Topic 4 Function
Part 1

Part 1

- main() function
- Function Definition and Invocation
- Parameter Passing: Call by Value/Reference
- const Parameter
- Return Value

Main Function

- A program shall contain a global function named main, which is the designated start of the program

```
int main ( ) { }
int main ( int argc, char* argv[ ] ) { }
```

- The main function is called at program startup after initialization of the non-local objects with static storage duration.
- It is the designated entry point to a program that is executed
- The main function has several special properties:
  - It cannot be used anywhere in the program (recursion(x), address(x))
  - It cannot be predefined and cannot be overloaded
  - The body of the main function does not need to contain the return statement
  - Execution of the return is equivalent to first leaving the function normally (which destroys the objects with automatic storage duration) and then calling std::exit with the same argument as the argument of the return

https://en.cppreference.com/w/cpp/language/main_function

Storage Class in C++

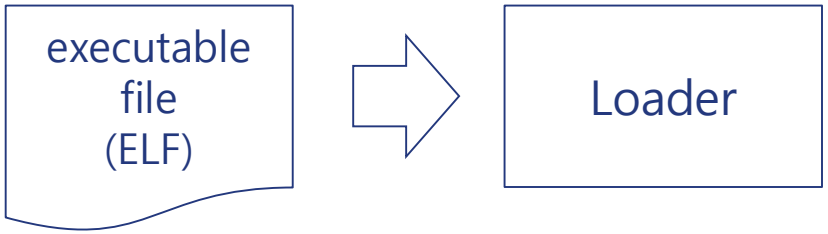
- The storage class specifiers control two independent properties of the name: its **storage duration** and **its linkage**

Keyword (기본값)	Storage Duration	Linkage	Linkage 예	Lifetime	Visibility (name lookup)	Initial Vaule
auto (지역변수)	automatic (block-scope)	no linkage	- 함수 파라미터 - non-extern block-scope 변수	Block ({ })	Local	Garbage
extern (모든함수, 파일범위 변수)	static (file-scope)	external linkage	- non-static 함수 - extern 변수 - file-scope non-static 변수	Whole Program	translation unit 외부 (다른 소스 파일)	Zero
static	static (file-scope)	internal linkage	- static file-scope 식별자(함수, 변수)	Whole Program	translation unit 내부 (동일한 소스 파일)	Zero
register	automatic	no linkage		Block ({ })	Local	Garbage

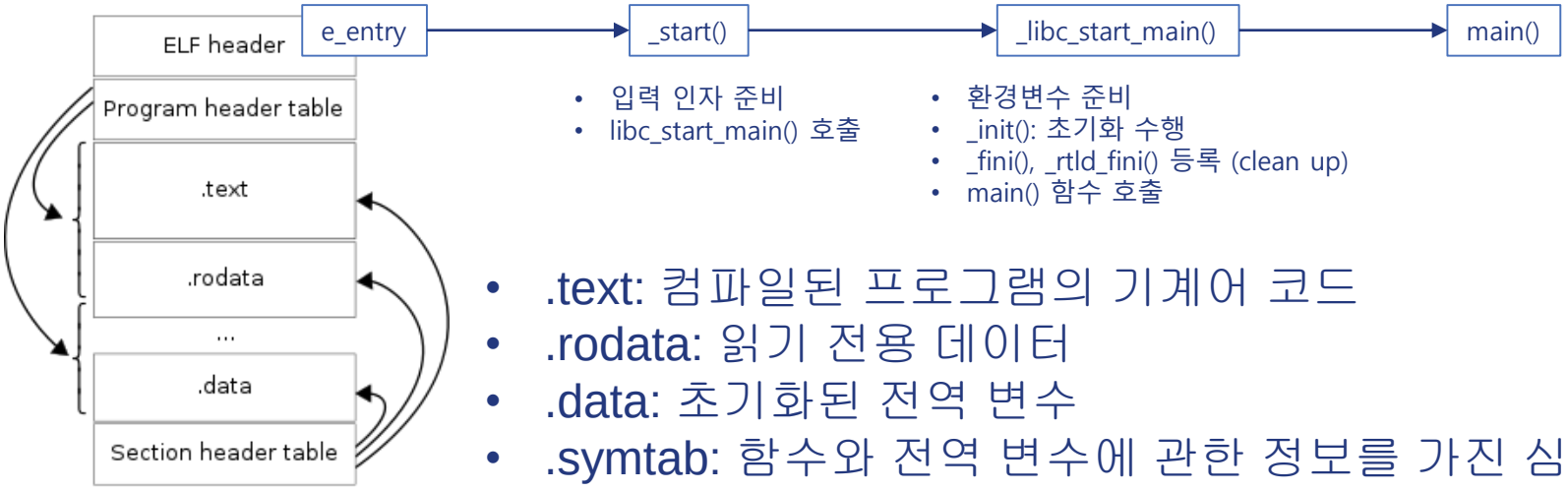
https://en.cppreference.com/w/c/language/storage_duration

Executing main() in C++ (Linux)

❖ Execution Process



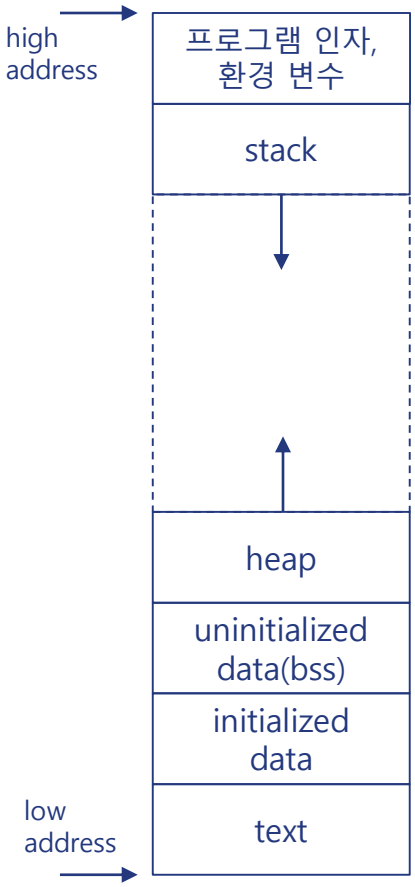
- 실행파일에 있는 코드와 데이터를 메모리에 복사
- 제어를 프로그램의 시작으로 옮김



- .text: 컴파일된 프로그램의 기계어 코드
- .rodata: 읽기 전용 데이터
- .data: 초기화된 전역 변수
- .symtab: 함수와 전역 변수에 관한 정보를 가진 심볼 테이블

https://www.geeksforgeeks.org/executing-main-in-c-behind-the-scene/
https://www.linuxjournal.com/article/6463
https://en.wikipedia.org/wiki/Executable_and_Linkable_Format

Memory Layout



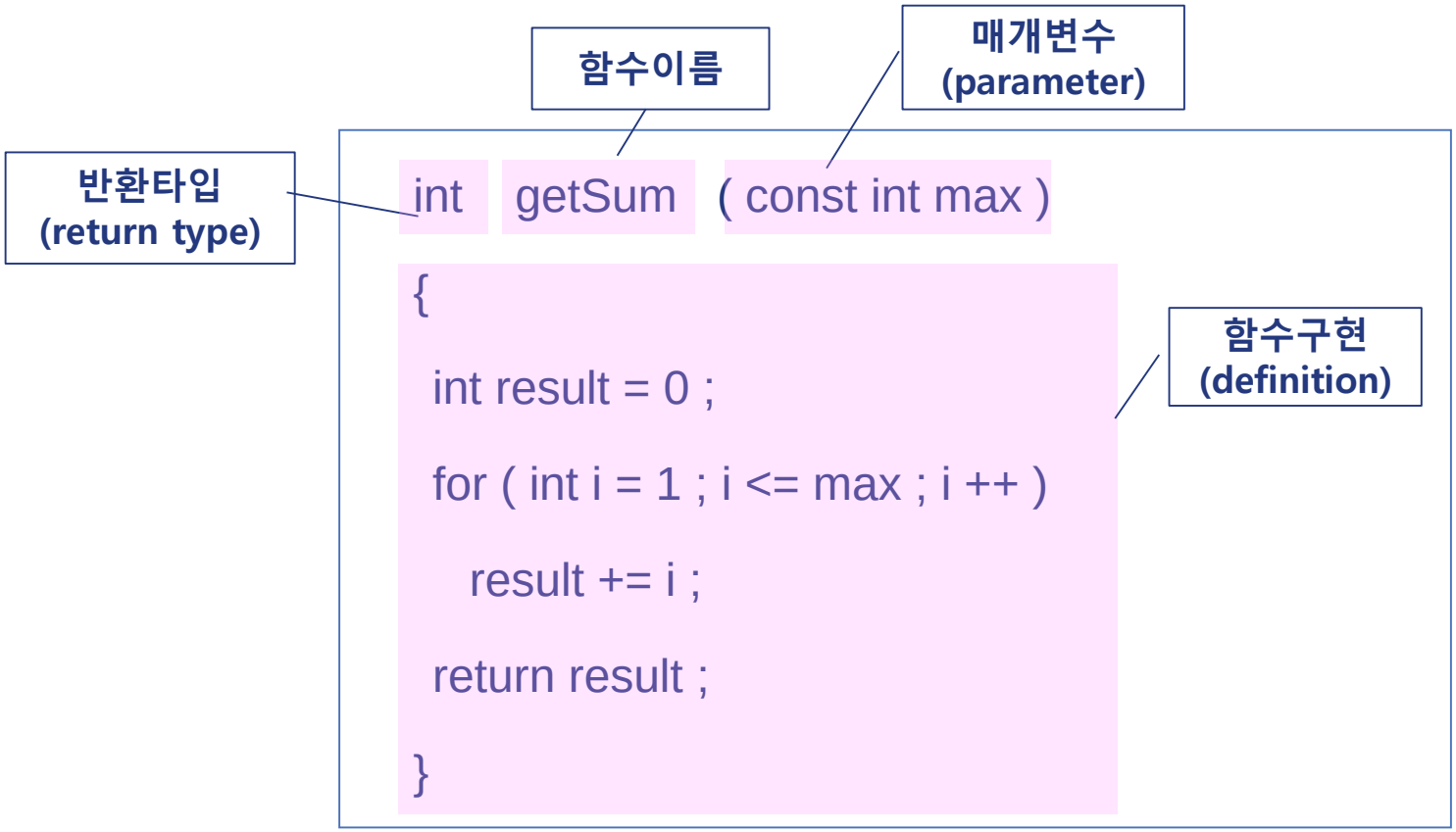
- arg.exe I am Kim → int main(argc, argv[])
- PATH=C:\Clion\bin
- stack: 함수 호출 시 함수 반환 주소, 인자 값, 로컬 변수 등이 추가됨
- heap: new, delete 연산자로 관리하는 메모리 공간
- uninitialized data : 프로그래머가 초기화 하지 않은 전역변수 및 static 변수가 위치함 (실행 시 0으로 초기화)
- initialized data : 프로그래머가 초기화한 global, static, const, extern 변수가 위치함 (읽기전용과 읽기쓰기 영역으로 나눠짐)
- text (read-only) : 실행 가능한 명령어

https://www.geeksforgeeks.org/memory-layout-of-c-program/

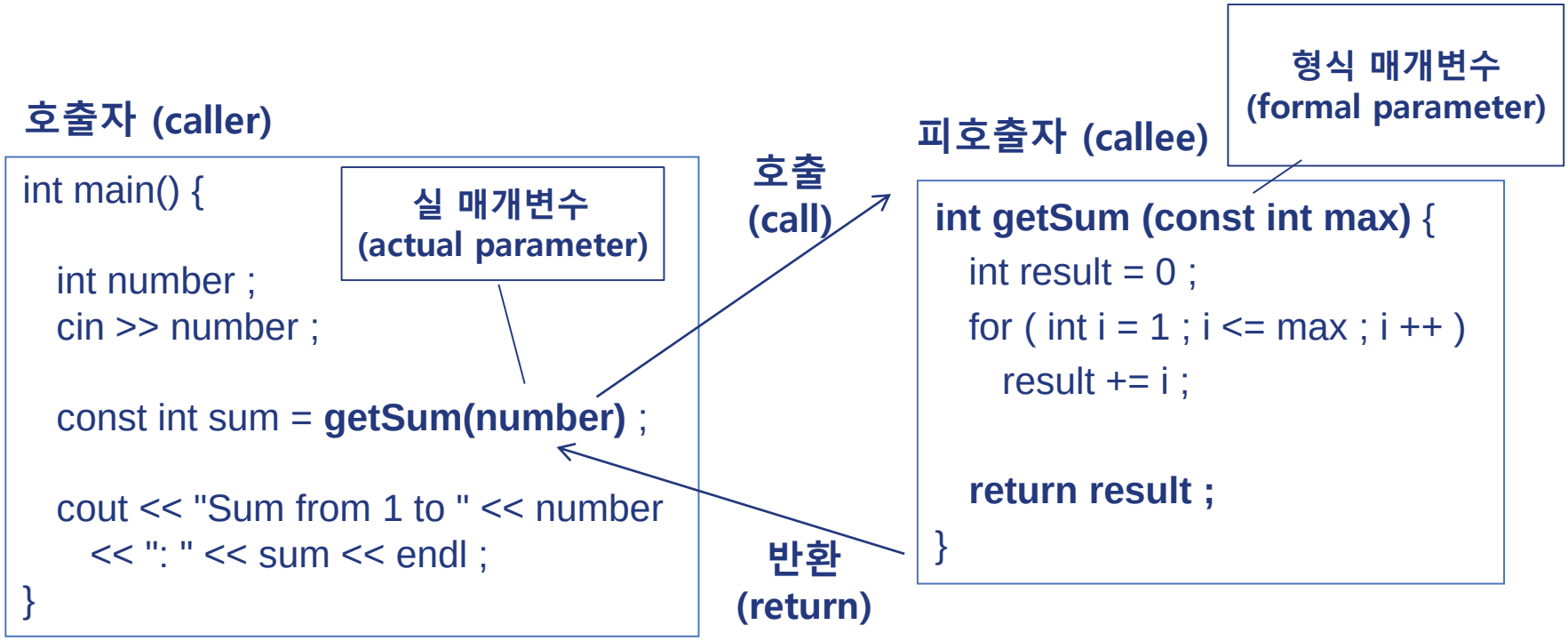
함수 정의와 호출

```
# include <iostream>
using namespace std ;

int getSum (const int max) {
   int result = 0 ;
   for ( int i = 1 ; i <= max ; i ++ )
      result += i ;

   return result ;
}

int main() {

   int number ;
   cin >> number ;

   const int sum = getSum(number) ;

   cout << "Sum from 1 to " << number << ": " << sum << endl ;
}
```

함수의 정의


함수이름
매개변수
(parameter)
반환타입
(return type)
int getSum (const int max)
{
int result = 0 ;
for (int i = 1 ; i <= max ; i ++)
result += i ;
return result ;
}
함수구현
(definition)

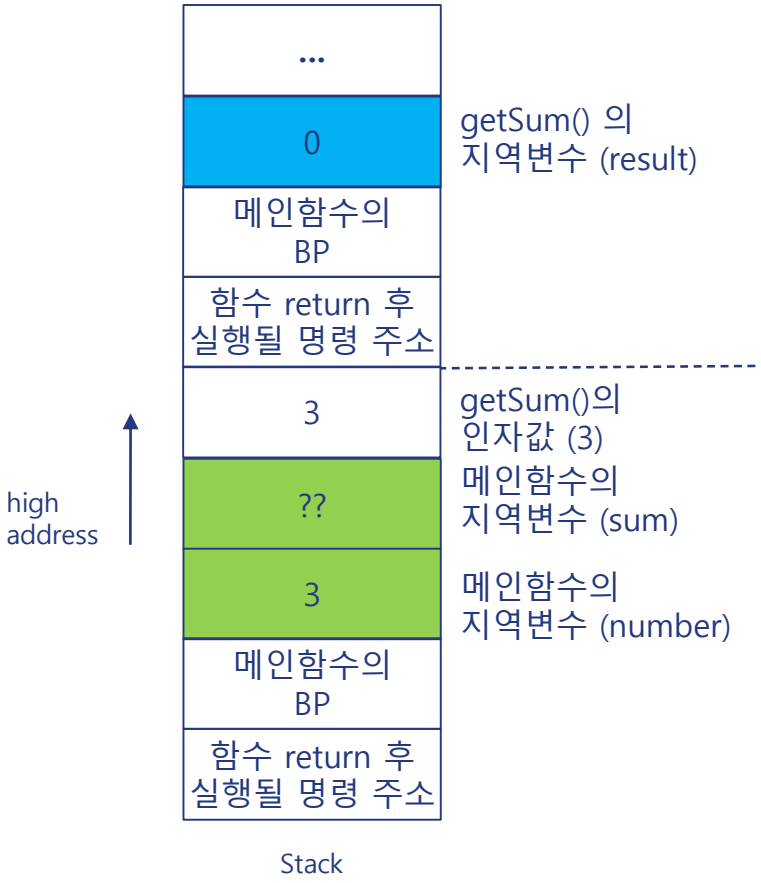
함수의 호출


호출자 (caller)
int main() {
int number ;
cin >> number ;
const int sum = getSum(number) ;
cout << "Sum from 1 to " << number
<< ": " << sum << endl ;
}
실 매개변수
(actual parameter)
호출
(call)
반환
(return)
피호출자 (callee)
형식 매개변수
(formal parameter)
int getSum (const int max) {
int result = 0 ;
for (int i = 1 ; i <= max ; i ++)
result += i ;
return result ;
}

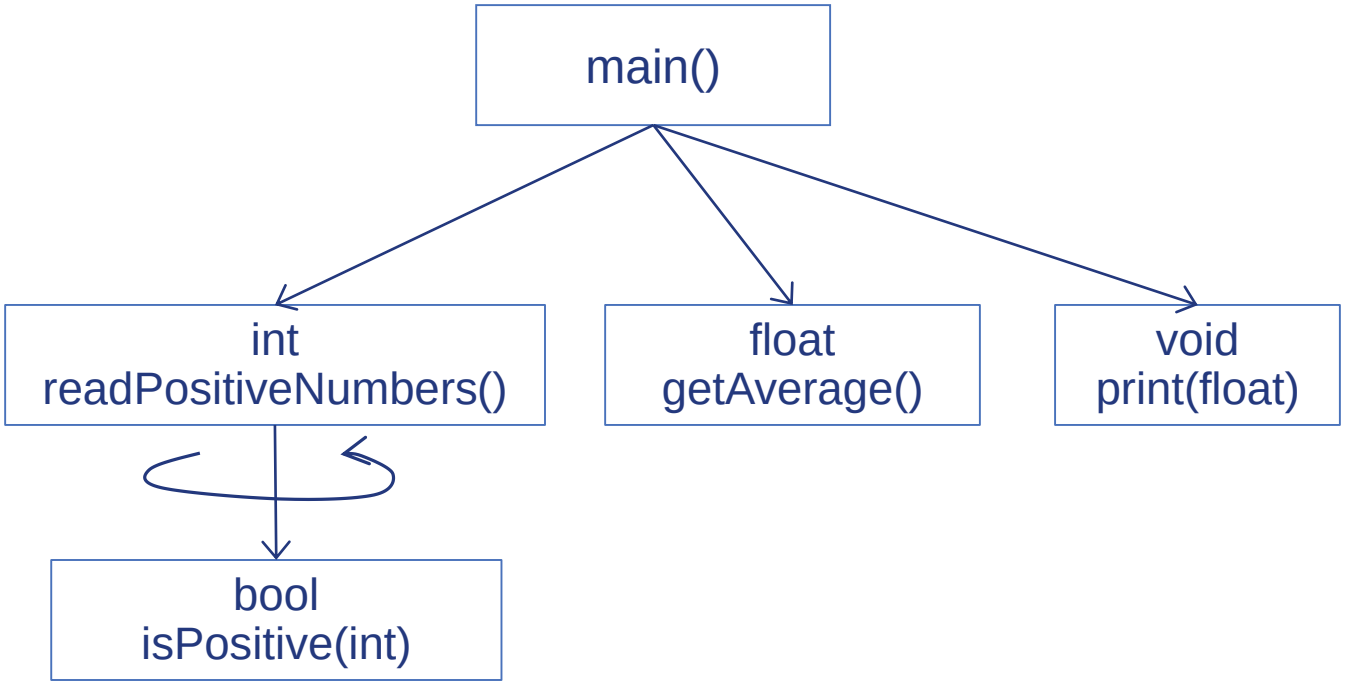
Call Stack

```
#include <iostream>

int getSum (const int max) {
   int result = 0 ;
   for ( int i = 1 ; i <= max ; i ++ )
      result += i ;
   return result ;
}
```

```
int main() {
   int number ;
   cin >> number ;  //3
   const int sum = getSum(number) ;

   cout << "Sum from 1 to " << number
      << ": " << sum << endl ;
}
```



함수 정의 및 호출: 예제


main()
int
readPositiveNumbers()
float
getAverage()
void
print(float)
bool
isPositive(int)

함수 정의와 호출

```
#include <vector>
#include <iostream>
using namespace std ;

vector<int> numbers ;

bool isPositive(const int n) {
   return n > 0 ;
}
int readPositiveNumbers() {
   while ( true ) {
      int number ;
      cin >> number ;
      if ( ! isPositive(number) ) break ;
      numbers.push_back(number) ;
   }
   return numbers.size() ;
}
```

```
float getAverage() {
   int sum = 0 ;

   for ( unsigned int i = 0 ; i < numbers.size() ; i ++ ) {
      sum += numbers[i] ;
   }
   return static_cast<float> (sum) / numbers.size() ;
}

void print(const float value) {
   cout << "The average is " << value << endl ;
}
int main() {
   readPositiveNumbers() ;
   const float average = getAverage() ;
   print(average) ;
}
```

Good Design

- 함수의 이름은 함수가 수행하는 기능의 결과만을 뜻해야 한다
  - 함수의 이름은 반드시 함수의 수행의 최종 결과를 뜻하도록
    - 예) sqrt(), getAverage(), 그리고 getSum()
  - 함수의 이름이 함수 수행의 전체 결과를 뜻해야 하며, 그 일부 결과 또는 과정을 뜻하지 않도록 해야 한다

```
int 0000(int destFloor, int curFloors[], int elevatorNo) {
    // step 1: 각 엘리베이터의 현재 위치(curFloors)와 목적지 층(destFloor)을
    // 바탕으로 목적지 층으로 이동시킬 엘리베이터를 선택한다.
    // step 2: 선택된 엘리베이터의 모터를 목적지 층 방향으로 작동시킨다.
    // step 3: 엘리베이터가 목적지 층에 도착하면 모터를 중단시키고 문을 연다.
    // step 4: 일정 시간이 경과하면 문을 닫는다.
}
```

Good Design

- 한줄에는 하나의 명령문을 사용
- side-effect가 있는 문장은 다른 문장과 독립적으로 하나의 문장

```
# include <iostream>
using namespace std ;
int add(int x, int y) { return x+y; }
int main() {
   int x = 0;
   cout << ++x << " " << add(x, x) << endl ;
   return 0;
}
```

Visual Studio 2010	MinGW GCC 4.7.2
1 2	1 0

Good Design: 함수의 응집도

- 함수는 오직 하나의 기능만을 제공해야 한다.
- 함수가 2개 이상의 기능을 제공할 경우 즉 낮은 응집도를 가질 경우 함수에 대한 이해 및 유지보수의 어려움이 초래된다

```
int getSumAndProduct(int flag, int val1, int val2) {
   int result = 0 ;
   switch ( flag ) {
      case 0 : result = val1 + val2 ; break ;
      case 1 : result = val1 * val2 ; break ;
   }
   return result ;
}
```

```
int manageFile(int flag, string filename) {
   switch ( flag ) {
      case 0 : // fileName의 파일 삭제
      case 1: // fileName의 파일 정보 출력
      case 2: // fileName의 파일 크기 반환
   }
}
```

함수 선언

```
#include <vector>
#include <iostream>
using namespace std ;


bool isPositive(const int n) ;
int readPositiveNumbers() ;
float getAverage() ;
void print(const float value) ;

vector<int> numbers ;

int main() {
   readPositiveNumbers() ;
   const float average = getAverage() ;
   print(average) ;
}
```

```
int readPositiveNumbers() {
   while ( true ) {
      int number ;
      cin >> number ;
      if (! isPositive(number) ) break ;
      numbers.push_back(number) ;
   }
   return numbers.size() ;
}
float getAverage() {
   int sum = 0 ;
   for ( unsigned int i = 0 ; i < numbers.size() ; i ++ ) {
      sum += numbers[i] ;
   }
   return static_cast<float> (sum) / numbers.size() ;
}
void print(const float value) {
   cout << "The average is " << value << endl ;
}
```

Header 파일을 이용한 함수 선언

```
// MyHeader.h

#ifndef __MY_HEADER_H
#define __MY_HEADER_H

bool isPositive(const int) ;
int readPositiveNumbers() ;
float getAverage() ;
void print(const float) ;

#endif
```

```
#include <vector>
#include <iostream>
#include "MyHeader.h“

using namespace std ;
vector<int> numbers ;
int main() {
   readPositiveNumbers() ;
   const float average = getAverage() ;
   print(average) ;
}
int readPositiveNumbers() {
   …
}
float getAverage()
   …
}
void print(const float value) {
   …
}
bool isPositive(const int n) {
   …
}
```

선언과 정의

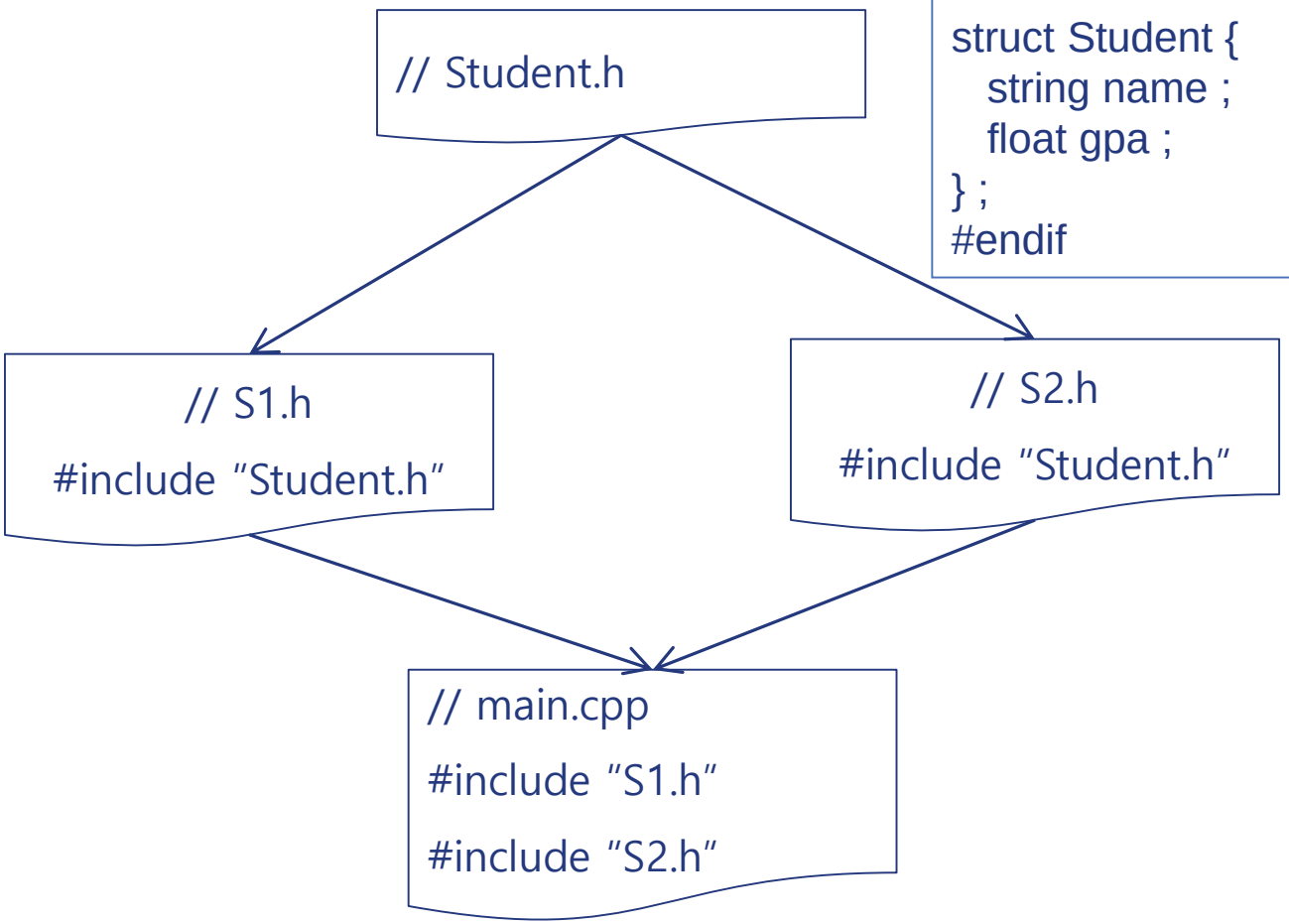
- 변수, 함수, enum, 클래스 등은 사용되기 전에 선언
- 선언은 여러 회 허용되지만 정의는 오직 1회만 가능 (One Definition Rule)

선언 (일반적으로 .h 파일)	정의 (일반적으로 cpp 파일)

int max(const int, const int) ;
struct Student ;
	inline int max(const int a, const int b) {
	  return (a>b) ? a : b ;
	}
	struct Student {
	  string name ;
	  float gpa ;
	} ;

헤더파일의 1회 포함의 보장

```
#ifndef __STUDENT_H
#define __STUDENT_H

struct Student {
   string name ;
   float gpa ;
} ;
#endif
```



#pragma once의 사용

A non-standard but widely supported preprocessor directive designed to cause the current source file to be included only once in a single compilation

```
// Student.h


# pragma once


struct Student {
    string name ;
    float gpa ;
} ;
```