Topic 4 Function

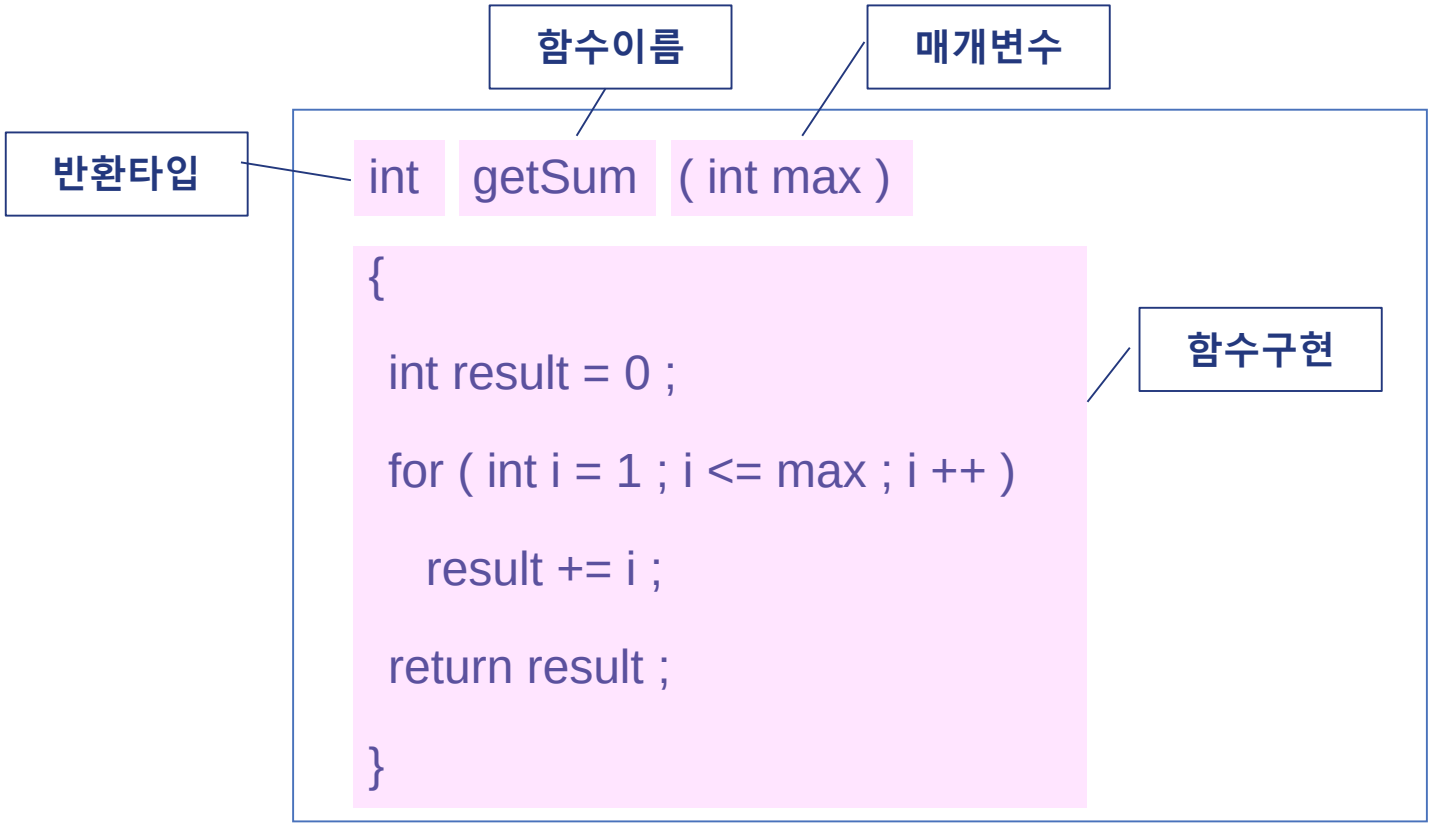
Function Definition and Invocation
Parameter Passing
inline function

Static local variable
Recursive function
Pre/post condition
File scope vs program scope

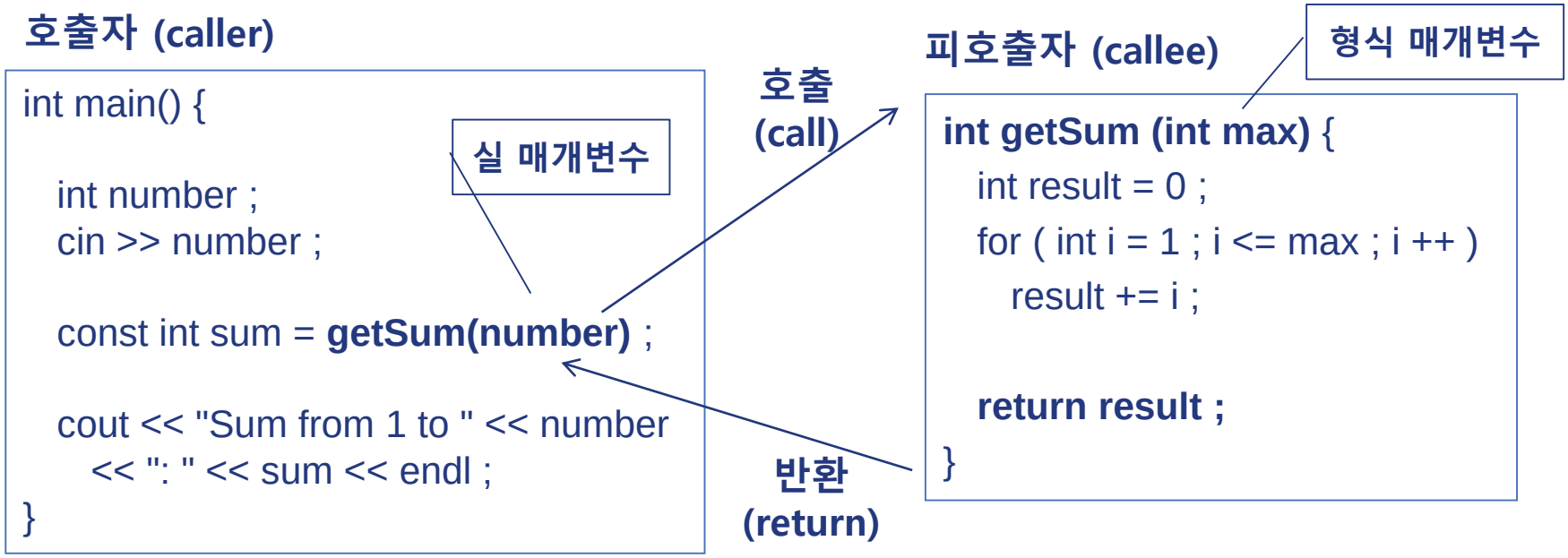
함수 정의와 호출

```
# 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 ;
}
```

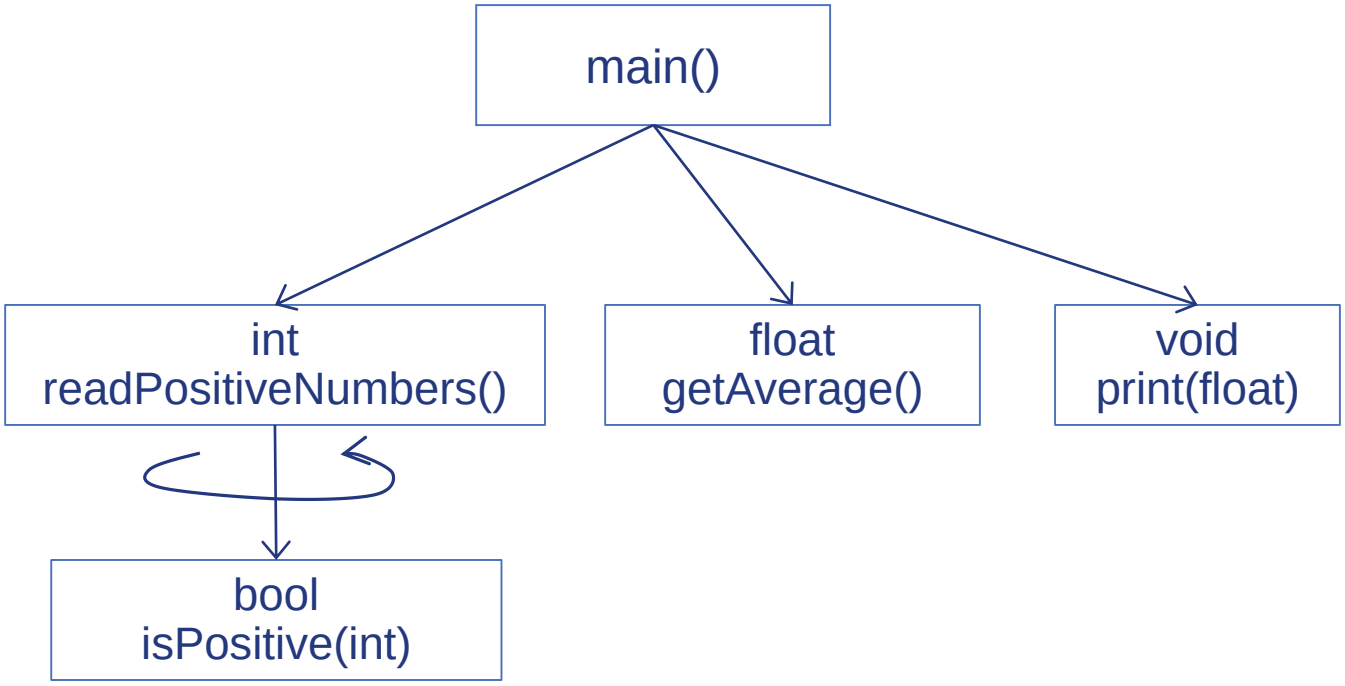
함수의 정의


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

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
호출
(call)
반환
(return)
피호출자 (callee)
형식 매개변수
int getSum (int max) {
int result = 0 ;
for (int i = 1 ; i <= max ; i ++)
result += i ;
return result ;
}

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 뜻해야하며, 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) {
   …
}
```

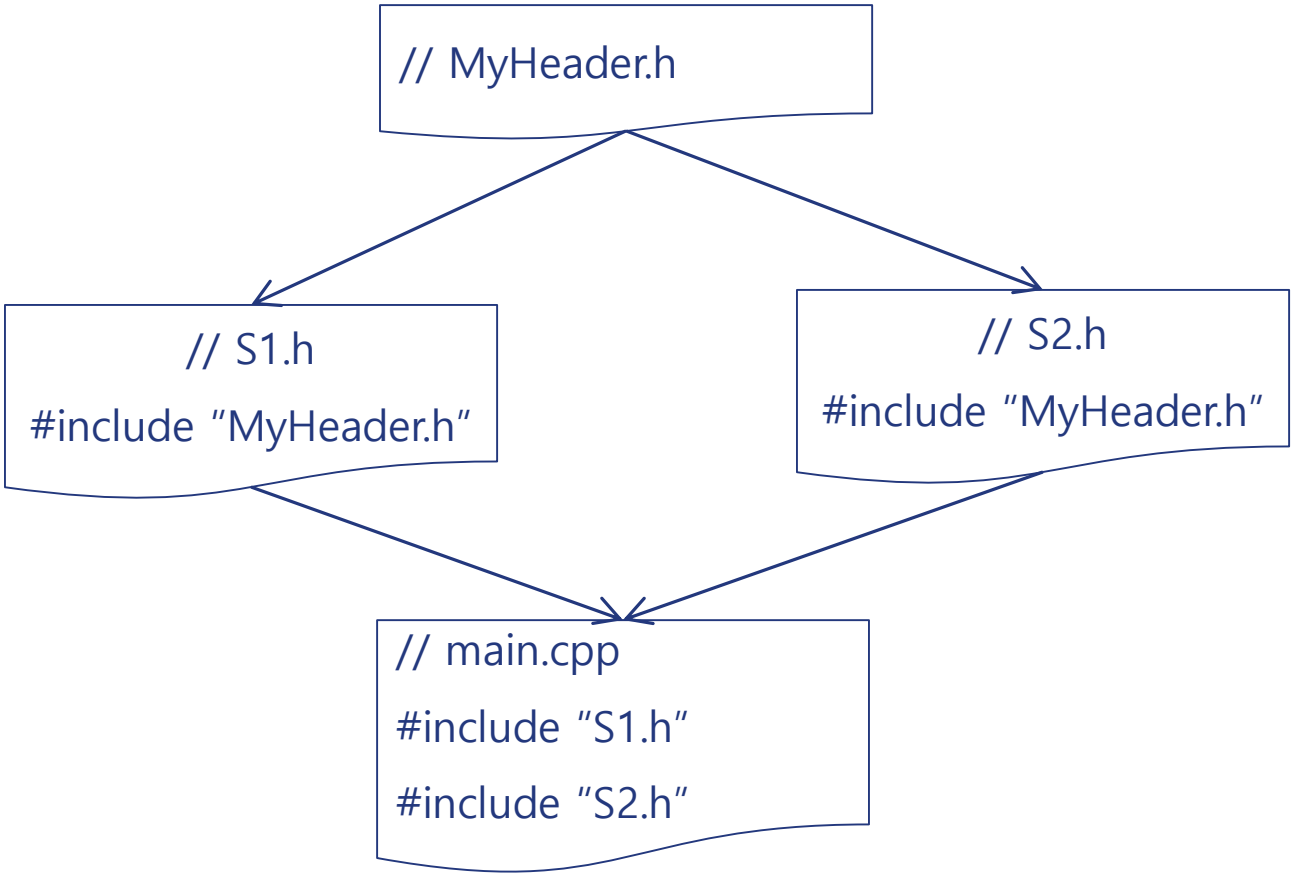
선언과 정의

- 선언은 여러 회 허용되지만 정의는 오직 1회만 가능

선언	정의

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



#pragma once의 사용

```
// MyHeader.h

# pragma once

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

인자 전달 방식: call by value & call by reference

- Argument passing
  - call by value: does not modify the actual argument
  - call by reference: change the value of the actual argument

```
void f(int x, int& y) { // 형식 매개변수 x, y
        x++ ;
        y++ ;
}
int main() {
        int i = 10 ;
        int j = 20 ;

        cout << i << '₩t' << j << endl ; // 10 20
        f(i, j) ; // 실 매개변수 i, j
        cout << i << '₩t' << j << endl ; // 10 21
}
```

call by value & call by reference: An Example

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

void callByValue(int, int ) ;
void callByReference(int&, int&) ;

int main() {
   int number1=10, number2=20;

   int v1 = number1, v2=number2 ;
   callByValue(v1, v2) ;
   cout  << v1 << '\t' << v2 << endl ;

   int r1 = number1, r2=number2 ;
   callByReference(r1, r2) ;
   cout << r1 << '\t' << r2 << endl ;
}
```

```
void callByValue(int n1, int n2) {
   int temp = n1 ;
   n1 = n2 ;
   n2 = temp ;
}

void callByReference(int& n1, int& n2) {
   int temp = n1 ;
   n1 = n2 ;
   n2 = temp ;
}
```

실 매개변수 값의 변경
Pointer and reference

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

void swapByPointer(int* n1, int* n2) {
   int temp = *n1 ;
   *n1 = *n2 ;
   *n2 = temp ;
}
void swapByReference(int& n1, int& n2)
{
   int temp = n1 ;
   n1 = n2 ;
   n2 = temp ;
}
```

```
int main() {
   cout << "Enter two integers !" << endl ;

   int number1, number2 ;
   cin >> number1 >> number2 ;

   swapByPointer(&number1, &number2) ;
   swapByReference(number1, number2) ;

   cout << number1 << '₩t' << number2
<< endl ;
}
```

실 매개변수 값의 변경
Pointer and reference

	포인터 방식	참조 방식
실 매개변수 명시	int x ; f(&x) ;	int x ; f(x) ;
형식 매개변수 선언	f(int* n)	f(int& n)
실 매개변수 접근 방법	*n = 10	n = 10

매개변수 전달 방법: 요약

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

void multiplyWithRef(
   int& number, int times) {
   number *= times ;
}

void multiplyWithValue(
   int number, int times) {
   number *= times ;
}

void multiplyWithPointer(
   int* number, int times) {
   *number *= times ;
}
```

```
int main() {

   int intVal ;
   int intTimes ;
   cin >> intVal >> intTimes ;

   // 참조를 이용한 매개 변수 전달
   multiplyWithRef(intVal, intTimes) ;
   cout << intVal << endl ;

   // 일반 변수를 이용한 매개변수 전달
   multiplyWithValue(intVal, intTimes) ;
   cout << intVal << endl ;

   // 포인터를 이용한 매개변수 전달
   multiplyWithPointer(&intVal, intTimes) ;
   cout << intVal << endl ;
}
```

인자 전달 방식: const 상수 매개변수

- 매개변수를 const로 선언함으로써 해당 매개변수의 값이 피호출함수에서 변경되는 것을 불허함

```
void f(int x, const int y) { // y는 상수 매개변수임
    cout << x << '₩t' << y << endl ;
    x++ ;
    y++ ;   // ERROR: y는 상수이므로 변경될 수 없음
}
```

인자 전달 방식: const
상수 매개변수

const 미사용시	const 사용시

void print(char* str) {
  cout << str << endl ;
  strcpy(str, "Hi") ;
}
int main() {
  char* msg = "Hello" ;
  print(msg) ; // Hello
  print(msg) ; // Hi
}
	void print(**const** char* str) {
	  cout << str << endl ;
	  strcpy(str, "Hi") ; // **ERROR**
	}
	int main() {
	  char* msg = "Hello" ;
	  print(msg) ; // Hello
	  print(msg) ; // Hello
	}

인자 전달 방식: const
상수 매개변수

To prevent the object from changing in the called function, declare it as "**const**"

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

struct Point { int x, y ; } ;
void readPoints(Point* const pts, const int size) ;
bool isEqual(const Point pt1, const Point pt2) ;
bool findPoint(const Point* const pts, const int size, const Point pt) ;
int main() {
  int no ;
  cin >> no ;
  Point* pts = new Point[no] ;

  readPoints(pts, no) ;

  Point pt ;
  cin >> pt.x >> pt.y ;

  string msg = findPoint(pts, no, pt) ? "Found." : "Not Found." ;
  cout << msg << endl ;
}
```

인자의 유형

유형	예
기본 타입	void print(**float** value) ; bool isPositive(**int** n) ;
나열형	enum Grade { FRESH=1, SOPHOMORE, JUNIOR, SENIOR } ; void upgrade(**Grade**&) ;
배열	int find(int data**[]**, int size, int value) ;
구조체	struct Student { string name ; float gpa ; } ; float getGPA(const **Student**&) ;

인자: 나열형

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

enum Grade { FRESH=1, SOPHOMORE, JUNIOR, SENIOR } ;

void print(const Grade grade) {
   string gradeLabels[] = { "Fresh", "Sophomore", "Junior", "Senior" } ;
   cout << gradeLabels[grade-1] << endl ;
}
void upgrade(Grade& now) {
   if ( now != SENIOR ) now = static_cast<Grade>(now+1) ;
}
int main() {
   Grade grade = FRESH ;
   print(grade) ;                              // “Fresh”
   upgrade(grade) ; print(grade) ;             // “Sophomore”
   upgrade(grade) ; print(grade) ;             // “Junior”
   upgrade(grade) ; print(grade) ;             // “Senior”
   upgrade(grade) ; print(grade) ;             // “Senior”
}
```

인자: 나열형의 배열

```
const int COUNT = 4 ;
enum Grade { FRESH=1, SOPHOMORE, JUNIOR, SENIOR } ;
void upgrade(Grade& now) {
  if ( now != SENIOR ) now = static_cast<Grade>(now+1) ;
}
void print(const Grade grade) {
  string gradeLabels[] =  { "Fresh", "Sophomore", "Junior", "Senior" } ;
  cout << gradeLabels[grade-1] << endl ;
}
void upgradeAll(Grade grades[], const int size) {
  for ( int i = 0 ; i < size ; i ++ ) upgrade(grades[i]) ;
}
void printAll(const Grade grades[], const int size) {
  for ( int i = 0 ; i < size ; i ++ ) print(grades[i]) ;
}
```

```
int main() {
  Grade grades[COUNT] = {FRESH, SOPHOMORE, JUNIOR, SENIOR} ;
  upgradeAll(grades, COUNT) ;
  printAll(grades, COUNT) ;
}
```

인자: 문자의 배열

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

int findChar(const char *const str, const char ch) ;

int main() {
   char str[] = "A string to be searched." ;
   cout << "Enter a character to find in " << str << endl ;
   char ch ;
   cin >> ch ;
   int index = findChar(str, ch) ;
   if ( index == -1 )
      cout << ch << " isn\'t in " << str << endl ;
   else
      cout << ch << " is found at "<< index << endl ;
}

int findChar(const char *const str, const char ch) {
   for ( int i = 0 ; i < strlen(str) ; i ++ )
      if ( str[i] == ch ) return i ;
   return -1 ;
}
```

인자: 구조체

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

struct Point {
  int x, y ;
} ;

bool isEqual(const Point pt1, const Point pt2) ;

int main() {
   Point point1, point2 ;
   cin >> point1.x >> point1.y ;
   cin >> point2.x >> point2.y ;

   string msg = isEqual(point1, point2) ? "Two points are equal." : "Two points are not equal." ;
   cout << msg << endl ;
}

bool isEqual(const Point pt1, const Point pt2) {
   return pt1.x == pt2.x && pt1.y == pt2.y ;
}
```

인자: 구조체의 배열

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

struct Point { int x, y ; } ;
void readPoints(Point* pts, int size) ;
bool isEqual(Point pt1, Point pt2) ;
bool findPoint(Point* pts, int size, Point pt) ;
int main() {
  int no ;
  cin >> no ;
  Point* pts = new Point[no] ;

  readPoints(pts, no) ;

  Point pt ;
  cin >> pt.x >> pt.y ;

  string msg = findPoint(pts, no, pt) ? "Found." : "Not Found." ;
  cout << msg << endl ;
}
```

```
void readPoints(Point* const pts, const int
size) {
  for ( int i = 0 ; i < size ; i ++ )
     pts[i].x = pts[i].y = i ;
}
bool isEqual(const Point pt1, const Point
pt2) {
  return pt1.x == pt2.x && pt1.y == pt2.y ;
}
bool findPoint(const Point* const pts,
const int size, const Point pt) {
  for ( int i = 0 ; i < size ; i ++ )
     if ( isEqual(pts[i], pt) ) return true ;

  return false ;
}
```

Call by reference

- Call by reference is useful when a large object is passed

```
struct StudentInfo {
    string name ;
    int age ;
    int year ;
    float gpa ;
} ;

void printStudentInfo(StudentInfo& st) {
    cout << st.name << '\'t ;
    cout << st.age << '\t' ;
    cout << st.year << '\t' ;
    cout << st.gpa << endl ;
}
```

Call by reference with const

- To prevent the object from changing in the called function, declare it as "const"

```
struct StudentInfo {
    string name ;
    int age ;
    int year ;
    float gpa ;
} ;

void printStudentInfo(const StudentInfo& st) {
    cin >> st.name ; // error
    cout << st.name << '\'t ;
    cout << st.age << '\t' ;
    cout << st.year << '\t' ;
    cout << st.gpa << endl ;
}
```

구조체의 전달: 요약

```
struct Point {
  int x, y ;
} ;
void readPoint(Point& pt) {
        cin >> pt.x >> pt.y ;
}
void printPoint(const Point& pt) {
        cout << pt.x << '₩t' << pt.y << endl ;
}
bool isEqualPoint(const Point& pt1, const Point& pt2) {
        return pt1.x == pt2.x && pt1.y == pt2.y ;
}
int main() {
        Point point1, point2 ;
        readPoint(point1) ;
        readPoint(point2) ;

        printPoint(point1) ;
        printPoint(point2) ;

        string msg = isEqualPoint(point1, point2) ? "Equal." : "Not equal." ;
        cout << msg << endl ;
}
```

구조체의 전달 요약

	형식 매개 변수 명시	예
실 매개변수의 변경 불가시	상수 참조 사용	printPoint(const Point&) ; isEqualPoint(const Point&, const Point&) ;
실 매개변수의 변경 허용시	참조 사용	readPoint(Point&) ;

인자 전달 방식: 요약

- in 매개변수: 호출자에서 전달된 매개변수의 값이 피호출자에서 변경되는 것을 허용하지 않는 경우
- out 매개 변수: 피호출함수에서의 변경이 호출함수의 실 매개변수에 반영되는 경우

유형	in 매개변수	out 매개변수
기본 또는 나열형	const	&
배열	상수에 대한 상수 포인터 const char* const str	상수 포인터 char* const str
구조체	const &	&

기본 매개변수 값

- 피호출 함수의 선언에서 생략된 매개변수에 대한 기본(default) 값을 미리 정의한 경우에만 실 매개변수 값의 생략이 가능

```
int multiplyBy(const int, const int = 1) ; // 2번째 매개변수의 기본 값은 1임
int main () {
  cout << multiply(10, 2) << endl ;  // 20
  cout << multiply(10) << endl ;     // 10
}
int multiplyBy(const int v1, const int v2) { return v1 * v2 ; }
```

기본 매개변수 값

- 뒤쪽의 매개변수부터 기본값을 차례로 지정

```
void f1(int a, int b, int c=5, int d=10) ;        // OK
void f2(int a, int b=5, int c, int d=10) ;        // ERROR
void f3(int a=10, int b, int c, int d) ;          // ERROR
```

- 기본 매개변수의 값은 오직 1회만 지정되어야 하며, 동일한 값이라 하더라도 여러 회 지정되는 것은 허용되지 않음

```
void scaleBy(Circle&, const float = 1.0F) ;       // OK
void scaleBy(Circle&, const float = 1.0F) ;       // ERROR
```

기본 매개변수 값

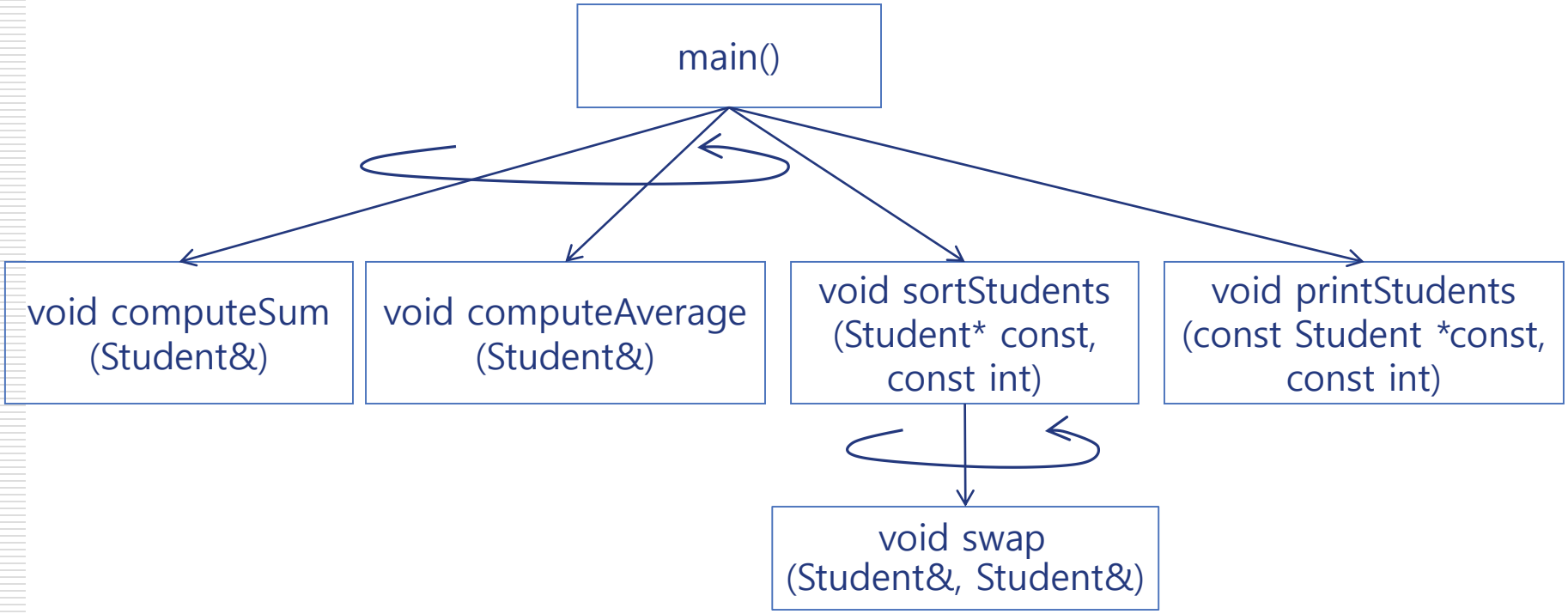
```
#include <iostream>
using namespace std ;

const float PI = 3.14F ;
struct Circle {
   int centerX, centerY ;
   float radius ;
} ;
void scaleBy(Circle& circle, const float ratio = 1.0F) ;
float getArea(const Circle& circle) ;

int main() {
   cout << "Enter the information of a circle: x y radius" << endl ;
   Circle circle ;
   cin >> circle.centerX >> circle.centerY >> circle.radius ;

   cout << "The area is " << getArea(circle) << endl ;
   scaleBy(circle, 2) ;
   cout << "The area is " << getArea(circle) << endl ;
   scaleBy(circle) ;
   cout << "The area is " << getArea(circle) << endl ;
}
void scaleBy(Circle& circle, const float ratio) { circle.radius *= ratio ; }
float getArea(const Circle& circle) { return PI * circle.radius * circle.radius ; }
```

종합 예제: 평균값 계산

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

bool isPositive(const int) ;
int readPositiveNumbers(vector<int>&) ;
float getAverage(const vector<int>&) ;
void printAverage(const float) ;
int main() {
  vector<int> numbers ;
  // 구조체; out 매개변수 => &
  readPositiveNumbers(numbers) ;
  // 구조체; in 매개변수 => const &
  float average = getAverage(numbers) ;
  // 기본 타입; in 매개변수 => const
  printAverage(average) ;
}
```

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

종합 예제 – 학생 성적 관리


main()
void computeSum (Student&)
void computeAverage (Student&)
void sortStudents (Student* const, const int)
void printStudents (const Student *const, const int)
void swap (Student&, Student&)

학생 성적 관리

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

const int SUBJECT_NO = 3 ;
struct Student {
   string name ;
   int score[SUBJECT_NO] ;
   int sum ;
   float average ;
} ;
void computeSum(Student& st) ;
void computeAverage(Student& st) ;
void swap(Student& st1, Student& st2) ;
void sortStudents(Student* const sts, const int no) ;
void printStudents(const Student *const sts, const int no) ;
```

```
void main() {
   int studentNo ;
   cin >> studentNo ;
   Student* students = new Student[studentNo] ;
   for ( int i = 0 ; i < studentNo ; i ++ ) {
     cin >> students[i].name ;
     for ( int subject = 0 ; subject < SUBJECT_NO ; subject ++ )
        cin >> students[i].score[subject] ;
     computeSum(students[i]) ;
     computeAverage(students[i]) ;
   }
   sortStudents(students, studentNo) ;
   cout << endl << "After sorting the student scores..." << endl ;
   printStudents(students, studentNo) ;
   delete [] students ;
}
void computeSum(Student& st) {
   for ( int subject = 0 ; subject < SUBJECT_NO ; subject ++ )
     st.sum += st.score[subject] ;
}
void computeAverage(Student& st) {
   computeSum(st) ;
   st.average = static_cast<float> (st.sum) / SUBJECT_NO ;
}
```

```
void swap(Student& st1, Student& st2) {
  Student temp = st1 ;
  st1 = st2 ;
  st2 = temp ;
}
void sortStudents(Student* const sts, const int no) {
  for ( int i = 0 ; i < no - 1 ; i ++ )
    for ( int j = i + 1 ; j < no ; j ++ )
      if ( sts[i].sum < sts[j].sum ) swap(sts[i], sts[j]) ;
}
void printStudents(const Student *const sts, const int no) {
  for ( int i = 0 ; i < no ; i ++ ) {
    cout << i + 1 << '₩t' ;
    cout << sts[i].name << '₩t' ;
    for ( int subject = 0 ; subject < SUBJECT_NO ; subject ++ )
      cout << sts[i].score[subject] << '₩t' ;
    cout << sts[i].sum << '₩t' << sts[i].average << endl ;
  }
}
```

반환 값

- 함수는 약속된 기능을 수행하고 그 결과를 호출함수에게 반환하는 것이 일반적이다

호출 함수	피 호출 함수

int main() {
  float result = **getAverageUpTo(10)** ;
  cout << result << endl ; // 5.5
}
	float getAverageUpTo(const int max) {
	  int sum = 0 ;
	  for (int i = 1 ; i <= max ; i ++)
	  sum += i ;
	  return static_cast<float>(sum) / max ;
	}

반환 값의 전달: return 문

- 피호출 함수에서는 return문을 이용해서 호출 함수에게 자신의 수행 결과를 반환

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

int find(const int* const data, const int size, const int target) {
   for ( int i = 0 ; i < size ; i ++ )
     if ( data[i] == target ) return i ; // 발견된 경우
   return -1 ; // 발견되지 않은 경우
}
int main() {
   const int SIZE = 6 ;
   int scores[SIZE] = {10, 20, 30, 40, 50, 60} ;
   cout << find(scores, SIZE, 50) << endl ; // 4
   cout << find(scores, SIZE, 55) << endl ; // -1
}
```

반환 값의 사용 여부

- 피호출함수가 반환한 값을 호출 함수에서는 활용
- 피호출함수의 반환 값을 호출 함수에서 사용하지 않는 것도 가능

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

int readPositiveNumbers(vector<int>& numbers) {
  while ( true ) {
    int n ;
    cin >> n ;
    if ( n <= 0 ) break ;
    numbers.push_back(n) ;
  }
  return numbers.size() ;
}
```

```
int printLargeNumbers(const vector<int>& numbers, const int base) {
  typedef vector<int>::const_iterator iterator ;
  int count = 0 ;
  for ( iterator it = numbers.begin() ; it != numbers.end() ; ++ it )
   if ( *it > base ) {
     cout << *it << endl ;
     count ++ ;
   }
  return count ;
}
int main() {
  vector<int> positiveNumbers ;
  // 피호출 함수가 반환한 값을 호출함수에서 사용하고 있지 않음
  readPositiveNumbers(positiveNumbers) ;
  printLargeNumbers(positiveNumbers, 50) ;
}
```

Good Design: 오류를 뜻하는 반환값은 호출함수에서 처리

- 반환한 값이 정상적인 계산/처리 결과가 아니라 비정상적인 처리 즉 오류를 뜻하는 경우에는 호출함수에서 이를 처리

```
int find(const int* const data, const int size, const int target) {
  for ( int i = 0 ; i < size ; i ++ )
    if ( data[i] == target ) return i ; // 발견된 경우
  return -1 ; // 발견되지 않은 경우
}
```

```
char* values = "Hello, C++!" ;
char target = '+' ;
int result = find(values, target) ;
if ( result == -1 ) // -1 즉 오류 상황 발생에 대한 처리를 해야 함
  cout << target << " Not Found in << values << endl ;
```

반환 값 타입: 기본 타입

- 자신의 기능에 따라서 int, float, char, bool 등의 기본 타입의 반환 값을 호출 함수에게 전달

```
bool isEqualPoint(const Point& pt1, const Point& pt2) ;
int getSum(const int max) ;
float getAverage(const int max) ;
int readPositiveNumbers(vector<int>& numbers) ;
int printLargeNumbers(const vector<int>& numbers, const int base) ;
```

- void 타입은 아무 값도 반환하지 않음

```
void upgrade(Grade& grade) ;
void printStudent(const Student& st) ;
```

반환 값 타입: 나열형

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

enum Grade { FRESH=1, SOPHOMORE, JUNIOR, SENIOR } ;
Grade upgrade(const Grade now) {
  Grade next = ( now == SENIOR ) ? SENIOR : static_cast<Grade>(now+1) ;
  return next ;
}
int main() {
  Grade current = FRESH ;
  Grade next = upgrade(current) ;
  cout << current << '₩t' << next << endl ; // 1 2
}
```

반환 값 타입: 구조체

- 구조체를 함수에서 반환하는 것도 가능

```
struct Student {
    string name ;
    Grade grade ;
    float gpa ;
} ;
Student readStudent() ;
Student findStudentByName(const vector<Student>& sts, const string& name) ;
```

반환 값 타입: 구조체

```
Student readStudent() { // 구조체 Student 반환
  string name ; int grade ; float gpa ;
  cin >> name >> grade >> gpa ;
  Student st ;
  st.name = name ;
  st.grade = static_cast<Grade>(grade) ;
  st.gpa = gpa ;
  return st ;
}
Student findStudentByName(const vector<Student>& sts, const string& name) {
  for ( unsigned int i = 0 ; i < sts.size() ; i ++ )
    if ( sts[i].name == name ) return sts[i] ;
  return Student() ;
}
int main() {
  cout << "Enter the number of students !" << endl ;
  int no ; cin >> no ;
  vector<Student> students(no) ; // 구조체 Student의 vector 생성
  for ( int i = 0 ; i < no ; i ++ )
    students[i] = readStudent() ; // 반환된 구조체 Student의 대입
  // 반환된 구조체 Student를 이용한 stJames의 초기화
  Student stJames = findStudentByName(students, "James") ;
}
```

반환 값 타입: 구조체

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

enum Grade { FRESH=1, SOPHOMORE, JUNIOR, SENIOR } ;
Grade upgrade(const Grade now) {
  Grade next ;
  next = ( now == SENIOR ) ? SENIOR : static_cast<Grade>(now+1) ;
  return next ;
}
string getGradeLabel(const Grade grade) {
  string gradeLabels[] = { "Fresh", "Sophomore", "Junior", "Senior" } ;
  return gradeLabels[grade-1] ;
}
int main() {
  Grade current = SOPHOMORE ;
  Grade next = upgrade(current) ;
  cout << getGradeLabel(current) << endl ;
  cout << getGradeLabel(next) << endl ;
}
```

반환 값 전달: 값에 의한 전달

- 기본적으로 피호출 함수에서 생성한 반환 값은 호출 함수로 복사된다. 즉 값에 의한 전달이 발생한다

```
int main() {
    int sum = getSum(10) ;
    float average = getAverage(10) ;
}
```

```
int getSum(const int max) {
    int result = 0 ;
    ...
    return result ;
}
float getAverage(const int max) {
    int result = 0 ;
    ...
    return static_cast<float>result / max ;
}
```

반환 값 전달: 값에 의한 전달

- 값으로 복사되는 방식은 반환 타입이 구조체/클래스와 같이 많은 메모리를 사용할 때 문제를 유발

```
Student readStudent() ;

Student findStudentByName(const vector<Student>& sts, const string& name) ;
```

```
string getGradeLabel(const Grade grade) ;
```

반환 값 전달: 참조에 의한 전달

```
Student& readStudent() ;
Student& findStudentByName(const vector<Student>& sts, const string& name) ;
string& getGradeLabel(const Grade grade) ;
```

- 상황에 따라서 위험한 상황을 초래

참조에 의한 반환의 문제: 예 1

```
struct Student {
  string name ;
  Grade grade ;
  float gpa ;
} ;
Student& readStudent() {
  string name ; int grade ; float gpa ;
  cin >> name >> grade >> gpa ;
  Student st ;
  st.name = name ;
  st.grade = static_cast<Grade>(grade) ;
  st.gpa = gpa ;
  return st ; // 지역변수 st에 대한 참조(주소)가 반환되므로 위험함
}
const Student& findStudentByName(
  const vector<Student>& sts, const string& name) {
  for ( unsigned int i = 0 ; i < sts.size() ; i ++ )
    if ( sts[i].name == name ) return sts[i] ; // 안전함
  return Student() ; // 위험함
}
int main() {
  Student newSt = readStudent() ;
  vector<Student> students ;
  const Student& stJames = findStudentByName(students, "James") ;
}
```

참조에 의한 반환의 문제: 예 2

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

enum Grade { FRESH=1, SOPHOMORE, JUNIOR, SENIOR } ;

Grade upgrade(const Grade now) {
  Grade next ;
  next = ( now == SENIOR ) ? SENIOR : static_cast<Grade>(now+1) ;
  return next ;
}
string& getGradeLabel(const Grade grade) {
  string gradeLabels[] = { "Fresh", "Sophomore", "Junior", "Senior" } ;
  return gradeLabels[grade-1] ; // 지역 변수에 대한 참조를 반환하므로 위험함
}
int main() {
  Grade current = FRESH ;
  Grade next = upgrade(current) ;
  cout << getGradeLabel(current) << endl ;
  cout << getGradeLabel(next) << endl ;
}
```

Good Design – 지역변수에 대한 참조/포인터 반환은 위험

지역변수에 대한 참조 반환	지역변수에 대한 주소(포인터) 반환

int& f() {
  int local ;
  …
  return local ;
}
	int* g() {
	  int local ;
	  …
	  return &local ;
	}

안전한 참조 및 포인터 반환

- 호출 함수에서도 유효한 메모리에 대한 참조 또는 포인터를 전달하는 것은 안전
  - 전역 변수에 대한 참조/포인터 반환
  - 참조 매개변수에 대한 참조/포인터 반환
  - 할당된 동적 메모리에 대한 포인터 반환
  - static 지역 변수에 대한 참조/포인터 반환

전역 변수에 대한 참조/포인터 반환

전역변수에 대한 참조 반환	전역변수에 대한 포인터 반환

int gCount ;
int& f() {
  gCount ++ ;
  return gCount ;
}
int main() {
  int x = f() ;
  cout << x ; // 1
}
	int gCount ;
	int* f() {
	  gCount ++ ;
	  return &gCount ;
	}
	int main() {
	  int* x = f() ;
	  cout << *x ; // 1
	}

참조 매개변수에 대한 참조/포인터 반환

- 전달받은 참조 매개변수에 대한 참조/포인터를 반환하는 것은 안전

참조 매개변수에 대한 참조 반환	참조 매개변수에 대한 포인터 반환

int& f(int**&** n) {
  n++ ;
  return n;
}
int main() {
  int x = 10 ;
  cout << f(x) ; // 11
}
	int* f(int& n) {
	  n++ ;
	  return &n ;
	}
	int main() {
	  int x = 10 ;
	  cout << *f(x) ; // 11
	}

할당된 동적 메모리에 대한 포인터 반환

- 피호출함수에서 new를 이용하여 동적으로 메모리를 할당하고 이에 대한 포인터를 호출함수에게 전달하는 것은 안전

```
int* f(int n) {
   int* const p = new int[n] ;
   for ( int i = 0 ; i < n ; i ++ )
      p[i] = i+10 ;
   return p ;
}
int main() {
   const int* const pVal = f(10) ;
   cout << pVal[0] ; // 10
   delete [] pVal ;
}
```

정적 지역 변수에 대한 참조/포인터 반환

- 정적 지역 변수에 대한 참조 및 포인터를 반환하고 이를 호출 함수에서 정적 지역 변수의 값을 접근하는 것은 안전

static 지역 변수에 대한 참조 반환	static 지역 변수에 대한 포인터 반환

int& f() {
  static int n = 0 ;
  n ++ ;
  return n;
}
int main() {
  cout << f() ; // 1
  cout << f() ; // 2
}
	int* f() {
	  static int n = 0 ;
	  n ++ ;
	  return &n ;
	}
	int main() {
	  cout << *f() ; // 1
	  cout << *f() ; // 2
	}

안전한 참조 및 포인터 반환 1

```
Student* readStudent() {        // 할당된 동적 메모리에 대한 포인터 반환
  string name ; int grade ; float gpa ;
  cin >> name >> grade >> gpa ;
  Student* const pSt = new Student ;
  pSt->name = name ;
  pSt->grade = static_cast<Grade>(grade) ;
  pSt->gpa = gpa ;
  return pSt ;
}
// 참조 매개변수에 대한 포인터 반환
const Student* findStudentByName(const vector<Student*>& sts, const string& name) {
  for ( unsigned int i = 0 ; i < sts.size() ; i ++ )
    if ( sts[i]->name == name ) return sts[i] ;
  return 0 ;
}
int main() {
  cout << "Enter the number of students !" << endl ;
  int no ; cin >> no ;
  vector<Student*> students(no) ;
  for ( int i = 0 ; i < no ; i ++ ) students[i] = readStudent() ;
  const Student* stJames = findStudentByName(students, "James") ;
  for ( unsigned int i = 0 ; i < students.size() ; i ++ ) delete students[i] ;
}
```

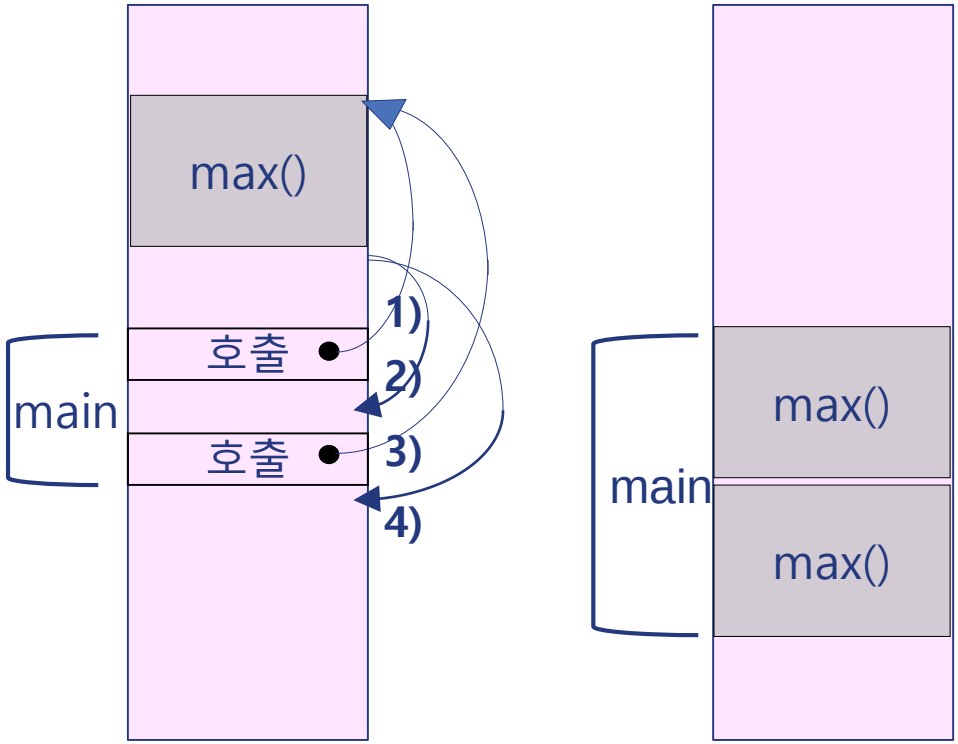
안전한 참조 및 포인터 반환 2

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

enum Grade { FRESH=1, SOPHOMORE, JUNIOR, SENIOR } ;
Grade upgrade(const Grade now) {
  Grade next ;
  next = ( now == SENIOR ) ? SENIOR : static_cast<Grade>(now+1) ;
  return next ;
}
// static 지역 변수에 대한 참조 반환
string& getGradeLabel(const Grade grade) {
  static string gradeLabels[] = { "Fresh", "Sophomore", "Junior", "Senior" } ;
  return gradeLabels[grade-1] ;
}
int main() {
  Grade current = FRESH ;
  Grade next = upgrade(current) ;
  cout << getGradeLabel(current) << endl ;
  cout << getGradeLabel(next) << endl ;
}
```

inline function

- The call to Inline function is replaced with the actual code instead of function call.

```
inline int max(int i, int j)
{
  return ( i > j ) ? i : j ;
}

int main() {
  max(10, 20) ;
  max(30, 20) ;
}
```



inline function

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

void readPositiveNumbers(vector<int>& numbers) ;
void sortNumbers(vector<int>& numbers) ;
inline void swap(int& n1, int& n2) ;
void printNumbers(const vector<int>& numbers) ;

int main() {
   vector<int> numbers ;

   readPositiveNumbers(numbers) ;
   sortNumbers(numbers) ;
   printNumbers(numbers) ;

}
```

```
void readPositiveNumbers(vector<int>& numbers) {
   cout << "Enter positive integers. A negative integer will stop." << endl ;
   while ( true ) {
      int number ;
      cin >> number ;
      if ( number <= 0 ) break ;
      numbers.push_back(number) ;
   }
}
void sortNumbers(vector<int>& numbers) {
   int size = numbers.size() ;
   for ( int i = 0 ; i < size - 1 ; i ++ )
      for ( int j = i + 1 ; j < size ; j ++ )
          if ( numbers[i] < numbers[j] ) swap(numbers[i], numbers[j]) ;
}

inline void swap(int& n1, int& n2) {
   int temp = n1 ;
   n1 = n2 ;
   n2 = temp ;
}
void printNumbers(const vector<int>& numbers) {
   typedef vector<int>::const_iterator it ;
   for ( it p = numbers.begin() ; p != numbers.end() ; ++p )
      cout << *p << endl ;
}
```

Header 파일을 이용한 inline함수 정의

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

#include "MyHeaderForInline.h"

int main() {
   vector<int> numbers ;
   readPositiveNumbers(numbers) ;
   sortNumbers(numbers) ;
   printNumbers(numbers) ;
}
void readPositiveNumbers(vector<int>& numbers) {
   …
}
void sortNumbers(vector<int>& numbers) {
   …
}
void printNumbers(const vector<int>& numbers) {
   …
}
```

```
// MyHeaderForInline.h

#ifndef __MY_HEADER_FOR_INLINE_H
#define __MY_HEADER_FOR_INLINE_H

#include <vector>

void readPositiveNumbers(vector<int>& numbers) ;
void sortNumbers(vector<int>& numbers) ;
inline void swap(int& n1, int& n2) ;
void printNumbers(const vector<int>& numbers) ;

inline void swap(int& n1, int& n2) {
   int temp = n1 ;
   n1 = n2 ;
   n2 = temp ;
}

#endif
```

매크로와 인라인 함수

인라인 함수	매크로

inline int safeABS(int i) {
  return i >= 0 ? i : -i;
}
	#define unsafeABS(i) ((i) >= 0 ? (i) : -(i))

매크로와 인라인 함수

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

#define unsafeABS(i)  ( (i) >= 0 ? (i) : -(i) )
inline int safeABS(int i) { return i >= 0 ? i : -i; }
int f() {
  static int i = 0 ;
  i-- ;
  return i ;
}
```

인라인 함수 이용시	매크로 이용시

void main() {
  int x = -10 ;
 	
  cout << safeABS(x++) << endl ; // 10
  cout << safeABS(f()) << endl ; // 1
}
	void main() {
	  int x = -10 ;
	
	  cout << unsafeABS(x++) << endl ; // 9
	  cout << unsafeABS(f()) << endl ; // 2
	}

정적 지역 변수

- 값이 함수 호출 간에 유지

```
# include <iostream>
using namespace std ;
 int incorrectAdd(const int val) {
  int sum = 0 ; // 일반 지역 변수
  sum += val ;
  return sum ;
}
int correctAdd(const int val) {
  static int sum = 0 ; // 정적 지역 변수
  sum += val ;
  return sum ;
}
int main() {
  cout << incorrectAdd(3) << endl ;
  cout << incorrectAdd(5) << endl ;
  cout << correctAdd(3) << endl ;
  cout << correctAdd(5) << endl ;
}
```

정적 지역 변수 예

```
# include <iostream>
using namespace std ;
int incrementCounter() {
   static int counter = 0 ;
   counter ++ ;
   return counter ;
}
int sumUpTo(const int upTo) {
   int sum = 0 ;
   for ( int i = 1 ; i <= upTo ; i ++ ) sum += i ;
   return sum ;
}
int main() {
   while ( true ) {
      int number ;
      cin >> number ;
      if ( number <= 0 ) break ;
      int counter = incrementCounter() ;
      cout << counter << " integer: " << number << endl ;
      int sum = sumUpTo(number) ;
      cout << "Sum of 1 to " << number << " : " << sum << endl ;
   }
}
```

정적 지역 변수 예

```
enum Grade { FRESH=1, SOPHOMORE, JUNIOR, SENIOR } ;
string getGradeLabel_1(const Grade grade) { // 방법 1: 낮은 성능 유발
  string gradeLabels[] = { "Fresh", "Sophomore", "Junior", "Senior" } ;
  return gradeLabels[grade-1] ;
}
string& getGradeLabel_2(const Grade grade) { // 방법 2: 안전하지 않음
  string gradeLabels[] = { "Fresh", "Sophomore", "Junior", "Senior" } ;
  return gradeLabels[grade-1] ;
}
string& getGradeLabel_3(const Grade grade) { // 방법 3: 가장 바람직함
  static string gradeLabels[] = { "Fresh", "Sophomore", "Junior", "Senior" } ;
  return gradeLabels[grade-1] ;
}
int main() {
  cout << getGradeLabel_1(FRESH) << endl ;
  cout << getGradeLabel_2(FRESH) << endl ;
  cout << getGradeLabel_3(FRESH) << endl ;
}
```

재귀적 함수

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

long factorial(const int number) ;

int main() {
  int number ;
  cin >> number ;

  long result = factorial(number) ;
  cout << "Factorial of " << number << " : " << result << endl ;
}

// 재귀적함수; stack overflow의 위험이 있으므로 반복문이 좋을수가 있음
long factorial(const int number) {
  if ( number == 1 ) return 1 ;

  return number * factorial(number-1) ;
}
```

재귀적 함수

```
int binarySearch(const vector<int>& numbers,
    int min, int max, int target) {
  int mid = (min + max) / 2 ;
  if ( numbers[mid] == target ) return mid ;
  if ( min >= max ) return -1 ;
  if ( target < numbers[mid] )
    binarySearch(numbers, min, mid-1, target) ;
  else
    binarySearch(numbers, mid+1, max, target) ;
}

int main() {
  int data[] = {1, 3, 5, 8, 20, 30} ;
  vector<int> numbers(data, data+sizeof(data)/sizeof(int)) ;
  cout << binarySearch(numbers, 0, numbers.size()-1, 50) ;
}
```

재귀적 함수의 위험성

- 재귀적 함수는 스택 공간의 부족(stack overflow)을 유발
- 대신에 반복문을 사용

```
long factorial(const int number) {
  long result = 1 ;
  for ( int i = 2 ; i ++ ; i <= number )
    result *= i ;
  return result ;
}
```

```
int binarySearch(const vector<int>& numbers, int min, int max, int target) {
  while ( max > min ) {
    int mid = (min + max) / 2 ;
    if ( numbers[mid] == target ) return mid ;
    if ( min >= max ) return -1 ;
    if ( target < numbers[mid] ) max = mid -1 ;
    else min = mid+1 ;
  }
  return -1 ;
}
```

함수의 선행 조건

```
# include <cassert>

long factorial(const int number) {
   assert (number >= 0 ) ;

   // 1만 비교하기 때문에 factorial( -10)을 호출하면 무한반복이 될수있다
   if ( number == 1 ) return 1 ;

   return number * factorial(number-1) ;
}

int findChar(const char *const str, const char ch) {
   // 선행조건; 기본적으로 포인터는 반드시 null이 아님이 확인되어야함
   assert ( str != 0 ) ;

   for ( int i = 0 ; i < strlen(str) ; i ++ )
      if ( str[i] == ch ) return i ;

   return -1 ;
}
```

함수의 후행 조건

```
# include <cassert>

void sortNumbers(vector<int>& numbers) {
   int size = numbers.size() ;
   for ( int i = 0 ; i < size - 1 ; i ++ )
      for ( int j = i + 1 ; j < size ; j ++ )
         if ( numbers[i] < numbers[j] ) swap(numbers[i], numbers[j]) ;

   // 후행조건 (이것도 2개 이상의 원소를 가정한 것임)
   for ( int i = 0 ; i <= numbers.size() - 2 ; i ++ )
      assert ( numbers[i] >= numbers[i+1] ) ;
}
```

파일 scope과 program scope

```
// static예제_1.cpp

# include <iostream>
using namespace std ;

extern void globalBFunction() ;

int globalCounter ;
static int staticCounter = 10 ;
static void staticFunction() {
  staticCounter ++ ;
}
void globalAFunction() {
  globalCounter ++ ;
  staticFunction() ;
  cout << globalCounter << '\t'
       << staticCounter << endl ;
}
int main() {
  globalAFunction() ;
  globalBFunction() ;
}
```

```
// static예제_2.cpp

# include <iostream>
using namespace std ;

extern int globalCounter ;
extern void globalAFunction() ;

static int staticCounter = 20 ;
static void staticFunction() {
  staticCounter ++ ;
}

void globalBFunction() {
  globalCounter ++ ;
  staticFunction() ;

  cout << globalCounter << '\t'
       << staticCounter << endl ;
}
```