

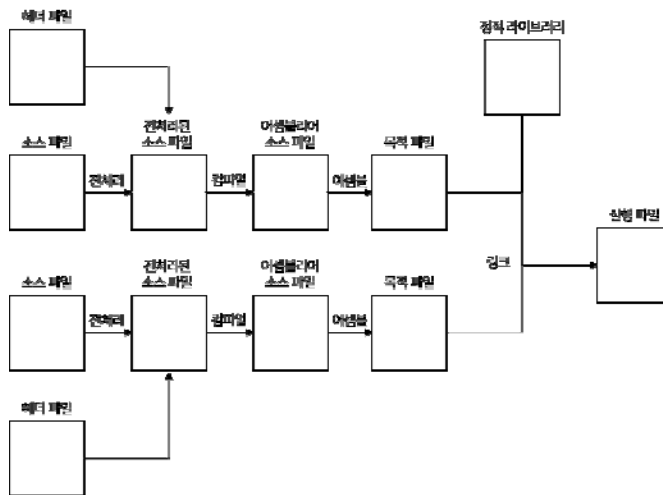
챕터13. 전처리기와 분할 컴파일

1절. 컴파일과 파일 분할

핵심 키워드	전처리, 컴파일, 어셈블, 링크
여기서는 무얼 배울까	
우리는 지금까지 C 언어를 구성하는 크고 작은 블록들이 무엇인지, 그리고 그 블록들을 이용하여 어떻게 큰 블록을 만들어낼지를 배웠다. 이제 남은 것은 그러한 조각들을 합쳐 하나의 완전한 작품을 만들어낼 차례이다. 이번 챕터에서는 C 언어로 완전한 실행 파일을 만들기 위한 자세한 과정과 함께, 큰 프로젝트를 위한 파일 분할 방법에 대해서 배운다.	

1. 빌드 과정

C 언어를 처음 배웠을 시기에는 컴파일 과정을 통해 소스 파일을 실행 파일로 만든다고 이야기했었다. 이 설명이 틀린 것은 아니지만, 실제 C 언어는 그보다 더 복잡한 과정을 통해 소스 파일을 실행 파일로 바꾼다.



위의 이미지는 C 언어의 전체 빌드 과정을 도식화한 것이다. 이 과정은 크게 전처리, 컴파일, 어셈블, 링크의 네 단계로 나눌 수 있다.

(1) 전처리

전처리는 소스 파일에 적힌 C 언어를 번역하기 전 과정으로 전처리 지시자로 적힌 명령을 전처리기가 해석하여 수정된 소스 파일을 만드는 과정이다.

DELL 2023/09/01 16:36

기초 용어 정리

전처리

C 언어를 번역하기 전 수정된 소스 파일을 만드는 과정

DELL 2023/09/01 16:36

기초 용어 정리

전처리기

전처리를 수행하는 프로그램

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

C 언어를 처음 배울 때부터 사용한 `#include`는 대표적인 전처리 지시자이다. `#include`가 포함된 소스 파일은 곧바로 C 언어를 번역하는 것이 아닌, `#include`가 있는 부분에 해당 파일의 모든 내용을 붙여넣는다. 이처럼 전처리기는 소스 파일의 번역 외적으로 사람이 하기 번거로운 작업이나 환경에 맞게 서로 다른 코드를 실행하는 등의 역할을 한다.

(2) 컴파일과 어셈블

컴파일은 컴파일러를 통해 어셈블리어로 작성된 파일로 만드는 과정이며, 어셈블은 어셈블러를 통해 기계어로 작성된 파일을 만드는 과정이다. 경우에 따라 컴파일과 어셈블을 하나로 합쳐 컴파일이라고 부르기도 하며, 컴파일러 역시 어셈블리어가 아닌 기계어를 곧바로 만들 수도 있다.



컴파일과 어셈블은 하나의 파일을 단위로 한다. 10개의 C 언어 파일이 있다면 10개의 파일에 대하여 각각 컴파일과 어셈블을 수행하는 것이다. 만약 한 파일에서 특정한 함수를 사용하지만 그 함수에 대한 정의가 없을 경우, 다른 파일에 있음을 가정하고 컴파일 및 어셈블을 수행한다.

(3) 링크

컴파일러와 어셈블 과정을 거친 기계어 파일을 목적 파일이라고 한다. 여러 파일을 통해 프로그램을 만들었다면 이러한 목적 파일의 양이 많고, 각각의 목적 파일은 서로 다른 목적 파일에 존재하는 변수와 함수가 필요하다.

DELL 2023/09/01 16:37

기초 용어 정리

컴파일

소스 코드를 어셈블리어로 작성된 파일을 만드는 과정

DELL 2023/09/01 16:37

기초 용어 정리

컴파일러

컴파일을 수행하는 프로그램

DELL 2023/09/01 16:38

기초 용어 정리

어셈블

기계어로 작성된 목적 파일을 만드는 과정

DELL 2023/09/01 16:38

기초 용어 정리

어셈블러

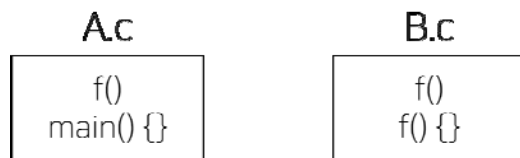
어셈블을 수행하는 프로그램



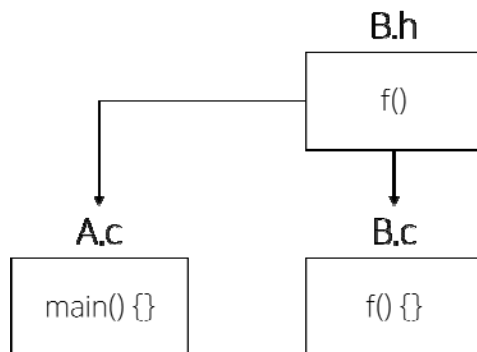
링크는 목적 파일과 라이브러리를 묶어 하나의 완전한 실행 파일을 만드는 과정으로, 링크를 수행하는 프로그램을 링커라고 한다.

2. 파일 분할

C 언어는 함수 단위로 프로그램을 만들기에 파일 분할 역시 함수 단위로 이루어진다. 프로그램 안에 많은 함수가 포함될 경우, 함수들을 적절히 분리하여 각각의 C 언어 파일로 만든다.



A 파일에는 main 함수, B 파일에는 f 함수가 있다고 가정하자. 여기서 main 함수가 f 함수를 필요로 한다면 A 파일의 상단에 f 함수의 선언을 작성한다. 두 함수는 서로 다른 파일로 분리가 되어있으나 이후 링크 단계에서 하나로 합쳐질 것이다.



DELL 2023/09/01 16:39
기초 용어 정리
링크
 목적 파일과 라이브러리를 묶어 하나의 완전한 실행 파일을 만드는 과정

DELL 2023/09/01 16:39
코멘트
 라이브러리는 자신이 아닌 다른 사람이 만든 기능들을 묶은 것을 말합니다. 라이브러리는 소스 파일이나 목적 파일의 형태, 또는 동적 라이브러리의 형태로 제공됩니다. 헤더 파일로 사용해왔던 많은 함수가 라이브러리에 포함된 것들입니다.

DELL 2023/09/01 16:39
기초 용어 정리
 링크를 수행하는 프로그램

사용하고자 하는 함수의 선언을 파일 상단에 작성하는 것은 함수의 수가 많아지고 프로젝트의 규모가 커질수록 번거롭고 사용하기 복잡해진다. 따라서 C 언어 파일에 있는 모든 함수들 대한 선언을 그 파일에 해당하는 헤더 파일로 분리하여 만드는 경우가 많다. 헤더 파일과 소스 파일이 분리되었다면, 이제 함수가 필요한 곳에서 #include를 통해 헤더 파일을 불러오면 된다.

손으로 익히는 코딩 (B.h)
<pre>int f(int x);</pre>

헤더 파일은 함수, 구조체, 변수 등 주로 선언만을 담는다. 여기서는 함수 f의 선언만을 작성하였다.

손으로 익히는 코딩 (B.c)
<pre>#include "B.h" int f(int x) { return 2 * x; }</pre>

B.c는 B.h를 include하고 실제 함수의 정의를 구현했다. #include에서 표준 헤더 파일과 같이 프로젝트 외부에서 사용하는 라이브러리는 <>를, 프로젝트 내부에 있는 파일은 ""를 사용한다. 여기서 "" 안에 들어가는 것은 파일의 경로이며, "B.h"라는 의미는 현재 파일과 같은 위치에 있는 B.h 파일이란 뜻이다.

손으로 익히는 코딩 (A.c)
<pre>#include <stdio.h> #include "B.h" int main() { printf("%d", f(2)); }</pre>

main 함수가 있는 A.c에서는 f 함수의 선언만 있으면 그 함수를 사용할 수 있다. 빌드 과정에서 링커가 A.c와 B.c를 묶어서 하나의 실행 파일로 만들기 때문이다.

(1) 한 번에 빌드하기

```
woolyung@DESKTOP-JVP38VM:~/C-programming$ ls
A.c B.c B.h
woolyung@DESKTOP-JVP38VM:~/C-programming$ gcc A.c B.c B.h -o exe
woolyung@DESKTOP-JVP38VM:~/C-programming$ ./exe
4woolyung@DESKTOP-JVP38VM:~/C-programming$
```

GCC 컴파일러는 빌드하고자 하는 모든 파일을 지정하여 한 번에 컴파일, 어셈블, 링크를 할 수 있다. 앞서 소개한 3개의 파일을 생성한 후 이를 묶어서 빌드하겠다고 지정하면, 최종 결과인 exe라는 이름의 실행 파일은 세 가지 파일을 링크하여 완성한다.

```
woolyung@DESKTOP-JVP38VM:~/C-programming$ gcc *.c -o exe
woolyung@DESKTOP-JVP38VM:~/C-programming$ ./exe
4woolyung@DESKTOP-JVP38VM:~/C-programming$
```

빌드에 필요한 파일이 많다면 리눅스의 와일드카드를 통해 확장자가 c인 파일을 빌드에 사용할 수 있다. 단 이때 디렉토리에는 빌드에 필요한 파일만 존재해야 한다.

DELL 2023/09/16 11:37

TIP
헤더 파일은 #include로 이미 소스 파일에 포함됩니다.

(2) 목적 파일 만들기

GCC 컴파일러는 기본적으로 컴파일, 어셈블, 링크를 한 번에 수행하지만, 프로그램이 전체적으로 수정되지 않고 특정 파일만 바뀔 경우, 바뀌지 않은 파일들은 컴파일과 어셈블을 또 할 필요가 없다. GCC 컴파일러는 컴파일과 어셈블이 필요하지 않다면 목적 파일을 컴파일에 그대로 사용하여 빌드 속도를 높일 수 있다.

```
woolyung@DESKTOP-JVP38VM:~/C-programming$ ls
A.c B.c B.h
woolyung@DESKTOP-JVP38VM:~/C-programming$ gcc B.c -o B.o -c
woolyung@DESKTOP-JVP38VM:~/C-programming$ ls
A.c B.c B.h B.o
woolyung@DESKTOP-JVP38VM:~/C-programming$ gcc A.c B.o -o exe
woolyung@DESKTOP-JVP38VM:~/C-programming$ ./exe
4woolyung@DESKTOP-JVP38VM:~/C-programming$
```

GCC의 -c 옵션은 대상을 컴파일 및 어셈블하여 중간 결과인 목적 파일로 만든다. gcc B.c -o B.o -c는 B.c 파일을 B.o라는 이름의 목적 파일로 만든다는 의미이다. B.o는 앞선 B.c과 같은 역할을 수행하므로 A.c와 B.o를 묶어 하나의 독립적인 실행 파일로 만든다. 프로젝트의 크기가 커지면서 구성 파일의 수가 많아지면 목적 파일을 사용할 때와 그렇지 않을 때의 빌드 시간의 차이는 눈에 띄게 벌어진다.

DELL 2023/09/16 11:36

TIP
큰 프로젝트의 빌드는 명령어로 모두 입력하기에는 그 수가 많기에 GCC와 함께 설치되는 make라는 이름의 프로그램을 함께 사용합니다.

3. 전역 변수

이제부터 파일이 둘 이상이 될 수 있음에 따라 변수의 스코프도 확장할 필요가 있다. C 언어에서는 분할된 파일에 대하여 변수의 스코프를 지정하는 두 가지 변수 지정자가 있다.

(1) static

static 키워드가 전역 변수에 사용되면 이를 정적 전역 변수라고 한다. 정적 전역 변수는 라

이프타임이 전역 변수와 같지만, 스코프는 그 변수가 선언된 파일로 제한된다.

```
#include <stdio.h>

static int cnt = 0;

void count()
{
    cnt++;
}

int main()
{
    for (int i = 0; i < 10; i++)
        count();
    printf("%d", cnt);
}
```

실행 결과
10

위 예제에서 변수 cnt는 정적 전역 변수이므로 다른 파일에서는 사용할 수 없는 전역 변수이다. 정적 전역 변수는 전역 변수의 특징을 이용해 프로그램을 만들어야 하는 경우, 다른 파일에서 수정할 수 없음을 보장할 수 있게 하여 더 안전하게 전역 변수를 사용할 수 있다.

(2) extern

extern 키워드가 전역 변수에 사용되면 이를 외부 변수라고 한다. extern 키워드가 붙은 전역 변수는 다른 파일에 존재하는 전역 변수를 가져와서 사용한다는 의미이며, 따라서 초기화 작업이 필요하지 않다.

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

extern int cnt;

int main()
{
```

```
f();
printf("%d", cnt);
}
```

위 예제에서 외부 변수 cnt가 있다. 이 외부 변수는 정확히 어느 파일에서 선언되었는지는 알 수 없지만, 링커가 링크하는 시점에서 다른 파일 어딘가에 존재하는 변수일 것이며, 그 변수와 이 파일에 있는 cnt는 같은 변수이다. 그러므로 다른 파일에서 cnt라는 이름의 전역 변수를 수정하면 그 값을 곧바로 main 함수에서도 사용할 수 있다.

다른 파일의 전역 변수를 사용하고자 한다면 원본 변수는 static이 아니어야 하고 다른 파일은 extern으로 선언되어야 한다. 이렇게 구성된 전역 변수는 링크된 이후 같은 변수를 나타내므로 서로 다른 파일에서 공유된 변수를 사용할 수 있다.

(3) 파일 분할과 변수의 스코프

아래는 여러 파일을 가지는 프로젝트에서 전역 변수를 사용하는 예제이다.

손으로 익히는 코딩 (max.h)
<code>void func(unsigned int n);</code>

max.h 파일에는 func 함수가 선언되어 있다. 프로그램이 시작된 후부터 매개변수로 전달받은 n 값 중에서 가장 큰 값을 저장하는 역할을 할 것이다.

손으로 익히는 코딩 (max.c)
<pre>#include "max.h" unsigned int max = 0; void func(unsigned int n) { if (max < n) max = n; }</pre>

max.c에는 전역 변수 max와 func 함수의 정의가 있다. func 함수는 n과 max를 비교하여 n이 더 크다면 이를 max에 저장한다. 그러므로 전역 변수 max에는 지금까지의 모든 n 중에서 가장 큰 값이 저장된다.

손으로 익히는 코딩 (main.c)
<code>#include <stdio.h></code>

```
#include "max.h"

extern unsigned int max;

int main()
{
    for (unsigned int i = 0; i < 10; i++)
        func(i);
    printf("%d", max);
}
```

실행 결과
9

main 함수는 max.h에서 선언된 func 함수를 호출한다. func 함수를 통해 저장된 가장 큰 값은 다른 파일의 전역 변수로 있으므로 이를 외부 변수로 선언하여 사용한다. max.c 파일의 max와 main.c 파일의 max는 같은 변수이므로 결과적으로 max 변수의 값인 9가 출력된다.

예리에서 배우기 : Windows 버전과 동일

2절. 전처리 지시자

Windows 버전의 2권 13챕터 2절과 동일

연습문제 : Windows 버전과 동일

한 장으로 요약하기 : Windows 버전과 동일

[1쪽] [메모:2] DELL 2023/09/01 16:36

기초 용어 정리

전처리

C 언어를 번역하기 전 수정된 소스 파일을 만드는 과정

[1쪽] [메모:1] DELL 2023/09/01 16:36

기초 용어 정리

전처리기

전처리를 수행하는 프로그램

[2쪽] [메모:3] DELL 2023/09/01 16:37

기초 용어 정리

컴파일

소스 코드를 어셈블리어로 작성된 파일을 만드는 과정

[2쪽] [메모:4] DELL 2023/09/01 16:37

기초 용어 정리

컴파일러

컴파일을 수행하는 프로그램

[2쪽] [메모:5] DELL 2023/09/01 16:38

기초 용어 정리

어셈블

기계어로 작성된 목적 파일을 만드는 과정

[2쪽] [메모:6] DELL 2023/09/01 16:38

기초 용어 정리

어셈블러

어셈블을 수행하는 프로그램

[3쪽] [메모:7] DELL 2023/09/01 16:39

기초 용어 정리

링크

목적 파일과 라이브러리를 묶어 하나의 완전한 실행 파일을 만드는 과정

[3쪽] [메모:9] DELL 2023/09/01 16:39

코멘트

라이브러리는 자신이 아닌 다른 사람이 만든 기능들을 묶은 것을 말합니다. 라이브러리는 소스 파일이나 목적 파일의 형태, 또는 동적 라이브러리의 형태로 제공됩니다. 헤더 파일로 사용해왔던 많은 함수가 라이브러리에 포함된 것들입니다.

[3쪽] [메모:8] DELL 2023/09/01 16:39

기초 용어 정리

링크를 수행하는 프로그램

[5쪽] [메모:11] DELL 2023/09/16 11:37

TIP

헤더 파일은 #include로 이미 소스 파일에 포함됩니다.

[5쪽] [메모:10] DELL 2023/09/16 11:36

TIP

큰 프로젝트의 빌드는 명령어로 모두 입력하기에는 그 수가 많기에 GCC와 함께 설치되는 make라는 이름의 프로그램을 함께 사용합니다.