

# C++ 프로그래밍



# 출석수업

류욱재

skyroom7@knou.ac.kr

# C++ 프로그래밍 출석수업

## 1 C++ 프로그래밍 첫걸음

2 변수와 배열

3 포인터와 참조

4 함수

## (1) C++ 소스 프로그램

### ■ FirstStep.cpp

```
1  #include <iostream>  선행처리기 지시어
2
3  int main()
4  {                      주석(comment)
5      // 표준 출력 스트림으로 문장을 출력함
6      std::cout << "나의 첫 번째 C++ 프로그램" << std::endl;
7      return 0;
8  }
```

## (1) C++ 소스 프로그램

### ■ 선행처리

- C++ 프로그램을 컴파일하기 전에 소스 프로그램을 가공하여 컴파일러가 실제로 번역할 소스 프로그램을 만드는 것
- 선행처리기 지시어(preprocessor directives)로 처리를 지시함
  - ◆ 선행처리기 지시어는 '#'로 시작함
  - ◆ 선행처리기 지시어 문장은 한 행에 한 개의 문장을 작성함
- 대표적인 선행처리
  - ◆ 헤더파일 삽입 : `#include`
  - ◆ 매크로 선언 및 해제 : `#define`, `#undef`
  - ◆ 조건부 컴파일 : `#if`, `#ifdef`, `#ifndef`

## (1) C++ 소스 프로그램

### ■ 선행처리의 예 - 헤더파일 삽입

a.cpp

```
#include "a.h"  
문장_1;  
문장_2;  
문장_3;  
.....
```

a.h

```
문장_h1;  
문장_h2;  
문장_h3;  
.....
```

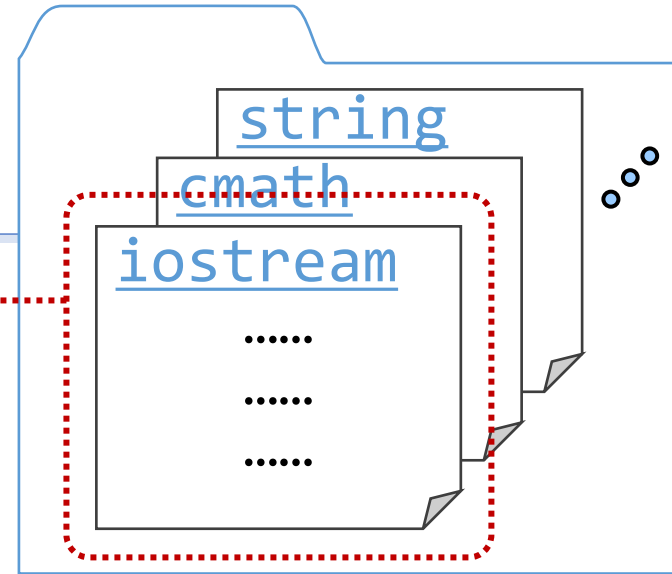
### 선행처리 결과

```
문장_h1;  
문장_h2;  
문장_h3;  
문장_1;  
문장_2;  
문장_3;  
.....
```

## (1) C++ 소스 프로그램

### ■ FirstStep.cpp

```
1  #include <iostream>
2
3  int main()
4  {
5      // 표준 출력 스트림으로 문장을 출력함
6      std::cout << "나의 첫 번째 C++ 프로그램" << std::endl;
7      return 0;
8  }
```



# (1) C++ 소스 프로그램

## ■ FirstStep.cpp

```
1  #include <iostream>
2
3  int main()
4  {
5      // 표준 출력 스트림으로 문장을 출력함
6      std::cout << "나의 첫 번째 C++ 프로그램" << std::endl;
7      return 0;
8  }
```

함수

머리부

몸체블록

## (2) 입출력 스트림

### ■ std::cout 객체

- 표준 출력 스트림 객체
- 데이터를 문자열로 변환하여 출력함
- 출력 연산자(삽입 연산자) : <<

```
std::cout << "나의 첫 번째 C++ 프로그램" << std::endl;  
int a = 10;  
std::cout << "a의 값은 ";  
std::cout << a << "입니다." << std::endl;
```



나의 첫 번째 C++ 프로그램  
a의 값은 10입니다.



## (2) 입출력 스트림

### ■ std::cin 객체

- 표준 입력 스트림 객체
- 문자열을 입력 변수의 자료형의 값으로 변환하여 입력함
- 입력 연산자(추출 연산자) : >>
- 📝 백색 공백 문자(빈칸, 탭, 새줄 문자)는 구분자로 취급함

```
int a;  
char str[100];  
std::cin >> a >> str;  
cout << a << endl;  
cout << str << endl;
```



10

KNOU Computer Sci.



출력

```
10  
KNOU
```

### (3) 명칭공간

#### ■ 명칭공간이란?

- 특정한 이름들이 인식되는 프로그램의 부분



명칭공간의 정의

명칭공간 이름

```
namespace myNSpc {  
    int count; // 명칭을 선언하는 문장 나열  
}
```



명칭공간에 정의된 명칭의 사용

```
myNSpc::count = 0;
```

- std 명칭공간 : 표준 C++ 라이브러리의 명칭들이 정의되어 있는 명칭공간

### (3) 명칭공간

#### ■ 명칭공간 사용 예

```
1  #include <iostream>
2
3  namespace NameSpace1 { int a = 10; }
4  namespace NameSpace2 { int a = 20; }
5  int a = 30;
6  namespace NameSpace1 { int b = 50; }
7  int main()
8  {
9      int a = 40;
10     std::cout << NameSpace1::a << std::endl;
11     std::cout << NameSpace2::a << std::endl;
12     std::cout << ::a << std::endl;
13     std::cout << a << std::endl;
14     std::cout << NameSpace1::b << std::endl;
15     return 0;
16 }
```

### (3) 명칭공간

#### ■ 'using'을 이용한 명칭공간 사용

- 특정 명칭공간이나 명칭공간 내의 특정 이름을 자주 사용하는 경우 명칭공간 지정을 간소화 할 수 있음

예 **using namespace** std;

예 **using** std::cout;  
**using** std::endl;

### (3) 명칭공간

#### ■ 명칭공간 사용 예

```
1  #include <iostream>
2  using namespace std;
3  namespace NameSpace1 { int a = 10; }
4  namespace NameSpace2 { int a = 20; }
5  int a = 30;
6  namespace NameSpace1 { int b = 50; }
7  int main()
8  {
9      int a = 40;
10     cout << NameSpace1::a << endl;
11     cout << NameSpace2::a << endl;
12     cout << ::a << endl;
13     cout << a << endl;
14     cout << NameSpace1::b << endl;
15     return 0;
16 }
```

### (3) 명칭공간

#### ■ 명칭공간 사용 예

- FirstStep.cpp

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      // 표준 출력 스트림으로 문장을 출력함
6      cout << "나의 첫 번째 C++ 프로그램" << endl;
7      return 0;
8  }
```

# C++ 프로그래밍 출석수업

- 1 C++ 프로그래밍 첫걸음
- 2 변수와 배열
- 3 포인터와 참조
- 4 함수

## (1) 변수

### ■ 변수란?

- 프로그램이 실행되는 동안 기억하고 있어야 하는 값들을 저장하는 메모리 영역
- 변수에는 이름이 지정되어야 함
- 모든 변수는 사용하기 전에 미리 선언해야 함

### ■ 변수 선언 위치

- 함수 내부    ➡    지역변수
- 함수 외부    ➡    전역변수



# (1) 변수

## ■ 변수 선언 형식



형식1. int     size;



변수의 이름

자료형 이름(int, float, double 등)



형식2. int     size, price, stock;



동일 자료형의 여러 개의 변수 선언


자료형 이름(int, float, double 등)

# (1) 변수


## ■ 변수의 초기화

 형식1. `int total = 0;`

 형식2. `int total(0);`

 `int x(1.5);`      `// 1로 초기화`  
`float y = x;`      `// 1로 초기화`

 형식3. `int total = {0};`

 `short x{total};`      `// 오류: 축소 변환`  
`float y{total};`      `// 오류: 축소 변환`

## (1) 변수

### ■ 자료형 추론

- 변수를 초기화할 때 초기화하는 값의 자료형으로 변수의 자료형을 추론함

예 `auto i(10);`    `// int i(10);` 과 동일함

### ■ const 한정어

- 변수의 값을 수정할 수 없게 함
- 초기화를 통해서만 값을 정할 수 있음

예 `const double PI {3.14159};`    `// 원주율 정의`

## (2) 배열

### ■ 배열이란?

- 동일한 자료형의 값을 여러 개 저장할 수 있는 연속적으로 할당된 공간을 묶어 하나의 이름을 갖는 변수로 만든 것
- 각각의 원소는 0번부터 시작하여 차례로 부여된 번호(첨자, 인덱스)를 이용하여 액세스 함
- 배열의 차원 : 배열의 첨자 개수

## (2) 배열

### ■ 1차원 배열의 선언

```
float fArray[4];
```

➔ 

fArray[0]	fArray[1]	fArray[2]	fArray[3]
-----------	-----------	-----------	-----------

### ■ 1차원 배열의 사용

```
float fArray[4];  
int i=0;  
fArray[i] = 10.0f;  
cin >> fArray[1] >> fArray[2] >> fArray[3];  
cout << fArray[1] * fArray[2];
```

## (2) 배열

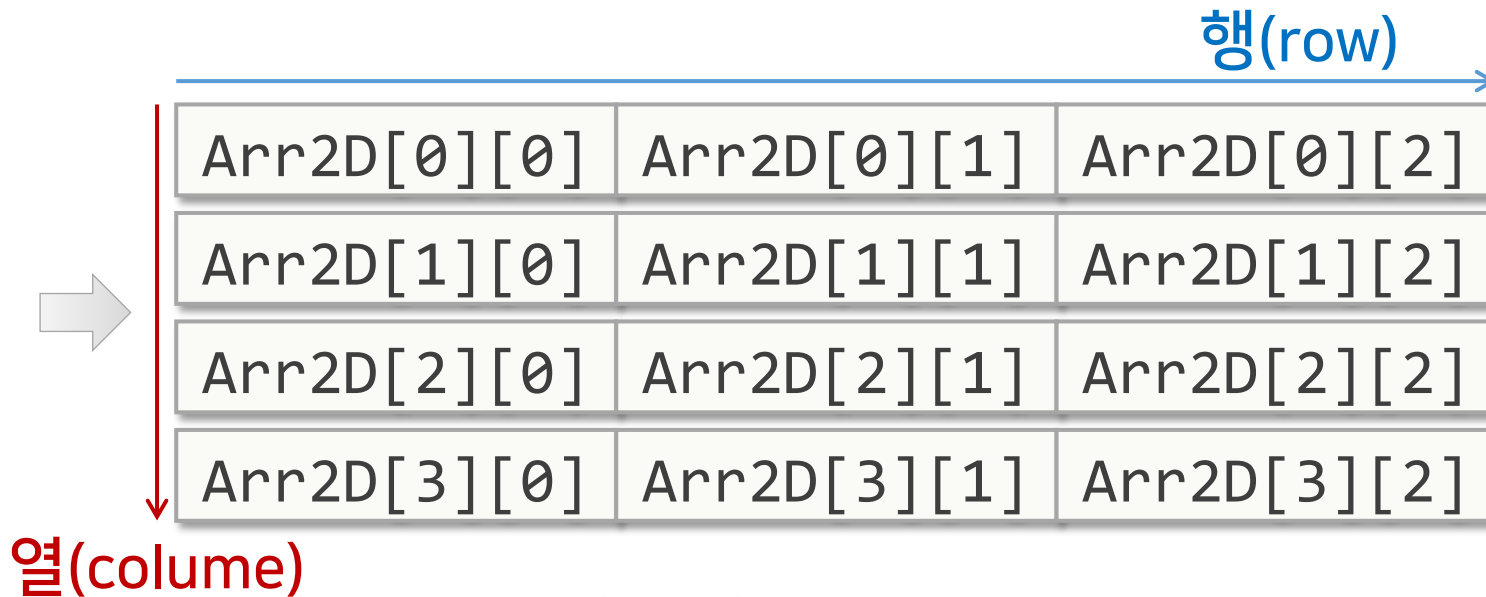
### ■ n차원 배열

- n-1차원 배열이 원소인 배열



2차원 배열

```
int Arr2D[4][3];
```



행 우선 순서(row-major order)

## (2) 배열

### ■ 배열의 초기화

```
int a[5] = { 1, 2, 3, 4, 5 }; // '='는 생략할 수 있음
```

```
int b[5] = { 1, 2, 3 }; // b[3]와 b[4]는 0으로 초기화됨
```

```
int c[] = { 1, 2, 3, 4, 5 }; // 배열의 크기는 5임
```

```
int d[2][4] = {{ 1, 2, 3, 4 }, // 2차원 배열의  
               { 5, 6, 7, 8 }}; // 초기화
```

## (2) 배열

### ■ 여러 개의 데이터 중 최댓값 구하기

#### 일반 변수를 사용한 프로그램

```
max = data0;  
if (max < data1) max = data1;  
if (max < data2) max = data2;  
if (max < data3) max = data3;  
if (max < data4) max = data4;  
if (max < data5) max = data5;  
if (max < data6) max = data6;  
if (max < data7) max = data7;  
if (max < data8) max = data8;  
if (max < data9) max = data9;
```

⇒ 배열을 이용하면 효율적인  
프로그래밍을 할 수 있음



## (2) 배열

## ■ 여러 개의 데이터 중 최대값 구하기 - MaxValue.cpp

```

1  #include      <iostream>
2  using namespace std;
3  int main()
4  {
5      int data[10] = { 10, 23, 5, 9, 22, 48, 12, 10, 55, 31 };
6      int (ㄱ); // data의 0번 값을 max로 가정함
7
8      cout << "데이터 : " << (ㄴ); // 0번 데이터 출력
9      for ( (ㄷ) ) { // 나머지 9개의 데이터 비교
10         cout << " " << (ㄹ); // i번 데이터 출력
11         if (max < (ㄺ)) // i번 데이터가 max보다 크면
12             max = (ㄻ); // i번 데이터 비교
13     }
14     cout << endl << endl;
15     cout << "배열의 최대값 : " << max << endl;
16     return 0;
17 }

```

# C++ 프로그래밍 출석수업

- 1 C++ 프로그래밍 첫걸음
- 2 변수와 배열
- 3 포인터와 참조**
- 4 함수

## (1) 포인터

### ■ 포인터란?

- 다른 변수, 구조체, 객체 등 값이 저장된 곳을 가리키는 변수

### ■ 포인터 선언 형식

```
TypeName *ptrVar;
```

- ◆ *TypeName* : 가리킬 값의 자료형
- ◆ *ptrVar* : 포인터 변수의 이름

예 `int *iPtr;`

## (1) 포인터

### ■ 포인터의 사용

- 포인터가 유효한 대상을 가리키게 한 후 사용해야 함

```
ptrVar = &var;    // ptrVar이 var를 가리키게 함  
*ptrVar = value;  // ptrVar을 이용하여 var를 액세스
```

- ◆ **&** : 주소 계산 연산자
- ◆ **\*ptrVar** : 포인터 ptrVar가 가리키는 곳

**예**

```
int a;  
int *iPtr = &a;  
*iPtr = 10;
```

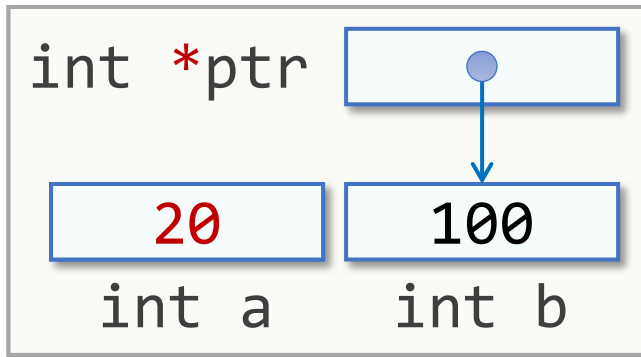
# (1) 포인터

## ■ 포인터의 사용 예

```

1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int  a = 10, b = 100;
7      int  ( ) ; // 포인터 ptr을 선언한 후에 a의 주소를 넣는다.
8
9      cout << "ptr이 가리키는 곳의 값 : " << (L) << endl;
10     (C) = 20; // ptr이 가리키는 곳에 20을 저장
11     cout << "변수 a의 값 : " << a << endl;
12     (E) ; // ptr이 b를 가리키게 함
13     cout << "변수 b의 값 : " << *ptr << endl;
14     return 0;
15 }

```



## (2) 동적 메모리 할당

### ■ 동적 메모리 할당이란?

- 프로그램 동작 중에 기억공간의 필요성 및 소요량을 결정하여 필요한 공간을 할당하는 것
  - ◆ 기억 공간의 생성 시점 : **new** 연산자의 실행 시점
  - ◆ 기억 공간의 소멸 시점 : **delete** 연산자의 실행 시점
- 포인터 변수가 할당된 기억 공간을 가리키게 함

## (2) 동적 메모리 할당

### ■ 메모리 할당 연산자

- 1 `ptrVar = new TypeName;`
- 2 `ptrVar = new TypeName[n];`

### ■ 메모리 반납 연산자

- 1 `delete ptrVar;`
- 2 `delete [] ptrVar;`

## (2) 동적 메모리 할당

### ■ 단일 데이터 공간의 할당 및 반환

```
int *intPtr;  
intPtr = new int;  
*intPtr = 10;
```

int \*intPtr



자유 메모리 공간

10





## (2) 동적 메모리 할당

### ■ 단일 데이터 공간의 할당 및 반환

```
int *intPtr;  
intPtr = new int;  
*intPtr = 10;  
. . . . .  
delete intPtr;  
intPtr = nullptr;
```

int \*intPtr  
nullptr

자유 메모리 공간

10



## (2) 동적 메모리 할당

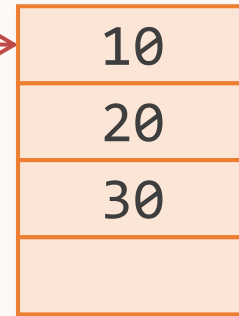
### ■ 배열 데이터 공간의 할당 및 반환

```
int *intPtr;  
intPtr = new int[4];  
*intPtr = 10;  
*(intPtr+1) = 20;  
intPtr[2] = 30;
```

int \*intPtr



자유 메모리 공간



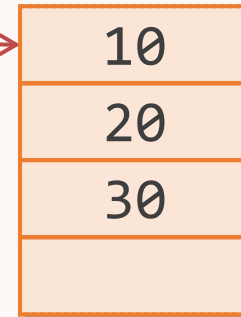
## (2) 동적 메모리 할당

### ■ 배열 데이터 공간의 할당 및 반환

```
int *intPtr;  
intPtr = new int[4];  
*intPtr = 10;  
*(intPtr+1) = 20;  
intPtr[2] = 30;  
.....  
delete [] intPtr;  
intPtr = nullptr;
```

int \*intPtr  
nullptr

자유 메모리 공간



## (2) 동적 메모리 할당

### ■ 배열 데이터 공간의 할당 및 반환

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

int main()
{
    int *intPtr;
    (ㄱ) // intPtr에 4개의 int 값을 저장할 메모리 할당
    *intPtr = 10;
    *(intPtr + 1) = 20;
    intPtr[2] = 30;
    intPtr[3] = 40;
    for (int *p = intPtr, i = 0; i < 4; i++)
        cout << (ㄴ) << " "; // p가 가리키는 곳의 값을 출력한 후 p를 다음으로 이동
    cout << endl;

    (ㄷ) // (ㄱ)에서 할당한 메모리 반환
}
```

## (3) 참조

### ■ 참조(reference)란?

- 어떠한 대상을 가리키는 값(포인터와 유사함)
- 참조 변수는 참조 대상의 별명처럼 사용할 수 있음
- l-value 참조 : 실체가 있는 대상(l-value)에 대한 참조

### ■ l-value 참조 변수의 선언 형식

```
TypeName &refVar = varName;
```

- ◆ *TypeName* : 참조 대상의 자료형
- ◆ *refVar* : 참조 변수의 이름
- ◆ *varName* : 참조 대상

### (3) 참조

#### ■ l-value 참조 변수의 사용 예

```
1 int a = 10, b = 20;  
2 int &aRef = a;  
3 cout << aRef << endl;  
4 aRef = 100;  
5 aRef = b;
```

VS.

```
1 int a = 10, b = 20;  
2 int *aPtr = &a;  
3 cout << *aPtr << endl;  
4 *aPtr = 100;  
5 *aPtr = b;
```

20

a

20

b

a의 참조

aRef

## (3) 참조

### ■ const 참조

- 참조를 이용하여 참조 대상의 값을 바꿀 수 없음

```
int x { 10 };  
const int &xRef = x;  
cout << xRef << endl; // x의 값을 읽어 출력함  
xRef += 10; // 오류: const 참조로 값을 수정할 수 없음
```

## (3) 참조

### ■ 참조 변수가 포인터와 다른 점

- 참조 변수를 이용하여 값을 읽거나 저장할 때 일반 변수를 사용하는 형식과 동일함
- 참조 변수는 초기화를 통해 반드시 어떤 대상을 참조해야 함
  - ◆ 초기화되지 않은 상태로 인해 무엇을 참조하고 있는지 알 수 없는 상황은 발생하지 않음
- 참조 변수는 초기화를 통해 지정된 참조 대상을 바꿀 수 없어 참조의 유효기간 동안 하나의 대상만 참조할 수 있음



## C++ 프로그래밍 출석수업

- 1 C++ 프로그래밍 첫걸음
- 2 변수와 배열
- 3 포인터와 참조
- 4 함수

## (1) 함수의 개념

### ■ 다음 프로그램이 하는 작업은?

```
int main()
{
    double  a[50], b[100];
    .....    // 생략: a와 b에 각각 50개와 100개의 데이터 입력
    double  sum = 0, sqSum = 0;
    for (int i=0; i < 50; i++) {
        sum += a[i];
        sqSum += a[i] * a[i];
    }
    cout << sqSum / 50 - sum * sum / (50 * 50) << endl;
    sum = sqSum = 0;
    for (int i=0; i < 100; i++) {
        sum += b[i];
        sqSum += b[i] * b[i];
    }
    cout << sqSum / 100 - sum * sum / (100 * 100) << endl;
}
```

# (1) 함수의 개념

다음 프로그램이 하는 작업은?

결괏값의  
자료형

`double variance(const double arr[], int len)`

함수의 머리(헤더)

`double sum = 0, sqSum = 0;` → 지역변수 선언문

`for (int i=0 ; i < len ; i++) {`

`sum += arr[i];`

`sqSum += arr[i] * arr[i];`

함수의 몸체 블록

`}`

`return sqSum/n - sum*sum/(len*len);` → 함수를 마치고 호출한 곳으로 복귀 (결괏값을 반환함)

`int main()`

`{`

`double a[50], b[100];`

`..... // 생략: a와 b에 각각 50개와 100개의 데이터 입력`

`cout << variance(a, 50) << endl;`

`cout << variance(b, 100) << endl;`

`}`

함수호출

## (1) 함수의 개념

### ■ 함수란?

- 필요한 작업을 수행하는 프로그램 문장들을 하나의 단위로 모아 놓고 이름을 부여한 것
- 함수에 정의된 처리가 필요한 부분에서 호출하여 사용함
  - ◆ 매개변수를 통하여 함수에서 처리할 데이터(인수)를 전달함
  - ◆ 호출된 함수로 이동하여 몸체 블록을 실행함
  - ◆ 정해진 처리를 한 후 결과값을 반환할 수 있음
  - ◆ 함수의 실행을 완료하면 호출한 곳으로 복귀함
- C++ 프로그램은 함수를 기본 단위로 하여 구성됨

## (2) 함수의 정의

### ■ int형 배열에서 최댓값을 구하는 함수의 정의

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

int getMax(      ( )      ) // 함수 머리부
{
    .....
}

int main()
{
    int data[10] = { 10, 23, 5, 9, 22, 48, 12, 10, 55, 31 };

    cout << "데이터 : ";
    for (int i = 0; i < 10; i++)
        cout << data[i] << " ";
    cout << endl << endl;
    cout << "배열의 최댓값 = " << getMax(data, 10) << endl;
}
```

## (2) 함수의 정의

### ■ int형 배열에서 최댓값을 구하는 함수의 정의

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

int getMax(const int arr[], int len) // 함수 머리부
{
    int max = INT_MIN; // int형의 최솟값을 max로 가정함
    for (int i = 0; i < len; i++)
        if (max < arr[i])
            max = arr[i];
    (L) // 결과의 반환
}

int main()
{
    .....
    cout << "배열의 최댓값 = " << getMax(data, 10) << endl;
}
```


## (2) 함수의 원형

### ■ 호출 위치의 뒤 또는 다른 파일에 정의된 함수의 호출

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

int main()
{
    .....
    cout << "배열의 최댓값 = " << getMax(data, 10) << endl;
}

int getMax(const int arr[], int len) // 함수 머리부
{
    int max = INT_MIN; // int형의 최솟값을 max로 가정함
    for (int i = 0; i < len; i++)
        if (max < arr[i])
            max = arr[i];
    return max; // 결과의 반환
}
```



## (2) 함수의 원형

### ■ 호출 위치의 뒤 또는 다른 파일에 정의된 함수의 호출

```
#include <iostream>
using namespace std;
int getMax(const int arr[], int len);    // 함수의 원형
int main()
{
    .....
    cout << "배열의 최댓값 = " << getMax(data, 10) << endl;
}

int getMax(const int arr[], int len)    // 함수 머리부
{
    int max = INT_MIN;    // int형의 최솟값을 max로 가정함
    for (int i = 0; i < len; i++)
        if (max < arr[i])
            max = arr[i];
    return max;    // 결과의 반환
}
```



## (2) 함수의 원형



### 장점

- 의미 있는 작업 단위로 모듈화
  - ➔ 간결하고 이해하기 쉬운 프로그램을 만들 수 있음
- 반복 사용되는 코드의 중복 방지
- 잘 설계된 함수는 다른 응용에서 재사용할 수 있음



### 단점

- 함수 호출과 복귀 과정에서 처리 시간이 추가됨
  - ➔ 매우 효율적으로 동작해야 하는 함수라면  
`inline` 함수로 선언함

### (3) 인수의 전달

#### ■ 인수

- 함수 호출 문장에서 함수에 전달하는 값
- 매개변수를 통해 인수를 전달함

##### 실 매개변수(actual parameter)

- 함수 호출 문장에서 함수의 형식 매개변수에 전달할 값

##### 형식 매개변수(formal parameter)

- 인수를 전달받기 위해 함수에 선언된 매개변수
- 함수 헤더에 매개변수의 자료형과 이름을 선언함

### (3) 인수의 전달

#### ■ 인수 전달 방법 - 값 호출(call-by-value)

```

.....
int main()
{
    int data[10] = { 10, 23, 5, 9, 22, 48, 12, 10, 55, 31 };
    int size = sizeof(data) / sizeof(int);
    cout << "배열의 최대값 = " << getMax(data, size) << endl;
    return 0;
}

int getMax(const int arr[], int len) // 함수 머리부
{
    int max = INT_MIN; // int형의 최솟값을 max로 가정함
    for (int i = 0; i < len; i++)
        if (max < arr[i]) max = arr[i];
    return max; // 결과의 반환
}

```

### (3) 인수의 전달

#### ■ 인수 전달 방법 - 값 호출(call-by-value)



##### 장점

- ◆ 실 매개변수와 형식 매개변수는 별개의 데이터이므로 불필요한 부작용이 발생하지 않음



##### 단점

- ◆ 실매개변수의 값을 함수에서 변경하도록 함수를 구현하는 것이 필요한 경우가 있음
- ◆ 구조체와 같이 많은 양의 데이터로 구성된 인수를 전달할 경우 데이터의 복사량이 많아짐

### (3) 인수의 전달

#### ■ 인수 전달 방법 - 참조 호출(call-by-reference)

- 실 매개변수의 참조를 형식 매개변수에 전달함
  - ➔ 형식 매개변수는 실 매개변수의 참조 변수
- 함수에서 형식 매개변수의 값을 변경하는 것은 실 매개변수의 값을 변경하는 것과 같음
  - ➔ 함수에서 변경한 형식 매개변수의 값이 실 매개변수에 전달되게 할 필요가 있을 경우 유용함
- 형식 매개변수에 복사되는 데이터의 양은 실 매개변수의 크기와 관계 없이 일정함
  - ➔ 많은 양의 데이터로 구성되는 구조체나 객체를 인수로 전달하는 경우 효율적임

### (3) 인수의 전달

#### ■ 인수 전달 방법 - 참조 호출(call-by-reference)

```
const float PI = 3.14159265;
struct Circle {
    float radius, cx, cy;
};

// 원의 데이터 입력
(ㄱ) inputData( (ㄴ) ) // 함수 머리부
{
    .....
}

int main()
{
    Circle circle = {1, 2, 3};
    inputData(circle);
    .....
}
```

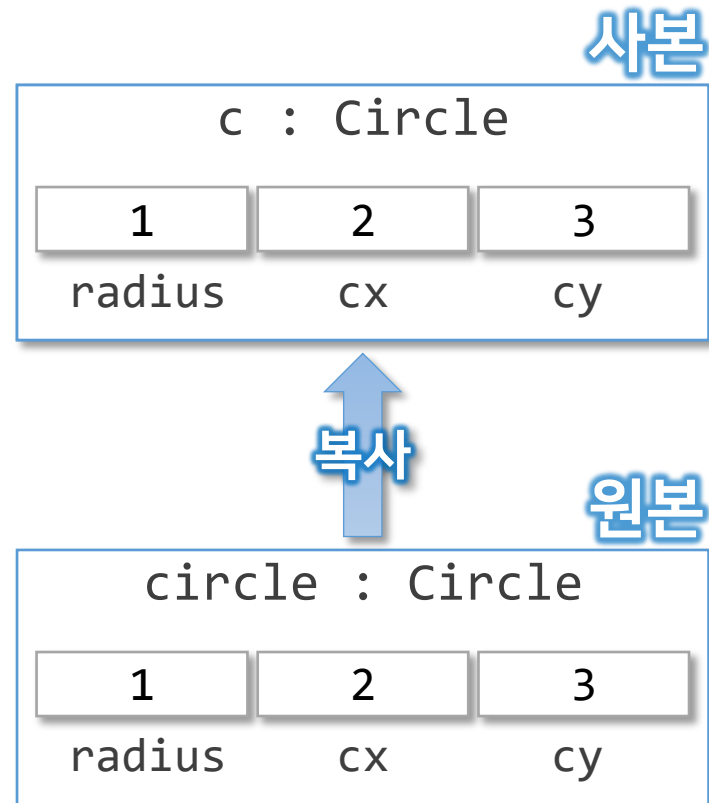
### (3) 인수의 전달

#### ■ 인수 전달 방법 - 참조 호출(call-by-reference)

// 원의 데이터 입력

(ㄱ) inputData((ㄴ)) // 함수 머리부

```
{
    cout << "원의 반지름 : ";
    cin >> c.radius;
    cout << "중심좌표(x) : ";
    cin >> c.cx;
    cout << "중심좌표(y) : ";
    cin >> c.cy;
}
.....
int main()
{
    Circle circle = {1, 2, 3};
    inputData(circle);
    .....
}
```



### (3) 인수의 전달

#### ■ 인수 전달 방법 - 참조 호출(call-by-reference)

```
// 원의 데이터 입력
void inputData(Circle& c) // 함수 머리부
{
    cout << "원의 반지름 : ";
    cin >> c.radius;
    cout << "중심좌표(x) : ";
    cin >> c.cx;
    cout << "중심좌표(y) : ";
    cin >> c.cy;
}
.....
int main()
{
    Circle circle = {1, 2, 3};
    inputData(circle);
    .....
}
```

c : Circle&  
circle의 참조

참조 전달

circle : Circle

1

radius

2

cx

3

cy



### (3) 인수의 전달

#### ■ 참조 호출을 통한 효율적인 인수 전달

- 참조 호출을 사용할 경우 형식 매개변수에 복사되는 데이터의 양은 실 매개변수의 크기와 관계 없이 일정함
  - ➡ 많은 양의 데이터로 구성되는 구조체나 객체를 인수로 전달하는 경우 효율적임
- 함수 호출의 효율성을 위해 참조 호출을 하지만 실 매개변수의 값이 변경되는 것을 원하지 않는 형식 매개변수에는 **const**를 지정하여 실 매개변수를 보호

### (3) 인수의 전달

#### ■ 인수 전달 방법 - 참조 호출(call-by-reference)

```
.....  
// 원의 데이터 출력  
void prData(const Circle& c) // 함수 머리부  
{  
    cout << " 반지름 = " << c.radius << endl;  
    cout << " 중심좌표 = (" << c.cx;  
    cout << ", " << c.cy << ")" << endl;  
}  
.....  
int main()  
{  
    Circle circle;  
    inputData(circle);  
    cout << "입력된 원의 정보" << endl;  
    prData(circle);  
    cout << "원의 면적 = " << area(circle) << endl;  
}
```

## (4) 디폴트 인수

### ■ 인수의 디폴트 값을 지정하는 방법

- 일반적으로 사용되는 디폴트 값이 있는 인수의 경우 함수를 선언할 때 그 값을 미리 지정할 수 있음



예

```
istream& get(char* pch, int max, char delim='\n');
```

char str[10];  
cin.get(str, 10); ⇨ cin.get(str, 10, '\n')과 동일

## (5) 함수의 다중정의

### ■ 다중정의(overloading)란?

- 동일한 이름에 대하여 여러 가지 의미를 부여하는 것

### ■ 함수 다중정의

- 동일한 이름을 갖는 함수를 여러 개 정의하는 것
- 동일한 개념의 처리를 여러 가지 데이터나 객체에 대해 각각의 대상에 맞는 처리를 해야 할 경우 사용함
- 다중정의된 함수의 구분 : 인수의 개수 및 자료형
  - ⚠ 함수의 반환 자료형으로 함수를 구분할 수 없음

## (5) 함수의 다중정의

### ■ int형 배열과 float형 배열 안의 최댓값을 구하는 함수 다중정의

```
int getMax(const int arr[], int len) // 함수 머리부
{
    int max = INT_MIN; // int형의 최솟값을 max로 가정함
    for (int i = 0; i < len; i++)
        if (max < arr[i])
            max = arr[i];
    return max; // 결과의 반환
}

float getMax(const float arr[], int len) // 함수 머리부
{
    float max = -numeric_limits<float>::max();
    for (int i = 0; i < len; i++)
        if (max < arr[i])
            max = arr[i];
    return max; // 결과의 반환
}
```

## (5) 함수의 다중정의

### ■ 함수 다중정의를 할 때 주의할 점 : 모호한 함수 다중정의

```
void f(int a) {  
    cout << a * a;  
}
```

```
void f(int a, int b) {  
    cout << a * b;  
}
```

```
int f(int c, int d) {  
    cout << c * d;  
}
```

```
int main() {  
    f(10);  
    f(10, 20);  
}
```

**에러!** (반환 자료형만 다르고  
형식 매개변수는 동일함)

## (5) 함수의 다중정의

### ■ 함수 다중정의를 할 때 주의할 점 : 모호한 함수 다중정의

```
void g(int a)
{
    cout << a * a;
}

void g(int a, int b = 100)
{
    cout << a * b;
}

int main()
{
    g(10, 20);
    g(10);    // 에러: 선택 기준이 모호함
    .....
}
```

## (5) 함수의 다중정의

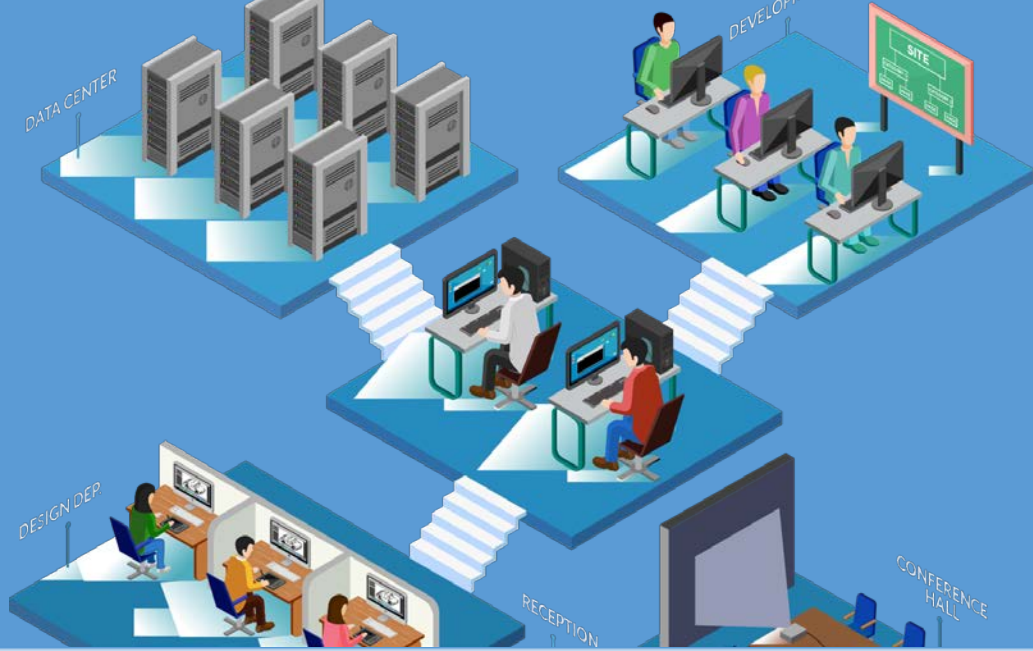
### ■ 함수 다중정의를 할 때 주의할 점 : 모호한 함수 다중정의

```
void h(int a)
{
    cout << a * a;
}

void h(float a)
{
    cout << a * b;
}

int main()
{
    h(10);
    h(10.0f);
    h(10.0); // 에러: 형 변환 대상이 모호함
}
```





수고하셨습니다.