1. C와 C++의 차이점

♲ C++ 언어란?

AT&T Bell 연구소의 비얀 스트로스트룹(Bjarne Stroustrup)이 1983년 C언어를 기반으로 객체지향이라는 개념이 바탕이 된 클래스, 상속, 가상함수, 연산자오버로딩, 자유로운 변수선언을 포함하는 C++ 이라는 언어를 개발했다. C++언어는 초기부터 C언어의 모든 기능을 포함할 수 있도록 설계되었음으로 C언어의 문법적인 특징과 기능을 모두 사용할 수 있다. 줄여서 말하자면 C++은 C언어의 기능에 객체지향이라는 개념이 추가되어 만들어진 언어이다.

♲ C언어와 C++ 의 차이점

1. C++에서 함수의 반환형은 항상 명시적이어야 한다.

설명 : C언어에서는 함수의 반환형(return type)을 명시하여 지정하지 않으면 int으로 가정하여 처리된다. 하지만 C++ 에서는 해당사항이 사라졌기 때문에 모든 함수의 반환형(return type)을 반드시 명시해야 한다.

예제 1-1 : C언어에서 return type을 명시하지 않을 경우

```
func(int data) {                // func함수의 return type의 생략되었음으로
                                // retuen type을 int로 가정하여 처리한다.
    printf("data=%dWn", data);
    return data;
}
```

예제 1-2 : C++ 에서 return type을 명시하지 않을 경우

```
int func(int data) {            // func함수의 return type을 명시하지 않으면
    printf("data=%dWn", data);
    return data;                // int형 data를 return시 에러가 발생한다.
}
```

2. 지역변수의 선언이 자유롭다.

설명 : C언어에서 지역변수는 반드시 함수의 블록이 시작하는 곳에서만 선언이 가능하다. 즉, 함수내의 실행문장 이전에 명시되어 있어야 한다. 하지만 C++에서는 지역변수를 프로그래머가 원하는 곳에서 정의하여 사용할 수 있게 되었다.

예제 2-1 : C언어의 변수선언 위치

```
void main( ) {
    int a = 10;                 // 함수의 시작 부분에 변수를 만들어야 한다.
    printf("a = %dWn", a);
    int b = 20;                 // 명령문 실행 후 변수를 만들면 에러가 발생한다.
    printf("b = %dWn", b);
}
```

예제 2-2 : C++ 의 변수선언 위치.

```
void main( ) {
    int a = 10;              // 함수의 시작 부분에 변수를 만들어야 한다.
    printf("a = %dWn", a);
    int b = 20;              // 명령문 실행 후 필요한 곳에서 변수를 만들어 사용 할 수 있다.
    printf("b = %dWn", b);
}
```

3. 입·출력과 관련된 헤더 파일이 변경되었다.

설명 : C언어에서는 입·출력 헤더 파일을 include 할 때 stdio.h를 사용하지만 C++에서는 iostream.h를 include해 사용한다. 또한 기본 입·출력도 printf()와 scanf() 함수 대신 cout 과 cin이라는 객체를 사용한다. (물론 C++에서도 printf()와 scanf()를 사용 할 수 있다.)

예제 3-1 : C언어에서의 입·출력

```
#include <stdio.h>              // 기본 입·출력 헤더파일로 stdio.h를 사용한다.
void main( ) {
    int a;
    printf("a의 값을 입력 :");    // printf()를 이용해 출력한다.
    scanf("%d", &a);            // scanf()를 이용해 입력받는다.
    printf("a = %dWn", a);
}
```

예제 3-2 : C++ 에서의 입·출력

```
#include <iostream.h>           // 기본 입·출력 헤더파일로 iostream.h를 사용한다.
void main( ) {
    int a;
    cout << "a의 값을 입력 :";   // cout을 이용해 출력한다.
    cin >> a;                   // cin을 이용해 입력받는다.
    cout << "a = " << a << endl;
}
```

iostream.h 헤더파일은 istream 과 ostream이 합해져 만들어진 헤더 파일이며 대표적인 입·출력 객체로 cin과 cout을 사용한다. cin과 cout 객체는 C언에서 사용하는 printf( ), scanf( ) 등의 함수와는 달리 입·출력 시 변수의 자료형에 따라 서식을 별도로 지정하지 않아도 된다. 따라서 보다 편리한 입·출력이 가능한 객체이다.

<table>
<tr><th>구 분</th><th>설 명</th></tr>
<tr><td>cin</td><td>C++에서 사용하는 대표적 입력 객체이다. "cin >> 변수명" 과 같은 형태로 사용하며 입력 서식을 사용할 필요가 없다. 또한 char 입력 시 입력 버퍼의 내용을 자동으로 비워줌으로 fflush(stdin);을 사용할 필요가 없다.

cin을 이용해 입력을 받는 방법 (정수, 문자, 실수, 문자열 등)
<table>
<tr>
<td>int a;
cin >> a;</td>
<td>char a;
cin >> a;</td>
<td>double a;
cin >> a;</td>
<td>char name[10];
cin >> name;</td>
<td>unsigned int a;
cin >> a;</td>
</tr>
</table>
cin을 이용해 여러개의 변수에 연달아 입력을 받는 방법 :
<table>
<tr>
<td>int a;
int b;
cin >> a >> b;</td>
<td>double a;
double b;
cin >> a >> b;</td>
<td>char name[11];
char add[100];
cin >> name >> add;</td>
</tr>
</table>
</td></tr>
</table>

cout

C++에서 사용하는 대표적 출력 객체이다. "cout << 출력할내용" 과 같은 형태로 사용하며 출력 서식을 사용할 필요가 없다. 같은 형태로 문자열과 포인터 역시 출력 할 수 있으며 실수 출력 시 소숫점 이하의 수는 자동으로 0이 아닌 수 까지만 출력 된다.

cout을 이용해 상수를 출력 하는 방법 (정수, 실수, 문자, 일반 문자열 상수)

```
cout << 10;
cout << 'Z';
cout << 3.14;
cout << "안녕하세요";
cout << "반갑습니다₩n";
cout << "문자 상수들을 출력합니다.₩n";
cout << "1₩b2₩t3₩a₩0";
cout << "₩n";
```

cout을 이용해 변수를 출력 하는 방법 (정수, 문자, 실수, 문자열, 포인터 등)

```
int a=10;
cout << a;
```

```
char a='Z';
cout << a;
```

```
double a=3.14;
cout << a;
```

```
char name[10]="korea";
cout << name;
```

```
int a;
cout << &a;
```

cout을 이용해 여러 개의 변수를 연달아 출력하는 방법 :

```
int a=10;
int b=20;
cout << a << b;
```

```
double a=3.14;
double b=2.48;
cout << a << b;
```

```
char name[11]="korea";
char add[100]="우리집";
cout << name << add;
```

cout을 이용해 문자열 상수와 변수를 같이 출력하는 방법

```
int a= 10;    char b = 'Z';    double c = 3.14;
cout << "a = " << a << "입니다.₩n";
cout << "b = " << b << "입니다.₩n";
cout << "c = " << c << "입니다.₩n";
```

※ visual studio 2005 이상 버전을 사용하는 사용자는 #include <iostream.h> 대신
#include <iostream>
using namespace std;
로 헤더 파일을 정의해야 한다. (namespace의 자세한 설명은 교재 뒷부분을 참고한다.)
(visual studio 2005 이상 버전에서는 iostream.h의 내용이 별도의 파일이 아닌 namespace std에 선언되어 있기 때문이다.)

cout과 cin사용시 각종 조정문자를 사용할 수 있다.

조정문자	용도	설 명
dec	출력시	정수 출력 시 해당 정수가 10진수로 출력된다.
hex	출력시	정수 출력 시 해당 정수가 16진수로 출력된다.
oct	출력시	정수 출력 시 해당 정수가 8진수로 출력된다.
endl	출력시	줄을 바꿔준다.
ends	출력시	NULL문자를 출력해준다.
flush	출력시	버퍼내의 자료를 제거한다.

iomanip.h 헤더파일을 include하면 아래의 조정 문자를 추가로 사용 할 수 있다.

조정문자	용도	설 명
setw(int)	출력시	가로 ( ) 안에 입력한 정수의 크기만큼 공간을 확보한 뒤 출력 대상을 확보한 공간 안에 출력해 준다. 출력 대상의 길이가 확보한 공간보다 크다면 공간 확보는 무시된다.
setfill(int)	출력시	setw( )로 확보된 공간에 대상을 출력한 뒤 빈 공간을 가로 ( ) 안의 값으로 대체해 준다. 가로 안에는 원하는 문자의 ascii 코드를 입력한다.
setprecision(int)	출력시	차후 출력할 실수가 소수점 이하 몇 번째 수까지 출력되게 할 것인지를 정할 수 있게 해준다. setprecision( )을 사용하려면 우선 setiosflags(ios::fixed)를 이용해 실수의 출력 형태를 고정소수점 형태로 지정해야 한다.
setbase(int)	출력시	출력할 정수의 진수를 설정할 수 있게 해준다. (8, 10, 16으로 설정가능하다.)
setiosflags(flag)	입출력시	가로 ( )안에 있는 flag로 출력 형식을 지정한다.
resetiosflags(flag)	입출력시	setiosflags( )를 이용해 지정한 형식을 해제한다.

setiosflags( ) 에 사용할 수 있는 flag들

flag	기 능
ios::left	확보된 공간 내에서 출력 대상이 왼쪽 정렬되어 출력된다.
ios::right	확보된 공간 내에서 출력 대상이 오른쪽 정렬되어 출력된다.
ios::dec	정수 출력 시 해당 정수가 10진수로 출력된다.
ios::oct	정수 출력 시 해당 정수가 8진수로 출력된다.
ios::hex	정수 출력 시 해당 정수가 16진수로 출력된다.
ios::uppercase	16진수 출력 시 a~f의 숫자가 대문자로 출력된다. ios::scientific을 이용해 실수 출력 시 지수표현인 e가 대문자로 출력된다.
ios::scientific	실수 출력 시 해당 실수를 지수형(부동소수점 형태)으로 출력되게 한다.
ios::fixed	실수 출력 시 해당 실수를 고정소수점 형태로 변환 한다.

4. 자료형(data type) bool이 추가되었다.

설명 : 1byte의 크기를 가지는 진위형 연산자 bool이 추가되었다. bool은 오직 참과 거짓(true, false)만을 저장하기 위해 사용한다. bool에 참을 보관하고 싶다면 true, 거짓을 보관 하고 싶다면 false를 대입하면 된다. bool형 변수 안의 값을 출력 할 경우 bool형 변수 안의 값이 참이면 1이 출력되고 거짓일 경우 0이 출력된다. 자료형 bool은 C++에서 보다 완벽하게 논리와 정수형을 구분 할 수 있는 방법을 제공해 준다.

예제 4-1 : bool형 변수의 정의방법, 크기, 값의 대입, 출력.

```
#include <iostream.h>
void main( ) {
    bool a;                     // 논리 참·거짓을 보관 할 수 있는 bool형 변수 a를 선언
    cout << sizeof(a) << endl;  // sizeof 연산자로 a의 크기를 출력한다. 1byte임으로 1이 나온다.
    a = true;                   // bool형 변수 a에 true(참) 값을 대입
    cout << a << endl;          // 변수 a의 값이 true임으로 1이 출력됨.
    a = false;                  // bool형 변수 a에 false(거짓) 값을 대입
    cout << a << endl;          // 변수 a의 값이 false임으로 0이 출력됨.
}
```

예제 4-2 : bool형 변수를 함수의 매개변수로 주는 방법과 리턴 받는 방법

```
#include <iostream.h>

bool func( bool x ) {           // bool형 data를 매개변수로 받아와 x에 저장하고 동작하여 결과로
                                // bool형 자료가 return 되는 함수 func()
    cout << x << endl;          // x가 true임으로 1이 출력됨.
    return x;                   // x의 값을 return
}

void main( ) {
    bool a, b;                  // 논리 참·거짓을 보관 할 수 있는 bool형 변수 a를 선언
    a = true;                   // bool형 변수 a에 true(참) 값을 대입
    b = func( a );              // 변수 a를 func함수에 매개변수로 건네준 뒤 그 결과를 b에 대입.
                                // func함수에서 return된 true값이 b에 대입된다.
    cout << b << endl;          // b가 true임으로 1이 출력됨.
}
```

5. 범위 지정 연산자 :: (scope연산자)의 추가

설명 : 변수 또는 함수를 호출 때 원하는 영역을 선택하여 변수나 함수를 호출 할 수 있도록 해주는 연산자 :: (scope연산자)가 추가되었다. C언어에서는 ::연산자가 없기 때문에 아래의 예제6과 같이 지역변수에 의해 전역변수가 가릴 경우 변수의 이름을 통한 전역변수의 호출이 불가능하다.

:: 연산자의 사용방법 : **호출하려는변수가있는영역명::호출하고싶은변수명**
호출하려는변수가있는영역명::호출하고싶은함수명
::호출하려는변수나함수명

:: 연산자의 동작형태 : A::B의 형태로 사용하면 B를 호출할 때 A영역에 안에서 호출한다.
::B와 같이 단항으로 사용하면 전역에서 B를 호출한다.

예제 5-1 : 전역변수와 지역변수의 사용가능 영역이 겹칠 경우 ::의 사용방법

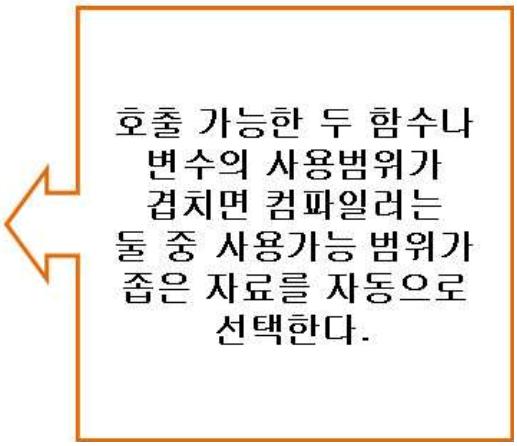
```
#include <stdio.h>              // iostram을 배우기 이전임으로 stdio.h를 include하였다.

int a = 10;                     // 전역변수 a를 선언하여 10으로 초기화 한다.

void main( ) {
    int a = 20;                 // 지역변수 a를 선언하여 20으로 초기화 한다.
                                // 전역변수와 지역변수는 메모리에 구현되는 위치가 다름으로 같은
                                // 이름으로 선언 될 수 있다. (전역변수는 data영역에 만들어지고
                                // 지역변수는 stack영역에 만들어 진다.)
    printf("a = %dWn", a);      // 전역변수 a와 지역변수 a의 사용영역이 겹칠 경우 컴파일러는
                                // 자동으로 지역변수를 선택하여 호출한다. 따라서 전역변수는
                                // 지역변수에의해 가려짐으로 C언어에서는 전역변수의 호출이
                                // 불가능하다.
    printf("a = %dWn", ::a);    // 전역변수 a를 호출하기 위하여 a앞에 ::를 붙였다.
                                // ::a는 전역에서 a를 호출하라는 의미가 된다.
}
```

예제 5-1의 해설 : 컴파일러는 호출 가능한 함수 또는 변수의 사용영역이 겹치면 무조건 둘 중 사용 가능 범위가 좁은 자료를 선택한다. (사용 가능 범위가 좁다는 것은 좀 더 가까이에서 만들었다는 의미이며 그 이름이 있지만 같은 이름으로 변수 또는 함수를 다시 만들었다는 것은 새로 만든 것을 사용하고 싶다는 것이라 판단하기 때문이다.)

```
#include <stdio.h>

int a = 10

void main( ) {

int a = 20

printf("a = %d₩n", a);

printf("a = %d₩n", ::a);

}
```

전역변수 a의 사용가능 영역

지역변수 a의 사용가능 영역



6. namespace(이름공간)의 추가

설명 : namespace(이름공간)는 말 그대로 프로그램의 특정 영역에 이름을 붙여주는 도구이다. 대형화된 프로젝트의 경우 여러명의 개발자가 만든 소스를 하나의 프로그램으로 합할 경우 각 개발자가 프로그램을 만들면서 사용한 함수등의 명칭에 중복 될 수 있다.

예제 6-1 : namespace(이름공간)을 사용하지 않은 경우 발생하는 문제

```
#include <iostream.h>

void func( ) {
    printf("개발자 A가 만든 func()₩n");
}

void func( ) {
    printf("개발자 B가 만든 func()₩n");
}

void main( ) {
    func();                    // 호출 가능한 func함수가 두 개임으로 어떠한 함수를 호출 할 것
                               // 인지 결정 할 수 없게 된다. 따라서 에러가 발생함.
}
```

namespace를 만드는 방법 : namespace 사용하고자하는이름공간명 {
내용....
.....
}

namespace의 사용방법 : namespace 사용하고자하는이름공간명;

namespace의 동작형태 : 앞으로 함수나 변수를 호출할 경우 지정한 namespace 안에서 호출한 이름을 가지는 함수와 변수를 찾아서 불러온다.

예제 6-2 : namespace(이름공간)를 만들고 사용하는 방법

```
#include <iostream.h>
namespace engineerA {                          // namespace(이름공간) engineerA를 만든다.

    void func( ) {
        printf("개발자 A가 만든 func()₩n");
    }

}
namespace engineerB {                          // namespace(이름공간) engineerB를 만든다.

    void func( ) {
        printf("개발자 B가 만든 func()₩n");
    }
}
void main( ) {
    engineerA :: func();      // func함수를 호출하기 위해 영역지정 연산자 :: 를 사용해
                              // engineerA 라는 namespace에서 func()를 호출하였다.
    engineerB :: func();      // func함수를 호출하기 위해 영역지정 연산자 :: 를 사용해
                              // engineerB 라는 namespace에서 func()를 호출하였다.
}
```

예제 6-2의 해설 : 두 func함수는 같은 이름이지만 서로 다른 namespace안에 있다. 또한 호출하여 사용 할 때 namespace를 지정하여 호출하였음으로 서로 충돌하지 않고 선택한 영역 안에 있는 함수를 호출 할 수 있게 된다.

7. 명시적 캐스팅

설명 : C언어에서 사용하던 "(자료형)변수명" 형태의 강제 형 변환 작업을 "자료형(변수명)"과 같은 식으로 사용 할 수 있게 되었다.

예제 7-1 : C언어와 C++언어의 강제 형 변환 방법

```
#include <iostream.h>

void main( ) {
    int a = 65;
    cout << a << endl;          // 65가 출력된다.
    cout << (char)a << endl;    // C언어의 방식으로 강제 형 변환을 했음으로 'A' 가 출력된다.
    cout << char(a) << endl;    // C++의 방식으로 강제 형 변환을 했음으로 'A' 가 출력된다.
}
```

예제 7-1의 해설 : 위의 예에서 (char)a 는 C언어의 강제 형 변환 방식이고 char(a)는 C++의 강제 형 변환 방식이다. C언어의 강제 형변환 방식인 (char)a는 (char)a&&b 와 같은 형태로 사용될 경우 연산 우선순위에 혼란이 올 수 있지만 char(a)는 char(a)&&b 와 같이 강제 형 변환 시킬 대상이 가로안에 있음으로 보다 명확하게 수식을 이해 할 수 있도록 해준다. 또한, char 함수의 매개변수로 a를 넘겨주는 것과 유사하게 보여 사용자가 수식을 보다 쉽게 이해 하는데 도움을 준다.

8. inline 함수의 추가 (확장함수)

설명 : 함수의 실행속도 향상을 위해 함수의 실행코드를 함수가 호출된 곳에 넣어 컴파일 되도록 해주는 inline함수가 추가되었다. 단, 모든 함수에 inline키워드를 붙인다고 해서 속도가 향상되는 것은 아니다. inline함수는 소스코드가 짧고 자주 호출되는 함수인 경우에 사용하는 것이 효율적이다.

inline 함수의 사용방법 : 함수를 만든 후 함수의 return type 앞쪽에 inline 키워드를 붙인다.

```
inline 반환형 함수명 (매개변수...) {
    확장함수의 내용;
}
```

inline 함수의 동작모양 : inline키워드를 붙인 함수는 컴파일 시 함수의 소스코드가 함수가 호출된 곳에 복사되어 실행된다.

예제 8-1 : inline 함수와 일반함수의 동작 차이 (위의 그림처럼 코드를 작성하면 아래처럼 동작한다.)

그림 1. 일 반 함 수

```
#include <iostream.h>
void func( ){              // 일반함수
    cout << "하하하" << endl;
}
void main( ) {
    func();
}
```

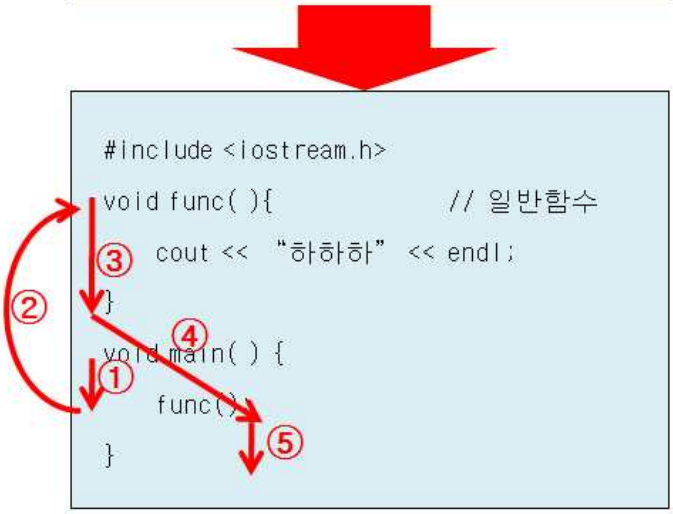
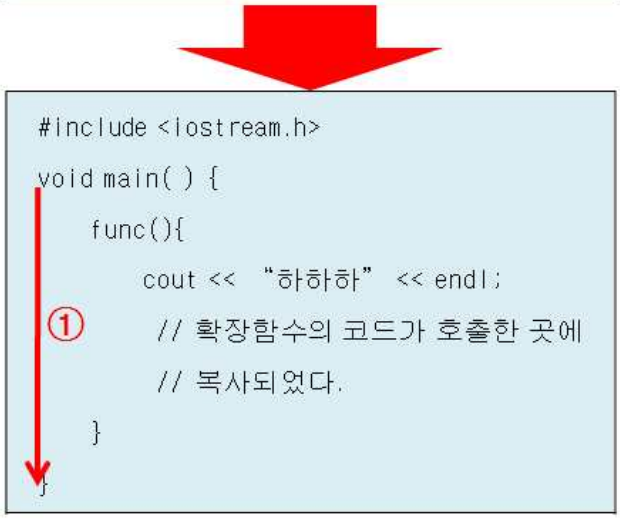


그림 2. 확 장 함 수

```
#include <iostream.h>
inline void func( ){       // 확장함수
    cout << "하하하" << endl;
}
void main( ) {
    func();
}
```



예제 8-1의 해설 : 그림 1. 일반함수의 경우는 함수를 호출하면 해당 함수가 있는 곳으로 이동하여 호출한 함수가 실행되고 함수가 종료하면 함수를 호출한 곳으로 복귀하게 된다. 따라서 이동시간이 발생하지만 그림 2. 의 경우 함수가 inline함수로 선언되었기 때문에 함수의 실행코드가 함수가 호출된 곳에 복사되었다. 따라서 함수의 호출에 따라 이동하지 않고 프로그램이 실행됨으로 그림 1. 보다 빠른 실행이 가능해 진다. (이동하지 않는다는 것은 분기가 발생하여 func함수의 실행을 위한 정보가 별도로 메모리에 구현되지 않는다는 의미이다.)

9. default parameter (디폴트 매개변수)의 추가

설명 : 함수의 호출시 함수가 받아오기로 했던 매개변수를 건네주지 않는다면 미리 지정한 값이 대신 매개변수에 전달되도록 해주는 것이 디폴트 매개변수이다. 단, 디폴트 매개변수는 항상 마지막 매개변수부터 지정해야 한다.

사용방법 : 함수선언 및 정의 시 매개변수에 default값을 지정해 준다.

```
void func(int x, int y = 10) {
    내용;
}
```

동작형태 : 디폴트 매개변수로 지정된 매개변수에 값이 넘어오지 않으면 자동으로 미리 지정한 값이 매개변수로 전달된다.

예제 9-1 : default 매개변수가 없을 경우

```
#include <stdio.h>

void func ( int a, int b ) {
    printf("매개변수 a = %d₩n", a);
    printf("매개변수 b = %d₩n", b);
}

void main( ) {
    func(10, 20);          // func함수에 매개변수로 int 두 개를 모두 건네주었음으로 문제없다.
    func(10);              // 두 개의 int가 매개변수로 필요한 func함수에 한 개의 int만 매개변수
                           // 로 건네주었음으로 에러가 발생한다.
}
```

예제 9-2 : default 매개변수의 사용

```
#include <stdio.h>

void func ( int a, int b = 10 ) {
    printf("매개변수 a = %d₩n", a);
    printf("매개변수 b = %d₩n", b);
}

void main( ) {
    func(10, 20);          // func함수에 매개변수로 int 두 개를 모두 건네주었음으로 문제없다.
    func(10);              // 두 개의 int가 매개변수로 필요한 func함수에 한 개의 int만 매개변수
                           // 로 건네주었지만 func함수의 두 번째 매개변수가 default 매개변수로
                           // 지정되어 있음으로 미지 지정한 10이 b에 들어가 함수가 동작한다.
}
```

예제 9-3 : default 매개변수의 활용

```
#include <stdio.h>

void func ( int mon, int day , int year = 2010 ) {
    printf("%d년 %d월 %d일입니다.₩n", year, mon, day);

}

void main( ) {
    func(1, 20, 2010);      // func함수는 "2010년 1월 20일입니다." 라는 문장을 출력한다.
    func(2, 20);            // year이 default 처리되면서 2010값이 전달된다. 따라서
                            // func함수는 "2010년 2월 20일입니다." 라는 문장을 출력한다.
}
```

예제 9-4 : default 매개변수의 사용 중 에러발생 (에러발생)

```
#include <stdio.h>

void func ( int a, int b = 10, int c) {   // 마지막 매개변수가 아닌 자료에 디폴트 매개변수를
                                          // 지정해 주면 에러가 발생한다.
    printf("매개변수 a = %d₩n", a);
    printf("매개변수 b = %d₩n", b);
    printf("매개변수 c = %d₩n", c);
}

void main( ) {

    func(10, 20);           // 세 번째 매개변수에 값이 넘어가지 않는다.
    func(10, 20, 30);       // 세 번째 매개변수에 값을 넣으려면 함수호출 시 매개변수를 세 개
                            // 넣어주어야 하지만 그렇다면 default매개변수의 의미가 없어진다.
}
```

예제 9-4 해설 : 디폴트 매개변수는 반드시 마지막 매개변수부터 지정되어야 한다.

뒤에서부터 확장하며 디폴트 매개변수로 지정하는 것은 가능하다.

잘된 예 :

```
void func( int a      , int b      , int c =30 )
void func( int a      , int b = 20, int c =30 )
void func( int a = 10, int b = 20, int c =30 )
```

앞이나 중간에서부터 디폴트 매개변수를 지정하면 에러가 발생한다.

잘못된 예 :

```
void func( int a = 10, int b      , int c      )
void func( int a = 10, int b = 20, int c      )
void func( int a      , int b = 20, int c      )
```

예제 9-5 : default 매개변수로 두 개 이상의 매개변수를 지정하는 방법

```
#include <stdio.h>

void func ( int a, int b = 10, int c = 20) {
    printf("매개변수 a = %d₩n", a);
    printf("매개변수 b = %d₩n", b);
    printf("매개변수 c = %d₩n", c);
}

void main( ) {
    func(50);              // a에는 50이 매개변수로 들어가고 default 매개변수의 의해 b에는 10이
                           // c에는 20 전달되어 함수가 동작한다.
    func(50, 60);          // a에는 50이 b에는 60이 매개변수로 들어가고 default 매개변수의 의해
                           // c에는 20 전달되어 함수가 동작한다.
    func(50, 60, 70);      // 매개변수 세 개의 값을 모두 넣어주었음으로 a에는 50이 b에는 60이
                           // c에는 70이 매개변수로 들어간다.
}
```

예제 9-6 : 모든 매개변수를 default 매개변수로 지정하는 방법

```
#include <stdio.h>

void func ( int a = 10, int b = 20, int c = 30) {
    printf("매개변수 a = %d₩n", a);
    printf("매개변수 b = %d₩n", b);
    printf("매개변수 c = %d₩n", c);
}

void main( ) {
    func();                // 매개변수의 값이 하나도 전달되지 않았음으로 default 매개변수에
                           // 의해 a에는 10이 b에는 20이 c에는 30이 전달되어 함수가 동작한다.
    func(50);              // a에는 50이 매개변수로 들어가고 default 매개변수의 의해 b에는 20이
                           // c에는 30 전달되어 함수가 동작한다.
    func(50, 60);          // a에는 50이 b에는 60이 매개변수로 들어가고 default 매개변수의 의해
                           // c에는 30 전달되어 함수가 동작한다.
    func(50, 60, 70);      // 매개변수 세 개의 값을 모두 넣어주었음으로 a에는 50이 b에는 60이
                           // c에는 70이 매개변수로 들어간다.
}
```

예제 9-7 : default 매개변수가 있는 함수의 원형(prototype)을 작성하는 방법

```
#include <stdio.h>

void func ( int a = 10, int b = 20, int c = 30);  // 함수의 프로토타입이 있는 경우에는 프로토
                                                  // 타입에 디폴트 매개변수를 지정해 준다.
void main( ) {
    func();               // 매개변수의 값이 하나도 전달되지 않았음으로 default 매개변수에
                          // 의해 a에는 10이 b에는 20이 c에는 30이 전달되어 함수가 동작한다.
    func(50);             // a에는 50이 매개변수로 들어가고 default 매개변수의 의해 b에는 20이
                          // c에는 30 전달되어 함수가 동작한다.
    func(50, 60);         // a에는 50이 b에는 60이 매개변수로 들어가고 default 매개변수의 의해
                          // c에는 30 전달되어 함수가 동작한다.
    func(50, 60, 70);     // 매개변수 세 개의 값을 모두 넣어주었음으로 a에는 50이 b에는 60이
                          // c에는 70이 매개변수로 들어간다.
}

void func ( int a , int b , int c ) {   // 함수의 프로토타입에서 디폴트 매개변수를 지정한 후
                           // 함수의 정의시 다시 디폴트 매개변수 값을 넣으면 에러가 발생한다.
    printf("매개변수 a = %d₩n", a);
    printf("매개변수 b = %d₩n", b);
    printf("매개변수 c = %d₩n", c);
}
```

예제 9-8 : char값을 default 매개변수로 지정하는 방법

```
#include <stdio.h>

void func1 ( char a = 'Z' );      // char형 변수 a의 default 매개변수로  'Z' 를 지장하였다.

void func2 ( char a = 'Z' ) {     // char형 변수 a의 default 매개변수로  'Z' 를 지장하였다.
    printf("매개변수 a = %c₩n", a);       // 'Z' 가 출력된다.
}

void main( ) {
    func1( );       // 함수의 프로토타입을 만든 함수의 호출
    func2( );       // 함수의 프로토타입을 만들의 않은 함수의 호출
}

void func1 ( char a ) {
    printf("매개변수 a = %c₩n", a);      // 'Z' 가 출력된다.
}
```

예제 9-9 : 문자열을 default 매개변수로 지정하는 방법

```
#include <stdio.h>

void func1 ( char *a = "버내너" );    // char형 변수 a의 default 매개변수로  'Z' 를 지장하였다.

void func2 ( char *a = "바나나" ) {   // char형 변수 a의 default 매개변수로  'Z' 를 지장하였다.
    printf("매개변수 a = %s₩n", a);       // 'Z' 가 출력된다.
}

void main( ) {
    func1( "홍길동" );                // 함수의 프로토타입을 만든 함수의 호출
    func2( "김을동" );                // 함수의 프로토타입을 만들의 않은 함수의 호출
}

void func1 ( char *a ) {
    printf("매개변수 a = %s₩n", a);      // 'Z' 가 출력된다.
}
```

10. 함수의 오버로딩(function overloading) - 중복함수

설명 : 매개변수의 개수나 자료형이 다르다면 같은 이름으로 함수를 여러개 만들어 사용할 수 있게 해주는 함수의 오버로딩 기능이 추가되었다. 오버로딩된 함수는 이름이 같음으로 함수 호출시 넘겨진 매개변수의 종류에 따라 호출될 함수가 결정된다. 함수의 오버로딩은 객체지향언어의 다형성을 지원하는 기능이며 같은 이름으로 그때 상황(건네준 매개변수)에 알맞은 함수가 호출되도록 해줌으로 보다 편하게 함수를 사용할 수 있도록 해준다.

예제 10-1 : 함수의 오버로딩 사용방법 (매개변수가 없는 func와 매개변수로 int를 받아가는 func)

```
#include <stdio.h>
void func ( ) {             // func함수가 호출되면서 매개변수가 넘어오지 않을 경우 호출된다.
    printf("난 매개변수가 없는 func() 함수₩n");
}

void func ( int data ) {     // func함수가 호출되면서 int가 매개변수로 넘어올 경우 호출된다.
    printf("난 %d를 매개변수로 받아온 func() 함수 ₩n", data);
}

void main( ) {
    func( ) ;          // 함수호출시 매개변수를 안 넘겨주었음으로 void func()가 호출된다.
    func( 10 ) ;       // 함수호출시 10을 매개변수로 주었음으로 void func(int data)가 호출된다.
}
```

예제 10-2 : 함수의 오버로딩 시 매개변수의 모양이 똑같다면 에러가 발생한다. (에러발생)

```
#include <stdio.h>
void func ( ) {             // func함수가 호출되면서 매개변수가 넘어오지 않을 경우 호출된다.
    printf("난 매개변수가 없는 첫 번째 func() 함수₩n");
}
void func ( ) {             // func함수가 호출되면서 매개변수가 넘어오지 않을 경우 호출된다.
    printf("난 매개변수가 없는 두 번째 func() 함수₩n");
}
void main( ) {
    func( ) ;          // 함수호출시 매개변수를 안 넘겨주었지만 위의 두 함수가 모두 매개변수를
                       // 필요로 하지 않음으로 어떠한 함수가 호출될지 결정 할 수 없다. 따라서
}                      // 에러가 발생한다.
```

예제 10-3 : 함수의 오버로딩 사용방법 (매개변수가 int인 func와 매개변수로 char를 받아가는 func)

```
#include <stdio.h>
void func ( int data ) {        // func함수가 호출되면서 int를 매개변수로 받아올 경우 호출된다.
    printf("난 data가 %d인 func함수₩n", data);
}
void func ( char data ) {       // func함수가 호출되면서 char를 매개변수로 받아올 경우 호출된다.
    printf("난 data가 %c인 func함수₩n", data);
}
void main( ) {
    func( 65 ) ;     // 함수호출시 65을 매개변수로 주었음으로 void func(int data)가 호출된다.
    func( 'B' ) ;    // 함수호출시 'B'를 매개변수로 주었음으로 void func(char data)가 호출된다.
}
```

예제 10-4 : 함수의 오버로딩 사용방법 (매개변수가 int인 func와 매개변수로 long을 받아가는 func)

```
#include <stdio.h>
void func ( int data ) {        // func함수가 호출되면서 int를 매개변수로 받아올 경우 호출된다.
    printf("난 data가 %d인 func(int data) 함수₩n", data);
}
void func ( long data ) {       // func함수가 호출되면서 long을 매개변수로 받아올 경우 호출된다.
    printf("난 data가 %d인 func(long data) 함수₩n", data);
}
void main( ) {
    func( 65 ) ;     // 함수호출시 65을 매개변수로 주었음으로 void func(int data)가 호출된다.
    func( 66l ) ;    // 함수호출시 66l을 매개변수로 주었음으로 void func(long data)가 호출된다.
}                    // cf : 정수형 상수 뒤쪽에 소문자 l을 붙이면 long형 자료가 된다.
```

예제 10-5 : 세 가지 이상으로 함수를 오버로딩 할 수 있다.

```
#include <stdio.h>
void func ( int data ) {      // func함수가 호출되면서 int를 매개변수로 받아올 경우 호출된다.
    printf("난 data가 %d인 func(int data) 함수₩n", data);
}
void func ( long data ) {     // func함수가 호출되면서 long을 매개변수로 받아올 경우 호출된다.
    printf("난 data가 %d인 func(long data) 함수₩n", data);
}
void func ( char data ) {     // func함수가 호출되면서 char을 매개변수로 받아올 경우 호출된다.
    printf("난 data가 %c인 func(char data) 함수₩n", data);
}
void func ( char* data ) {    // func함수가 호출되면서 char *을 매개변수로 받아올 경우 호출된다.
    printf("난 data가 %s인 func(char* data) 함수₩n", data);
}
void main( ) {
    func( 65 ) ;     // 함수호출시 65을 매개변수로 주었음으로 void func(int data)가 호출된다.
    func( 66l ) ;    // 함수호출시 66l을 매개변수로 주었음으로 void func(long data)가 호출된다.
    func( 'Z' ) ;    // 함수호출시 'Z'를 매개변수로 주었음으로 void func(char data)가 호출된다.
    func("홍길동");  // 함수호출시 "홍길동"을 매개변수로 주었음으로 void func(char* data)가
                     // 호출된다.
}
```

예제 10-6 : 매개변수의 개수를 다르게 한 함수의 오버로딩

```
#include <stdio.h>
void func ( int data ) {   // func함수가 호출되면서 int 한 개를 매개변수로 받아올 경우 호출된다.
    printf("난 %d 를 매개변수로 받아온 func함수₩n", data);
}
void func ( int data, int data2 ) {
                           // func함수가 호출되면서 int 두 개를 매개변수로 받아올 경우 호출된다.
    printf("난 %d, %d 를 매개변수로 받아온 func함수₩n", data, data2);
}
void func ( int data, int data2, int data3 ) {
                           // func함수가 호출되면서 int 세 개를 매개변수로 받아올 경우 호출된다.
    printf("난 %d, %d, %d를 매개변수로 받아온 func함수₩n", data, data2, data3);
}
void main( ) {
    func( 10 ) ;         // 10을 매개변수로 주었음으로 void func(int data)가 호출된다.
    func( 10, 20 ) ;     // 10,20을 매개변수로 주었음으로 void func(int data, int data2)가
                         // 호출된다.
    func( 10, 20, 30 ) ; // 10,20,30을 매개변수로 주었음으로 void func(int data, int data2,
                         // int data3)가 호출된다.
}
```

11. 구조체 변수 선언 시 struct 키워드 없이 구조체 이름만 사용해 바로 변수를 선언 할 수 있다.

설명 : C언어 에서는 구조체가 완벽한 자료형으로 인정받지 못했기 때문에 구조체를 이용해 변수를 만들기 위해서는 구조체명 앞쪽에 struct 키워드를 붙여 사용해야 했지만 C++에서는 구조체가 완벽한 하나의 자료형으로 인정받게 되었음(멤버로 함수를 가질 수 있게 됨으로 스스로의 동작을 정의 할 수 있게 되었다.)으로 별도의 키워드 없이 구조체 명으로 즉시 구조체형 변수를 만들어 사용할 수 있게 되었다.

사용방법 : 만들어 놓은 구조체 이름이 std일 경우 C언어에서는 구조체형 변수를 만들 때
`struct std ob;` 와 같은 식으로 앞에 struct 키워드를 붙여야 하지만 C++ 에서는 struct 없이
`std ob;` 와 같은 형태로 변수를 만들 수 있다.

예제 11-1 : 구조체형 변수를 만드는 방법

```
#include <stdio.h>
struct std {                    // 이름이 std인 구조체 선언
    int num;
    char munja;
};

void main(){
    std obj;                    // 구조체 이름 std로 변수 obj를 선언
                                // 구조체형 변수 선언시 앞쪽에 struct를 붙이지 않아도 된다.
    obj.num=7;
    obj.munja='a';
    printf("obj.num = %d, obj.munja = %c ₩n", obj.num, obj.munja);
}
```

예제 11-2 : C++에서의 구조체는 멤버로 함수를 가질 수 있다.

```
#include <stdio.h>
struct std {                    // 이름이 std인 구조체 선언
    int num;
    char munja;
    void print ( ) {            // C++에서는 구조체의 멤버로 함수를 넣을 수 있게 되었다.
        printf("num = %d, munja = %c ₩n", num, munja);
    }
};

void main(){
    std obj;
    obj.num=7;
    obj.munja='a';
    printf("obj.num = %d, obj.munja = %c ₩n", obj.num, obj.munja);
    obj.print();                // 구조체 안의 멤버 중 함수를 호출하는 방법
}
```

예제 11-2 설명 : 멤버함수에 대한 자세한 설명은 3챕터 객체지향언어의 설명 부분을 참고한다.

12. 공용체 선언 시 이름 없는 공용체의 선언이 가능해졌다.

설명 : 여러개의 멤버가 하나의 기억장소를 공유해 사용할 수 있게 해주는 공용체를 선언 할 때 이름 없이 선언해 사용할 수 있다.

예제 12-1 : 이름 없는 공용체의 선언 방법

```
#include <stdio.h>
union {                          // 이름을 지정하지 않고 공용체를 정의한다.
    int num;
    char munja;
} ob;                            // 공용체형 변수 ob를 만든다.

void main(){
    ob.num=7;                    // 공용체의 멤버변수를 호출한다.
    printf("num = %d₩n", ob.num);
    ob.munja='a';                // 공용체의 멤버변수를 호출한다.
    printf("munja = %c₩n", ob.munja);
}
```

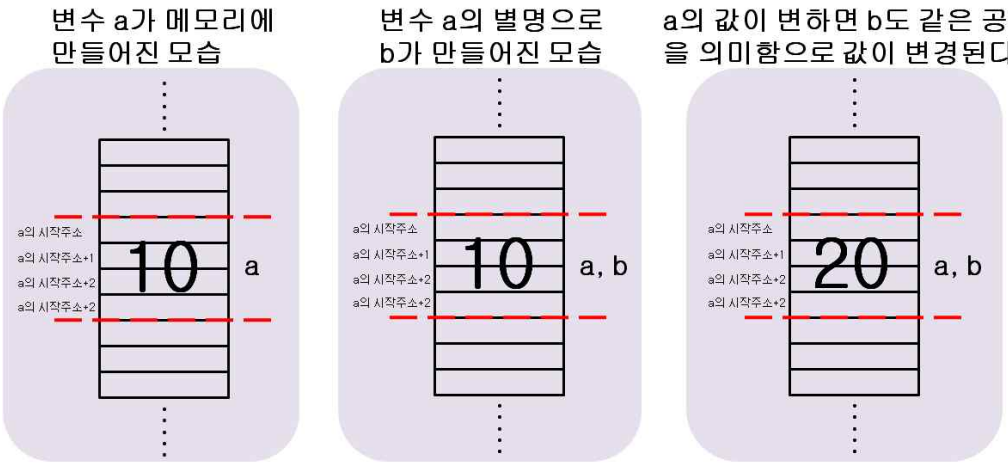
13. 참조연산자 &(레퍼런스 연산자)가 추가되었다.

설명 : 만들어 놓은 변수에 대해 또 다른 이름(별명)을 부여해 줄 수 있는 &연산자가 추가되었다. 기존 C 언어의 & 연산자는 **"&변수명"** 과 같은 형태로 사용되어 변수의 주소값을 구하기 위해서 사용했지만 C++에서는 위의 기능뿐 아니라 "자료형 &별명=기존변수명" 과 같은 형태로 사용해 기존에 생성한 변수에 대한 별명을 만들 수 있는 기능이 추가되었다. 생성된 별명(레퍼런스)은 기존 변수명과 동일한 용도로 사용할 수 있으며 생성한 별명(레퍼런스)을 함수의 매개변수나 return type에 기재할 수 도 있다. 별명을 만들때는 반드시 기존변수명과 같은 자료형으로 생성해야 한다.

예제 13-1 : &연산자를 이용한 별명 만들기

```
#include <stdio.h>
void main(){
    int a = 10;
    int &b = a;   // 기존에 생성한 변수 a의 별명을 b로 지정하였다.
                  // 변수를 만들면서 변수명 앞쪽에 &를 붙이면 별명을 만들겠다는 의미가 된다.
                  // "&변수명" 은 변수의 주소값을 구하라는 뜻이며 위의 문장과는 의미부터 다르다.
    printf(" &a = %p ₩n", &a); // 변수 a의 주소값이 변수 b의 주소값과 같다.
    printf(" &b = %p ₩n", &b); // 즉, 두 변수가 메모리의 같은 공간을 의미한다는 것을 알 수 있다.

    printf(" a = %d ₩n", a);     // 10 이 출력된다.
    printf(" b = %d ₩n", b);     // 10 이 출력된다.
    a = 20;
    printf(" a = %d ₩n", a);     // 20 이 출력된다.
    printf(" b = %d ₩n", b);     // 20 이 출력된다.
    b = 30;
    printf(" a = %d ₩n", a);     // 30 이 출력된다.
    printf(" b = %d ₩n", b);     // 30 이 출력된다.
}
```

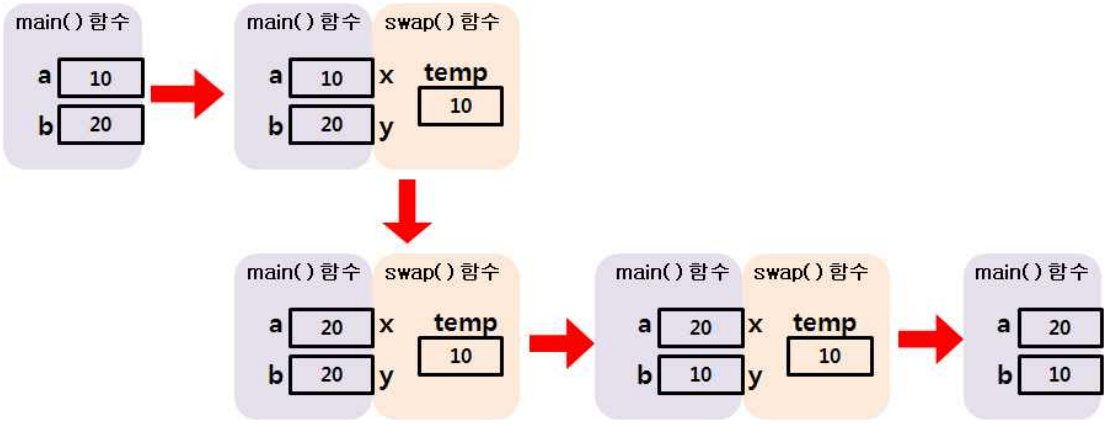
예제 13-1 해설 : 별명은 메모리의 같은 공간의 이름을 하나 추가 한다는 의미이다. 따라서 아래의 그림처럼 변수 a에 대한 별명이 b라는 이름으로 만들어 진다.



예제 13-2 : 별명을 매개변수로 받아가는 방법 (함수 호출의 세 가지 형태 중 Call By reference)

```
#include <stdio.h>

void swap( int &x, int &y) {       // 매개변수로 넘어온 자료에 별명을 x와 y로 부여한다.
    printf("swap 함수에서 작업 전 x = %d, y = %d ₩n", x, y);  // 10, 20 이 출력된다.
    int temp = x;                  // x와 y의 값을 바꾼다.
    x = y;
    y = temp;
    printf("swap 함수에서 작업 후 x = %d, y = %d ₩n", x, y);  // 20, 10 이 출력된다.
}

void main(){
    int a = 10, b = 20;
    printf("main 함수에서 func()호출 전 a = %d, b = %d ₩n", a, b);   // 10, 20 이 출력된다.
    swap(a, b);                    // a와 b를 func()함수의 매개변수로 보낸다.
    printf("main 함수에서 func()호출 후 a = %d, b = %d ₩n", a, b);   // 20, 10 이 출력된다.
}
```

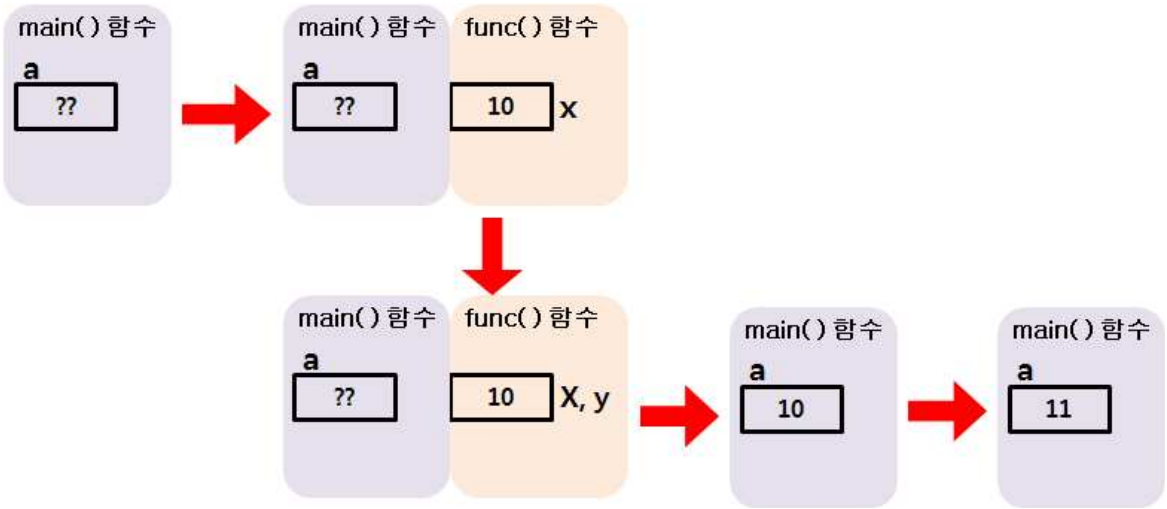
예제 13-2 해설 : 아래의 그림과 같이 main함수와 swap함수가 동작 하게 된다.



예제 13-3 : 별명을 return하는 방법

```
#include <stdio.h>
int & func ( ) {          // func함수에서 만든 별명 y를 리턴 함으로 return type이 int & 이다.
    int x = 10;
    int &y = x ;          // x에 대한 별명을 y로 선언
    return y;             // y를 리턴
}

void main(){
    int a;
    a = func( );                      // func()의 실행 결과를 a에 삽입
    printf("리턴된 a = %d ₩n", a);     // 리턴 된 값인 10 이 출력된다.
    a++;                              // a의 값을 1증가 시킨다.
    printf("a 값을 변경한 후 출력 a = %d ₩n", a);   // 11 이 출력된다.

}
```

예제 13-3 해설 : 아래의 그림과 같이 main함수와 func 함수가 동작 하게 된다.



14. 새로운 예약어와 연산자가 추가되었다.

class의 생성과 관련해 추가된 예약어 : class, friend, virtual, this

멤버에 대한 접근과 관련해 추가된 예약어 : private, protected, public

연산자와 관련해 추가된 예약어 : operator

예외처리를 위해 추가된 예약어 : try, catch, throw

template을 위해 추기된 예약어 : template

새롭게 추가된 연산자 : new, delete, ::,

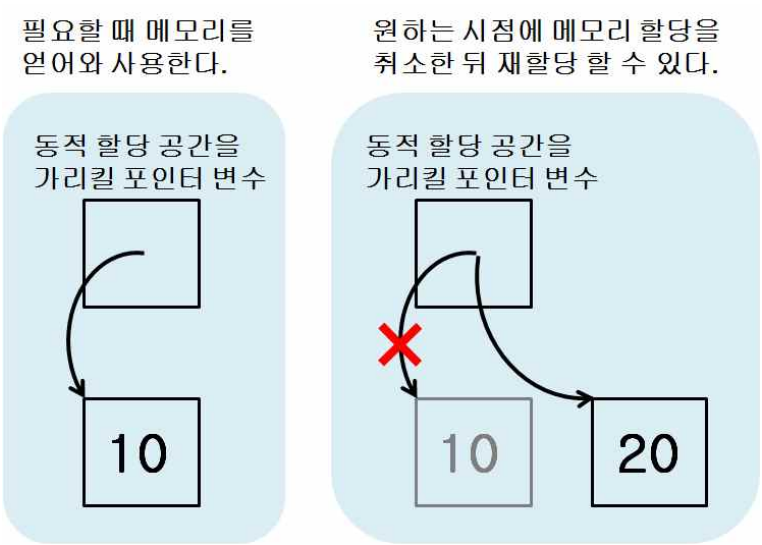
추가된 연산자가 포함된 C++ 연산자의 우선순위표

우선순위	연산자의 종류	결합 규칙
1(최우선)	() : 소괄호 [] : 대괄호 :: : scope (영역지정 연산자) . : 도트 (직접멤버 호출 연산자) -> : right arrow (간접멤버 호출 연산자)	왼쪽에로 오른쪽으로 (→)
2	! : not ~ : bit not + : 양수 (단항으로 사용될 경우) - : 부호변경 (단항으로 사용될 경우) ++ : 플러스플러스, 값 1증가 -- : 마이너스마이너스, 값 1감소 & : 레퍼런스 (단항으로 사용될 경우) * : 역참조 (단항으로 사용될 경우) sizeof : 자료를 크기 측정 new : 동적메모리할당 delete : 동적메모리할당 해제 type cast : 강제 형 변환	오른쪽에서 왼쪽으로 (←)
3	* : 곱하기 / : 나누기 % : 나머지	왼쪽에로 오른쪽으로 (→)
4	+ : 더하기 - : 빼기	왼쪽에로 오른쪽으로 (→)
5	<< : 왼쪽 쉬프트 >> : 오른쪽 쉬프트	왼쪽에로 오른쪽으로 (→)
6	< : 작다 <= : 작거나 같다 > : 크다 >= : 크거나 같다	왼쪽에로 오른쪽으로 (→)
7	== : 같다 != : 같지 않다	왼쪽에로 오른쪽으로 (→)
8	& : bit and (비트논리 곱 연산자)	왼쪽에로 오른쪽으로 (→)
9	^ : bit xor (비트논리 배타적논리합 연산자)	왼쪽에로 오른쪽으로 (→)
10	| : bit or (비트논리 합 연산자)	왼쪽에로 오른쪽으로 (→)
11	&& : and (일반논리 곱 연산자)	왼쪽에로 오른쪽으로 (→)
12	|| : or (일반논리 합 연산자)	왼쪽에로 오른쪽으로 (→)
13	?: : 삼항조건 연산자	왼쪽에로 오른쪽으로 (→)
14	= : 순수대입 연산자 *=, /=, %=. +=, -=, >>=, <<=, &=, ^=, |= : 연산대입 연산자.	오른쪽에서 왼쪽으로 (←)
15(최하위)	, : 그리고 (콤마 연산자)	왼쪽에로 오른쪽으로 (→)

1-2. 동적 메모리 할당

♲ 동적 메모리 할당

프로그램의 실행 중 사용자가 필요한 메모리를 할당받아 사용 할 수 있도록 해주는 기능이다. 일반적인 변수나 배열의 메모리 할당은 컴파일 작업 시 프로그램이 사용할 메모리의 크기를 계산하여 이루어지지만 동적 메모리 할당은 프로그램의 실행 중 이루어 지기 때문에 프로그램의 작성 시 자료의 크기를 예측하기 어려운 경우 유용하게 사용 할 수 다. 또한 동적으로 할당된 메모리 공간은 프로그래머가 명시적으로 해제하거나 쓰레기 수집이 일어나기 전까지 그대로 유지됨으로 함수가 종료되거나 스코프를 벗어나면 자동으로 공간 해제가 이루어지는 정적 메모리 할당과 대조적이다. 동적으로 획득한 메모리는 동적메모리 할당 해제 기능을 통해 원하는 시점에 메모리 획득을 취소 할 수도 있어 보다 효율적인 메모리 관리를 할 수 있게 해준다. 동적 메모리는 힙 영역에 할당된다.



1. 동적 메모리 할당의 장점과 단점

장점 : 동적 메모리 할당은 프로그래밍 당시 예측하기 어려운 자료의 크기를 프로그램의 실행 중 판단하여 메모리를 얻어올 수 있도록 해주며 메모리가 필요 없어질 경우 즉시 동적 메모리 할당 해제 작업을 통해 메모리를 반납 할 수 있어 효율적인 메모리 관리를 할 수 있도록 해준다.

단점 : 동적으로 할당한 메모리는 동적메모리 할당 해제를 해주지 않으면 메모리에 계속 남아 있게 됨으로 메모리의 누수가 발생 할 수 있다.

2. C언어의 동적 메모리 할당

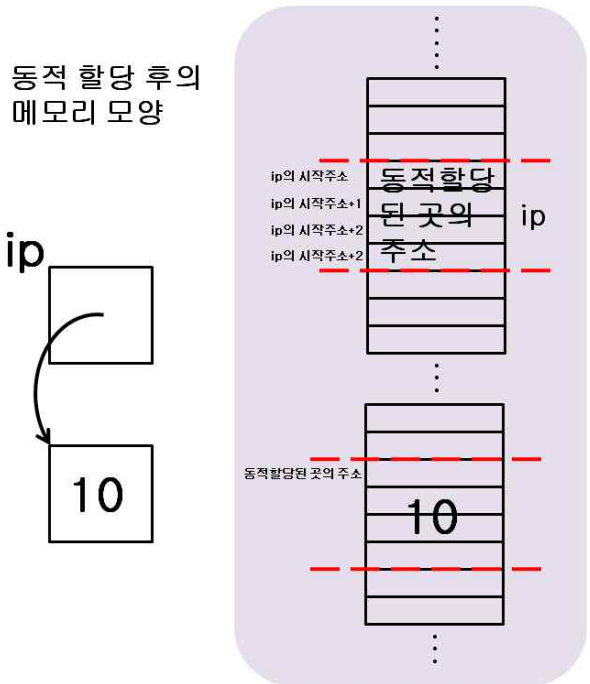
동적할당 방법 : 포인터 변수 = (포인터 변수의 자료형)malloc(동적할당 할 공간의 크기);

동적할당 해제 방법 : free(동적할당한 곳을 주소값을 가지는 포인터);

예제 2-1 : C언어 에서의 동적 메모리 할당

```
#include <stdio.h>

void main ( ) {
    int *ip;                        // 동적메모리 할당한 곳의 주소값을 보관할 포인터 변수
                                    // 를 만든다.
    ip = (int *)malloc(4);          // ip에 4바이트의 크기로 동적 메모리 할당을 해 할당한
                                    // 곳의 주소값을 넣는다.
    *ip = 10;                       // ip를 역참조해 동적할당한 공간에 10을 집어 넣는다.
    printf("*ip = %d₩n", *ip);      // ip를 역참조해 그곳에 있는 값을 출력한다.
    printf(" ip = %p₩n", ip);       // ip를 출력해 동적메모리 할당한 곳의 위치를 출력
                                    // 한다.
    free(ip);                       // 동적 메모리 할당을 해제한다.
}
```

예제 2-1 해설 : malloc함수는 자신의 뒤쪽 가로 안의 숫자 크기에 해당하는 동적으로 메모리를 할당받아 할당받은 곳의 주소값을 리턴 해준다. 단, 이 주소값은 자료형이 결정되지 않은 void *형임으로 역참조를 통해 값을 집어넣거나 꺼내올 수 없다 따라서 강제 형변환을 통해 주소값의 자료형을 변경 한 후 포인터에 대입해 사용하는 것이다.



3. C++의 동적 메모리 할당

동적할당 방법 : 포인터 변수 = new 동적할당할자료형;
포인터 변수 = new 동적할당할자료형[동적할당할 자료의 갯수];

동적할당 해제 방법 : delete 동적할당한 곳을 주소값을 가지는 포인터;
delete []동적할당한 곳을 주소값을 가지는 포인터;

예제 3-1 : C++ 에서의 동적 메모리 할당

```
#include <iostream.h>

void main ( ) {
    int *ip;                        // 동적메모리 할당한 곳의 주소값을 보관할 포인터 변수
                                    // 를 만든다.
    ip = new int;                   // int의 크기를 가지는 공간만큼 동적 메모리 할당을 해
                                    // 할당한 곳의 주소값을 ip에 넣는다.
    *ip = 10;                       // ip를 역참조해 동적할당 한 공간에 10을 집어 넣는다.
    cout << *ip << endl;            // ip를 역참조해 그곳에 있는 값을 출력한다.
    cout << ip << endl;             // ip를 출력해 동적메모리 할당한 곳의 위치를 출력
                                    // 한다.
    delete ip;                      // 동적 메모리 할당을 해제 한다.
}
```

예제 3-1 해설 : new 연산자는 자신의 뒤쪽에 있는 자료형의 크기만큼 동적으로 메모리를 할당받는다. 자료형을 정해 동적메모리 할당을 하기 때문에 C언어의 malloc 함수와 같이 강제 형변환을 할 필요가 없고 자료의 형태를 인지하기가 편하다. 동적 메모리 할당한 공간의 사용이 끝났다면 delete 연산자를 이용해 동적 메모리 할당 해제 작업을 한다.

예제 3-2 : 배열의 형태로 동적 메모리 할당을 하는 방법

```
#include <iostream.h>

void main ( ) {
    int *ip;                        // 동적메모리 할당한 곳의 주소값을 보관할 포인터 변수를
                                    // 만든다.
    ip = new int[4];                // [4]의 크기를 가지는 배열로 동적메모리 할당 하는 방법
    ip[0] = 10;                     // *ip = 10;
    ip[1] = 20;                     // *(ip+1) = 20;
    ip[2] = 30;                     // *(ip+2) = 30;
    ip[3] = 40;                     // *(ip+3) = 40;
    cout << ip[0] << ip[1] << ip[2] << ip[3] << endl;
                                    // cout << *ip << *(ip+1) << *(ip+2) << *(ip+3) << endl;
    delete ip;
}
```

예제 3-2 해설 : 동적 메모리 할당 시 동적메모리 할당을 하려는 자료형 뒤쪽에 [크기]를 쓰면 해당 사이즈만큼의 배열로 동적 메모리 할당을 할 수 있다.

예제 3-3 : 2차원 배열의 형태로 동적 메모리 할당을 하는 방법

```
#include <iostream.h>

void main ( ) {
    int (*ip)[3];                   // 동적메모리 할당한 곳의 주소값을 보관할 포인터 변수를
                                    // 만든다.
    ip = new int[2][3];             // [2][3]의 크기를 가지는 배열로 동적메모리 할당 하는
                                    // 방법
    ip[0][0] = 10;                  // *(*(ip+0)+0) = 10;
    ip[0][1] = 20;                  // *(*(ip+0)+1) = 20;
    ip[0][2] = 30;                  // *(*(ip+0)+2) = 30;
    ip[1][0] = 40;                  // *(*(ip+1)+0) = 40;
    ip[1][1] = 50;                  // *(*(ip+1)+1) = 50;
    ip[1][2] = 60;                  // *(*(ip+1)+2) = 60;

    cout << ip[0][0] << ip[0][1] << ip[0][2] << endl;
    cout << ip[1][0] << ip[1][1] << ip[1][2] << endl;

    delete []ip;
}
```

예제 3-3 해설 : 동적 메모리 할당 시 동적메모리 할당을 하려는 자료형 뒤쪽에 인덱스를 두 개 사용하면 이차원 배열만큼 동적 메모리 할당을 할 수 있다.

예제 3-4 : 2차원 배열과 같은 방식으로 사용 할 수 있도록 동적메모리 할당을 하는 방법

```
#include <iostream.h>

void main ( ) {
    int **dp;
    dp = new int*[2];               // 중첩포인터에 int *를 두 개 만큼 동적 할당 한다.
    dp[0] = new int[3];             // 각 포인터에 3칸짜리 int형 배열을 동적 할당 한다.
    dp[1] = new int[3];
    dp[0][0] = 10;                  // *(*(dp+0)+0) = 10;
    dp[0][1] = 20;                  // *(*(dp+0)+1) = 20;
    dp[0][2] = 30;                  // *(*(dp+0)+2) = 30;
    dp[1][0] = 40;                  // *(*(dp+1)+0) = 40;
    dp[1][1] = 50;                  // *(*(dp+1)+1) = 50;
    dp[1][2] = 60;                  // *(*(dp+1)+2) = 60;
    cout << dp[0][0] << dp[0][1] << dp[0][2] << endl;
    cout << dp[1][0] << dp[1][1] << dp[1][2] << endl;
    delete dp[0];                   // 각 1차원 배열을 동적 할당 해제 한다.
    delete dp[1];
    delete dp;                      // 포인터를 동적 할당 해제 한다.
}
```

예제 3-4 해설 : 위의 예제는 포인터 두 개를 동적 메모리 할당으로 만든 후 만들어진 포인터에 1차원 배열로 동적 할당을 한 예제이다. 실제 만들어 지는 것은 이차원 배열이 아니지만 이차원 배열과 같은 형태로 호출이 가능하다.

예제 3-5 : 입력받은 크기만큼 동적 메모리 할당을 하는 방법

```
#include <iostream.h>

void main ( ) {
    int *ip;                          // 동적메모리 할당한 곳의 주소값을 보관할 포인터 변수를
                                      // 만든다.
    int x;
    cout << "동적 메모리 할당 할 크기를 입력 하세요 : " ;
    cin >> x;                         // 동적 할당하려는 크기를 입력받는다.
    ip = new int[x];                  // 사용자가 입력한 크기만큼 동적메모리 할당이 된다.
    // 하고 싶은 동작을 한다.
    delete ip;                        // 동적메모리 할당을 해제한다.
}
```

예제 3-5 해설 : 동적 메모리 할당은 프로그램의 실행 중 일어남으로 사용자에게 크기를 입력받아 입력받은 크기만큼 메모리를 얻어오는 것도 가능하다.

예제 3-6 : 동적 메모리 할당을 이용한 배열의 리턴

```
#include <iostream.h>

int *func( ) {
       int arr[4] = {10,20,30,40};     // 함수내에서 만든 배열
       return arr;                     // 배열의 시작주소를 리턴
}
int *func2( ) {
       int *arr = new int[4];          // 함수 내에서 동적메모리 할당으로 만든 배열
       arr[0] = 10;
       arr[1] = 20;
       arr[2] = 30;
       arr[3] = 40;
       return arr;                     // 동적 할당 한 곳의 주소값을 리턴
}
void main ( ) {
       int *ip;
       ip = func();            // 함수가 종료되면 배열도 사라짐으로 배열은 리턴 할 수 없다.
       cout << ip[0] << endl;
       cout << ip[1] << endl;
       cout << ip[2] << endl;
       cout << ip[3] << endl;
       ip = func2();           // 동적메모리 할당으로 만든 배열은 함수가 끝나도 사라지지
                               // 않기 때문에 함수내에서 만든 배열도 리턴 받아 사용할 수 있다.
       cout << ip[0] << endl;
       cout << ip[1] << endl;
       cout << ip[2] << endl;
       cout << ip[3] << endl;
       delete ip;              // ip가 동적할당 한 메모리를 가리키고 있음으로 ip가 가리키는
                               // 곳의 동적메모리 할당해제 작업을 한다.
}
```

예제 3-6 해설 : 함수내에서 만든 배열은 함수가 종료되면 사라짐으로 리턴을 할 수 없다. (배열을 리턴할 경우 배열의 있던 곳의 주소만 리턴 되는 것이지 배열안의 data가 리턴 되는 것이 아니다.) 하지만 배열을 동적 메모리 할당으로 만들었다면 함수가 종료되어도 동적 메모리 할당 한 공간은 사지지지 않음으로 배열을 리턴 할 수 있게 되는 것이다. 또한 함수 종료 후에 동적 할당한 공간의 주소는 main 함수에 있는 ip가 알고 있음으로 main함수에서 동적 메모리 할당 해제를 하는 것도 가능하다.

예제 3-7 : 동적 메모리 할당의 활용 예제

```
#include <iostream.h>
#include <string.h>

char * input() {
        char inp[100];                  // 이름을 입력받아 저장할 배열을 넉넉한 크기로 만든다.
        cout << "이름을 입력해 주세요 : ";
        cin >> inp;                     // 이름을 입력받는다.
        char *cp;
        cp = new char[strlen(inp)+1];   // 입력받은 이름의 길이보다 1칸 더 크게 cp에
                                        // 동적 메모리 할당을 한다.
        strcpy( cp, inp );              // 입력받은 이름을 cp에 복사한다.
        return cp;                      // cp를 리턴 한다.
}

void main ( ) {
        char *name;
        name = input();                 // input함수에서 리턴받은 값을 name에 저장한다.
        cout << "당신의 이름은 " << name << "입니다." << endl;
        delete name;
}
```

예제 3-7 해설 : 길이가 얼마일지 모르는 이름을 입력받아 저장하기 위해 input() 함수를 사용하였다. input 함수는 넉넉하게 큰 배열 inp를 만들어 입력 받은 이름을 보관 한 뒤 입력받은 이름을 보관할 만큼만 동적 메모리 할당 작업을 해 입력받은 문자열을 복사해 넣고 동적 할당 한 곳의 주소값을 리턴 한다. input함수는 종료 되면서 함수내에서 선언한 배열과 변수가 모두 사라지지만 동적 메모리 할당한 공간은 사라지지 않는다. 따라서 main함수로 복귀한 뒤에는 입력받은 이름을 보관한 동적 메모리 할당 공간만 남게 된다.