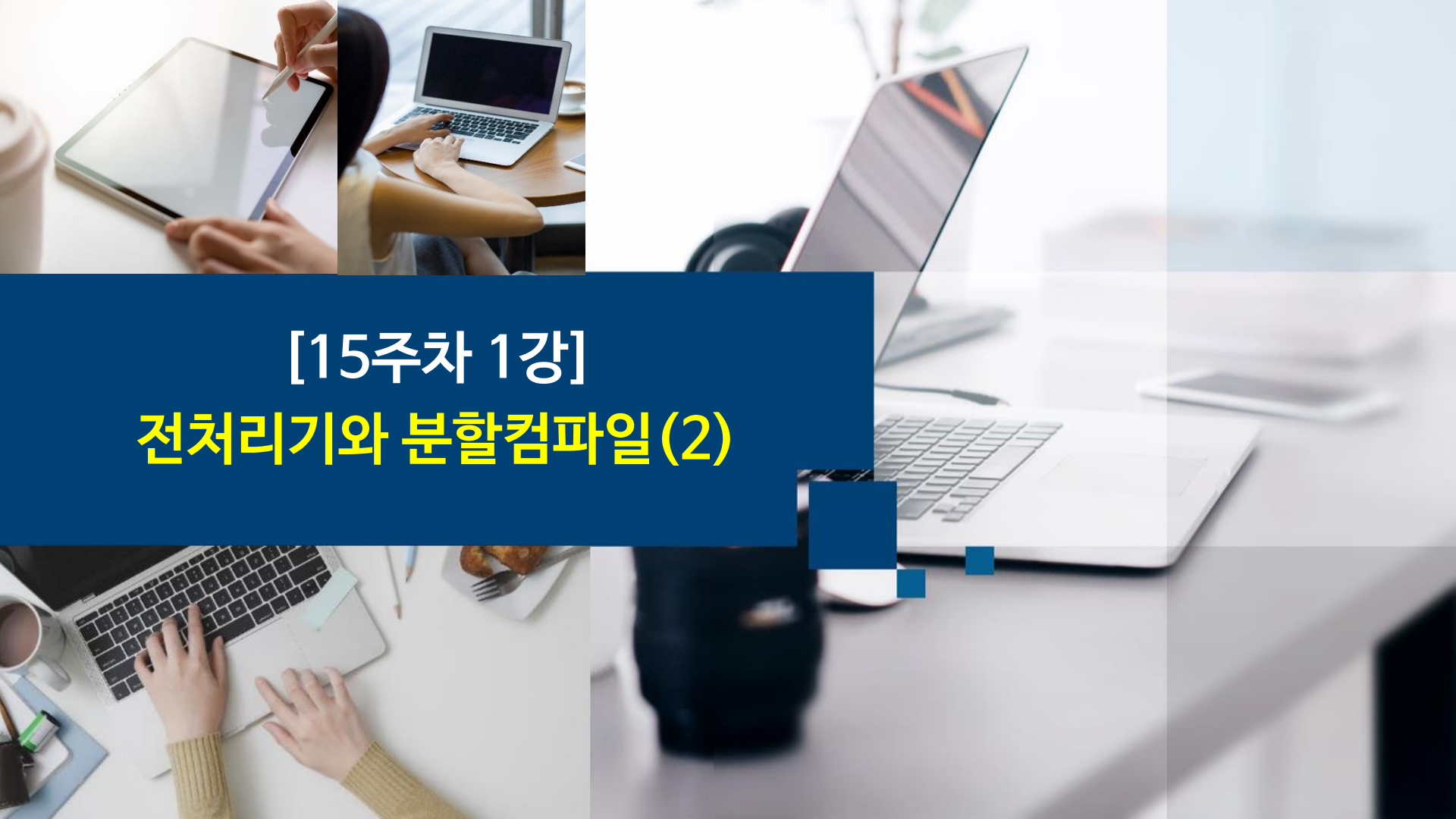


[15주차 1강]
전처리기와 분할컴파일(2)

학습 내용

- 15.2 분할 컴파일 (1)

학습 목표

- 15.2 분할 컴파일을 이해하고 활용할 수 있다.

C프로그래밍및실습

1. 전처리기
2. 분할 컴파일 (1)
3. 변수의 사용범위와 지속기간

2. 분할 컴파일

분할 컴파일이란?

- 하나의 프로젝트를 모듈 별로 **여러 소스 파일에 나누어 작성**하고, 소스 파일 별로 컴파일하는 것
  - ✓ 모듈(module) : 논리적으로 특정 기능 구현을 위한 함수 그룹
- 필요성
  - ✓ 대형 프로그램 작성 시, 작업 분담 목적
  - ✓ 모듈의 재사용성을 높임
  - ✓ 프로그램 관리의 효율성을 향상시킴
  - ✓ 프로그램의 일부를 변경/확장하기 위해 프로그램 전체를 컴파일해야 하는 번거로움을 덜 수 있음

2. 분할 컴파일

분할 컴파일 따라 해보기

- 하나의 소스 프로그램인 original.c를 3개의 소스파일로 분할
  - ✓ main.c
  - ✓ myfunc1.c
  - ✓ myfunc2.c
- 각 파일을 따로 컴파일 한 후, 링커를 통해 하나의 실행 파일을 생성
  - ➢ Visual studio에서는 “빌드”를 통해 해당 작업을 수행

2. 분할 컴파일

original.c

```c
#include <stdio.h>

int myfunc1(int x, int y);
void myfunc2(int n);

int main() {
  int a, b, cnt = 0;
  scanf("%d %d", &a, &b);
  cnt = myfunc1(a, b);
  myfunc2(cnt);
  return 0;
}

int myfunc1(int x, int y) {
  return (x * y - x);
}

void myfunc2(int n) {
  int i;
  for (i = 0; i < n; i++)
    printf("count - %d\n", i+1);
}
```

main.c

```c
#include <stdio.h>

int myfunc1(int x, int y);
void myfunc2(int n);

int main() {
  int a, b, cnt = 0;
  scanf("%d %d", &a, &b);
  cnt = myfunc1(a, b);
  myfunc2(cnt);
  return 0;
}
```

myfunc1.c

```c
int myfunc1(int x, int y) {
  return (x * y - x);
}
```

myfunc2.c

```c
#include <stdio.h>

void myfunc2(int n) {
  int i;
  for (i = 0; i < n; i++)
    printf("%d\n", i+1);
}
```

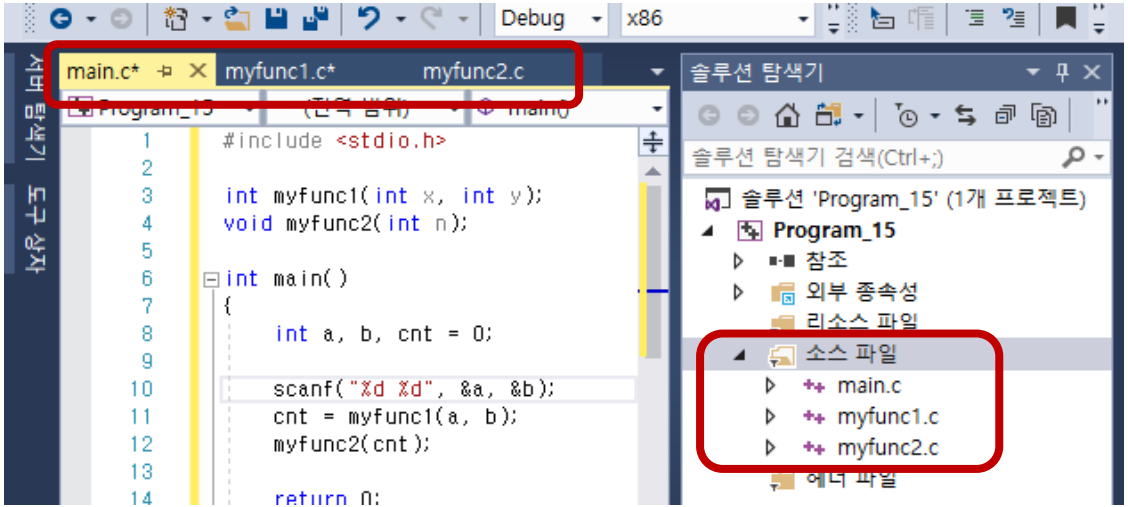
2. 분할 컴파일

Visual studio 환경에서 앞의 예제 실행하는 방법

① Project_15라는 이름으로 프로젝트 하나를 생성

② 소스 파일 폴더 내에 main.c, myfunc1.c, myfunc2.c를 생성 후, 각 파일에 앞의 소스 작성

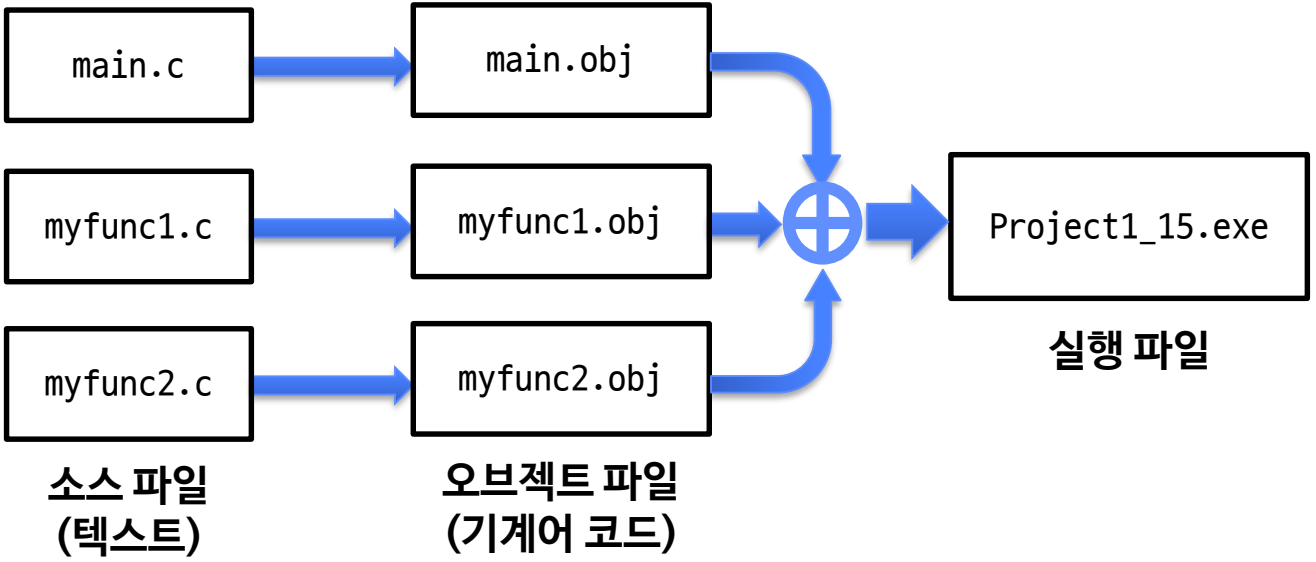
③ Project1_15 빌드



2. 분할 컴파일

분할 컴파일 과정(주요 부분만)

- 각 소스 파일마다 오브젝트 파일 생성
- 오브젝트 파일을 결합하여 실행 파일 생성



2. 분할 컴파일

무엇을 분할(+통합)하는가?

- **함수** 정의 + **전역 변수** 선언(정의)
  - ✓ 하나의 소스 파일에만 있어야 함(중복 불가)
    - ✓ myfunc2.c에도 myfunc1 () 함수 정의를 추가하면?
    - ✓ main.c와 myfunc1.c에 전역변수 'int Z=1;'을 추가하면?

※ 나머지는 함수 정의와 변수 선언(정의)에 필요한 정보

- ✓ 매크로, 자료형 정의 등
- ✓ 여러 소스 파일에 중복되어도 무방
  - ✓ 여러 소스에서 매크로 상수 Z를 정의하고 사용하면? (심지어 값이 달라도 무방)

분할 컴파일 시 주의사항

- 각 소스 파일은 독립적으로 오류 없이 컴파일 되어야 함
  - ✓ 각 소스 파일 컴파일에 필요한 정보가 해당 소스에 포함되어야 함
    - ✓ main.c에서 myfunc2() 함수 원형 선언을 삭제하면?
    - ✓ 다음과 같이 전역변수 Z를 선언하고 사용하면?
      - ✓ main.c에 추가 : int Z=1;
      - ✓ myfunc1.c에서 수정 : return (x * y – Z);
- 다른 파일에 정의된 함수나 전역 변수를 사용하려면?
  - ✓ 해당 함수나 변수에 대한 정보를 알려주어야 함 → extern 선언

2. 분할 컴파일

extern 키워드

- 다른 소스 파일에 정의된 함수나 전역 변수를 해당 소스 파일에서 사용하고자 할 때 사용
  - ✓ 함수 또는 전역 변수가 외부에 정의되어 있다는 사실만 컴파일러에게 알려줌
  - ✓ 구체적으로 어느 파일에 정의되어 있는지는 몰라도 됨
  - ✓ extern 키워드가 붙은 변수를 외부 변수라고도 함

- 형식
  - ✓ extern 변수선언/함수선언;
    - ✓ 함수의 경우 extern을 생략할 수 있음

2. 분할 컴파일

extern 선언 사용 예

메모리가 할당된 실제 변수

main.c

```
#include <stdio.h>
void count();      // 외부함수 선언 (extern 생략 가능)
int num = 1;       // 전역 변수 선언
int main() {
   printf("before - main.c: num = %d\n", num);
   count();                  // 외부함수 호출
   printf("after - main.c: num = %d\n", num);
   return 0;
}
```

이 선언으로 인해 메모리 공간이 할당되지는 않음

counter.c

```
#include <stdio.h>
extern int num;  // 외부변수 선언 (extern 생략 불가능)
void count() {   // count() 함수 정의 부분
   num++;        // 외부 변수 사용
   printf("count() - count.c: num = %d\n", num);}
```

2. 분할 컴파일

static 키워드

- 앞 예제에서 변수 num은 전역 변수이자 외부변수
  - ✓ 즉, 다른 파일에서도 extern 변수로 선언하면 자유로이 사용 가능

- 변수가 선언된 파일 안에서는 전역 변수처럼 쓰지만, **외부(다른 파일)로부터의 접근을 제한하려면**?
  - ➢ 해당 전역 변수 선언 시 앞에 **static 키워드**를 붙이면 됨 (**정적 전역 변수**)
    - ✓ 함수도 동일하게 적용

- 참고) 정적 지역 변수 (8장에서 학습)
  - ✓ 함수 내에서 선언된 정적 변수로, 사용 범위는 해당 함수 내부로 제한
  - ✓ 지속 시간은 프로그램 실행 기간 전체

extern vs. static 함수 사용 예

- 함수 eval()은 main.c에서 호출 가능
  - ✓ 이를 위해 main.c에서 extern 으로 선언 (extern 생략 가능)
- 함수 f()는 static 이므로 func.c 내에서만 호출 가능
  - ✓ main.c에서 호출 불가능

main.c

```
#include <stdio.h>
extern int eval(int n);

int main()
{
    printf("%d\n", eval(5));
    return 0;
}
```

func.c

```
static int f(int x)
{
    return x*4+3;
}

int eval(int n)
{
    return 3*f(n);
}
```

학습 정리

- 하나의 프로그램은 여러 개의 **파일**로 분할하여 작성할 수 있음
- 각 소스 파일은 독립적으로 오류 없이 컴파일 되어야 함
- **extern** 키워드는 다른 소스 파일에 선언된 변수 또는 함수를 사용하고자 할 때 사용
- **정적 변수**는 변수 앞에 **static** 키워드를 붙인 변수이고,
  선언 위치에 따라서 함수 내 또는 파일 내에서만 사용 가능