C 언어 보충자료

데이터 구조론 수강을 위한 보충자료

C언어 보충자료 내용

- 메모리의 동적 할당
- 다수의 소스파일, 헤더파일로 프로그램 구성
- 헤더파일 의 정의
- 매크로 #ifndef ~ #endif의 의미

운영체제가 할당하는 메모리 공간의 구성

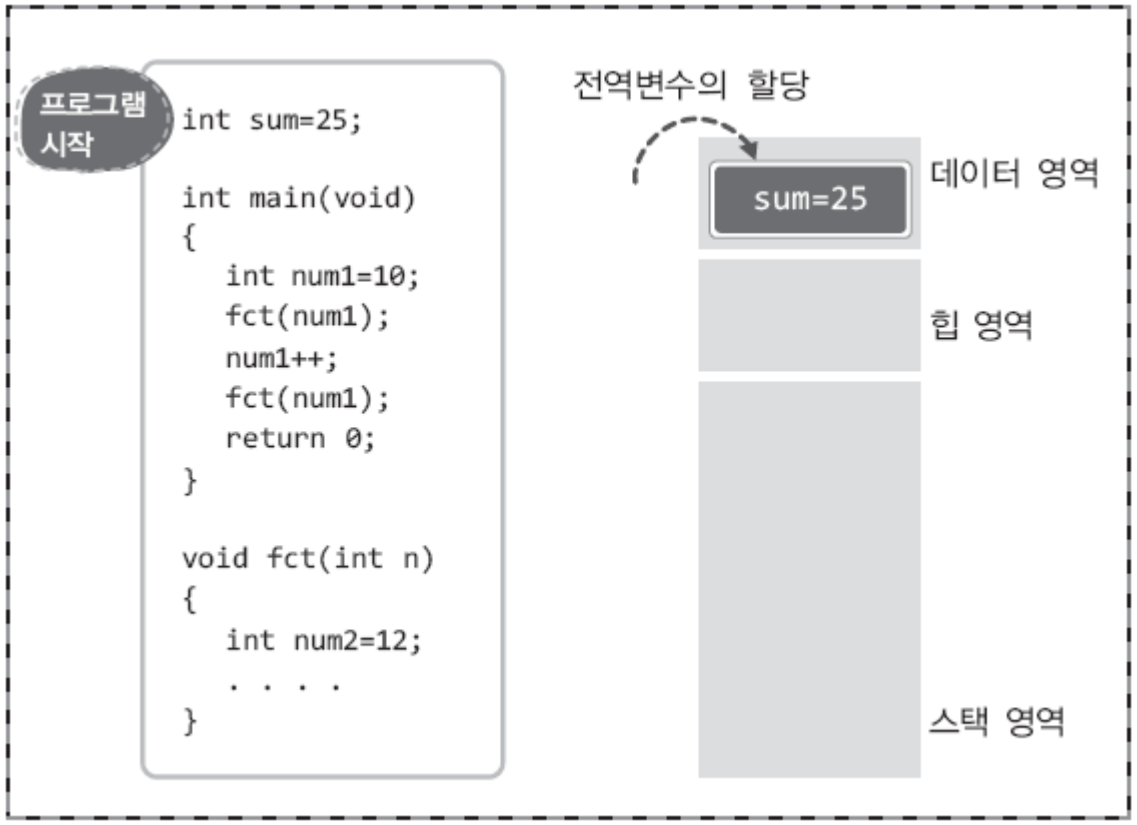
실행할 프로그램의 코드가 저장되는 메모리 공간.
CPU는 코드 영역에 저장된 명령문을 하나씩 가져다가 실행

전역변수와 static 변수가 할당되는 영역.
프로그램 시작과 동시에 할당되어 종료 시까지 남아있는 특징의 변수가 저장되는 영역

프로그래머가 원하는 시점에 메모리 공간에 할당 및 소멸을 하기 위한 영역

지역변수와 매개변수가 할당되는 영역
함수를 빠져나가면 소멸되는 변수를 저장하는 영역

프로그램의 실행에 따른 메모리의 상태 변화1

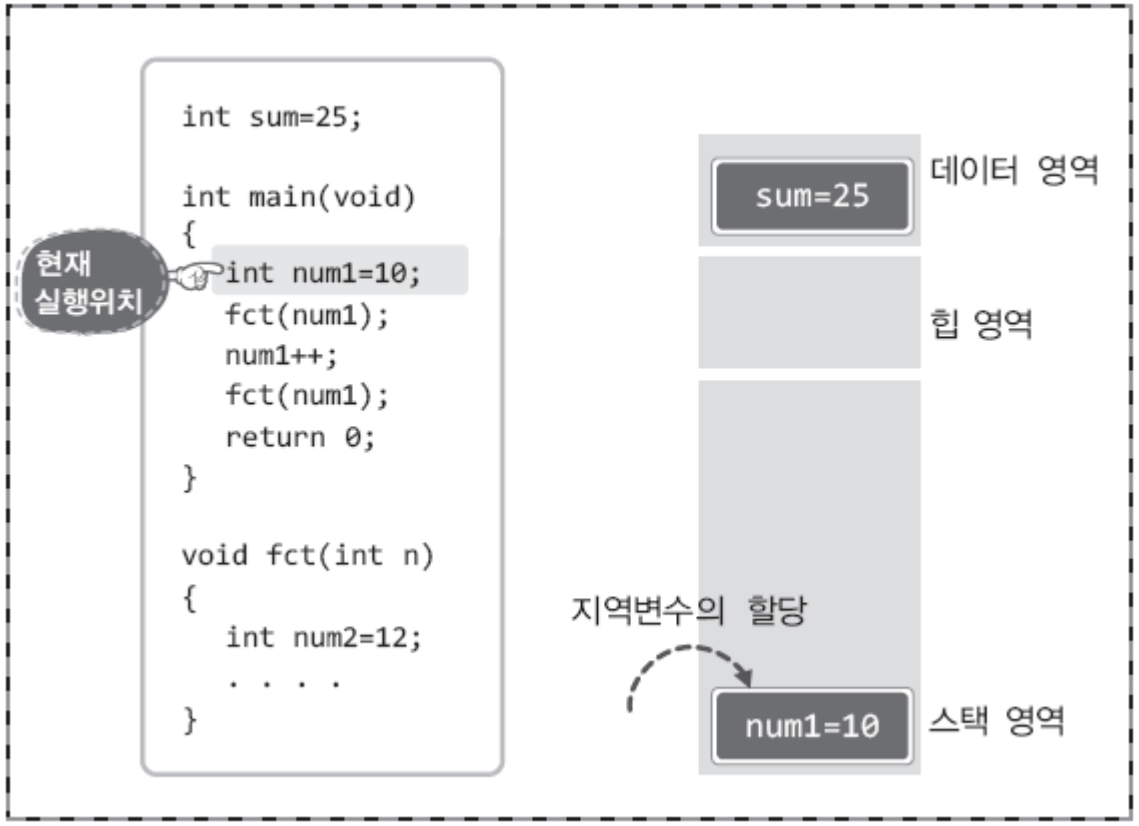


프로그램의 시작:

전역변수의 할당 및 초기화

실행의 흐름/

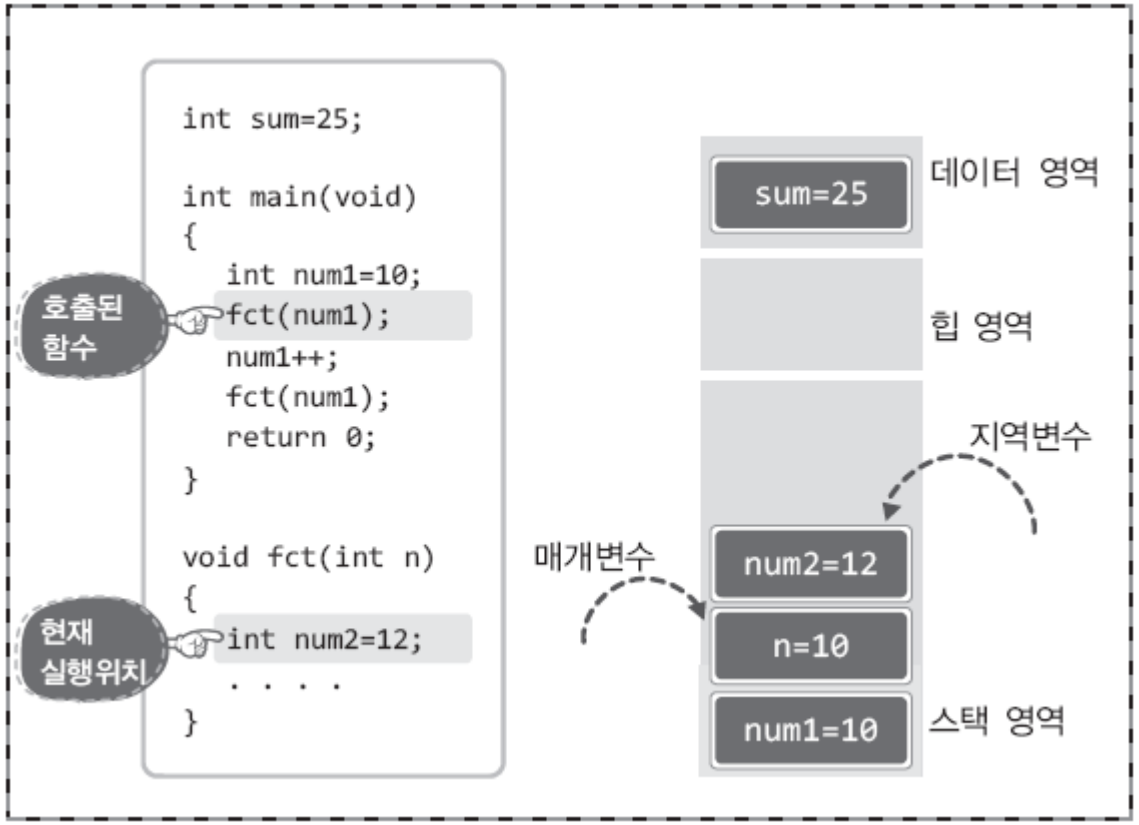
프로그램의 실행에 따른 메모리의 상태 변화2



main 함수의 호출 및 실행

실행의 흐름2

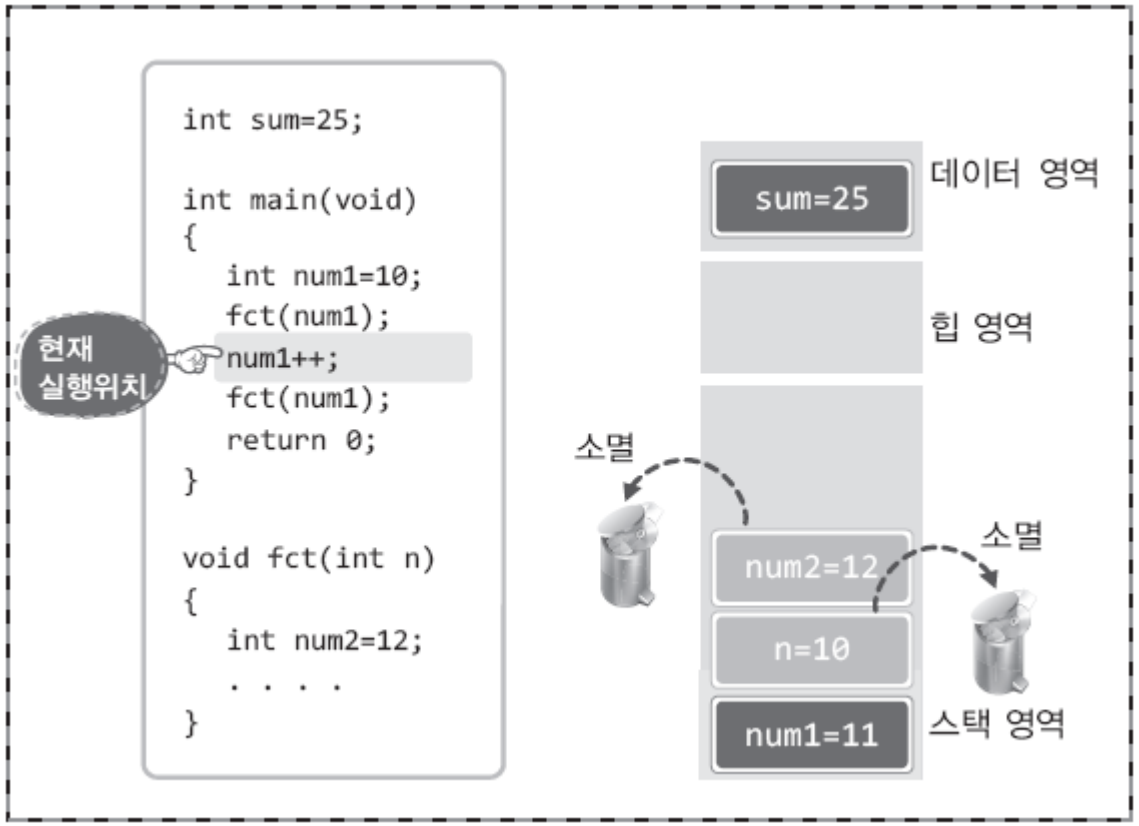
프로그램의 실행에 따른 메모리의 상태 변화3



fct 함수의 호출

실행의 흐름3

프로그램의 실행에 따른 메모리의 상태 변화4

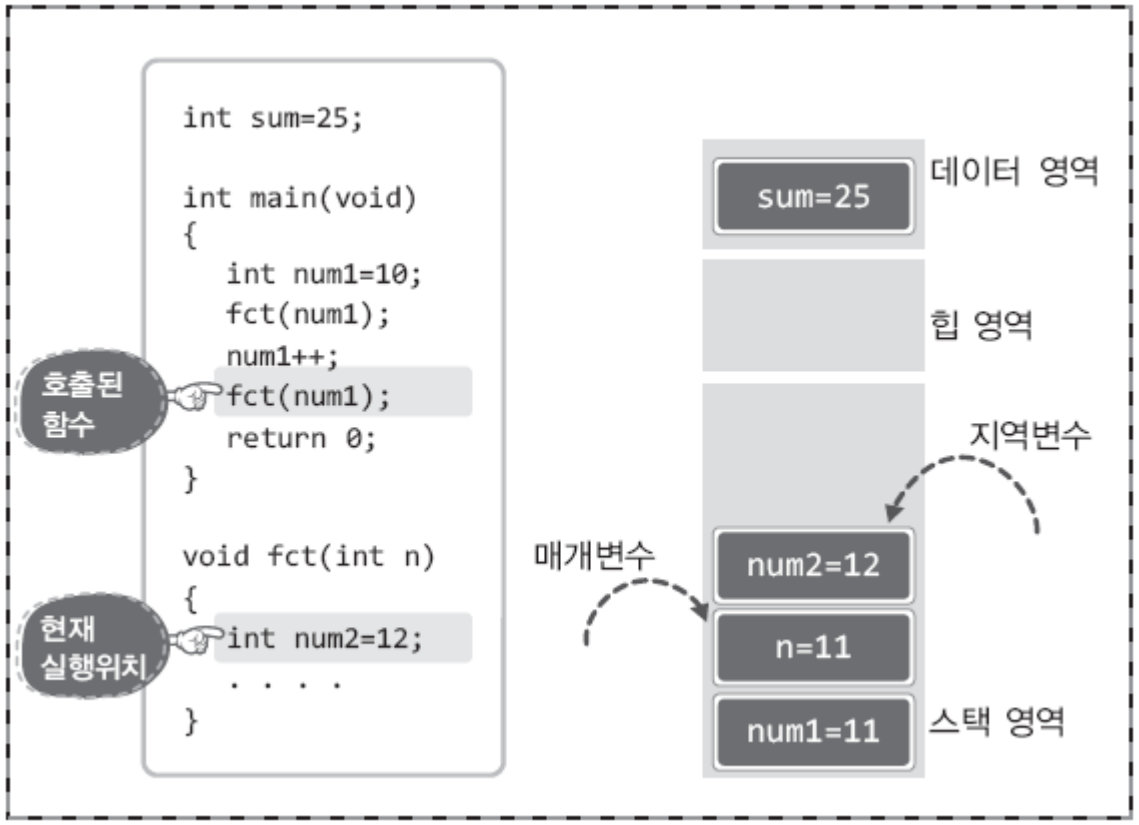


fct 함수의 반환

그리고 main 함수 이어서 실행

실행의 흐름4

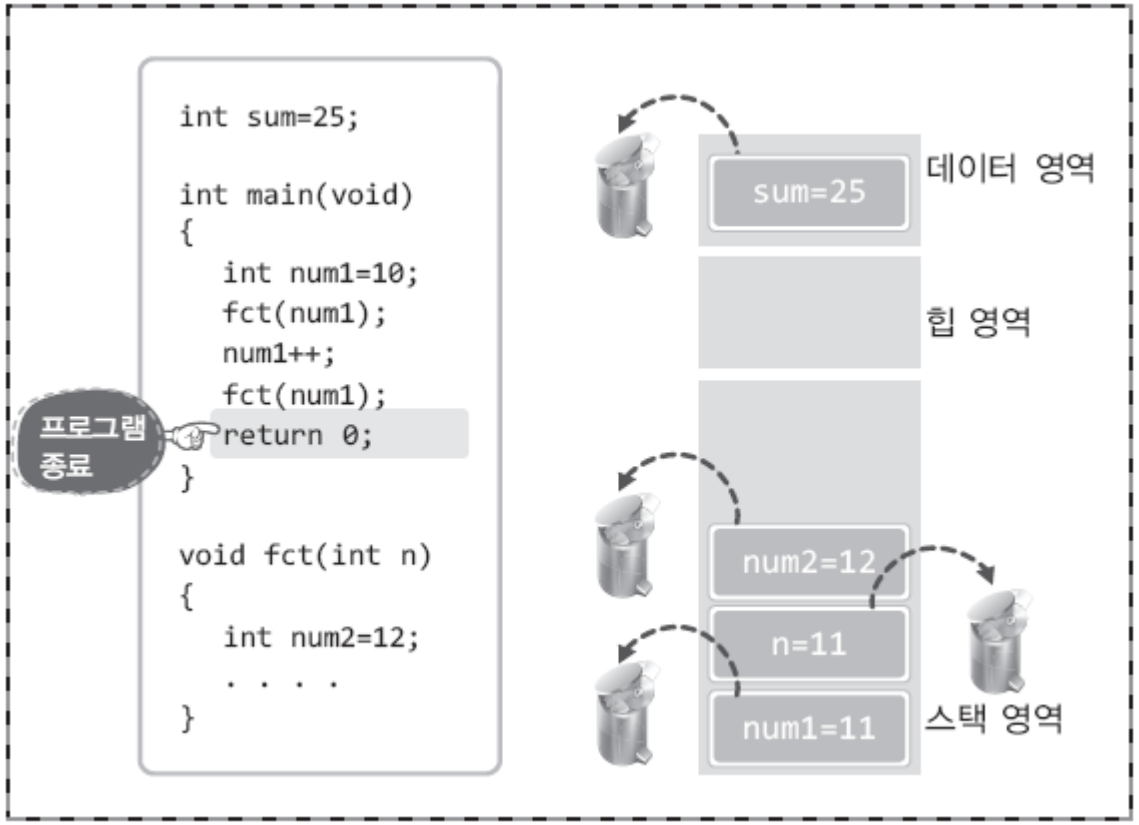
프로그램의 실행에 따른 메모리의 상태 변화5



fct 함수의 재호출 및 실행

실행의 흐름5

프로그램의 실행에 따른 메모리의 상태 변화6



fct 함수의 반환

및 main 함수의 반환

실행의 흐름6 (프로그램 종료)

함수의 호출순서가 main → fct1 → fct2이라면 스택의 반환은(지역변수의 소멸은) 그의 역순인 fct2 → fct1 → main으로 이루어진다는 특징을 기억하자!

메모리 동적 할당:힙 영역의 메모리 공간 할당과 해제

반환형이 void형 포인터임에 주목!

```
#include <stdlib.h>
void * malloc(size_t size);       // 힙 영역으로의 메모리 공간 할당
void free(void * ptr);     // 힙 영역에 할당된 메모리 공간 해제
```

➡ malloc 함수는 성공 시 할당된 메모리의 주소 값, 실패 시 NULL 반환

malloc 함수는 인자로 숫자만 하나 전달받을 뿐이니 할당하는 메모리의 용도를 알지 못한다. 따라서 메모리의 포인터 형을 결정짓지 못한다. 따라서 다음과 같이 형 변환의 과정을 거쳐서 할당된 메모리의 주소 값을 저장해야 한다.

```
int * ptr1 = (int *)malloc(sizeof(int));
double * ptr2 = (double *)malloc(sizeof(double));
int * ptr3 = (int *)malloc(sizeof(int)*7);
double * ptr4 = (double *)malloc(sizeof(double)*9);
```

malloc 함수의
가장 모범적인 호출형태

힙 영역으로의 접근

```
int main(void)
{
    int * ptr1 = (int *)malloc(sizeof(int));
    int * ptr2 = (int *)malloc(sizeof(int)*7);
    int i;

    *ptr1 = 20;
    for(i=0; i<7; i++)
        ptr2[i]=i+1;

    printf("%d \n", *ptr1);
    for(i=0; i<7; i++)
        printf("%d ", ptr2[i]);

    free(ptr1);
    free(ptr2);
    return 0;
}
```

```
int * ptr = (int *)malloc(sizeof(int));
if(ptr==NULL)
{
    // 메모리 할당 실패에 따른 오류의 처리
}
```

메모리 할당 실패 시 malloc 함수는 NULL을 반환

이렇듯 힙 영역으로의 접근은 포인터를 통해서만 이뤄진다.

실행결과

```
20
1 2 3 4 5 6 7
```

'동적 할당'이라 하는 이유!

컴파일 시 할당에 필요한 메모리 공간이 계산되지 않고, 실행 시 할당에 필요한 메모리 공간이 계산되므로!

free 함수를 호출하지 않으면?

- free 함수를 호출하지 않으면?

  할당된 메모리 공간은 메모리라는 중요한 리소스를 계속 차지하게 된다.

- free 함수를 호출하지 않으면 프로그램 종료 후에도 메모리를 차지하는가?

  프로그램이 종료되면 프로그램 실행 시 할당된 모든 자원이 반환된다.

- 꼭 free 함수를 호출해야 하는 이유는 무엇인가?

  fopen 함수와 쌍을 이루어 fclose 함수를 호출하는 것과 유사하다.

파일 분리의 필요성

- 프로그램의 유지보수에 편리
  - 연관된 내용만을 한 소스 파일로 구성하면 보기 편함
  - 컴파일 시간 단축
    - Visual studio에서 컴파일은 변경된 소스 파일만 함.

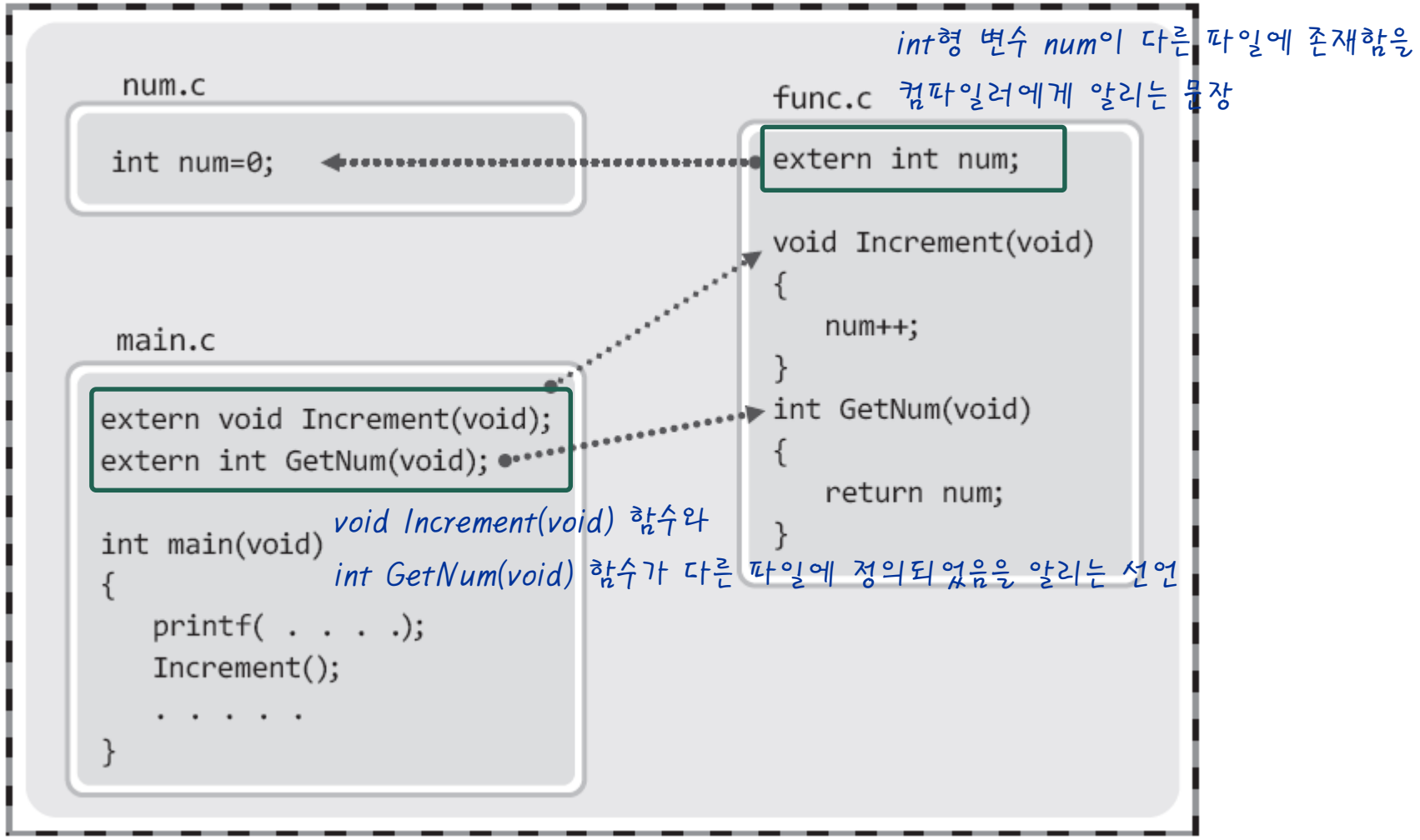
- 본 과목에서 구현하는 데이터 구조는 필요에 따라 다른 과목의 프로젝트를 수행하는 데 사용 가능
  - 데이터 구조를 구현한 소스와 그것을 응용하는 프로그램을 작성하는 소스를 구분해 놓으면 편리

파일의 분할

- 컴파일러는 파일 단위로 컴파일 함.
  - 컴파일 하는 소스 파일에서 사용하는 변수, 함수가 모두 인식되어야 함.
  - 아닌 경우 컴파일 오류 발생

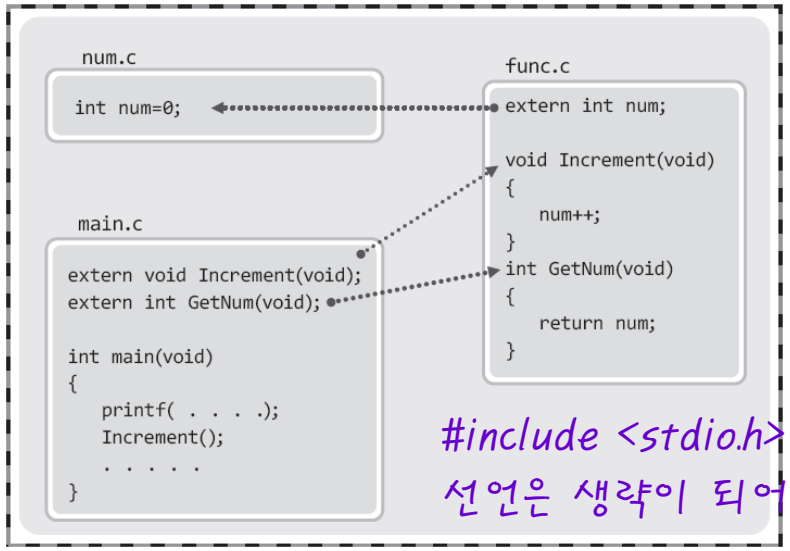
- 파일을 분할하여 다른 파일에 있는 변수나 함수를 사용하는 경우
  - 외부 선언 및 정의 사실을 컴파일러에게 알려줘야 컴파일 오류 발생 안함.
  - 키워드 *extern* 이 외부에 존재함을 알리는 용도로 사용.

파일 분리의 예


num.c
int num=0;
func.c
extern int num;
void Increment(void)
{
num++;
}
int GetNum(void)
{
return num;
}
main.c
extern void Increment(void);
extern int GetNum(void);
int main(void)
{
printf(. . . .);
Increment();
. . . . .
}
int형 변수 num이 다른 파일에 존재함을
컴파일러에게 알리는 문장
void Increment(void) 함수와
int GetNum(void) 함수가 다른 파일에 정의되었음을 알리는 선언

여러 파일을 컴파일 하는 방법

- 파일 정리



#include <stdio.h>의 선언은 생략이 되어 있다!

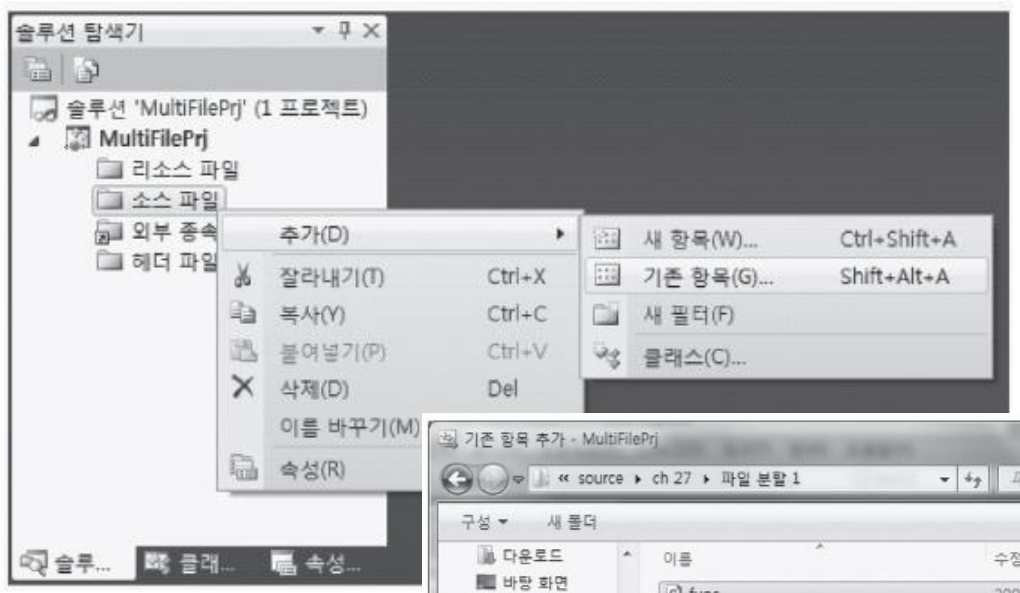
이 세 개의 파일을 하나의 프로젝트 안에 담아서 하나의 실행파일을 생성해 보는 것이 목적!

다중파일 컴파일 방법 두 가지

· 첫 번째 방법
→ 파일을 먼저 생성해서 코드를 삽입한 다음에 프로젝트에 추가한다.

· 두 번째 방법
→ 프로젝트에 파일을 추가한 다음에 코드를 삽입한다.

존재하는 파일, 프로젝트에 추가하는 방법



1단계

아래에서 보이듯이 다수의 파일이 하나의 프로젝트 안에 포함되었음이 솔루션 탐색기에 나타나야 한다.

2단계

추가결과 확인

헤더파일을 include 하는 방법

- 표준헤더 파일을 포함시킬 때 사용하는 방식

```
#include <헤더파일 이름>
```

  - 표준헤더 파일이 저장된 디렉터리에서 헤더파일을 찾아서 포함을 시킨다.

- 프로그래머가 정의한 헤더파일을 포함시킬 때 사용하는 방식

```
#include "헤더파일 이름"
```

  - 이 문장을 포함하는 소스파일이 저장된 디렉터리에서 헤더파일을 찾게 된다.
  - 상대경로를 사용하여 헤더파일 이름 지정할 것

절대경로의 지정과 그에 따른 단점

```
#include  "C:\CPower\MyProject\header.h"
```

Windows의 절대경로 지정방식.

```
#include  "/CPower/MyProject/header.h"
```

Linux의 절대경로 지정방식

▶ 절대경로를 지정하면 프로그램의 소스파일과 헤더파일을 임의의 위치로 이동시킬 수 없다.

▶ 운영체제가 달라지면 디렉터리의 구조가 달라지기 때문에 경로지정에 대한 부분을 전면적으로 수정해야 한다.

상대경로의 지정 방법

상대경로 기반의 #include 선언

```
#include "header.h"
#include "Release₩header0.h"
#include "..₩CProg₩header1.h"
#include "..₩..₩MyHeader₩header2.h"
```

이 문장을 포함하는 소스파일이 저장된 디렉터리 :현재 디렉터리

현재 디렉터리의 서브인 Release 디렉터리

현재 디렉터리의 상위 디렉터리 (..) 의 서브인 Cporg 디렉터리

현재 디렉터리의 상위 디렉터리의 상위 디렉터리의 서브인 MyHeader 디렉터리

상대경로의 지정방식을 기반으로 헤더파일 경로를 명시하면 프로그램의 소스코드가 저장되어 있는 디렉터리를 통째로 이동하는 경우 어디서든 컴파일 및 실행이 가능해진다.

헤더파일에 포함하는 일반적인 내용

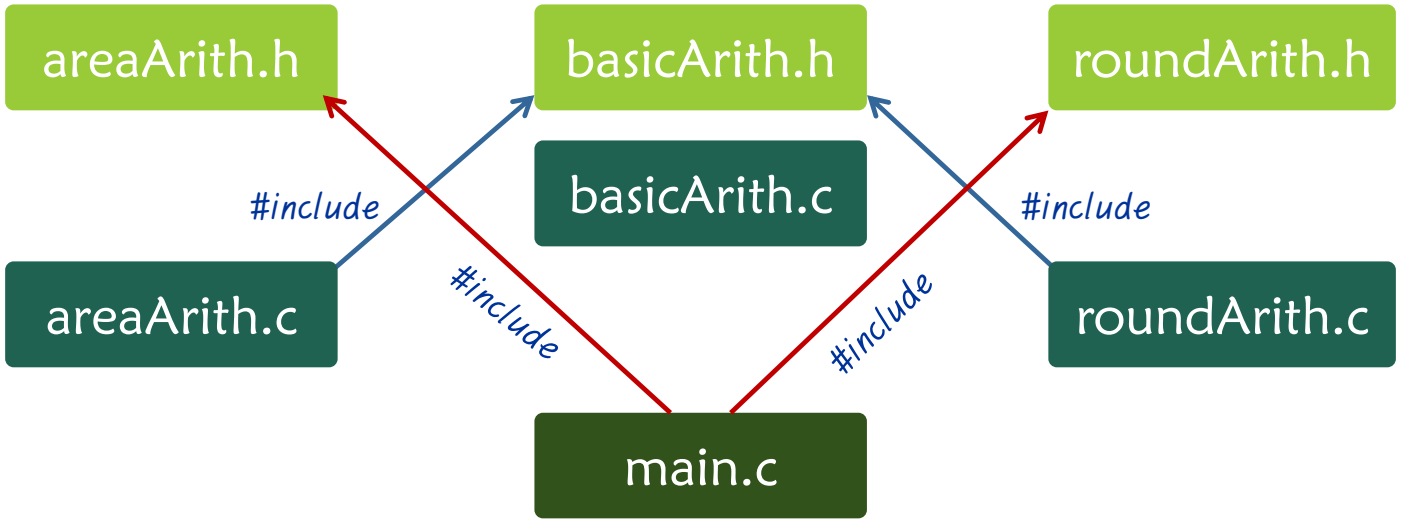
- 전역변수
- 함수의 원형
- 구조체 정의

헤더파일에 삽입이 되는 가장 일반적인 선언의 유형

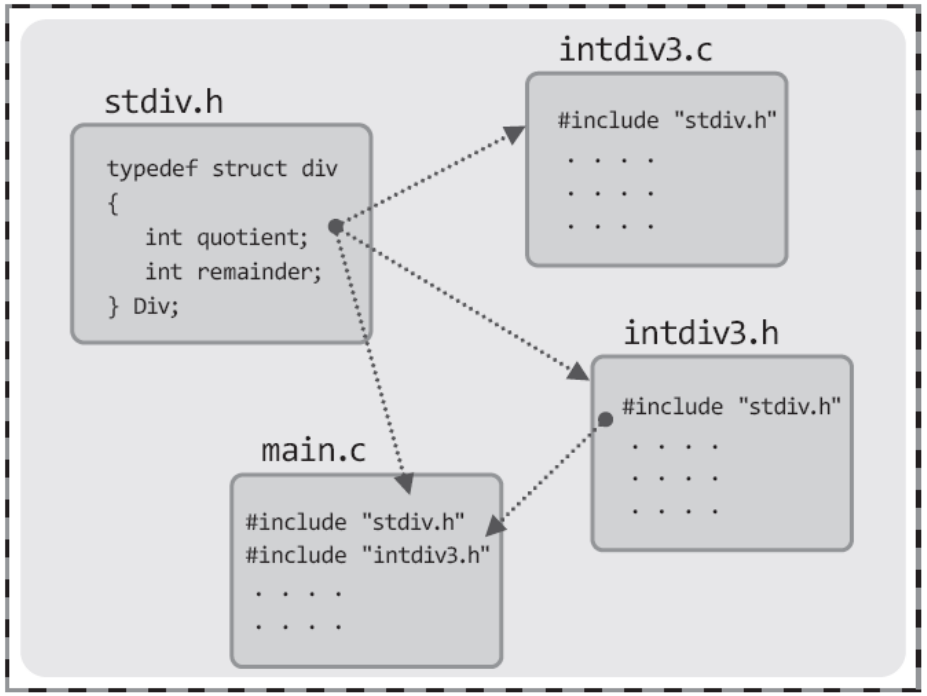
```
extern int num;                 // 외부 파일에 있는 전역변수
extern int GetNum(void);  // 외부 파일에 있는 함수의 원형
// extern 생략 가능
```

헤더파일과 소스파일의 포함관계

- 예제의 소스파일과 헤더파일의 구성 및 내용
  - basicArith.h basicArith.c → 수학과 관련된 기본적인 연산의 함수의 정의 및 선언
  - areaArith.h areaArith.c → 넓이계산과 관련된 함수의 정의 및 선언
  - roundArith.h roundArith.c → 둘레계산과 관련된 함수의 정의 및 선언
  - main.c



헤더파일의 중복삽입 문제

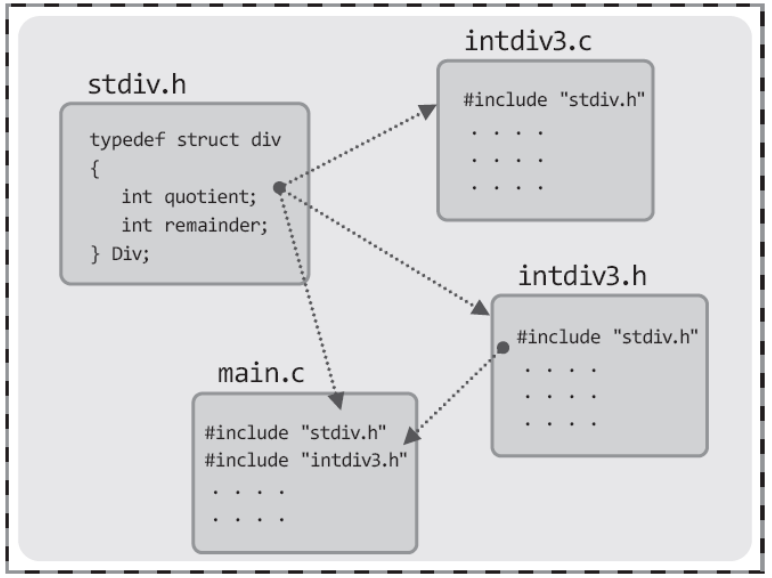


일반적으로 선언(예로 함수의 선언)은 두 번 이상 포함시켜도 문제되지 않는다. 그러나 정의(예로 구조체 및 함수의 정의)는 두 번 이상 포함시키면 문제가 된다.

main.c는 결과적으로 구조체 Div의 정의를 두 번 포함하는 꼴이 된다! 그런데 구조체의 정의는 하나의 소스 파일 내에서 중복될 수 없다!

조건부 컴파일을 활용한 중복삽입 문제의 해결

#ifndef . . . #endif : 정의되지 않았다면



중복 삽입문제의 해결책

```
#ifndef __STDIV2_H__
#define __STDIV2_H__
typedef struct div
{
    int quotient;     // 몫
    int remainder;    // 나머지
} Div;
#endif
```

__STDIV2_H__가 정의되지 않았다면 ~endif 까지 컴파일 대상에 포함

매크로 __STDIV2_H__ 와 #ifndef의 효과가 main.c에서 어떻게 나타나는지 그려보자!

위와 같은 이유로 모든 헤더파일은 #ifndef~#endif로 감싸는 것이 안전하고 또 일반적이다!