main(void)
{
        printf("5 factorial : %d", facto2(5));
        fflush(stdout);
        return 0;
}

int facto2(int n)
{
        int i, result;
        result=1;
        if(n==0)
                return 0;
        for(i=n; i>0; i--)
                result=result *i;
        return result;
}
```

부록 1.6 구조체

◆ 구조체의 정의

– 하나(보통 둘) 이상의 기본 자료형을 기반으로 사용자 정의 자료형을 만들 수 있는 문법 요소입니다.

```
struct point   // point 라는 이름의 구조체 선언
{
Int x;       // 구조체 멤버 int x
Int y;        // 구조체 멤버 int y.
};
```

구조체 선언과 변수 선언을 동시에!!

```
struct 구조체명 {
        구조체 멤버선언;
        구조체 멤버선언;
              :
} 구조체 변수명1, 구조체변수명2…;
```

구조체 맴버

구조체 선언과 변수 선언을 따로!!

```
struct 구조체명 {
        구조체 멤버선언;
        구조체 멤버선언;
              :
};
struct 구조체명 구조체변수명1, 구조체변수명2…;
```

◆ 구조체의 특징

구조체 변수는 함수의 전달인자로 사용가능합니다.
구조체는 기본자료형과 같이 배열을 선언하여 사용할 수 있습니다.

```
예) struct phone list[10];  // 구조체 배열의 선언
    list[3].age; // list배열의 네번째배열요소의
                 // age멤버 참조
```

◆ 구조체 포인터

• 구조체는 마치 일반변수와 같이 주소 연산이 가능하고 구조체 변수 전체를 가리키는 포인터를 가질 수 있다.

```
예) 구조체 포인터 변수의 선언과 초기화
    struct score *sp = &a;

예) 구조체 포인터를 이용한 멤버 참조
    (*sp).kor;
```

◆ 간접 멤버 참조 연산자

• 간접 멤버 참조 연산자 : (–〉)

```
예) 간접 멤버 참조 연산자 사용법
    (*sp).kor  == sp–〉kor

예) lp[i].name          // 배열표현
    == (*(lp+i)).name;      // 포인터표현
    == (lp+i)–〉name;      //  간접멤버 참조 연산자를 이용한 표현
```

◆ 자기참조 구조체

● 개념

구조체 멤버 중에서 자기 자신을 가리키는 포인터를 가지고 있는 구조체

```
예) struct list{
            int I;
            struct list *ptr;
    } s1, s2, s3;

    s1.prt=&s2;
    s2.prt=&s3;
    s3.prt=&s1;
```

◆ 공용체

구조체 선언과 변수 선언을 동시에!!

```
union 공용체명 {
        공용체 멤버선언;
        공용체 멤버선언;
              :
} 공용체 변수명1, 공용체변수명2…;
```

구조체 선언과 변수 선언을 따로!!

```
union 공용체명 {
        공용체 멤버선언;
        공용체 멤버선언;
              :
};
union 공용체명 공용체변수명1, 공용체변수명2…;
```

◆ typedef

구조체, 공용체, 열거형과 같은 응용자료형을 사용할 때 길어지는 선언문을 간결하게 작성할 수 있도록 자료형의 이름을 재정의한다.

```
예) struct student{
      int num;
      double grade;
    };
    typeof struct student Student;
```

◆ 열거형

- 개념
  - 기억공간에 저장될 데이터의 집합을 정의한다.
  - 열거형 멤버들은 정수형 상수로 취급 당한다.
  - 프로그램 가독성을 높일 수 있다.
  - 특정 정수값에 의미를 부여할 수 있다.

```
예)  enum color {RED=1, GREEN=3, BLUE=5};
       enum color {sun=5, mon, tue, wed=10}
```

부록 1.7 메모리

◆ 메모리 동적할당

※ 스택, 힙 그리고 데이터 영역

- 프로그램의 실행을 위해 기본적으로 할당하는 메모리 공간입니다.
- 컴파일 타임에 함수에서 요구하는 스택의 크기 결정되어야 합니다.
- OS 가 관리해주는 메모리 구조이고, C++, JAVA 등도 실행되면, 아래와 같이 메모리 관리됨

※ 데이터 영역 : 전역변수, Static 변수 저장하기 위한 장소입니다.

※ 스택 영역 : 지역변수, 매개 변수를 저장하기 위한 장소입니다.

- 컴파일 타임에 함수에서 요구하는 스택의 크기 결정되어야 합니다.
- 반드시 컴파일 타임에 결정됩니다.

● malloc함수

heap영역에 메모리를 할당합니다.

```
void* malloc(size_t size)
```

● free함수

heap영역에 할당된 메모리를 해제합니다

```
void free(void* ptr)
```

부록2 예제 요약표

chap–ter	항목	문법	파일명	Input	Output	예제 (# 로봇동작)
1.3	C언어 훑어 보기		hhello.c		모터1개	C언어 전반적인 설명을 위한 예제 전처리기, 메인함수(변수, 반복, 분기, 함수 호출 이 포함됨), 함수 # 오른쪽 팔을 천장으로 드는 동작
2.1	변수	변수와 연산자	hvar.c		모터5개	# 모터를 하나씩 제어해서 웨이브 동작 만들기 (visual logic 예제 와 동일)
3.1	반복	while	hwhile.c		모션,소리	# 앉았다 일어나는 모션– 반복, 한번씩 일어날때마다 음악을 그 일어난 숫자 만큼 반복
3.2		do~while	hdo–while.c		모션	조건과 상관없이 반드시 한번은 실행해야하는 do~ while 문을 사용 # 모션중 원투권투 동작, 한번 반복할 때마다 좌측으로 이동, 총5번 좌측으로 이동시 동작 멈춤
3.3		for	hfor.c		모션,소리	중첩 for 문을 써서 모션과 소리가 번갈아 나오게 함 # 원투권투 동작 + 소리 2번을 10번 진행
3.4	조건 분기	if	hif.c	거리센서	모션,소리	PSD 앞에 무언가가 얼마나 가까운지를 화면상으로 출력하고 # 10 cm 이내로 접근시 권투 모션을 실행하고 종료
3.5		if~else	hifelse.c	빛센서	모터–LED	# CDS를 손가락으로 가리면 오른쪽 팔을 하늘로 올리면서 모터의 LED를 파란색으로, 떼면 반대동작으로 LED를 빨간색으로 변하게 함
3.6		if, else if, else	hifelseif.c	소리센서	모션	# 왼쪽에서 박수치면 왼쪽 손을 들고, 오른쪽에서 박수 치면 오른쪽 손을 든다. 그 외에는 차렷자세
3.7		switch ~ case	hswitch.c	키보드	모션	키보드로 모션 번호를 입력 받아 번호에 따라서 # 다른 모션을 실행함.
4.1	함수	메인함수	hfunc–main.c	리모콘	모션, 모터 LED	리모콘을 받을 수 있는 상태가 되는 함수 정의, 메인함수로 정의 # 리모콘 파워버튼을 누르면 앉았다가 일어나서 모든 모터의 LED 가 세번 깜빡임
4.2		함수호출	hfunccall.c	리모콘	모션, 소리	위 리모콘 받을 수 있는 메인 함수명을 ir–recieve 라는 함수명으로 바꾸고, 메인함수에서 호출함 # 리모콘 파워버튼을 누르면 앉았다 일어난다. 그리고 모든 모터의 LED가 세번 깜빡임

4.3		변수범위 지역/전역 변수	hfunc_scope.c	소리 센서, 리모콘	모션, 소리	전역변수_ 모션한 한 개는 어떠한 상황이라도 실행하면 그 동작을 취함—〉 차려자세 # 차렷자세를 취한 후 소리가 들리는 쪽의 손을 들어올린 후 리모컨의 OK 버튼(15)를 입력받으면 다시 차렷자세를 취함
5.1	배열	1차원 배열	harr.c	키보드	모션	char 선언후 문자열 키보드에서 입력 # ""left"" 왼팔 듦, ""right"" 오른팔 듦, ""down"" 팔 내림, ""sit"" 앉음, ""stand"" 일어남
5.2		다차원 배열	harrtwo.c	키보드	모터LED	float 2차원 배열 선언 후 자세 위치 값을 저장. 숫자 입력 받아서 # 자세를 취함
5.3	포인터	포인터란	hpointer.c	키보드	모터 LED 모니터	모터별 변수 선언, 그 변수선언된 곳 주소값 출력 # 해당 모터 LED 깜빡거림
5.4		포인터와 배열	hparr.c	키보드	모니터, 모터 LED 모션	배열의 이름과 배열의 각 원소의 주소가 어떤 관계인지 모니터 출력을 통해 알아본다. 또한 포인터 변수를 선언하고 전역 변수 배열의 주소를 포인터 변수에 넣어서 주소를 서로 비교하고, # 선언한 포인터 변수를 통해 로봇을 제어한다.
5.5		포인터 함수	hpfunc.c		모터	값에 의한 호출예제_call by value와 참조에 의한 호출 예제_ call by reference (예제 두개 제공 요망, 값호출 과 참조호출 예제) # 1.오른팔을 드는 함수 —〉 2.왼팔을 들도록 바꿈(값에의한 호출), 하지만 여전히 오른팔을 듬 –〉 3.(참조의의한 호출)이때 왼팔을 듬
6.1	구조체	구조체 정의	hstruct.c	키보드	모터	struct 로 왼팔어깨 모터 한개와 오른팔 어깨 모터 한개씩 정의해서 묶어놓고 # 왼팔 오른팔 올리기, 입력값에 따라 각도 조절
7.1	메모리	메모리 동적할당	hmalloc_free.c	리모콘	모터 LED	malloc free 함수를 써서 예제 작성 # 리모콘의 숫자 1~9 중 한개를 누르면 전체 모터 LED 가 그 숫자 만큼 반복해서 순차적으로 켜졌다 꺼짐
8.1	매크로와 전처리기		hdefine.c	키보드	모터 LED	define으로 선언한 NUMBER, MELODY에 따라, # 모션을 실행하고 MELODY 번의 멜로디를 2번 실행하는 걸 NUMBER 번 반복한다.

부록3 로봇 제어 함수

C 프로그래밍을 위해서는 동부로봇에서 제공하는 기본 API 와 로봇제어 함수를 추가하여 합니다. API 는 dll 형태로 제공하여 소스 파일을 볼 수 없으나, www.hovis.co.kr/guide 에 들어가면 PC program 탭에 API 를 클릭하면, 제어기 DRC와 모터의 모든 API 를 볼 수 있습니다. 로봇을 세부적으로 제어하고 싶으면 그 API 를 활용하여 프로그래밍할 수 있습니다.

본 교재에서 제공하는 hovis.c 는 GCC 컴파일러용으로 만든 것으로서 API 를 더욱 간략화하여 사용의 편의를 위해 만든 파일입니다. 여기서는 hovis.c 에 포함된 함수만으로 모든 제어가 가능합니다. 또한 hovis.c 를 사용자가 편의데로 고칠 수 있도록 하기위해 함수 소스를 오픈하였습니다.

교재에서 제공하는 모든 프로젝트에는 drApi.dll, drApi.h, hovis.c, hovis.h 네 개의 파일을 import 해준 후에 Build 를 해줘야 에러가 없이 Run 시킬 수 있습니다.

부록3.1 Initialize

```
/*****************************************************************
Function name : initialize

Author : Dongbu Robot
Date : 12.09.28
Arguments : const TCHAR* pszPort – 열 COM 포트의 이름(문자열)
                    DWORD dwBaudRate – 시리얼 통신 속도(기본 115200 bps)
Return : int(성공 시 1을, 실패 시 0을 리턴)
Description :
HOVIS LITE 16DOF를 사용하기 위한 초기화 함수다.
nPortNum의 번호를 가진 시리얼 포트를 nBaudRate의 속도로 열고,
모터의 방향을 설정한다.
*****************************************************************/
```

함수코드

```
int initialize(const TCHAR* pszPort, DWORD dwBaudRate)
{
        int nResult;
                                // 실행 결과를 받아올 변수 선언


        dr_common_destroy( g_hDr );
                        // 기존 g_hDr가 있을 경우 메모리 해제

        nResult = dr_common_initialize( &g_hDr );
// g_hDr의 메모리 초기화
        if(nResult != _DR_SUCCESS){
                        // 메모리 초기화를 실패한 경우
                terminate();
                                // 종료 함수를 실행
                return 0;
                                // 실패했다는 의미로 0을 반환
        }

        nResult = dr_common_connect( g_hDr, pszPort, dwBaudRate );
// g_hDr에서 pszPort 포트를 dwBaudRate의 속도로 연다.
        if(nResult != _DR_SUCCESS){
                        // 포트 열기를 실패한 경우
                terminate();
                                // 종료 함수를 실행
                return 0;
                                // 실패했다는 의미로 0을 반환
        }

        dr_motor_init(g_hDr);
                        // 모터를 제어하기 위한 메모리 초기화

        // 모터의 방향 설정
        // 좌우 대칭이므로 모터의 방향이 서로 반대이지만
        // 좌우를 같은 각도로 제어하기 위해서 정/역방향 설정을 한다.
```

```
dr_motor_set_param_dir(1, 1);
        // 오른쪽 윗 팔(1번 모터)을 역방향 설정
        dr_motor_set_param_dir(2, 1);
                // 오른쪽 아래 팔(2번 모터)을 역방향 설정
        dr_motor_set_param_dir(3, 1);
                // 왼쪽 어깨(3번 모터)를 역방향 설정
        dr_motor_set_param_dir(7, 1);
                // 오른쪽 다리 앞뒤 방향(7번 모터)을 역방향 설정
        dr_motor_set_param_dir(8, 1);
                // 오른쪽 무릎(8번 모터)을 역방향 설정
        dr_motor_set_param_dir(10, 1);
                // 오른쪽 발 좌우 방향(10번 모터)을 역방향 설정
        dr_motor_set_param_dir(11, 1);
                // 왼쪽 다리 좌우 방향(11번 모터)을 역방향 설정
        dr_motor_set_param_dir(14, 1);
                // 왼쪽 발 앞뒤 방향(14번 모터)을 역방향 설정

        return 1;
                                        // 성공했다는 의미로 1을 반환
}
```

부록3.2 Run

/***

Function name : run
Author : Dongbu Robot
Date : 12.09.28
Arguments : int nTime – 모터 명령의 실행 시간(몇 ms 동안 목표위치에 도달할지)
Return : void
Description :
HOVIS LITE 16DOF를 실제로 움직이는 함수다.
위에 선언된 g_fMotorPos[16], g_ucMotorGreenLed[16], g_ucMotorBlueLed[16],
g_ucMotorRedLed[16], g_ucMotorStop[16], g_nDrcMelody에 값을 넣고
run() 함수를 실행하면 실제로 로봇이 그 값에 따라서 동작하게 된다.
static 변수들을 선언해, 직전에 run() 함수를 호출 했을 때의
전역 변수 값들을 저장해 놓고 다음 호출 시에는
이전과 비교해 변화된 모터만 반영하여 실행한다.

단, 맨 처음 호출한 경우에는 모두 반영하여 실행한다.
nTime의 범위는 0~2844까지다. 그 이상의 시간 동안은 동작시킬 수 없으며
이 범위를 벗어난 경우는 자동으로 범위 내로 수정해서 동작된다.

**********/

함수코드

```
void run(int nTime)
{
	int i;
                                                        // for문에서 사용할 변수 선언
	static float fLastPos[16] = { 0. };
		// 직전에 실행한 위치 값을 저장할 static 변수 선언
	static unsigned char ucLastGreenLed[16] = {0};
// 직전에 실행한 녹색 LED 값을 저장할 static 변수 선언
	static unsigned char ucLastBlueLed[16] = {0};
// 직전에 실행한 청색 LED 값을 저장할 static 변수 선언
	static unsigned char ucLastRedLed[16] = {0};
// 직전에 실행한 적색 LED 값을 저장할 static 변수 선언
	static unsigned char ucLastStop[16] = {0};
	// 직전에 실행한 정지 플래그 값을 저장할 static 변수 선언
	static int nFirst = 0;
			// 맨 처음 호출된 여부를 판단하기 위한 static 변수 선언

	if(nTime 〈 0)
                                                // 만약 nTime이 0보다 작다면
		nTime = 0;
                                                        // nTime을 0으로 만든다.
	else if(nTime 〉 2844)
                                        // 만약 nTime이 2844보다 크다면
		nTime = 2844;
                                                // nTime을 2844로 만든다.

	if(nFirst == 0){
                                        // nFirst가 0인 경우(run() 함수가 처음 호출된 경우)
		dr_motor_send_driver_servo_on_off( g_hDr, 254, 1, 1 );		/  /
모든 서보 모터에 토크를 인가한다.
		dr_motor_set_command_init();
			// 명령 변수를 초기화 한다.
		for(i=0;i〈16;i++){
                                        // 0~15번 모터에 대해서 반복
			dr_motor_set_command_angle( i, g_fMotorPos[i] );
// g_fMotorPos[i]의 값으로 모터 위치 설정
```

```
                    dr_motor_set_command_flag(  i,  g_ucMotorStop[i],  0,  g_
ucMotorGreenLed[i], g_ucMotorBlueLed[i], g_ucMotorRedLed[i], 0 );

                                                        // 정지 플래그, 녹색, 청
색, 적색 LED 값을 설정

                    fLastPos[i] = g_fMotorPos[i];
          // static 변수에 현재 값 저장
                    ucLastGreenLed[i] = g_ucMotorGreenLed[i];
     // static 변수에 현재 값 저장
                    ucLastBlueLed[i] = g_ucMotorBlueLed[i];
          // static 변수에 현재 값 저장
                    ucLastRedLed[i] = g_ucMotorRedLed[i];
          // static 변수에 현재 값 저장
                    ucLastStop[i] = g_ucMotorStop[i];
          // static 변수에 현재 값 저장
          }
          dr_motor_send_move(g_hDr, nTime);
          // nTime(ms)의 시간 동안 모터 동작
          nFirst = 1;
                                                  // nFirst를 1로 만들어 줌
     }
     else{
                                                  // nFirst가 0이 아닌 경우(run() 함
수가 예전에도 호출 되었던 경우)
          dr_motor_set_command_init();
               // 명령 변수를 초기화 한다.
          for(i=0;i<16;i++){
                    // 0~15번 모터에 대해서 반복
               if(ucLastGreenLed[i] != g_ucMotorGreenLed[i] ||
// 이전 실행에 비해 녹색 LED값이 변했거나
                    ucLastBlueLed[i] != g_ucMotorBlueLed[i] ||
// 청색 LED값이 변했거나
                    ucLastRedLed[i] != g_ucMotorRedLed[i] ||
     // 적색 LED값이 변했거나
                    ucLastStop[i] != g_ucMotorStop[i] ||
     // 정지 플래그가 변했거나
                    fLastPos[i] != g_fMotorPos[i]){
          // 모터 위치가 변했다면,

                         dr_motor_set_command_angle( i, g_fMotorPos[i] );
// g_fMotorPos[i]의 값으로 모터 위치 설정
                         dr_motor_set_command_flag( i, g_ucMotorStop[i], 0,
g_ucMotorGreenLed[i], g_ucMotorBlueLed[i], g_ucMotorRedLed[i], 0 );
```

```
// 정지 플래그, 녹색, 청색, 적색 LED 값을 설정
                                        fLastPos[i] = g_fMotorPos[i];
                        // static 변수에 현재 값 저장
                                        ucLastGreenLed[i] = g_ucMotorGreenLed[i];
            // static 변수에 현재 값 저장
                                        ucLastBlueLed[i] = g_ucMotorBlueLed[i];
                        // static 변수에 현재 값 저장
                                        ucLastRedLed[i] = g_ucMotorRedLed[i];
                        // static 변수에 현재 값 저장
                                        ucLastStop[i] = g_ucMotorStop[i];
                        // static 변수에 현재 값 저장
                                    }
                        }
                        dr_motor_send_move(g_hDr, nTime);
                        // nTime(ms)의 시간 동안 모터 동작
            }

            if(g_nDrcMelody){
                                                        // 만약 g_nDrcMelody가 0이 아니
라면
                        dr_controller_send_play_ledbuzz( g_hDr, 253, 0, g_nDrcMelody );
// DRC에 Melody 실행 명령 보내기
                        g_nDrcMelody = 0;
                                                            //  g_nDrcMelody  다시
0으로 초기화
            }
}
```

부록3.3 Motion

/***

Function name : motion
Author : Dongbu Robot
Date : 12.09.28
Arguments : int nMotionNum – 실행할 모션 번호
int nReady – 모션을 준비자세만 실행할지 여부(1이 준비자세, 0이면 전부)
Return : void
Description :
HOVIS LITE 16DOF에 저장된 모션을 실행하는 함수다.
nMotionNum은 저장된 모션의 번호이고, 0~127의 값을 가진다.

nReady는 준비자세 실행 여부로, 0~1의 값을 가진다.
nReady가 1로 입력된 경우 모션의 첫 프레임만 천천히 실행한다.
nReady가 0으로 입력된 경우, 모션의 모든 프레임을 실행한다.
범위 밖의 값이 입력된 경우 실행하지 않는다.

**********/

함수코드

```
void motion(int nMotionNum, int nReady)
{
        if( nMotionNum < 0 || nMotionNum > 127 || nReady < 0 || nReady > 1 )        /   /
nMotionNum이 0~127, nReady가 0~1이 아니라면
                return;
                                                                                    // 함수를 종료
하고 리턴한다.

        dr_controller_send_play_motion( g_hDr, 253, nMotionNum, nReady);
// DRC에 모션 실행 명령 보내기
}

/*****************************************************************************
Function name : motion_wait

Author : Dongbu Robot
Date : 12.09.28
Arguments : 없음
Return : void
Description :
HOVIS LITE 16DOF에 저장된 모션을 실행 후, 모션이 끝날 때까지 기다리는 함수다.
DRC의 Playing Motion이라는 flag는 모션이 실행중일 때 1이고, 끝나면 0이 된다.
do-while문을 사용해 Playing Motion flag를 계속 읽으면서 1일 동안 계속 반복하면
0으로 되는 순간 do-while문을 빠져나가고 함수가 종료된다.
*****************************************************************************/
void motion_wait()
{
        unsigned char ucPlayingMotion;
// 모션 실행 여부를 저장할 변수 선언

        do{
```

```
// do-while문으로 반복한다.
                dr_controller_receive_ram_playing_motion( g_hDr, 253 );
                // 모션 실행 여부를 읽어오는 패킷을 보낸다.
                if( dr_wait_busy( g_hDr, 100 ) == _DR_SUCCESS ){
                        // 응답 패킷을 기다려서, 응답이 성공적으로 왔다면
                        dr_controller_get_ram_playing_motion( g_hDr, &ucPlayingMo-
tion );     // ucPlayingMotion으로 실행 여부를 복사해 온다.
                }
                else{
                        // 응답을 받는 데에 실패했다면
                        printf("Communication Error₩n"); fflush(stdout);
                        fflush(stdout);
                // 에러 메시지를 출력한다.
                }
                delay(5);
                                                                        // 5ms 동안
대기하는 함수 실행
        }while(ucPlayingMotion);
                                                        // ucPlayingMotion이 0이 아닐동
안 반복한다.
}
```

부록3.4 Read

```
/*********************************************************************

Function name : read
Author : Dongbu Robot
Date : 12.09.28
Arguments : 없음
Return : void
Description :
HOVIS LITE 16DOF의 DRC로부터 센서 값을 읽어서 센서 변수에 업데이트 한다.
read() 함수를 실행하면 g_nButton, g_nBattery, g_nAdcDist[2] 등의 전역 변수가
새로 업데이트 된 값으로 바뀌게 된다.

*********************************************************************/
```

함수코드

```
void read(){
        unsigned char aucData[100];
                                        // 모션 실행 여부를 저장할 변수 선언

        dr_controller_receive_ram( g_hDr, 253, 73, 27 );
// DRC의 여러 센서를 읽어오는 패킷을 보낸다.

        if(dr_wait_busy( g_hDr, 100 )==_DR_SUCCESS){
        // 응답 패킷을 기다려서, 응답이 성공적으로 왔다면
                dr_controller_get_ram( g_hDr, 73, 27, aucData);
// aucData에 27바이트의 센서 값들을 복사해 온다.

                g_nButton = aucData[0];
                                        // g_nButton에 버튼 값 대입
                g_nRemoconLength = aucData[1];
                        // g_nRemoconLength에 리모컨 눌린 길이 대입
                g_nRemoconData = aucData[2];
                        // g_nRemoconData에 눌린 리모컨 버튼 값 대입
                g_nBattery = aucData[3];
                        // g_nBattery에 배터리 전압 대입
                g_nTemperature = aucData[4];
                        // g_nTemperature에 온도 대입
                g_nLight = aucData[5];
                                        // g_nLight에 빛 센서 값 대입

                g_nAdcType[0] = aucData[6];
                                        // g_nAdcType[0]에 연결된 ADC 센서의 종류(좌측 포
트) 대입
                g_nAdcType[1] = aucData[7];
                                        // g_nAdcType[1]에 연결된 ADC 센서의 종류(우측 포
트) 대입
                g_nAdcDist[0] = aucData[8] | (aucData[9]<<8);
// g_nAdcDist[0]에 연결된 ADC 센서의 값(좌측 포트) 대입
                g_nAdcDist[1] = aucData[10] | (aucData[11]<<8);
// g_nAdcDist[1]에 연결된 ADC 센서의 값(우측 포트) 대입

                g_nAcc[0] = (short)(aucData[13] | (aucData[14]<<8));
// g_nAcc[0]에 가속도 센서 값(X축) 대입
                g_nAcc[1] = (short)(aucData[15] | (aucData[16]<<8));
// g_nAcc[1]에 가속도 센서 값(Y축) 대입
                g_nAcc[2] = (short)(aucData[17] | (aucData[18]<<8));
// g_nAcc[2]에 가속도 센서 값(Z축) 대입
```

```
                    g_nGyro[0] = (short)(aucData[19] | (aucData[20]〈〈8));
// g_nGyro[0]에 자이로 센서 값(X축) 대입
                    g_nGyro[1] = (short)(aucData[21] | (aucData[22]〈〈8));
// g_nGyro[1]에 자이로 센서 값(Y축) 대입
                    g_nGyro[2] = (short)(aucData[23] | (aucData[24]〈〈8));
// g_nGyro[2]에 자이로 센서 값(Z축) 대입

                    g_nSoundCount = aucData[25];
                              // g_nSoundCount에 연속적으로 소리가 들어온 횟수 대입
                    g_nSoundDirection = (char)aucData[26];
                    // g_nSoundDirection에 마지막 들어온 소리의 방향 대입
          }
          else{
                                                        // 응답을 받는 데에 실패했다면
                    printf("Communication Error₩n"); fflush(stdout);
                     fflush(stdout);
// 에러 메시지를 출력한다.
          }
}
```

부록3.5 delay

/**

Function name : delay
Author : Dongbu Robot
Date : 12.09.28
Arguments : int nTime – 대기할 시간(ms)
Return : void
Description :
nTime(ms)의 시간동안 아무 일도 하지 않고 대기한다.

**/

함수코드

```
void delay(int nTime){
        dr_wait_delay(nTime);                                   //  nTime(ms)동안  대기
하는 함수 실행
}
```

부록3.6 terminate

/***

Function name : terminate
Author : Dongbu Robot
Date : 12.09.28
Arguments : 없음
Return : void
Description :
제어를 종료하는 함수다. g_hDr 핸들의 시리얼 통신 연결을 해제하고
모터와 관련된 메모리를 해제 후 g_hDr 핸들의 메모리까지 해제 후
g_hDr 값을 NULL로 만들어 준다.

***/

함수코드

```
void terminate(){
        dr_common_disconnect( g_hDr );          // g_hDr 핸들의 시리얼 통신 연결
해제
        dr_wait_delay(100);                                     // 100ms 동안 대기
        dr_motor_destroy();                                     // 모터 제어와
관련된 메모리 해제
        dr_common_destroy( g_hDr );                             // g_hDr 핸들의 메모리
해제
        g_hDr = NULL;                                           //  g_hDr  값
NULL로 초기화
}
```

부록3.7 hovis.c

hovis.c 전체 소스입니다.

hovis.c

```
#include <stdio.h>
#include <stdlib.h>
#include "hovis.h"

float g_fMotorPos[16] = {45., 90., 0., 45., 90., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.};
unsigned char g_ucMotorGreenLed[16] = {0};
unsigned char g_ucMotorBlueLed[16] = {0};
unsigned char g_ucMotorRedLed[16] = {0};
unsigned char g_ucMotorStop[16] = {0};
int g_nDrcMelody = 0;

int g_nBattery = 0;
int g_nTemperature = 0;
int g_nLight = 0;
int g_nAdcType[2] = {0};
int g_nAdcDist[2] = {0};
int g_nAcc[3] = {0};
int g_nGyro[3] = {0};
int g_nButton = 0;
int g_nRemoconLength = 0;
int g_nRemoconData = 0xFE;
int g_nSoundCount = 0;
int g_nSoundDirection = 0;

HANDLE g_hDr = NULL;

typedef enum EHovisControllerAddr_t{
        _HOVIS_CONTROLLER_RAM_ADDR_BTN          = 73,
        _HOVIS_CONTROLLER_RAM_ADDR_RMCLENGTH    = 74,
        _HOVIS_CONTROLLER_RAM_ADDR_RMCDATA      = 75,
        _HOVIS_CONTROLLER_RAM_ADDR_BATT         = 76,
        _HOVIS_CONTROLLER_RAM_ADDR_TEMP         = 77,
        _HOVIS_CONTROLLER_RAM_ADDR_LIGHT        = 78,
        _HOVIS_CONTROLLER_RAM_ADDR_ADCTYPE1     = 79,
        _HOVIS_CONTROLLER_RAM_ADDR_ADCTYPE2     = 80,
        _HOVIS_CONTROLLER_RAM_ADDR_ADCVAL1      = 81,
```

```
	_HOVIS_CONTROLLER_RAM_ADDR_ADCVAL1		= 81,
	_HOVIS_CONTROLLER_RAM_ADDR_ADCVAL2		= 83,
	_HOVIS_CONTROLLER_RAM_ADDR_ACCGYROCON	= 85,
	_HOVIS_CONTROLLER_RAM_ADDR_ACCX			= 86,
	_HOVIS_CONTROLLER_RAM_ADDR_ACCY			= 88,
	_HOVIS_CONTROLLER_RAM_ADDR_ACCZ			= 90,
	_HOVIS_CONTROLLER_RAM_ADDR_GYROX		= 92,
	_HOVIS_CONTROLLER_RAM_ADDR_GYROY		= 94,
	_HOVIS_CONTROLLER_RAM_ADDR_GYROZ		= 96,
	_HOVIS_CONTROLLER_RAM_ADDR_SOUNDCNT		= 98,
	_HOVIS_CONTROLLER_RAM_ADDR_SOUNDDIR		= 99,

	_HOVIS_CONTROLLER_RAM_ADDR_START		= _HOVIS_CONTROL–
LER_RAM_ADDR_BTN,
	_HOVIS_CONTROLLER_RAM_ADDR_END			= _HOVIS_
CONTROLLER_RAM_ADDR_SOUNDDIR,
} EHovisControllerAddr_t;

int initialize(const TCHAR* pszPort, DWORD dwBaudRate){
	int nResult;

	dr_common_destroy( g_hDr );

	nResult = dr_common_initialize( &g_hDr );
	if(nResult != _DR_SUCCESS){
		terminate();
		return 0;
	}

	nResult = dr_common_connect( g_hDr, pszPort, dwBaudRate ) ;
	if(nResult != _DR_SUCCESS){
		terminate();
		return 0;
	}

	dr_motor_init(g_hDr);

	dr_motor_set_param_dir(1, 1);
	dr_motor_set_param_dir(2, 1);

	dr_motor_set_param_dir(3, 1);
```

```
	dr_motor_set_param_dir(7, 1);
	dr_motor_set_param_dir(8, 1);
	dr_motor_set_param_dir(10, 1);

	dr_motor_set_param_dir(11, 1);
	dr_motor_set_param_dir(14, 1);

	return 1;
}

void run(int nTime){
	int i;
	static float fLastPos[16] = { 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.};
	static unsigned char ucLastGreenLed[16] = {0};
	static unsigned char ucLastBlueLed[16] = {0};
	static unsigned char ucLastRedLed[16] = {0};
	static unsigned char ucLastStop[16] = {0};
	static int nFirst = 0;

	if(nFirst == 0){
		dr_motor_send_driver_servo_on_off( g_hDr, 254, 1, 1 );
		dr_motor_set_command_init();
		for(i=0;i<16;i++){
			dr_motor_set_command_angle( i, g_fMotorPos[i] );
			dr_motor_set_command_flag( i, g_ucMotorStop[i], 0, g_
ucMotorGreenLed[i], g_ucMotorBlueLed[i], g_ucMotorRedLed[i], 0 );
			fLastPos[i] = g_fMotorPos[i];
			ucLastGreenLed[i] = g_ucMotorGreenLed[i];
			ucLastBlueLed[i] = g_ucMotorBlueLed[i];
			ucLastRedLed[i] = g_ucMotorRedLed[i];
			ucLastStop[i] = g_ucMotorStop[i];
		}
		dr_motor_send_move(g_hDr, nTime);
		nFirst = 1;
	}
	else{
		dr_motor_set_command_init();
		for(i=0;i<16;i++){
			if(ucLastGreenLed[i] != g_ucMotorGreenLed[i]){
				dr_motor_set_command_flag( i, g_ucMotorStop[i], 0,
g_ucMotorGreenLed[i], g_ucMotorBlueLed[i], g_ucMotorRedLed[i], 1 );
				ucLastGreenLed[i] = g_ucMotorGreenLed[i];
			}
			if(ucLastBlueLed[i] != g_ucMotorBlueLed[i]){
```

```
g_ucMotorGreenLed[i], g_ucMotorBlueLed[i], g_ucMotorRedLed[i], 1 );
                                        ucLastRedLed[i] = g_ucMotorRedLed[i];
                                }
                                if(ucLastStop[i] != g_ucMotorStop[i]){
                                        dr_motor_set_command_flag( i, g_ucMotorStop[i], 0,
g_ucMotorGreenLed[i], g_ucMotorBlueLed[i], g_ucMotorRedLed[i], 1 );
                                        ucLastStop[i] = g_ucMotorStop[i];
                                }
                                if(fLastPos[i] != g_fMotorPos[i]){
                                        dr_motor_set_command_angle( i, g_fMotorPos[i] );
                                        fLastPos[i] = g_fMotorPos[i];
                                }
                        }
                        dr_motor_send_move(g_hDr, nTime);
                }

                if(g_nDrcMelody){
                        dr_controller_send_play_ledbuzz( g_hDr, 253, 0, g_nDrcMelody );
                        g_nDrcMelody = 0;
                }
}

void motion(int nMotionNum, int nReady)
{
        dr_controller_send_play_motion( g_hDr, 253, nMotionNum, nReady);
}

void motion_wait()
{
        int nResult;
        unsigned char ucPlayingMotion = 1;

        while(ucPlayingMotion){
                dr_controller_receive_ram_playing_motion( g_hDr, 253 );
                nResult = dr_wait_busy( g_hDr, 100 );
                if( nResult == _DR_SUCCESS ){
                        dr_controller_get_ram_playing_motion( g_hDr, &ucPlayingMo–
tion );
                }
                else{
                        printf("Communication Error₩n");
                        fflush(stdout);
                        break;
                }
        }
}
```

```
void read(){
	unsigned char aucData[100];

	dr_controller_receive_ram( g_hDr, 253, _HOVIS_CONTROLLER_RAM_ADDR_
START,
		_HOVIS_CONTROLLER_RAM_ADDR_END-_HOVIS_CONTROLLER_
RAM_ADDR_START+1 );

	if(dr_wait_busy( g_hDr, 100 )==_DR_SUCCESS){
		dr_controller_get_ram( g_hDr, _HOVIS_CONTROLLER_RAM_ADDR_
START,
				_HOVIS_CONTROLLER_RAM_ADDR_END-_HOVIS_CON-
TROLLER_RAM_ADDR_START+1, aucData);

		g_nBattery = aucData[_HOVIS_CONTROLLER_RAM_ADDR_BATT-_
HOVIS_CONTROLLER_RAM_ADDR_START];

		g_nTemperature = aucData[_HOVIS_CONTROLLER_RAM_ADDR_
TEMP-_HOVIS_CONTROLLER_RAM_ADDR_START];

		g_nLight = aucData[_HOVIS_CONTROLLER_RAM_ADDR_LIGHT-_
HOVIS_CONTROLLER_RAM_ADDR_START];

		g_nAdcType[0] = aucData[_HOVIS_CONTROLLER_RAM_ADDR_ADC-
TYPE1-_HOVIS_CONTROLLER_RAM_ADDR_START];
		g_nAdcType[1] = aucData[_HOVIS_CONTROLLER_RAM_ADDR_ADC-
TYPE2-_HOVIS_CONTROLLER_RAM_ADDR_START];
		g_nAdcDist[0] = aucData[_HOVIS_CONTROLLER_RAM_ADDR_AD-
CVAL1-_HOVIS_CONTROLLER_RAM_ADDR_START] |
					(aucData[_HOVIS_CONTROL-
LER_RAM_ADDR_ADCVAL1+1-_HOVIS_CONTROLLER_RAM_ADDR_START]<<8);
		g_nAdcDist[1] = aucData[_HOVIS_CONTROLLER_RAM_ADDR_AD-
CVAL2-_HOVIS_CONTROLLER_RAM_ADDR_START] |
					(aucData[_HOVIS_CONTROL-
LER_RAM_ADDR_ADCVAL2+1-_HOVIS_CONTROLLER_RAM_ADDR_START]<<8);

		if(aucData[_HOVIS_CONTROLLER_RAM_ADDR_ACCGYROCON-_
HOVIS_CONTROLLER_RAM_ADDR_START]==1){
			g_nAcc[0] = aucData[_HOVIS_CONTROLLER_RAM_ADDR_
ACCX-_HOVIS_CONTROLLER_RAM_ADDR_START];
			g_nAcc[1] = aucData[_HOVIS_CONTROLLER_RAM_ADDR_
ACCY-_HOVIS_CONTROLLER_RAM_ADDR_START];
```

```
                        g_nAcc[2] = aucData[_HOVIS_CONTROLLER_RAM_ADDR_
ACCZ-_HOVIS_CONTROLLER_RAM_ADDR_START];
                        g_nGyro[0] = aucData[_HOVIS_CONTROLLER_RAM_ADDR_
GYROX-_HOVIS_CONTROLLER_RAM_ADDR_START];
                        g_nGyro[1] = aucData[_HOVIS_CONTROLLER_RAM_ADDR_
GYROY-_HOVIS_CONTROLLER_RAM_ADDR_START];
                        g_nGyro[2] = aucData[_HOVIS_CONTROLLER_RAM_ADDR_
GYROZ-_HOVIS_CONTROLLER_RAM_ADDR_START];
                }
                else{
                        g_nAcc[0] = 0;
                        g_nAcc[1] = 0;
                        g_nAcc[2] = 0;
                        g_nGyro[0] = 0;
                        g_nGyro[1] = 0;
                        g_nGyro[2] = 0;
                }

                g_nButton = aucData[_HOVIS_CONTROLLER_RAM_ADDR_BTN-_
HOVIS_CONTROLLER_RAM_ADDR_START];
                g_nRemoconLength = aucData[_HOVIS_CONTROLLER_RAM_ADDR_
RMCLENGTH-_HOVIS_CONTROLLER_RAM_ADDR_START];
                g_nRemoconData = aucData[_HOVIS_CONTROLLER_RAM_ADDR_RM-
CDATA-_HOVIS_CONTROLLER_RAM_ADDR_START];
                g_nSoundCount = aucData[_HOVIS_CONTROLLER_RAM_ADDR_
SOUNDCNT-_HOVIS_CONTROLLER_RAM_ADDR_START];
                g_nSoundDirection = (char)aucData[_HOVIS_CONTROLLER_RAM_
ADDR_SOUNDDIR-_HOVIS_CONTROLLER_RAM_ADDR_START];
        }
        else{
                printf("Communication Error₩n");
                 fflush(stdout);
        }
}

void delay(int nTime){
        dr_wait_delay(nTime);
}

void terminate(){
        dr_common_disconnect( g_hDr );
        dr_wait_delay(100);
        dr_motor_destroy();
        dr_common_destroy( g_hDr );
}
```

부록3.8 drApi.h

drApi.h 설명 소스입니다.

drApi.h

```
/*
        파일 명                    : hovis.h
        제어로봇 형태      : 16dof 휴머노이드
        설명                       : HOVIS Lite 16DOF를 제어하기 위한 모듈화된 함수를
제공하는 파일이다.
                                                예제 파일에서 접근 가능한 전역 변수와, 그
전역 변수를 사용해서 로봇을 제어하는 함수들을 제공한다.
*/

// hovis.h를 한 번만 컴파일 하도록 하는 부분
#ifndef _HOVIS_H                                        // _HOVIS_H라는 기호가 정의되지
않았었다면 맨 밑의 #endif 까지를 컴파일한다.
#define _HOVIS_H                                        // _HOVIS_H라는 기호를 정의한
다.

#ifdef __cplusplus                                      // C++일 경우 C로 작성된 함수는
그냥 불러올 수 없다.
extern "C" {                                            // 따라서 extern "C"를
선언해 괄호 안의 부분을 C 형식으로 컴파일 하라고 지시한다.
#endif                                                  // #ifdef __
cplusplus 와 대응되는 #endif

#include "drApi.h"                                      // 동부로봇의 api 를 사용하기위해
반드시 이렇게 선언 해 두어야 한다.

// 다른 소스(hovis.c)에 선언된 전역 변수를 접근하기 위해 선언
// 제어 변수
extern float g_fMotorPos[16];                           // 모터의 위치(도(de-
gree) 기준)
extern unsigned char g_ucMotorGreenLed[16];             // 모터의 녹색 LED 점등 여부(1/0)
extern unsigned char g_ucMotorBlueLed[16];              // 모터의 청색 LED 점등 여부(1/0)
extern unsigned char g_ucMotorRedLed[16];               // 모터의 적색 LED 점등 여부(1/0)
extern unsigned char g_ucMotorStop[16];                 // 모터를 즉시 멈추는 플
래그 사용 여부(1/0)
extern int g_nDrcMelody;                                // DRC에서
울렸으면 하는 멜로디 번호
```

```
// 센서 변수
extern int g_nBattery;                                  // 배터리 전압
값
extern int g_nTemperature;                              // 현재 온도 값
extern int g_nLight;                                    // 빛 센서 값
extern int g_nAdcType[2];                               //       연결된
ADC 센서의 종류
extern int g_nAdcDist[2];                               //       연결된
ADC 센서의 값
extern int g_nAcc[3];                                   // 연결된 가속
도/자이로 센서의 가속도 값
extern int g_nGyro[3];                                  // 연결된 가속
도/자이로 센서의 자이로
extern int g_nButton;                                   // 버튼 값
extern int g_nRemoconLength;                            // 리모컨 버튼
이 눌린 길이
extern int g_nRemoconData;                              // 눌린 리모컨
버튼 값
extern int g_nSoundCount;                               //  연속적으로
소리가 들어온 횟수
extern int g_nSoundDirection;                           // 마지막 들어
온 소리의 방향(-2~2, -가 왼쪽)

// 모듈화 된 함수
int initialize(const TCHAR* pszPort, DWORD dwBaudRate);
void run(int nTime);                                    //  명령  실행
함수
void motion(int nMotionNum, int nReady);                // 모션 실행 함수
void motion_wait();                                     // 모션 실행이
끝날 때 까지 기다리는 함수
void read();
// 센서 값을 읽어오는 함수
void delay(int nTime);                                  // 실행을 지연
하는 함수
void terminate();                                       // 종료하는 함
수

#ifdef __cplusplus                                      // C++일 경우
}                                                       // extern "C"
를 선언하며 열은 괄호를 닫는다.
#endif                                                  //  #ifdef  __
cplusplus 와 대응되는 #endif

#endif                                                  //  #ifndef  _
HOVIS_H 와 대응되는 #endif
```

부록3.9 hovis.h

hovis.h 설명 소스입니다.

hovis.h

```
#ifndef _DRAPI_H
#define _DRAPI_H

#ifdef __cplusplus
extern "C" {
#endif

// 이 클래스는 drApi.dll에서 내보낸 것입니다.
#include <windows.h>
#include <limits.h>

typedef enum EDrModule_t{
        _DR_MODULE_NONE                                         = 0,
        _DR_MODULE_MOTOR,
        _DR_MODULE_HEADLED,
        _DR_MODULE_CONTROLLER,
        _DR_MODULE_SENSOR,

        _DR_MODULE_MIN              = _DR_MODULE_MOTOR,
        _DR_MODULE_MAX                      = _DR_MODULE_SENSOR,
} EDrModule_t;

typedef enum EDrResult_t
{
        _DR_SUCCESS
        = 0,
        _DR_NULL_HANDLE,
        _DR_COM_OPEN_FAIL,
        _DR_NULL_DATA_ARG,
        _DR_WRONG_AXIS,

        _DR_SEND_INVALID_ARG_MODULE                             = 10,
        _DR_SEND_INVALID_ARG_SIZE,
        _DR_SEND_INVALID_ARG_ID,
        _DR_SEND_INVALID_ARG_CMD,
        _DR_SEND_INVALID_ARG_DATA,
        _DR_SEND_FAIL,
```

```
		_DR_RECV_ERR_TIMEOUT					= 20,
		_DR_RECV_ERR_CHECKSUM,
		_DR_RECV_ERR_PACKET_SIZE_TOO_BIG,
		_DR_RECV_ERR_MOTOR_INVALID_CMD,
		_DR_RECV_ERR_MOTOR_INVALID_ID,
		_DR_RECV_ERR_MOTOR_INVALID_SIZE,
		_DR_RECV_ERR_MOTOR_INVALID_REG_RANGE,
		_DR_RECV_ERR_HEADLED_INVALID_CMD,
		_DR_RECV_ERR_HEADLED_INVALID_ID,
		_DR_RECV_ERR_HEADLED_INVALID_SIZE,
		_DR_RECV_ERR_HEADLED_INVALID_REG_RANGE,
		_DR_RECV_ERR_CONTROLLER_INVALID_CMD,
		_DR_RECV_ERR_CONTROLLER_INVALID_ID,
		_DR_RECV_ERR_CONTROLLER_INVALID_SIZE,
		_DR_RECV_ERR_CONTROLLER_INVALID_REG_RANGE,
		_DR_RECV_ERR_SENSOR_INVALID_CMD,
		_DR_RECV_ERR_SENSOR_INVALID_ID,
		_DR_RECV_ERR_SENSOR_INVALID_SIZE,
		_DR_RECV_ERR_SENSOR_INVALID_REG_RANGE,
} EDrResult_t;

// << Common >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
//
__declspec(dllimport) EDrResult_t	dr_common_initialize( HANDLE* hDr );
__declspec(dllimport) EDrResult_t	dr_common_destroy( HANDLE hDr );
__declspec(dllimport) EDrResult_t	dr_common_get_connection( HANDLE hDr, BOOL* pbConnection );
__declspec(dllimport) EDrResult_t	dr_common_connect( HANDLE hDr, const TCHAR* pszPort, DWORD dwBaudRate );
__declspec(dllimport) EDrResult_t	dr_common_disconnect( HANDLE hDr );
__declspec(dllimport) EDrResult_t	dr_common_send( HANDLE hDr, EDrModule_t ETargetModule, BYTE byPacketSize, BYTE byId, BYTE byCmd, BYTE* pbyData );
// << Controller >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> //
///////////////////////////////////////////////////////////////////////////////
// Controller
///////////////////////////////////////////////////////////////////////////////
typedef enum EDrControllerAxis_t
{
		_DR_CONTROLLER_AXIS_X,
		_DR_CONTROLLER_AXIS_Y,
		_DR_CONTROLLER_AXIS_Z,
		_DR_CONTROLLER_AXIS_ALL

} EDrControllerAxis_t;
```

```
__declspec(dllimport) EDrResult_tdr_controller_receive_rom( HANDLE hDr, BYTE byId, BYTE byAddress,
BYTE byLength );
__declspec(dllimport) EDrResult_tdr_controller_get_rom( HANDLE hDr, BYTE byAddress, BYTE byLength,
BYTE* pbyData );
__declspec(dllimport) EDrResult_tdr_controller_send_rom( HANDLE hDr, BYTE byId, BYTE byAddress,
BYTE byLength, BYTE* pbyData );
__declspec(dllimport) EDrResult_tdr_controller_receive_ram( HANDLE hDr, BYTE byId, BYTE byAddress,
BYTE byLength );
__declspec(dllimport) EDrResult_tdr_controller_get_ram( HANDLE hDr, BYTE byAddress, BYTE byLength,
BYTE* pbyData );
__declspec(dllimport) EDrResult_tdr_controller_send_ram( HANDLE hDr, BYTE byId, BYTE byAddress,
BYTE byLength, BYTE* pbyData );
__declspec(dllimport) EDrResult_tdr_controller_receive_connection_check( HANDLE hDr, BYTE byId,
BYTE byLength, BYTE* pbyMotorId );
__declspec(dllimport) EDrResult_tdr_controller_get_connection_check( HANDLE hDr, BYTE* pbyLength,
BYTE* pbyData );
__declspec(dllimport) EDrResult_tdr_controller_send_play_motion( HANDLE hDr, BYTE byId, BYTE
byMotion, BYTE byReady );
__declspec(dllimport) EDrResult_tdr_controller_send_stop_motion( HANDLE hDr, BYTE byId );
__declspec(dllimport) EDrResult_tdr_controller_send_play_task( HANDLE hDr, BYTE byId );
__declspec(dllimport) EDrResult_tdr_controller_send_stop_task( HANDLE hDr, BYTE byId );
__declspec(dllimport) EDrResult_tdr_controller_receive_debug_task_start( HANDLE hDr, BYTE byId );
__declspec(dllimport) EDrResult_tdr_controller_receive_debug_task_step( HANDLE hDr, BYTE byId );
__declspec(dllimport) EDrResult_t dr_controller_get_debug_task_counter( HANDLE hDr, WORD* pwData );
__declspec(dllimport) EDrResult_tdr_controller_send_play_ledbuzz( HANDLE hDr, BYTE byId, BYTE
byLed, BYTE byBuzz );
__declspec(dllimport) EDrResult_tdr_controller_receive_stat( HANDLE hDr, BYTE byId );
__declspec(dllimport) EDrResult_tdr_controller_get_stat( HANDLE hDr, BYTE* pbyData );
__declspec(dllimport) EDrResult_tdr_controller_send_rollback( HANDLE hDr, BYTE byId, BYTE byIdSkip,
BYTE byBaudSkip );
__declspec(dllimport) EDrResult_tdr_controller_send_reboot( HANDLE hDr, BYTE byId );
__declspec(dllimport) EDrResult_tdr_controller_receive_zigbee( HANDLE hDr, BYTE byId, BYTE byInst );
__declspec(dllimport) EDrResult_tdr_controller_get_zigbee( HANDLE hDr, BYTE* pbyData );
__declspec(dllimport) EDrResult_tdr_controller_send_remocon( HANDLE hDr, BYTE byId, BYTE
byChannel, BYTE byLength, BYTE byData );
__declspec(dllimport) EDrResult_tdr_controller_send_fw_update( HANDLE hDr, BYTE byId );

__declspec(dllimport) EDrResult_tdr_controller_receive_rom_version( HANDLE hDr, BYTE byId );
__declspec(dllimport) EDrResult_tdr_controller_get_rom_version( HANDLE hDr, BYTE* pbyData );
__declspec(dllimport) EDrResult_tdr_controller_receive_rom_id( HANDLE hDr, BYTE byId );
__declspec(dllimport) EDrResult_tdr_controller_get_rom_id( HANDLE hDr, BYTE* pbyData );
__declspec(dllimport) EDrResult_tdr_controller_send_rom_id( HANDLE hDr, BYTE byId, BYTE byNewId );
__declspec(dllimport) EDrResult_tdr_controller_receive_ram_id( HANDLE hDr, BYTE byId );
__declspec(dllimport) EDrResult_tdr_controller_get_ram_id( HANDLE hDr, BYTE* pbyData );
__declspec(dllimport) EDrResult_tdr_controller_send_ram_id( HANDLE hDr, BYTE byId, BYTE byNewId );
```

```
__declspec(dllimport) EDrResult_tdr_controller_receive_rom_rmc_channel( HANDLE hDr, BYTE byId );
__declspec(dllimport) EDrResult_tdr_controller_get_rom_rmc_channel( HANDLE hDr, BYTE* pbyData );
__declspec(dllimport) EDrResult_tdr_controller_send_rom_rmc_channel( HANDLE hDr, BYTE byId, BYTE
byNewChannel );
__declspec(dllimport) EDrResult_tdr_controller_receive_ram_rmc_channel( HANDLE hDr, BYTE byId );
__declspec(dllimport) EDrResult_tdr_controller_get_ram_rmc_channel( HANDLE hDr, BYTE* pbyData );
__declspec(dllimport) EDrResult_tdr_controller_send_ram_rmc_channel( HANDLE hDr, BYTE byId, BYTE
byNewChannel );
__declspec(dllimport) EDrResult_tdr_controller_receive_ram_status( HANDLE hDr, BYTE byId );
__declspec(dllimport) EDrResult_tdr_controller_get_ram_status( HANDLE hDr, BYTE* pbyData );
__declspec(dllimport) EDrResult_tdr_controller_clear_ram_status( HANDLE hDr, BYTE byId );
__declspec(dllimport) EDrResult_tdr_controller_receive_ram_led_control( HANDLE hDr, BYTE byId );
__declspec(dllimport) EDrResult_tdr_controller_get_ram_led_control( HANDLE hDr, BYTE* pbyData );
__declspec(dllimport) EDrResult_tdr_controller_send_ram_led_control( HANDLE hDr, BYTE byId, BYTE
byNewControl );
__declspec(dllimport) EDrResult_tdr_controller_receive_ram_zigbee_channel( HANDLE hDr, BYTE byId );
__declspec(dllimport) EDrResult_t dr_controller_get_ram_zigbee_channel( HANDLE hDr, BYTE* pbyData );
__declspec(dllimport) EDrResult_tdr_controller_send_ram_zigbee_channel( HANDLE hDr, BYTE byId,
BYTE byNewChannel );
__declspec(dllimport) EDrResult_tdr_controller_receive_ram_zigbee_panid( HANDLE hDr, BYTE byId );
__declspec(dllimport) EDrResult_tdr_controller_get_ram_zigbee_panid( HANDLE hDr, BYTE* pbyData );
__declspec(dllimport) EDrResult_tdr_controller_send_ram_zigbee_panid( HANDLE hDr, BYTE byId,
WORD wNewPANID );
__declspec(dllimport) EDrResult_tdr_controller_receive_ram_zigbee_saddr( HANDLE hDr, BYTE byId );
__declspec(dllimport) EDrResult_tdr_controller_get_ram_zigbee_saddr( HANDLE hDr, BYTE* pbyData );
__declspec(dllimport) EDrResult_tdr_controller_send_ram_zigbee_saddr( HANDLE hDr, BYTE byId,
WORD wNewSADDR );
__declspec(dllimport) EDrResult_tdr_controller_receive_ram_zigbee_dstaddr( HANDLE hDr, BYTE byId );
__declspec(dllimport) EDrResult_tdr_controller_get_ram_zigbee_dstaddr( HANDLE hDr, BYTE* pbyData );
__declspec(dllimport) EDrResult_tdr_controller_send_ram_zigbee_dstaddr( HANDLE hDr, BYTE byId,
WORD wNewDSTADDR );
__declspec(dllimport) EDrResult_tdr_controller_receive_ram_zigbee_ackreq( HANDLE hDr, BYTE byId );
__declspec(dllimport) EDrResult_tdr_controller_get_ram_zigbee_ackreq( HANDLE hDr, BYTE*
pbyData );
__declspec(dllimport) EDrResult_tdr_controller_send_ram_zigbee_ackreq( HANDLE hDr, BYTE byId,
BYTE byNewACKREQ );
__declspec(dllimport) EDrResult_tdr_controller_receive_ram_zigbee_backoff( HANDLE hDr, BYTE byId );
__declspec(dllimport) EDrResult_tdr_controller_get_ram_zigbee_backoff( HANDLE hDr, BYTE* pbyData );
__declspec(dllimport) EDrResult_tdr_controller_send_ram_zigbee_backoff( HANDLE hDr, BYTE byId,
BYTE byNewBACKOFF );
```

```
__declspec(dllimport) EDrResult_tdr_controller_receive_ram_servo_count( HANDLE hDr, BYTE byId );
__declspec(dllimport) EDrResult_tdr_controller_get_ram_servo_count( HANDLE hDr, BYTE* pbyData );
__declspec(dllimport) EDrResult_tdr_controller_receive_ram_servo_id( HANDLE hDr, BYTE byId );
__declspec(dllimport) EDrResult_tdr_controller_get_ram_servo_id( HANDLE hDr, BYTE* pbyData );
__declspec(dllimport) EDrResult_tdr_controller_receive_ram_playing_motion( HANDLE hDr, BYTE byId );
__declspec(dllimport) EDrResult_tdr_controller_get_ram_playing_motion( HANDLE hDr, BYTE* pbyData );
__declspec(dllimport) EDrResult_tdr_controller_receive_ram_playing_task( HANDLE hDr, BYTE byId );
__declspec(dllimport) EDrResult_tdr_controller_get_ram_playing_task( HANDLE hDr, BYTE* pbyData );
__declspec(dllimport) EDrResult_tdr_controller_receive_ram_button_status( HANDLE hDr, BYTE byId );
__declspec(dllimport) EDrResult_tdr_controller_get_ram_button_status( HANDLE hDr, BYTE* pbyData );
__declspec(dllimport) EDrResult_tdr_controller_receive_ram_rmc_length( HANDLE hDr, BYTE byId );
__declspec(dllimport) EDrResult_tdr_controller_get_ram_rmc_length( HANDLE hDr, BYTE* pbyData );
__declspec(dllimport) EDrResult_tdr_controller_receive_ram_rmc_data( HANDLE hDr, BYTE byId );
__declspec(dllimport) EDrResult_tdr_controller_get_ram_rmc_data( HANDLE hDr, BYTE* pbyData );
__declspec(dllimport) EDrResult_tdr_controller_receive_ram_voltage( HANDLE hDr, BYTE byId );
__declspec(dllimport) EDrResult_tdr_controller_get_ram_voltage( HANDLE hDr, BYTE* pbyData );
__declspec(dllimport) EDrResult_tdr_controller_receive_ram_temperature( HANDLE hDr, BYTE byId );
__declspec(dllimport) EDrResult_tdr_controller_get_ram_temperature( HANDLE hDr, BYTE* pbyData );
__declspec(dllimport) EDrResult_tdr_controller_receive_ram_light_sensor( HANDLE hDr, BYTE byId );
__declspec(dllimport) EDrResult_tdr_controller_get_ram_light_sensor( HANDLE hDr, BYTE* pbyData );
__declspec(dllimport) EDrResult_tdr_controller_receive_ram_adc1_type( HANDLE hDr, BYTE byId );
__declspec(dllimport) EDrResult_tdr_controller_get_ram_adc1_type( HANDLE hDr, BYTE* pbyData );
__declspec(dllimport) EDrResult_tdr_controller_receive_ram_adc2_type( HANDLE hDr, BYTE byId );
__declspec(dllimport) EDrResult_tdr_controller_get_ram_adc2_type( HANDLE hDr, BYTE* pbyData );
__declspec(dllimport) EDrResult_tdr_controller_receive_ram_adc1_value( HANDLE hDr, BYTE byId );
__declspec(dllimport) EDrResult_tdr_controller_get_ram_adc1_value( HANDLE hDr, BYTE* pbyData );
__declspec(dllimport) EDrResult_tdr_controller_receive_ram_adc2_value( HANDLE hDr, BYTE byId );
__declspec(dllimport) EDrResult_tdr_controller_get_ram_adc2_value( HANDLE hDr, BYTE* pbyData );
__declspec(dllimport) EDrResult_tdr_controller_receive_ram_accgyro_connected( HANDLE hDr, BYTE
byId );
__declspec(dllimport) EDrResult_tdr_controller_get_ram_accgyro_connected( HANDLE hDr, BYTE*
pbyData );
__declspec(dllimport) EDrResult_tdr_controller_receive_ram_acc_value( HANDLE hDr, BYTE byId,
EDrControllerAxis_t EDrAxis );
__declspec(dllimport) EDrResult_tdr_controller_get_ram_acc_value( HANDLE hDr, BYTE* pbyData,
EDrControllerAxis_t EDrAxis );
__declspec(dllimport) EDrResult_tdr_controller_receive_ram_gyro_value( HANDLE hDr, BYTE byId,
EDrControllerAxis_t EDrAxis );
__declspec(dllimport) EDrResult_tdr_controller_get_ram_gyro_value( HANDLE hDr, BYTE* pbyData,
EDrControllerAxis_t EDrAxis );
```

```
__declspec(dllimport) EDrResult_tdr_controller_receive_ram_sound_detection( HANDLE hDr, BYTE byId );
__declspec(dllimport) EDrResult_tdr_controller_get_ram_sound_detection( HANDLE hDr, BYTE* pbyData
);
__declspec(dllimport) EDrResult_tdr_controller_receive_ram_sound_direction( HANDLE hDr, BYTE byId );
__declspec(dllimport) EDrResult_tdr_controller_get_ram_sound_direction( HANDLE hDr, BYTE* pbyData
);

// << Timer >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> //
/* 주의 – 반드시 Init 을 해야 사용 가능하다 – */
/////////////////////////////////////////////////////////////////////////////////
// Timer(Init, Destroy, Set, Get) – Init 을 해야 쓸 수 있는 함수들...
/////////////////////////////////////////////////////////////////////////////////
__declspec(dllimport) char                              dr_wait_timer_init(int nCnt);
// 내가 사용할 만큼(nCnt)의 타이머를 생성한다. (ex:nCnt = 3 -> 0 ~ 2 까지의 타이머 사용가능)
__declspec(dllimport) void                              dr_wait_timer_destroy();
// 타이머 메모리 해제 – 이제부터 작동 안됨
__declspec(dllimport) void                              dr_wait_timer_set(int nIndex);
// timer_get 과 쌍으로 set 한 시점에서 시간을 0 으로 본다.
__declspec(dllimport) unsigned long          dr_wait_timer_get(int nIndex);
// timer_set 한 후 얼마나 지났는가를 확인하는 타이머
__declspec(dllimport) char                              dr_wait_timer_check(int nIndex, unsigned long t);
// timer_set 한 후 지정한 시간이 되면 true 를 반환, 단, 에러 발생시 혹은 시간이 안되면 false

/* Init 할 필요 없다. */
/////////////////////////////////////////////////////////////////////////////////
// Time(년,월,일,시,분,초,현재시간 등
// 기준 시간은 1970년 1월 1일 0시 0분 0초 부터... (유닉스 타임)
/////////////////////////////////////////////////////////////////////////////////
__declspec(dllimport) unsigned long          dr_wait_timer_curr();
// 현재의 시간을 ms 로 리턴하는 함수
__declspec(dllimport) int                               dr_wait_timer_year();
// 컴퓨터의 year 값 리턴(4자리 ex:2012)
__declspec(dllimport) int                               dr_wait_timer_month();
```

```
__declspec(dllimport) int                               dr_wait_timer_day();
// 컴퓨터의 날짜 값 리턴
__declspec(dllimport) int                               dr_wait_timer_week();
// 컴퓨터의 요일 값 리턴(일(0), 월(1), 화(2), 수(3), 목(4), 금(5), 토(6))
__declspec(dllimport) int                               dr_wait_timer_hour();
// 컴퓨터의 시간 값 리턴
__declspec(dllimport) int                               dr_wait_timer_minute();
// 컴퓨터의 minute 값 리턴
__declspec(dllimport) int                               dr_wait_timer_second();
// 컴퓨터의 second 값 리턴
__declspec(dllimport) int                               dr_wait_timer_millisecond();
// 컴퓨터의 millisecond 값 리턴

/* 주의 – dr_wait_break 를 사용 시 반드시 dr_wait_reset 을 해야만 다시 사용이 가능하다 – */
//////////////////////////////////////////////////////////////////////////////
// Delay
//////////////////////////////////////////////////////////////////////////////
__declspec(dllimport) void                              dr_wait_delay_break();
// 모든 대기(wait – dr_wait_delay) 함수를 탈출하도록 한다.
__declspec(dllimport) void                              dr_wait_delay_break_once();
// 동작하고 있는 대기(wait – dr_wait_delay) 함수를 한번 탈출하도록 한다
__declspec(dllimport) void                              dr_wait_delay_reset();
// break 명령으로 사용할 수 없게 된 delay 함수를 다시 사용 가능하게 해 준다.
__declspec(dllimport) void                              dr_wait_delay(unsigned long);
// 지정한 시간(ms) 만큼 함수를 대기시킨다.

//////////////////////////////////////////////////////////////////////////////
// Wait Busy
//////////////////////////////////////////////////////////////////////////////
__declspec(dllimport) EDrResult_tdr_wait_busy( HANDLE hDr, DWORD dwMilliSecond );
// << Motor >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> //
//////// Data ////////
//////////////////////////////////////////////////////////////////////////////
// 초기화 함수
//////////////////////////////////////////////////////////////////////////////
```

```
__declspec(dllimport) EDrResult_t dr_motor_init(HANDLE hDr);
                                                    // 모터 제어 전 반드시 한번 호출 하는
함수로 최초 한번만 실행해 주면 된다.
__declspec(dllimport) void dr_motor_destroy();

// 모터 제어를 종료할 때, 혹은 프로그램을 종료하기 전에 반드시 한번 호출 해 주어야 하는 함수
////////////////////////////////////////////////////////////////////////////
// 파라미터 설정 함수
////////////////////////////////////////////////////////////////////////////
// 실제 ID 에 가상의 AliasID 를 부여
__declspec(dllimport) void dr_motor_set_param_alias_id(int nID, int nAliasID);
                                    // 특정 아이디에 AliasID 를 부여하는 함수
__declspec(dllimport) int dr_motor_get_param_alias_id_by_id(int nID);
// 해당 아이디에 셋 되어 있는 AliasID를 가져오는 함수
__declspec(dllimport) void dr_motor_set_param_dir(int nID, char cReverse);
                                        // cReverse = 0(정방향), cReverse = 1(역방향)
__declspec(dllimport) char dr_motor_get_param_dir(int nID);
// return value = 0(정방향), return value = 1(역방향)
__declspec(dllimport) int dr_motor_get_param_id_by_alias_id(int nAliasID);
                                        // AliasID 를 이용해서 실제 아이디 값을 가져오기

////////////////////////////////////////////////////////////////////////////
// motor 구동 함수 모음(led(default=off), mode(default=위치제어), stop(default=false), evd/angle,
// noaction(default=true) => 단, dr_motor_set_command_position/angle 의 위치제어함수를 사용시
자동으로 NoAction = false 가 되어 모터가 동작한다.(즉, 위치값을 넣는 순간 모터는 동작 가능상태가
된다.)
////////////////////////////////////////////////////////////////////////////
// 구동시 초기화 함수
__declspec(dllimport) void dr_motor_set_command_init();
// 모터제어 명령 전에 반드시 호출해 주어야 하는 함수. 이 함수를 호출하면 변수전체가 리셋되고, Stop
명령의 플래그가 클리어 된다.
// Data
__declspec(dllimport) void dr_motor_set_command_position(int nID, int nPos);
                                    // Evd 위치제어시 사용하는 함수
__declspec(dllimport) void dr_motor_set_command_angle(int nID, float fAngle);
                                        // 각도를 이용한 위치제어시 사용하는 함수
```

```
__declspec(dllimport) int dr_motor_get_command_position(int nID);
// 현재 명령이 내려진 Evd 값을 읽어온다.
__declspec(dllimport) float dr_motor_get_command_angle(int nID);
// 현재 명령이 내려진 Angle 값을 읽어온다.
__declspec(dllimport) void dr_motor_set_command_flag(int nID, BOOL bStop, BOOL bMode_Speed,
BOOL bLed_Green, BOOL bLed_Blue, BOOL bLed_Red, BOOL bNoAction); // flag 설정을 한번에 전부
할수 있는 명령(정지, 모드설정, LED, NoAction)
__declspec(dllimport) int dr_motor_get_command_flag(int nID);
/ 현재 설정된 flag 설정을 한번에 가져오는 명령어
__declspec(dllimport) void dr_motor_set_command_flag_stop(int nID, BOOL bStop);
// flag 명령: 모터동작 지령전 이 함수를 set(bStop = true) 하면 모터의 지령이 "정지"로 변경된다.(단,
m_bStop 내부변수가 셋되는 명령이 아니다. 정지도 하나의 제어로 생각하는 함수)
__declspec(dllimport) void dr_motor_set_command_flag_mode(int nID, BOOL bMode_Speed);
// flag 명령: 모터동작 지령전 이 함수를 call 하면 모터의 제어상태(0–위치, 1–속도)가 바뀐다.
__declspec(dllimport) void dr_motor_set_command_flag_led(int nID, BOOL bGreen, BOOL bBlue, BOOL
bRed);      // flag 명령: 모터동작 지령전 이 함수를 call 하면 모터의 LED 상태를 변경할 수 있다.
__declspec(dllimport) void dr_motor_set_command_flag_led_green(int nID, BOOL bGreen);
// flag 명령: 모터동작 지령전 이 함수를 call 하면 모터의 LED 상태(녹색)를 변경할 수 있다.
__declspec(dllimport) void dr_motor_set_command_flag_led_blue(int nID, BOOL bBlue);
// flag 명령: 모터동작 지령전 이 함수를 call 하면 모터의 LED 상태(청색)를 변경할 수 있다.
__declspec(dllimport) void dr_motor_set_command_flag_led_red(int nID, BOOL bRed);
// flag 명령: 모터동작 지령전 이 함수를 call 하면 모터의 LED 상태(적색)를 변경할 수 있다.
__declspec(dllimport) void dr_motor_set_command_flag_noaction(int nID, BOOL bNoAction);
// flag 명령: 모터동작 지령전 이 함수를 set(bNoAction = true) 하면 모터가 동작하지 않는다.

__declspec(dllimport) BOOL dr_motor_get_command_flag_led_green(int nID);
// flag: 모터의 LED 지령상태(현재상태 아님) 중 녹색 설정 여부를 확인한다.
__declspec(dllimport) BOOL dr_motor_get_command_flag_led_blue(int nID);
// flag: 모터의 LED 지령상태(현재상태 아님) 중 청색 설정 여부를 확인한다.
__declspec(dllimport) BOOL dr_motor_get_command_flag_led_red(int nID);
// flag: 모터의 LED 지령상태(현재상태 아님) 중 적색 설정 여부를 확인한다.
```

```
__declspec(dllimport) void dr_motor_send_move(HANDLE hDr, int nTime);
                                                    // 실제 지령: 실제 모터의 지령, 기
설정된 명령데로 모터를 동작시키는 명령, nTime ms(1/1000s) 의 속도로 동작(모터 동작시)

/* 초기화 함수 이후 현 모터의 기본 설정으로 계산됨 */
////////////////////////////////////////////////////////////////////////////
// 계산함수 모음
////////////////////////////////////////////////////////////////////////////
////////// calc ///////////////
__declspec(dllimport) int dr_motor_calc_angle2evd(int nID, float fValue);
                                        // 해당 모터의 각도값을 Evd 값으로 변경해 주는 함수
__declspec(dllimport) float dr_motor_calc_evd2angle(int nID, int nValue);
                                        // 해당 모터의 Evd 값을 각도값으로 변경해 주는 함수
__declspec(dllimport) int dr_motor_calc_time2tick(int nTime);
                                                    // 해당 모터의 시간값(ms)을 실제
모터에서 사용하는 Tick 값으로 변경하는 함수(1 Tick 당 11.2 ms)
////////////////////////////////////////////////////////////////////////////
// 값 제한 함수 모음
////////////////////////////////////////////////////////////////////////////
//////// Limit ////////
__declspec(dllimport) void dr_motor_set_limit_en(BOOL bOn);
                                                                    // 이 함수의
bOn = off(Default On) 한 경우 모터의 limit 파라미터 설정을 무시한다. 즉, 리미트 없이 동작한다.(데드존
및 기구적 충돌 주의 필요)
__declspec(dllimport) BOOL dr_motor_get_limit_en();
                                                                    // Limit 설정이
되어 있는지를 확인하는 함수
__declspec(dllimport) int Clip(int nLimitValue_Up, int nLimitValue_Dn, int nData);
                                    // 내부적으로 사용하는 함수로 clipping 함수(정수)
__declspec(dllimport) float Clipf(float fLimitValue_Up, float fLimitValue_Dn, float fData);
            // 내부적으로 사용하는 함수로 clipping 함수(실수)
__declspec(dllimport) int dr_motor_calc_limit_evd(int nID, int nValue);
                                                    // evd 를 limit 값을 넘지 않게 클리핑하는 함수로
(파라미터 셋 필요–Init 함수 콜시 기본 설정이 된다.) nValue 에 값을 넣으면 결과값을 리턴받는다.
__declspec(dllimport) float dr_motor_calc_limit_angle(int nID, float fValue);
                                    // 각도를 limit 값을 넘지 않게 클리핑하는 함수로(파라미터 셋 필요–Init
함수 콜시 기본 설정이 된다.) fValue 에 값을 넣으면 결과값을 리턴받는다.
//////////////////////
```

```
////////////////////////////////////////////////////////////////////////////
// Stop & Ems & Reset
////////////////////////////////////////////////////////////////////////////
__declspec(dllimport) BOOL dr_motor_get_status_stop();
                                                                    // 정지 명령이
설정되어 있는지 확인(init 함수 call 시, 혹은 리셋관련함수 call 시 해제 됨)
__declspec(dllimport) BOOL dr_motor_get_status_Ems();
                                                                    // 정지 명령이
설정되어 있는지 확인(리셋 시 해제 됨)

////////////////////////////////////////////////////////////////////////////
// Reset 관련 함수
////////////////////////////////////////////////////////////////////////////
__declspec(dllimport) void dr_motor_send_reset(HANDLE hDr, int nID);
                                                // 모터에 reset 명령을 내리는 함수, stop/ems(
비상정지) 도 초기화 된다.
__declspec(dllimport) void dr_motor_set_reset_stop();
                                                                // stop 플래그만 reset 하는
함수
__declspec(dllimport) void dr_motor_set_reset_ems();
                                                                //      비상정지
상태만 해제하는 함수, 이 함수를 call 하면 이후 모터 동작지령이 가능하다. 단, 사전에 에러요인을 없애는
것이 먼저 필요하다.(기구적 충돌 등)
__declspec(dllimport) void dr_motor_send_rollback(int nID, BOOL bIgnoreID, BOOL bIgnoreBaudrate);
        // 모터에 공장초기화 명령을 내리는 함수, 이 함수를 call 하면 최초 판매상태로 모터가 변경이
된다.

////////////////////////////////////////////////////////////////////////////
__declspec(dllimport) EDrResult_t dr_motor_receive_rom( HANDLE hDr, BYTE byId, BYTE byAddress,
BYTE byLength );                                                    // (rom)모터에  직접명령:
해당 모터(byID)의 byAddress 부터 byLength 만큼의 설정값을 가져오도록 지령하는 함수
__declspec(dllimport) EDrResult_t dr_motor_get_rom( HANDLE hDr, BYTE byAddress, BYTE byLength,
BYTE* pbyData );                                                    // (rom)올라온  모터값을
확인하는 함수, byAddress 부터 byLength 만큼의 설정값을 pbyData 에 기록하여 준다.
__declspec(dllimport) EDrResult_t dr_motor_send_rom( HANDLE hDr, BYTE byId, BYTE byAddress, BYTE
byLength, BYTE* pbyData );                                  // (rom)모터에 byAddress 부터 byLength 만큼의
설정값을 직접 써 넣어주는 함수
__declspec(dllimport) EDrResult_t dr_motor_receive_ram( HANDLE hDr, BYTE byId, BYTE byAddress,
BYTE byLength );                                                    // (ram)모터에  직접명령:
해당 모터(byID)의 byAddress 부터 byLength 만큼의 설정값을 가져오도록 지령하는 함수
```

```
__declspec(dllimport) EDrResult_t dr_motor_get_ram( HANDLE hDr, BYTE byAddress, BYTE byLength, BYTE* pbyData );  // (ram)올라온 모터값을 확인하는 함수, byAddress 부터 byLength 만큼의 설정값을 pbyData 에 기록하여 준다.
__declspec(dllimport) EDrResult_t dr_motor_send_ram( HANDLE hDr, BYTE byId, BYTE byAddress, BYTE byLength, BYTE* pbyData );  // (ram)모터에 byAddress 부터 byLength 만큼의 설정값을 직접 써 넣어주는 함수
__declspec(dllimport) EDrResult_t dr_motor_send_command( HANDLE hDr, BYTE byId, BYTE byCommand );
// 모터에 직접 명령: command 명령 하나만 내릴 때 사용
__declspec(dllimport) EDrResult_t dr_motor_send_ram_one_data( HANDLE hDr, BYTE byId, BYTE byAddress, BYTE byData );  // 모터에 직접 명령: command, 하나의 data 명령을 내릴 때 사용
__declspec(dllimport) EDrResult_t dr_motor_send_ram_two_data( HANDLE hDr, BYTE byId, BYTE byAddress, BYTE byData0, BYTE byData1 );// 모터에 직접 명령: command, 두개의 data 명령을 내릴 때 사용
__declspec(dllimport) EDrResult_t dr_motor_send_data( HANDLE hDr, BYTE byId, BYTE byCommand, BYTE byLength, BYTE* pbyData );  // 모터에 직접 명령: command, 여러개의 data 명령을 내릴 때 사용
__declspec(dllimport) EDrResult_t dr_motor_send_driver_servo_on_off(HANDLE hDr, int nID, BYTE byDriverOn, BYTE byServoOn);  // 모터에 직접 명령: Servo/Driver 의 On/Off 지령

#ifdef __cplusplus
}
#endif

#endif
```

로봇으로 쉽게 배우는

C 프로그래밍

C PROGRAMMING

- C 언어를 활용하여 로봇 모션, 센서 제어하기
- Eclips 편집기와 Windows GCC 컴파일러를 통한 C 문법익히기
- C 언어 문법 기준으로 편성된 다양한 로봇 예제 프로그래밍

93550
9 788998 873011
ISBN 978-89-98873-01-1

Dongbu Robot

(주) 동부로봇 www.dongburobot.com

본 사 : 경기도 부천시 원미구 약대동 193 부천테크노파크 401동 11층, TEL : 032-329-5551 (내선 103), FAX : 032-329-5569, E-MAIL : robotsales@dongbu.com

공 장 : 충청남도 천안시 서북구 직산읍 4산단 6길 27, TEL : 041-590-1700, FAX : 041-590-1701