펭귄 스터디 내용16

작성자 : 류성수
작성일자 : 24.07.05

url : https://www.youtube.com/watch?v=fK0Y3mAN1eI&list=PLz—ENLG_8TMdMJIwyqDIpcEOysvNoonf&index=18

● gcc의 옵션명령어

1. 옵션명령어가 뭔가?

터미널 창에서, gcc 컴파일러를 실행할 때, 기존까지는 당연하게 터미널에 이렇게 적었다.
gcc -g 소스.c -o 소스.out
여기서, 옵션 명령어는 gcc를 제외한 모든 명령어를 말한다.

2. 왜 이렇게 실행될 수 있는건가?

C의 기초를 배울 때는 당연하게 main함수의 파라미터 창을 비워놓는다.
그러나, 어떻게 보면, 해당 main 함수도 함수이고, 엄연히 파라미터를 받을 수 있다. 그렇기에 메인함수에 정석적으로 이렇게 파라미터를 받을수 있다.

```c
#include <stdio.h>

void main(int argc, char* argv[])
{
        // argument vector는 문자열 배열이기 때문에
        if(argc > 0)
                for(int i =0; i <argc; i++)
                        printf("op[%d] : %s\n",i,argv[i]);
}
```

이를 확인하기 위해, 터미널에서 실행을 해보자.

```
root@LAPTOP-BQ4AK11N:/project/24-Cstudy16# ./main_parameter.out 1 2 3 4 5 6 7
op[0] : ./main_parameter.out
op[1] : 1
op[2] : 2
op[3] : 3
op[4] : 4
op[5] : 5
op[6] : 6
op[7] : 7
```

gcc 컴파일러도 엄연히 정의를 따지자면, 빌드해주는 프로그램이고, 이 역시 C로 짜여있을 것이다.
그렇기에, 숫자말고 진짜 찐 명령어를 가정하고 실행할 때, 문자들을 넘겨주면

```
root@LAPTOP-BQ4AK11N:/project/24-Cstudy16# ./main_parameter.out -o test temp.c op.c -I./dir -D
asdf
op[0] : ./main_parameter.out
op[1] : -o
op[2] : test
op[3] : temp.c
op[4] : op.c
op[5] : -I./dir
op[6] : -Dasdf
```

이렇게 실행이 된다. 여기서 중요한건, 첫 번째로 들어오는 벡터 파라미터는 실행파일 자기자신을 실행시키는 명령이라는 것이다.
이를 이해하면, gcc가 대충 어떻게 옵션명령어들을 받을 수 있는지를 이해할 수 있다.
이 옵션명령어들에는 역시나 보이는것처럼 순서는 별로 중요치 않고, 무엇인지가 더 중요하다.

● 파일 나눠서 실행하기

VScode로 C/C++의 소스와 헤더들을 나눠서 실행할 때, 나눠서 구현한 함수들을 찾을 수 없다고 계속 경고가 떳던 것을 기억한다.
언제나 비주얼 스튜디오 혹은 누가 만들어준 편리한 IDE만을 써왔기에 기존처럼 소스를 나눠서 컴파일하면 오류가 발생해왔다.

VScode 역시, gcc와 g++ 그리고 gdb 디버거를 이용해 가상리눅스 환경에서 컴파일되고 실행되기 때문에, 이 방법을 모른다면 기존의 비주얼스튜디오처럼 같은 디렉터리라고 나눠진 소스들을 실행할 수 없다.

```c
#include<stdio.h>
#include "sum.h"

void main()
{
        int c = sum(3,4);

        printf("c = %d\n",c);
}
```

linker_test.c (메인 소스)

```c
#include "sum.h"

int sum(int a, int b)
{
        return a+b;
}
```

sum.c (보조 소스)

이렇게 나눠서 실행할 때, gcc의 메인파라미터에다가 보조소스 파일을 추가로 적어주기만 하면 된다.

```
root@LAPTOP-BQ4AK11N:/project/24-Cstudy16# gcc -g linker_test.c sum.c -o linker_test.out
root@LAPTOP-BQ4AK11N:/project/24-Cstudy16# ls
linker_test.c  linker_test.out  sum.c  sum.h
```

```
root@LAPTOP-BQ4AK11N:/project/24-Cstudy16# ./linker_test.out
c = 7
```

실행하면 다음과 같이 나온다.

1. 헤더파일의 중복

이전에 임베디드 수업을 들을 때, 헤더파일이 중복선언되는 것을 방지하기 위해서 헤더에 #ifndef 과 #endif를 썼었던 것을 기억한다.
만약, 이러한 부분을 작성하지 않으면 어떻게 될까?

이를 알아보기 위해, 헤더파일에 똑같은 함수를 중복선언해보고 다 해봤는데, 어쩐 일인지 컴파일이 되고 실행이 잘된다.
그래서, 구조체를 중복선언해봤다.

```c
#include "redefine_struct.h"

typedef struct
{
        int a;
        int b;
        int c;
}test;
```

redefination.h (헤더)

```c
#include<stdio.h>

typedef struct
{
        char d[20];
}test;
```

redefine_struct.h (헤더)

이를 메인 소스에서 include 순서를 redefine_struct.h -> redefination.h 순서로 하면, redefine_struct.h에서 구조체가 계속 바뀌기에 컴파일 시, 아래와 같은 오류가 발생한다.

```
root@LAPTOP-BQ4AK11N:/project/24-Cstudy16# gcc redefination.c -o redefination.out
redefination.c:3:1: error: duplicate 'typedef'
    3 | typedef struct
      | ^~~~~~~
```

그래서, #pragma once 같은 명령어 (이전 AVR IDE에서는 안됐던걸로 기억), #ifndef 같은 전처리기문을 헤더 맨위에 생성하여 중복을 막는다.

그리고, 왜 함수는 헤더에서 중복선언해도 잘되나 봤더니, gcc에도 컴파일 최적화 옵션이 있어서 중복함수선언은 자연적으로 걸러낸다고 한다.

● ifdef과 else 전처리기문과 컴파일 옵션

ifdef과 else 전처리기 문으로, 똑같은 함수를 선언하더라도, 컴파일 옵션에 따라서 전혀 다른 출력을 할 수 있다.
이는 임베디드 (chip마다 함수를 다르게), 모바일 앱 (버전, 기기마다 다르게)등 여러분야에서 많이 쓰이는 기법이다.

ifdef_prac.c (메인소스)

```c
#include "calculate.h"

void main()
{
        int a = calculate(3,4);
        int b = calculate(5,6);

        printf("a = %d\n",a);
        printf("b = %d\n",b);
}
```

calculate.c (보조소스)

```c
#pragma once

#include <stdio.h>

#ifdef SUM

int calculate(int a, int b)
{
        printf("add\n");
        return a+b;
}

#else

int calculate(int a, int b)
{
        printf("multiple!\n");
        return a*b;
}

#endif
```

이런식으로 선언했을 경우, SUM이 define 되어있으면, calculate함수가 덧셈을하고, 안되어있으면 곱셈을 한다.
여기서 터미널 명령어에서 calculate의 내용을 바꿔버릴수 있다.

그냥 컴파일하고 실행

```
multiple!
multiple!
c = 12
d = 56
```

터미널 명령에서 SUM define해줄시

```
add
add
a = 7
b = 11
```

gcc를 실행해서 컴파일시, 다음과 같은 명령을 주면된다.

```
root@LAPTOP-BQ4AK11N:/project/24-Cstudy16# gcc -g -DSUM ifdef_prac.c -o ifdef_prac2.out
```

여기서 -D는 그다음의 문자열을 컴파일 하기전에 define 시키고 실행한다는 gcc 컴파일 옵션명령어이다.

● 헤더나 보조소스가 다른곳에 존재할시,

```
calculate.h   ifdef_prac.out   linker_test.c    main_parameter.c    redefination.c    sum.c
ifdef_prac.c  ifdef_prac2.out  linker_test.out  main_parameter.out  redefine_struct.h sum.h
```

위 사진처럼, 한 디렉터리에 여러 종류의 확장자가 섞여있으면 알아보기 곤란하다.
그래서, 헤더파일과 보조소스들을 비주얼스튜디오처럼 따로 헤더전용 위치파일이나 소스전용 위치파일에 넣어서 컴파일 하려면

1. 헤더파일

```
root@LAPTOP-BQ4AK11N:/project/24-Cstudy16# mv *.h ./headers
```

일단, 헤더를 넣을 디렉터리를 생성하고 .h 확장자들을 해당 파일로 전부 옮겨준다.

```
former_exe  headers  ifdef_prac.c  linker_test.c  main_parameter.c  redefination.c  sum.c
root@LAPTOP-BQ4AK11N:/project/24-Cstudy16#
```

확인해 보면, 헤더 파일들이 없어진걸 확인할 수 있다.

```
root@LAPTOP-BQ4AK11N:/project/24-Cstudy16# gcc -g linker_test.c sum.c -o linker_test_noHeader.
out
linker_test.c:2:10: fatal error: sum.h: No such file or directory
    2 | #include "sum.h"
      |          ^~~~~~~
compilation terminated.
sum.c:1:10: fatal error: sum.h: No such file or directory
    1 | #include "sum.h"
      |          ^~~~~~~
compilation terminated.
```

이상태에서 그냥 컴파일을 하면, 에러가 발생한다.

그래서 컴파일에 디렉터리를 참조하라고 옵션을 줄 것이다.

```
root@LAPTOP-BQ4AK11N:/project/24-Cstudy16# gcc -I ./headers -o linker_test_noHeader.out linker
_test.c sum.c
```

-I ./디렉터리

이는 전처리 과정에서 h파일을 탐색하는 기본 디렉터리를 추가할 때 사용하는 옵션이다.

```
root@LAPTOP-BQ4AK11N:/project/24-Cstudy16# gcc -I ./headers -o linker_test_noHeader.out linker
_test.c sum.c
root@LAPTOP-BQ4AK11N:/project/24-Cstudy16# ls'
> ^C
root@LAPTOP-BQ4AK11N:/project/24-Cstudy16# ls
former_exe  ifdef_prac.c   linker_test_noHeader.out  redefination.c
headers     linker_test.c  main_parameter.c          sum.c
```

잘 실행된 것을 확인할 수 있다.

2. 소스파일

소스파일도 마찬가지로 다른 디렉터리에 둘 수있다.

보조 소스들을 sources 디렉터리에 전부 옮기고 컴파일 해보자.

```
root@LAPTOP-BQ4AK11N:/project/24-Cstudy16# mv sum.c ./sources
root@LAPTOP-BQ4AK11N:/project/24-Cstudy16# ls
former_exe  ifdef_prac.c   linker_test_noHeader.out  redefination.c
headers     linker_test.c  main_parameter.c          sources
```

```
root@LAPTOP-BQ4AK11N:/project/24-Cstudy16# gcc -I ./headers -o linker_test_onlyMain.out linker
_test.c  ./sources/sum.c
```

터미널 명령시, 소스파일의 이름만 쓰는 것이 아니라, 어디에 위치해있는지를 적어주어야 한다.

그럼 정상적으로 컴파일이 수행된다.

```
root@LAPTOP-BQ4AK11N:/project/24-Cstudy16# gcc -I ./headers -o linker_test_onlyMain.out linker
_test.c  ./sources/sum.c
root@LAPTOP-BQ4AK11N:/project/24-Cstudy16# ls
former_exe  ifdef_prac.c   linker_test_noHeader.out  main_parameter.c  sources
headers     linker_test.c  linker_test_onlyMain.out  redefination.c
root@LAPTOP-BQ4AK11N:/project/24-Cstudy16# ./linker_test_onlyMain.out
c = 7
```