

# C++ 프로그래밍

## □ C++ 기초

### ○ 프로그래밍 언어

- \* 기계어 : 0, 1 로 이루어진 언어
- \* 어셈블리어 : 기계어의 명령을 ADD, SUB, MOVE 등과 같이 상징적인 니모닉 기호로 일대일 대응시킨 언어
- \* 고급언어 : 사람이 이해하기 쉬운 언어

### ○ C++에 추가된 기능

- \* 함수 중복 (function overloading)
  - 매개 변수의 개수나 타입이 다른 동일한 이름의 함수들 선언
- \* 디폴트 매개 변수 (default parameter)
  - 매개 변수에 디폴트 값이 전달되도록 함수 선언
- \* 참조와 참조 변수 (reference)
  - 하나의 변수에 별명을 사용하는 참조 변수 도입
- \* 참조에 의한 호출 (call-by-reference)
  - 함수 호출 시 참조 전달
- \* new / delete 연산자
  - 동적 메모리 할당/해제를 위해 new와 delete 연산자 도입
- \* 인라인 함수
  - 함수 호출 대신 함수 코드의 확장 삽입
  - C언어가 자바보다 빠르다.
- \* 연산자 재정의 (overriding)
  - 기존 C++ 연산자에 새로운 연산 정의
- \* 제너릭 함수와 클래스 (일반화 프로그래밍)
  - 데이터 타입에 의존하지 않고 일반화시킨 함수나 클래스 작성 가능
  - 어떤 데이터 타입이든지 적용하면 사용할 수 있음

### ○ C++ 장단점

- \* 장점 : 기존에 개발된 C프로그램 코드 활용
- \* 단점 : 캡슐화의 원칙이 무너지짐
  - C++에서 전역 변수와 전역 함수를 사용할 수밖에 없음

### ○ 기본 구조

#### 소스 코드

```
#include<iostream> // .h가 없음

using namespace std; // 작성 안할 경우 std::cout 식으로 일일이 붙여줘야함

int main()
{
    cout << "야 시인난다 ~~\n";
    return 0;
}
```

- #include<iostream> // .h가 없음
- #include<string> // string 타입을 사용하기 위함
- #include<iomanip> // 입출력 조작자를 사용하기 위함

## ○ 입출력

- \* `cout << "글자" << "글자" << 5.5 << c << true << endl;` // 인자타입이 자동으로 설정된다. `endl`은 줄바꿈
- \* `cin >> 변수1 >> 변수2;` // 변수 2개를 입력받음
- \* `>>` 은 입력받은 값을 스트림(임시 저장)에 넘겨주는 것

소스 코드

```
#include <iostream>
using namespace std;
void main(){
    int width;
    int height;

    cout << "너비 입력 : ";
    cin >> width;
    cout << "높이 입력 : ";
    cin >> height;

    cout << "너비 : " << width << "높이 : " << height << "넓이 : " << width*height << endl;
}
```

## \* 입출력 조작자

- \* `cout.int width(int i);` : 최소 필드 너비를 조정 // 한 번 사용하면 사라진다, 마지막 숫자는 반올림된다.  
- 디폴트 값 : 6
- \* `cout.char fill(char c);` : 필드 내의 공백 자리에 채워질 문자 설정 // 한 번 설정하면 계속 남아있다.
- \* `cout.setf(ios::left);` : 어느 방향으로 정렬할 것인지 설정 // 한 번 설정하면 계속 남아있다.
- \* `cout.int precision(int p);` : 실수 출력 시 출력되는 총 자릿수, 출력 형식이 fixed 또는 scientific이라면 소수점 이하 자릿수 // 한 번 설정하면 계속 남아있다.

소스 코드

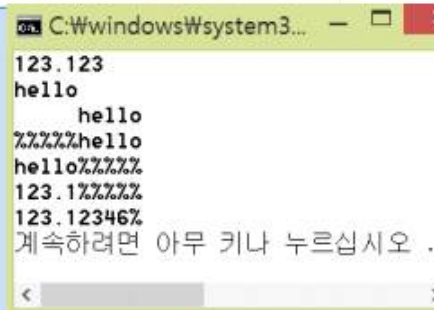
```
#include <iostream>
using namespace std;
int main(void)
{
    cout << 123.1234567 << endl;
    cout << "hello" << endl ;
    cout.width(10);cout << "hello" << endl;
    cout.fill('%');cout.width(10); cout << "hello" << endl ;
    cout.setf(ios::left); cout.width(10);cout << "hello" << endl ;
    cout.width(10);cout.precision(4);cout << 123.1234567 << endl;
    cout.width(10);cout.precision(8);cout << 123.1234567 << endl;
    return 0;
}
```

## \* 입출력 조작자 (전역 함수) // #include<iomanip>를 포함하고 cout 내부에 작성해야함

- \* `setw(int)` : 필드 너비 조정, 이후 한 번의 출력 후 디폴트로 환원됨
- \* `setfill(char)` : 공백 자리 채움 문자 지정
- \* `setprecision(int)` : 실수 출력 자릿수 설정
- \* `flush` : 스트림을 비움

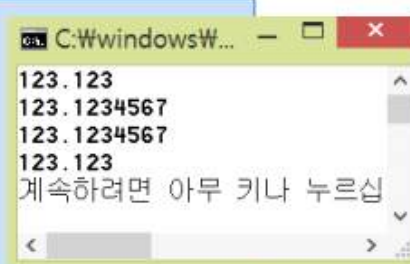
## 소스 코드

```
#include <iostream>
#include <iomanip>
using namespace std;
int main(void)
{
    cout << 123.1234567 << endl;
    cout << "hello" << endl;
    cout << setw(10) << "hello" << endl ;
    cout << setfill('%') << setw(10) << "hello" << endl ;
    cout << setw(10) << left << "hello" << endl ;
    cout << setw(10) << setprecision(4) << 123.1234567 << endl;
    cout << setw(10) << setprecision(8) << 123.1234567 << endl;
    return 0;
}
```



```
C:\Windows\system32\cmd.exe
123.123
hello
hello
%%%hello
hello%%%
123.1%%%
123.12346%
계속하려면 아무 키나 누르십시오 . . .
```

```
int main(void) {
    double dvalue = 123.1234567;
    int oldpre = cout.precision();
    cout << dvalue << endl;
    cout << setprecision( 10 ) << dvalue << endl ;
    cout << dvalue << endl;
    cout << setprecision(oldpre) << dvalue << endl;
    return 0;
}
```

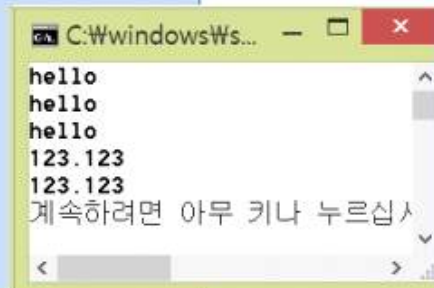


```
C:\Windows\system32\cmd.exe
123.123
123.1234567
123.1234567
123.123
계속하려면 아무 키나 누르십시오 . . .
```

```
int main(void)
{
    setw(10);cout << "hello" << endl;
    setfill('%');setw(10); cout << "hello" << endl ;

    cout.setf(ios::left); setw(10);cout << "hello" << endl ;
    setw(10);setprecision(4);cout << 123.1234567 << endl;
    setw(10);setprecision(3);cout << 123.1234567 << endl;

    return 0;
}
```



```
C:\Windows\system32\cmd.exe
hello
hello
hello
123.123
123.123
계속하려면 아무 키나 누르십시오 . . .
```

## 실습 예제

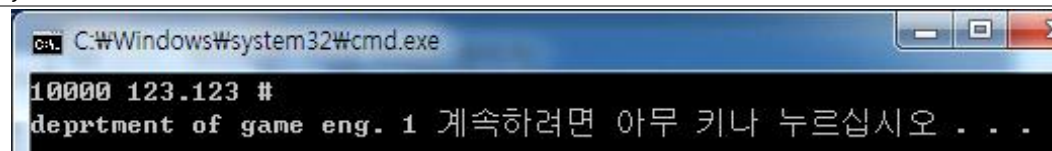
```
#include<iostream> // .h가 없음
#include<string> // string 타입을 사용하기 위함

using namespace std; // 작성 안할 경우 std::cout 식으로 일일이 붙여줘야함

int main()
{
    int inum = 10000;
    double dnum = 123.123456789;
    char nnum = '#';
    string str = "deptment of game eng.";
    bool flag = true;

    cout.setf(ios::left);cout.fill('#');
    cout << inum << " " << dnum << " " << nnum << " " <<endl;
    cout << str << " " << flag << " ";

    return 0;
}
```



```
C:\Windows\system32\cmd.exe
10000 123.123 #
deptment of game eng. 1
계속하려면 아무 키나 누르십시오 . . .
```

## ○ 문자열

\* C 스트링 방식 : -'\0'으로 끝나는 문자 배열

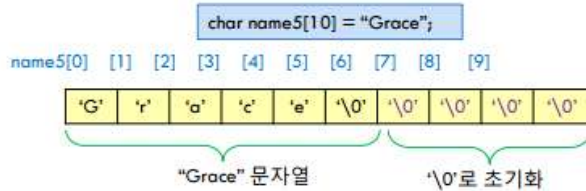
- 문자 하나하나를 넣을 때는 '\0'을 반드시 사용해야 문자열이 된다. (쓰지 않으면 단순 문자 배열)

● C++의 문자열 표현 방식 : 2가지

▶ C-스트링 방식 - '\0'로 끝나는 문자 배열

C-스트링 문자열  
단순 문자 배열

```
char name1[6] = {'G', 'r', 'a', 'c', 'e', '\0'}; // name1은 문자열 "Grace"  
char name2[5] = {'G', 'r', 'a', 'c', 'e'}; // name2는 문자열이 아니고 단순 문자 배열
```



### C-스트링 방식 실습 예제

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

void main(){
    char name[20];
    char password[20]="12510096 JKMeen";

    while(1){
        cout << "input password >> ";
        cin.getline(name, 30, '\n'); //\n을 할 때까지 문자열을 입력받는다.
        if (strcmp(name, password)){ //문자열을 비교, 참이면 0, 거짓이면 1
            cout << "< no match password! >" << endl << endl ;
        } else {
            cout << "< match password! >" << endl << endl;
            break;
        }
    }
}
```



## \* string 클래스 방식

- #include <string> 헤더 파일에 선언
- strcmp() : 스트링 값이 같으면 0 다르면 1을 리턴
- strlen() : 스트링 길이
- strcpy() : 스트링 복사
- 다양한 멤버 함수 제공, 문자열 비교, 복사, 수정 등

\* cin.getline(charbuf[], int size, char delimiter) : 공백이 낀 문자열을 입력 받는 방법

- delimiter : 문자열을 끝낼 명령을 입력 // ex) \n

\* getline(cin, singer); : string 타입의 문자열을 입력받기 위해 제공되는 전역 함수

\* getchar(); : 버퍼를 비워줌

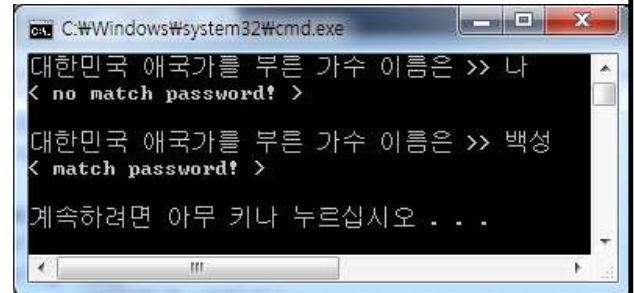
\* cin.ignore(INT\_MAX, '\n'); : cin에 \n을 했을 경우 버퍼를 비워줌

### string 실습 예제

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

void main(){
    string song, singer;
    string name;
    song = "대한민국 애국가";
    singer = "백성";

    while(1){
        cout << song <<"를 부른 가수
이름은 >> ";
        // cin >> name; // string을
        두 가지 방법으로 입력받을 수 있다.
        getline(cin, name);
        if (singer == name){
            cout << "< match
password! >" << endl << endl;
            break;
        } else {
            cout << "< no match
password! >" << endl << endl;
        }
    }
}
```



### string 실습 예제2

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

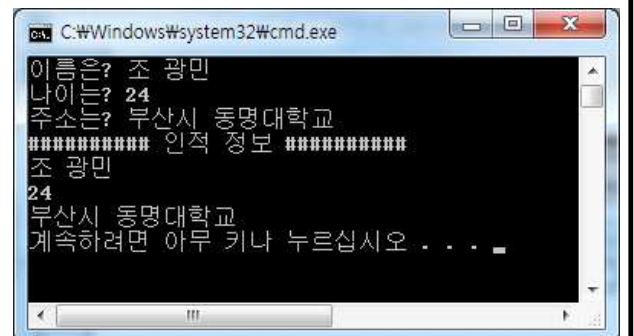
void main(){
    string name, year, home;

    cout << "이름은? ";
    getline(cin, name);

    cout << "나이는? ";
    getline(cin, year);

    cout << "주소는? ";
    getline(cin, home);

    cout << "##### 인적 정보
#####" << endl;
    cout << name << endl << year << endl
<< home << endl;
}
```





## 곱셈 테이블 실습 예제

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

void main(){

cout << "          곱셈 테이블          " <<
endl << "-----" <<
endl;
for (int i=1; i<=12; i++){
    for (int j=1; j<=10; j++){
        cout.width(3); cout << i*j;
    }
    cout << endl;
}

cout << "-----" <<
endl;
}
```

- \* `isalpha` : 알파벳인지 체크
- \* `tolower` : 대문자를 소문자로 바꿈
- \* `strlen(변수)` : 배열 변수의 길이를 구함

## 영어 히스토그램 실습 예제

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

void main(){
    char get[10000];
    int world[26]={0};

    cout << "입력" << endl;
    cin.getline(get, 10000, ':');

    int length = strlen(get);

    for (int i=0; i<length; i++){
        if (isalpha(get[i])){
            char ch =
tolower(get[i]);
            world[ch-'a']++;
        }
    }
    int total = 0;
    for(int i=0; i<26; i++){
        total += world[i];
    }
    cout << "총 알파벳 개수는 : " << total
<< endl;

    for(int i=0; i<26; i++){
        cout << (char)('a'+i) << ": ( "
<< world[i] << " ) : ";
        for(int j=0; j<world[i]; j++){
            cout << "*";
        }
        cout << endl;
    }
}
```

## □ 클래스와 객체

### ○ C++클래스 만들기

\* **클래스** : 객체를 만드는 설계도

\* **캡슐화** : 객체의 본질적인 특성, 객체를 캡슐로 싸서 그 내부를 보호하고 볼 수 없게 함  
- private

\* **인터페이스** : 외부에 객체의 일부분을 공개하는 것  
- Tv-리모컨, 사람-눈, 코, 입, 귀, 피부  
- public

\* **클래스 작성**

\* C++ 객체는 멤버 함수(행동)와 멤버 변수(상태)로 구성

\* 클래스 선언부와 클래스 구현부로 구성

\* **클래스 선언부**

\* class 키워드를 이용하여 클래스 선언

\* 멤버 변수와 멤버 함수 선언

- 멤버 변수는 클래스 선언 내에서 초기화할 수 없음
- 멤버 함수는 원형 형태로 선언

\* 멤버에 대한 접근 권한 지정

- private, public, protected 중 하나
- 디폴트는 private
- public : 다른 모든 클래스나 객체에서 멤버의 접근이 가능함을 표시

\* **클래스 구현**

\* 클래스에 정의된 모든 멤버 함수 구현

#### 헤더파일, 클래스의 활용 예제 소스 코드

```
// Circle.h //  
class Circle{  
public:  
    int radius; // 멤버 변수  
  
public:  
    double getArea(); // 멤버 함수  
};
```

```
// Circle.cpp //  
#include "Circle.h"  
  
double Circle::getArea(){  
    return 3.14* this->radius*this->radius;  
}
```

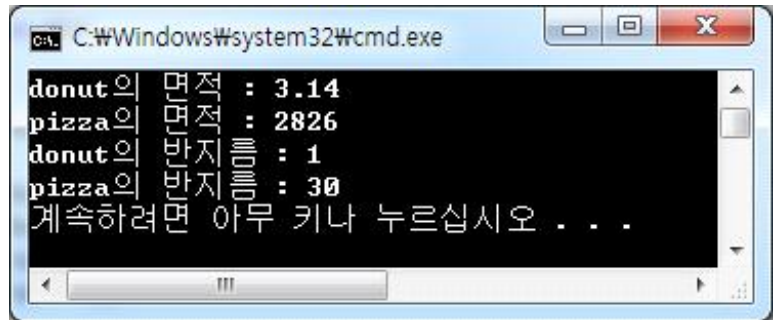
```
// circle_main.cpp //  
#include "Circle.h"  
#include <iostream>  
using namespace std;  
  
void main(){  
    Circle donut, pizza;  
  
    donut.radius = 1;  
    pizza.radius = 30;  
  
    double area = pizza.getArea();  
  
    cout << "donut의 면적 : " << donut.getArea() << endl << "pizza의 면적 : " << area << endl;  
}
```

## ○ 객체생성과 활용

- \* **헤더파일 선언유무 확인** : Circle파일을 여러 번 반복해서 포함시키더라도 if 문 때문에 바로 endif
  - \* **#ifndef** CIRCLE\_H : 만약 헤더파일이 선언되어있지 않으면,
  - \* **#define** CIRCLE\_H : 헤더파일을 포함시켜준다.
  - \* **#endif** : 만약 선언되어 있으면 if를 끝낸다.
- \* **setRadius, getRadius 함수를 선언하여 이용한 방법**

### 헤더파일, 클래스의 활용 예제 소스 코드

```
// Circle.h //  
#ifndef CIRCLE_H  
#define CIRCLE_H  
  
class Circle{  
public:  
    int radius;  
  
public:  
    double getArea();  
    void setRadius( int r );  
    int getRadius();  
};  
  
#endif
```



```
// Circle.cpp //  
#include "Circle.h"  
  
double Circle::getArea(){  
    return 3.14* this->radius*this->radius;  
}  
  
void Circle::setRadius(int r){  
    this->radius = r;  
}  
  
int Circle::getRadius(){  
    return radius;  
}
```

```
// circle_main.cpp //  
#include "Circle.h"  
#include <iostream>  
using namespace std;  
  
void main(){  
    Circle donut, pizza;  
  
    donut.setRadius(1);  
    pizza.setRadius(30);  
    donut.radius = 1;  
    pizza.radius = 30;  
  
    double area = pizza.getArea();  
  
    cout << "donut의 면적 : " << donut.getArea() << endl << "pizza의 면적 : " << area << endl;  
    cout << "donut의 반지름 : " << donut.getRadius() << endl << "pizza의 반지름 : " << pizza.getRadius() << endl;  
}
```



○ 생성자 (constructor) : 자바의 클래스 함수와 같음

- \* 객체가 생성되는 시점에서 자동으로 호출되는 멤버 함수
- \* 클래스 이름과 동일한 멤버 함수
- \* 생성자의 목적 : 객체가 생성될 때 객체가 필요한 초기화를 위해
  - 멤버 변수 값 초기화, 메모리 할당, 파일 열기, 네트워크 연결 등
- \* 객체 생성 시 오직 한 번만 호출(자동으로 호출됨)
- \* 생성자는 중복 가능
  - 한 클래스 내에 여러개 가능
  - 중복된 생성자 중 하나만 실행
- \* 생성자가 선언되어 있지 않으면 기본 생성자 자동으로 생성
  - 기본 생성자 - 매개 변수가 없는 생성자
  - 컴파일러에 의해 자동 생성

생성자 예제1 소스 코드

```
// Circle.h //
#ifndef CIRCLE_H
#define CIRCLE_H
#include <iostream>
using namespace std;

class Circle{
private:
    double radius;
public:
    Circle();
    Circle(double);
    double getArea();
    void setArea( double r );
};
```

#endif

```
// Circle.cpp //
#include "Circle.h"

Circle::Circle(){
    this->radius = 1;
    cout << "기본생성자 호출\n";
}

Circle::Circle(double r){
    this->radius = r;
    cout << "생성자 호출, 반지름 : " << this->radius << endl;
}

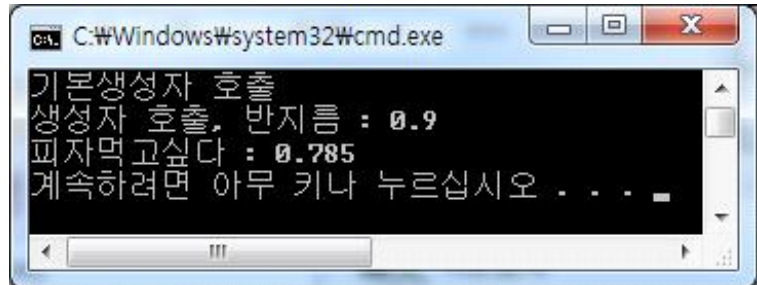
double Circle::getArea(){
    return 3.14*this->radius*this->radius;
}

void Circle::setArea(double r){
    this->radius = r;
}
```

```
// circle_main.cpp //
#include "Circle.h"

using namespace std;

void main(){
    Circle pizza;
    pizza.setArea(0.5);
    Circle(0.9);
    cout << "피자먹고싶다 : " << pizza.getArea() << endl;
}
```



## 생성자 예제2 소스 코드

```
// Rect_main.cpp //
#include "Rect.h"

void main(){
    Rect();
    Rect(3);
    Rect(3, 5);
}

// Rect.cpp //
#include "Rect.h"

Rect::Rect(){
    this->width = 1;
    this->height = 1;
    cout << "Default 생성자, " << this->width << ", " << this->height << endl;
}

Rect::Rect(int w){
    this->width = w;
    this->height = w;
    cout << "매개변수1개, " << this->width << ", " << this->height << endl;
}

Rect::Rect(int w, int h){
    this->width = w;
    this->height = h;
    cout << "매개변수2개, " << this->width << ", " << this->height << endl;
}

int Rect::getArea(){
    return this->width * this->height;
}

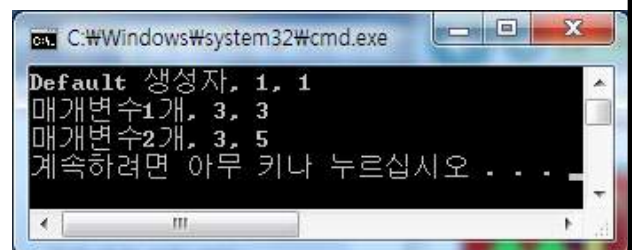
void Rect::setWidth(int w){
    this->width = w;
}

void Rect::setHeight(int h){
    this->height = h;
}
```

```
// Rect.h //
#ifndef RECT_H
#define RECT_H
#include <iostream>

using namespace std;
class Rect{
private:
    int width, height;
public:
    Rect();
    Rect(int w, int h);
    Rect(int w);
    int getArea();
    void setWidth(int w);
    void setHeight(int h);
};

#endif
```



○ **소멸자** : 객체가 소멸되는 시점에서 자동으로 호출되는 함수

\* 생성자가 호출한 순서의 역순으로 소멸자가 호출된다.



\* 오직 한 번만 자동 호출, 임의로 호출할 수 없음

\* 객체 메모리 소멸 직전 호출됨

\* **소멸자 함수의 이름은 클래스 이름 앞에 ~를 붙인다**

- ex) `Circle::~~Circle(){...}`

\* 중복 불가능

\* 객체가 선언된 위치에 따른 분류

- **지역 객체** : 함수 내에 선언된 객체, 함수가 종료하면 소멸

- **전역 객체** : 함수의 바깥에 선언된 객체로서, 프로그램이 종료할 때 소멸

○ **접근지정**

\* **public** : 공개

\* **private** : 비공개

\* **protect** : 일부에게만 공개

○ **인라인 함수(Inline)** : **inline** 키워드로 선언된 함수

\* 소스코드를 따로 안 만들고 헤더파일에서 구현을 안 하고, inline함수로 만들 경우, **메인 함수가 있는 파일에서 정의해야한다.**

\* 인라인 함수를 호출하는 곳에 인라인 함수 코드를 확장 삽입

- 매크로와 유사, 코드 확장 후 인라인 함수는 사라짐

\* **인라인 함수 호출**

- 함수 호출에 따른 오버헤드 존재하지 않음

- 프로그램의 실행 속도 개선

\* 컴파일러에 의해 이루어짐

\* **C++프로그램의 실행 속도 향상**

- 자주 호출되는 짧은 코드의 함수 호출에 대한 시간 소모를 줄임

- C++에는 짧은 코드의 멤버 함수가 많기 때문

\* **장점** : 프로그램의 실행 시간이 빨라진다

\* **단점** : 인라인 함수 코드의 삽입으로 컴파일된 전체 코드 크기 증가

과제 //////////////

연습문제 1 : 랜덤 클래스를 활용하여, 랜덤한 정수 10개 출력

랜덤 클래스의 생성자, `next()`, `nextIntRange()`의 3개의 멤버 함수를 가지도록 작성

랜덤 수의 범위 : 0~32767까지

## 생성자 + 소멸자 + 인라인 함수 예제1 소스 코드

```
#include "Calculator.h"
```

```
void main(){
    Calculator cal;
    cal.run();
}
```

```
#include "Calculator.h"
```

```
#include "Adder.h"
```

```
#include "Sub.h"
```

```
#include "Mul.h"
```

```
#include "Div.h"
```

```
void Calculator::run(){
    cout<<"숫자 두개 입력 >>";
    int a,b;
    cin>>a>>b;
    Adder adder(a, b);
    Sub sub(a, b);
    Mul mul(a, b);
    Div div(a, b);
    cout << "덧셈 결과 : "<<adder.processor()<<endl;
    cout << "나눗셈 결과 : "<<div.processor()<<endl;
}
```

```
#ifndef CALCULATOR_H
```

```
#define CALCULATOR_H
```

```
#include <iostream>
```

```
using namespace std;
```

```
class Calculator{
public:
    void run();
};
```

```
#endif
```

```
#ifndef ADDER_H
```

```
#define ADDER_H
```

```
class Adder{
    int a,b;
public:
    Adder(int aa, int bb){
        this->a = aa; this->b = bb;
    }
    int processor(){
        return this->a + this->b;
    }
};
```

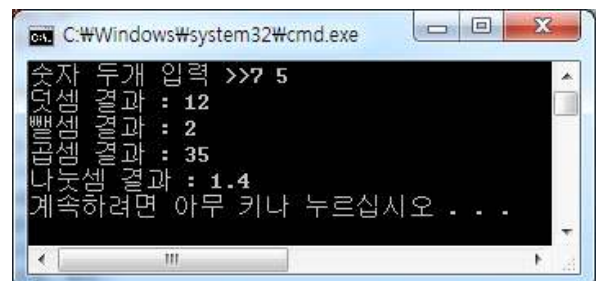
```
#endif
```

```
#ifndef DIV_H
```

```
#define DIV_H
```

```
class Div{
    double a,b;
public:
    Div(int aa, int bb){
        this->a = aa; this->b = bb;
    }
    double processor(){
        return this->a / this->b;
    }
};
```

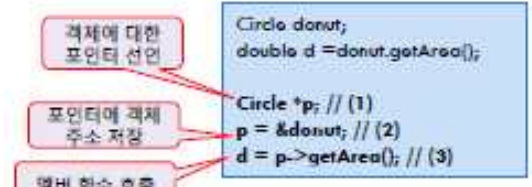
```
#endif
```



## □ 객체 포인터와 객체 배열, 객체의 동적 생성

### ○ 객체 포인터

- \* **동적 메모리 할당** : 동적으로 할당된 객체에 접근할 때 사용(힙 영역으로부터 할당)
- \* **함수에서 객체 배열을 인수로 사용** : 함수의 인수로 객체 배열을 넘길 때 사용
- \* **객체에 대한 포인터**
  - C언어의 포인터와 동일
  - 객체의 주소 값을 가지는 변수
- \* **포인터 멤버를 접근할 때** : 객체 포인터 -> 멤버



// Circle \*p처럼 포인터 변수 앞에는 변수타입이 붙는다. 그 주소를 따라가면 해당 변수타입이 있어야 한다.

- \* **스택 영역** : 함수들마다 별도로 사용할 수 있는 영역, 정해진 것들만 사용  
/ 프로그램이 수행(메인함수가 수행)되면 불러 졌다가 끝나면 사라짐
- \* **힙 영역** : 프로그램들이 공동으로 사용하는 영역 (운영체제가 관리)

#### 객체 포인터 예제1 소스 코드

```
#ifndef COLOR_H
#define COLOR_H
#include <iostream>
using namespace std;

class Color{
    int red, green, blue;

public:
    Color() { red = green = blue = 0; }
    Color(int r, int g, int b){
        red = r; green = g; blue = b;
    }

    void setColor(int r, int g, int b);
    void show();
};

#endif

#include "Color.h"

inline void Color::setColor( int r, int g, int b ){ // 인라인 함수
    red = r; green = g; blue = b;
}

inline void Color::show(){
    cout << "색상(red, green, blue) : " << red << " ";
    cout << green << " " << blue << endl;
}

void main(){
    Color screenColor(255, 0, 0); // 빨간색의 screenColor 객체 생성
    Color *p;
    p = &screenColor;
    /*두 가지의 형태는 같다*/
    p->show();
    (*p).show();

    Color colors[3];
    p = colors;
    p->setColor(255, 0, 0);
    p++;
    p->setColor(0, 255, 0);
    p = p+1;
    p->setColor(0, 0, 255);

    p = colors; // p[i]를 쓸 때 반드시 p를 초기값으로 바꾸고 실행해야 한다.
    for (int i=0; i<3; i++){
        /* 네가지의 경우가 다 같다 */
        (*p).show(); p++; // 첫 번째 방법
        //(p+i).show(); // 두 번째 방법
        p[i].show(); // 세 번째 방법
        colors[i].show(); // 네 번째 방법
    }
}
```

```
색상<red, green, blue> : 255 0 0
색상<red, green, blue> : 255 0 0
색상<red, green, blue> : 255 0 0
색상<red, green, blue> : 0 255 0
색상<red, green, blue> : 0 0 255
계속하려면 아무 키나 누르십시오 . . .
```

## ○ 객체 배열

\* 2차원 배열 : ex) Color colors[3][2] // 행(세로) : 3, 열(가로) : 2

\* 2차원 배열에서 포인터로 나타낼 경우

- 첫 번째 방법 : Color \*p = colors[0];
- 두 번째 방법 : Color \*p = &colors[0][0];

### 객체 포인터 예제1 소스 코드

```
#ifndef COLOR_H
#define COLOR_H
#include <iostream>
using namespace std;

class Color{
    int red, green, blue;

public:
    Color() { red = green = blue = 0; }
    Color(int r, int g, int b){
        red = r; green = g; blue = b;
    }

    void setColor(int r, int g, int b);

    void show();
};

#endif

#include "Color.h"

inline void Color::setColor( int r, int g, int b ){
    red = r; green = g; blue = b;
}

inline void Color::show(){
    cout << "색상(red, green, blue) : " << red << " ";
    cout << green << " " << blue << endl;
}

void main(){
    Color colors[3][2]={Color(255, 0, 0), Color(0, 255, 0),
                        Color(0, 0, 255), Color(0, 255, 255),
                        Color(10, 10, 10), Color(20, 20, 20)};

    //Color *p = colors[0];
    Color *p = &colors[0][0];

    for (int i =0; i<3; i++){
        for (int j=0; j<2; j++){
            //(*p+i).show();
            colors[i][j].show();
        }
    }
}
```

```
색상<red, green, blue> : 255 0 0
색상<red, green, blue> : 255 0 0
색상<red, green, blue> : 255 0 0
색상<red, green, blue> : 0 255 0
색상<red, green, blue> : 0 0 255
계속하려면 아무 키나 누르십시오 . . .
```



## ○ 정적 할당

### \* 변수 선언을 통해 필요한 메모리 할당

- 스택(stack)으로부터 할당
- 많은 양의 메모리는 배열 선언을 통해 할당

## ○ 동적 메모리 할당 및 반환 (생성자 / 소멸자)

### \* 프로그램이 실행된 후에 필요한 메모리를 할당 받음 : 필요한 만큼 메모리를 할당

### \* 실행 중에 운영체제로부터 할당 받음

- 힙(heap)으로부터 할당
- 힙은 운영체제가 소유하고 관리하는 메모리로 모든 프로세스가 공유할 수 있는 메모리

### \* C++의 동적 메모리 할당 / 반환 : new 연산자, delete 연산자

#### \* new 연산자

- 기본 타입 메모리 할당, 배열 할당, 객체 할당, 객체 배열 할당
- 객체의 동적 생성 - 힙 메모리로부터 객체를 위한 메모리 할당 요청
- 객체 할당 시 생성자 호출

#### \* delete 연산자

- new로 할당 받은 메모리 반환
- 객체의 동적 소멸 - 소멸자 호출 뒤 객체를 힙에 반환
- 배열일 경우 `delete [] p;` 형식으로 소멸

### \* new / delete 연산자의 사용 형식

- 반드시 포인터 타입으로 선언

```
데이터타입 *포인터변수(스택영역에 저장) = new 데이터타입(힙 영역에 저장);  
delete 포인터변수;
```

### \* delete 사용 시 주의사항

- 동적으로 할당 받지 않는 메모리 반환 -> 오류

```
int n;  
int *p = &n;  
delete p; // 실행 시간 오류  
// 포인터 p가 가리키는 메모리는 동적으로 할당 받은 것이 아님
```

- 동일한 메모리 두 번 반환 -> 오류

```
int *p = new int;  
delete p; // 정상적인 메모리 반환  
delete p; // 실행 시간 오류. 이미 반환한 메모리를 중복 반환할 수 없음
```

### \* 객체의 동적 생성 및 반환

- \* default 함수는 ()를 생략해줘도 된다. ex) `Circle *p Circle(); => Circle *p Circle;`

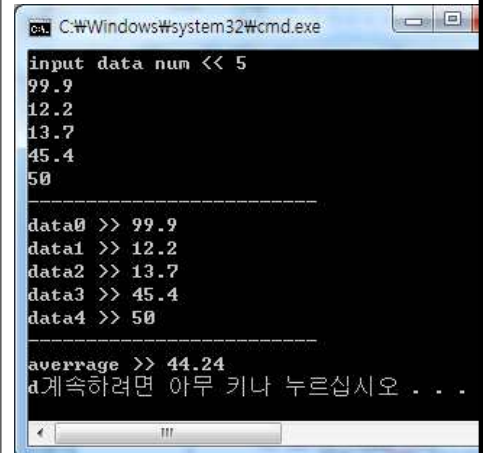
```
클래스이름 *포인터변수 = new 클래스이름;  
클래스이름 *포인터변수 = new 클래스이름(생성자매개변수리스트);  
delete 포인터변수;
```

### \* 메모리 누수

- 프로그램이 종료되면, 운영체제는 누수 메모리를 모두 힙에 반환함
- ex) `char *p = new char[1024]; p=&n;` 을 하면 이전의 \*p주소는 사라져서 누수가 발생함.

## 생성자, 소멸자 예제 소스 코드

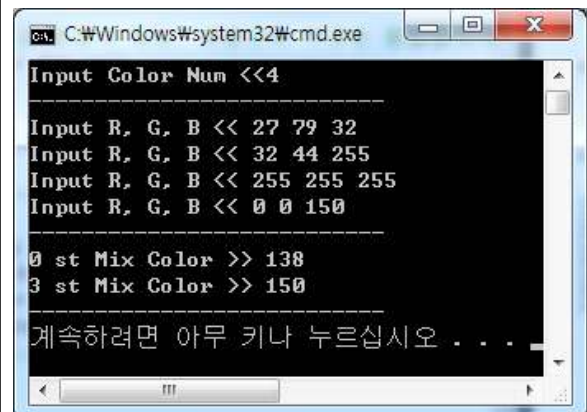
```
#include <iostream>
using namespace std;
void main(){
    int num;
    double *pdata, sum=0;
    cout << "input data num << ";
    cin >> num;
    pdata = new double[num];
    for (int i=0; i<num; i++){
        cin >> pdata[i];
        sum += pdata[i];
    }
    cout << "-----" << endl;
    for (int i=0; i<num; i++){
        cout << "data" << i << " >> " << (*pdata+i) << endl;
        /*cout << *pdata << " ";
        pdata++;*/ // 마지막 다음주소를 가리키기 때문에
        delete [] pdata; //에서 오류가 남
    }
    cout << "-----" << endl;
    cout << "averrage >> " << sum/num << endl;
    cout << "-----" << endl;
    delete [] pdata;
}
```



```
C:\Windows\system32\cmd.exe
input data num << 5
99.9
12.2
13.7
45.4
50
-----
data0 >> 99.9
data1 >> 12.2
data2 >> 13.7
data3 >> 45.4
data4 >> 50
-----
averrage >> 44.24
계속하려면 아무 키나 누르십시오 . . .
```

## 생성자, 소멸자 예제 소스 코드2

```
#ifndef COLOR_H
#define COLOR_H
#include <iostream>
using namespace std;
class Color{
    int red, green, blue;
public:
    Color() { red = green = blue = 0; }
    Color(int r, int g, int b){
        red = r; green = g; blue = b;
    }
    void setColor(int r, int g, int b);
    void show();
    int getRed() { return red; }
    int getGreen() { return green; }
    int getBlue() { return blue; }
    int getMix() { return red + green + blue; }
};
#endif
```



```
C:\Windows\system32\cmd.exe
Input Color Num <<4
-----
Input R, G, B << 27 79 32
Input R, G, B << 32 44 255
Input R, G, B << 255 255 255
Input R, G, B << 0 0 150
-----
0 st Mix Color >> 138
3 st Mix Color >> 150
계속하려면 아무 키나 누르십시오 . . .
```

```
#include "Color.h"
inline void Color::setColor( int r, int g, int b ){
    red = r; green = g; blue = b;
}
inline void Color::show(){
    cout << "색상(red, green, blue) : " << red << " ";
    cout << green << " " << blue << endl;
}
void main(){
    Color *colors;
    int r, g, b;
    int num; // 칼라 갯수 입력받고 배열에 넣음, 배열안에 저장된 혼합칼라색이 100~200인것만 출력
    cout << "Input Color Num <<";
    cin >> num;
    colors = new Color[num];
    cout << "-----" << endl;
    for (int i=0; i<num; i++){
        cout << "Input R, G, B << ";
        cin >> r >> g >> b;
        colors[i].setColor(r, g, b); // 배열 함수불러올때 . 연산자
        //(*colors+i).setColor(r, g, b);
    }
    cout << "-----" << endl;
    for (int i=0; i<num; i++){
        if (colors[i].getMix() >= 100 && colors[i].getMix() <= 200) {
            cout << i << " st Mix Color >> " << colors[i].getMix() << endl;
        }
    }
    cout << "-----" << endl;
    delete [] colors;
}
```

## ○ 객체와 객체 배열의 동적 생성 및 반환

### \* 동적으로 생성된 배열도 보통 배열처럼 사용

- ex) `pArray[0].setRadius[10];` // 배열의 첫 번째 객체의 `setRadius()` 멤버 함수 호출

### \* 포인터로 배열 접근

- ex) `pArray->setRadius[10];`

### \* 배열 소멸

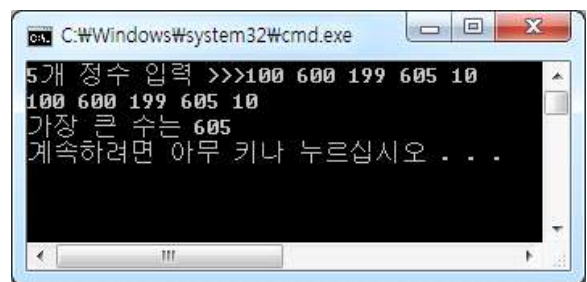
- ex) `delete [] pArray;`

## 연습문제 1 - Sample 클래스

```
#ifndef SAMPLE_H
#define SAMPLE_H
#include <iostream>

using namespace std;

class Sample{
    int *p;
    int size;
public:
    Sample(int n){ // 생성자
        size = n;
        p=new int [n]; // n개 정수 배열의 동적 생성
    }
    void read(); // 100 600 199 605 10
    void write();
    int big();
    ~Sample();
    /*
    *Sample(){
        delete[] p;
    }*/
};
#endif
```



```
#include "Sample.h"

void Sample::read(){
    cout << this->size<<"개 정수 입력 >>>";
    for (int i=0; i<this->size; i++){
        cin >> this->p[i];
    }
} // 100 600 199 605 10

void Sample::write(){
    for (int i=0; i<this->size; i++){
        cout << p[i] << " ";
    }
    cout << endl;
}

int Sample::big(){
    int bigsize = p[0];
    for (int i=0; i<this->size; i++){
        if (bigsize <= p[i]){
            bigsize = p[i];
        }
    }
    return bigsize;
}

Sample::~Sample(){
    delete [] p;
}

#include "Sample.h"

void main(){
    Sample s(5); // 10개 정수 배열을 가진 Sample 객체 생성
    s.read(); // 키보드에서 정수 배열 읽기
    s.write(); // 정수 배열 출력
    cout << "가장 큰 수는 " << s.big() << endl; // 가장 큰 수 출력
}
```

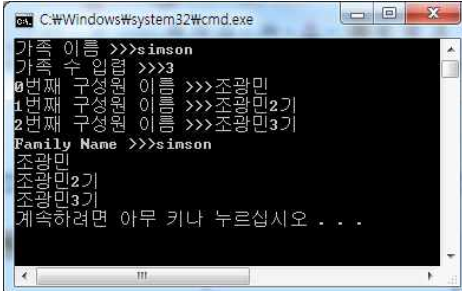
### 연습문제 3 - Family 클래스

```
#ifndef FAMILY_H
#define FAMILY_H
#include <iostream>
#include <string>
using namespace std;

class Person{
    string name;
public:
    Person(){name="";}
    Person(string name){this->name=name;}
    string getName(){ return name; }
    void setName(string name){ this->name = name; }
};

class Family{
    string name;
    Person* p; // Person 배열 포인터
    int size; // Person 배열의 크기, 가족 구성원 수
public:
    Family(); // size 개수 만큼 Person 배열 동적 생성
    void setName(); // 순서/이름
    void show(); // 모든 가족 구성원 출력
    ~Family();
};

#endif
```



```
C:\Windows\system32\cmd.exe
가족 이름 >>>simson
가족 수 입력 >>>3
0번째 구성원 이름 >>>조광민
1번째 구성원 이름 >>>조광민2기
2번째 구성원 이름 >>>조광민3기
Family Name >>>simson
조광민
조광민2기
조광민3기
계속하려면 아무 키나 누르십시오 . . .
```

```
#include "Family.h"

Family::Family(){
    string name;
    int size;

    cout << "가족 이름 >>> ";
    cin >> name;
    this->name = name;
    cout << "가족 수 입력 >>>";
    cin >> size;
    this->size = size;
    p = new Person[size];
} // size 개수 만큼 Person 배열 동적 생성

void Family::setName(){
    string name;
    for (int i=0; i<this->size; i++){
        cout << i << "번째 구성원 이름 >>>";
        cin >> name;
        p[i].setName(name);
    }
} // 순서/이름

void Family::show(){
    cout << "Family Name >>> " << this->name << endl;
    for (int i=0; i<this->size; i++){
        cout << p[i].getName()<<endl;
    }
} // 모든 가족 구성원 출력

Family::~Family(){
    delete [] p;
}

#include "Family.h"

void main(){
    Family *simpson = new Family(); // 3명으로 구성된 Simpson 가족
    simpson->setName();
    simpson->show();
    delete simpson;
}
```

## ○ This 포인터

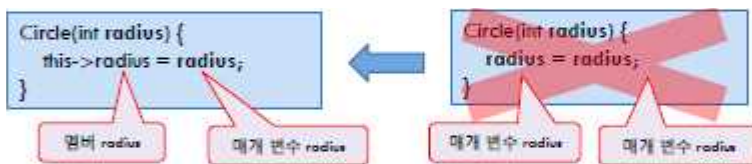
- \* 포인터, 객체 자신 포인터
- \* 클래스의 멤버 함수 내에서만 사용
- \* 개발자가 선언한 변수가 아닌, 컴파일러가 선언한 변수
  - 멤버 함수에 컴파일러에 의해 묵시적으로 삽입 선언되는 매개 변수

### 생성자, 소멸자 예제 소스 코드

```
class Circle {  
    int radius;  
public:  
    Circle() { this->radius=1; }  
    Circle(int radius) { this->radius = radius; } // radius = radius로 하면 오류 발생  
    void setRadius(int radius) { this->radius = radius; }  
    ....  
};
```

### \* this가 필요한 경우

- 매개변수의 이름과 멤버 변수의 이름이 같은 경우



- 멤버 함수가 객체 자신의 주소를 리턴 할 때 // 연산자 중복 시에 매우 필요

```
class Sample {  
public:  
    Sample* f() {  
        ....  
        return this;  
    }  
};
```

### \* this를 사용할 수 없는 경우

- 멤버 함수가 아닌 함수에서 this 사용 불가 // 객체와의 관련성이 없기 때문
- Static 멤버 함수에서 this 사용 불가 // 객체가 생기기 전에 static 함수 호출이 있을 수 있기 때문  
// Static 변수 : 프로그램 생존기간이랑 같다. (콘솔창이 켜질 때 생성, 닫을 때 파괴)  
// Static 함수 : 객체를 생성하지 않고도 호출할 수 있다.

## ○ String 클래스

- \* <string>을 헤더 파일에 선언
- \* String 사용법 (4가지)

### 가변 크기의 문자열

```
#include <iostream>  
#include <string>  
using namespace std;  
  
void main(){  
    char msg[]="I love C++ class";  
    string str1;  
    string str2("I love our department"); // string 바로 작성  
    string str3(str2); // string을 string으로  
    string str4(msg); // char를 string으로  
}
```

\* 가변 크기의 문자열

가변 크기의 문자열
<pre>string str = "I love"; // I love (7개의 문자로 구성) string.append("C++"); // I love C++ 이 됨 (11개의 문자)</pre>

\* 문자열 생성

문자열 생성
<pre>string str; // 빈 문자열을 가진 스트링 객체 string address("서울시 성북구 삼선동 389"); // 문자열 리터럴로 초기화 string copyAddress(address); // address를 복사한 copyAddress 생성 // C-스트링(char [] 배열)으로부터 스트링 객체 생성 char text[] = {'L', 'o', 'v', 'e', ' ', 'C', '+', '+', '\0'}; string title(text); // "Love C++" 문자열을 가진 title 생성</pre>

\* String 객체의 동적 생성

\* new/delete를 이용하여 문자열 동적 생성/반환

문자열 동적 생성/반환
<pre>string *p = new string("C++"); // 스트링 객체 동적 생성 cout &lt;&lt; *p; // "C++" 출력 p-&gt;append(" Great!!"); // p가 가리키는 스트링이 "C++ Great!!"이 됨 cout &lt;&lt; *p; // "C++ Great!!" 출력 delete p; // 스트링 객체 반환</pre>

\* String 주요 멤버 함수

멤버 함수	설명
string& append(string& str)	문자열 뒤에 str 추가
string& insert(int pos, string& str)	문자열 pos위치에 str 삽입
string& replace(int pos, int n, string& str)	문자열의 pos 위치부터 n개 문자를 str문자로 바꿈
int size()	문자열의 길이 리턴, 바이트 수
int length()	문자열의 길이 리턴, size()와 동일
string& erase(int pos, int n)	pos위치부터 n개의 문자 삭제
void clear()	문자열 모두 삭제, 크기를 0으로 만듦
char& at(int pos)	pos 위치의 문자 리턴
int find(string& str)	문자열의 처음부터 str을 검색하여 발견한 처음 인덱스 리턴, 없으면 -1 리턴 (int로 형변환 해야함)
int compare(string& str)	문자열과 str을 비교하여 같으면 0을, 사전 순으로 현재 문자열이 앞에 오면 음수, 뒤에오면 양수 리턴
string substr(int pos, int n)	pos 위치부터 n개 문자를 새로운 서브스트링으로 생성, 리턴
void swap(string& str1, string str2)	str1과 str2를 서로 교환



## string 주요 멤버 함수 예제

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

void main(){
    char msg[]="I love C++ class";
    string str1;
    string str2("I love our department");
    string str3(str2);
    string str4(msg);

    str1.append("C++");
    cout << str1 << endl << str2 << endl <<
str3 << endl << str4 << endl << endl;

    str1.insert(0, str2);
    cout << str1 << endl << str2 << endl <<
str3 << endl << str4 << endl << endl;

    str4.replace(2, 4, "dislike");
    cout << str1 << endl << str2 << endl << str3 << endl << str4 << endl << endl;

    cout << "str4의 문자열 길이 >>> " << str4.size()<<endl;

    str4.erase(1, 8);
    cout << str1 << endl << str2 << endl << str3 << endl << str4 << endl << endl;

    str4.clear();
    cout << "str2의 다섯번째 문자 >>" << str2.at(5) << endl;
    cout << "str2에서 love 위치 >>" << str2.find("love") << endl;
    cout << "str4에서 love 위치 >>" << (int)str4.find("love") << endl;
}
```

```
C:\Windows\system32\cmd.exe
C++
I love our department
I love our department
I love C++ class

I love our departmentC++
I love our department
I love our department
I love C++ class

I love our departmentC++
I love our department
I love our department
I dislike C++ class

str4의 문자열 길이 >>> 19
I love our departmentC++
I love our department
I love our department
I C++ class

str2의 다섯번째 문자 >>e
str2에서 love 위치 >>2
str4에서 love 위치 >>-1
계속하려면 아무 키나 누르십시오 . . .
```

### \* string 클래스의 연산자

- String s="C++", String s1="C", String s2="Java"

연산자	설명	사용 예	결과
s1 = s2	s2를 s1에 치환	s1 = s2	s1 = "Java"
s[]	s의 []인덱스에 있는 문자	char c = s[1]	c="+"
s1 + s2	s1과 s2를 연결한 새로운 문자열	s1 + s2	"CJava"
s1+=s2	s1에 s2문자열과 연결	s1 += s2	s1="CJava"
s1 == s2	s1과 s2가 같은 문자열이면 true	s1 == s2	false
s1 != s2	s1과 s2가 다른 문자열이면 true	s1 != s2	true
s1 < s2	s1과 s2를 비교 (사전적 순서에 따름)	s1 < s2	true
s1 > s2		s1 > s2	false
s1 <= s2		s1 <= s2	true
s1 >= s2		s1 >= s2	false

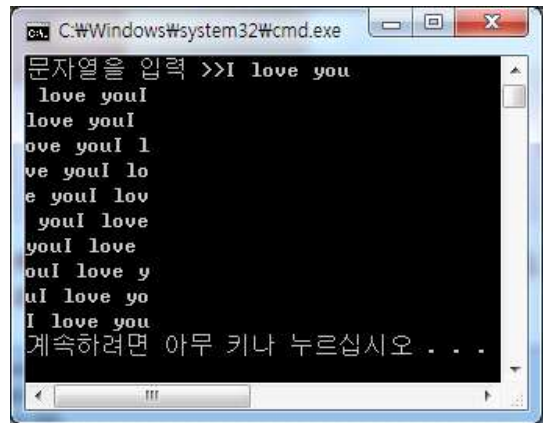
### string 예제 4-13 문자열 입력 및 회전시키기

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

void main(){
    string s;

    cout << "문자열을 입력 >>";
    getline(cin, s, '\n'); // 문자열 입력
    int len = s.length(); // 문자열의 길이
    //int len = str.size(); // 문자열의 길이

    for (int i=0; i< len; i++){
        string first = s.substr(0,1);
        string second = s.substr(1, len-1);
        s = second+first;
        cout << s << endl;
    }
}
```



### string 예제 4-13 문자열 입력 및 회전시키기

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

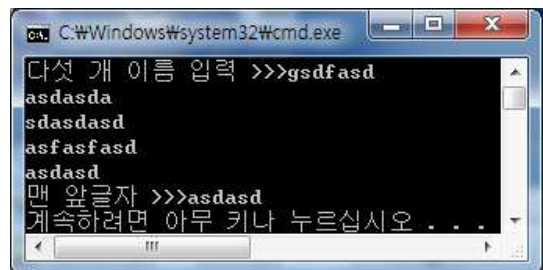
void main(){
    string name[5];

    cout << "다섯 개 이름 입력 >>>"<<endl;
    for (int i=0; i<5; i++){
        getline(cin, name[i], '\n');
    }

    string min = name[0];

    for (int i=1; i<5; i++){
        if(min >= name[i])
            min = name[i];
    }

    cout << "맨 앞글자 >>>" << min <<endl;
}
```



중간고사 : 화요일 11시, ppt 4장 까지

- OX문제, 단답형, 빈 칸, 필기시험

기말 : 실기, 오픈 북

중간고사 끝

=====

## ○ cpp 호출 종류

- 값에 의한 호출 / 주소에 의한 호출 / 참조에 의한 호출

## ○ 함수의 인자 전달 방식 리뷰

## \* 값에 의한 호출, call by value

- \* 함수가 호출하면 매개 변수가 스택에 생성됨
- \* 호출하는 코드에서 값을 넘겨줌
- \* 호출하는 코드에서 넘어온 값이 매개 변수에 복사됨

## \* 주소에 의한 호출, call by address

- \* 함수의 매개 변수는 포인터 타입
  - 함수가 호출되면 포인터 타입의 매개 변수가 스택에 생성됨
- \* 호출하는 코드에서는 명시적으로 주소를 넘겨줌
  - 기본 타입 변수나 객체의 경우, 주소 전달
  - 배열의 경우, 배열의 이름
- \* 호출하는 코드에서 넘어온 주소 값이 매개 변수에 저장됨

## string 예제 4-13 문자열 입력 및 회전시키기

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

// 값에 의한 호출
void swap( string str1, string str2 )
{
    string tmp;
    tmp = str1;
    str1 = str2;
    str2 = tmp;
}

// 주소에 의한 호출
void swap( string *str1, string *str2 )
{
    string tmp;
    tmp = *str1;
    *str1 = *str2;
    *str2 = tmp;
}

int main(void)
{
    string str1("게임공학과");
    string str2 = "동명대학교";

    swap(&str1, &str2);
    swap(str1, str2);

    cout << "str1 : " << str1 << "str2 : " << str2 << endl;

    return 0;
}
```

## ○ “값에 의한 호출”로 객체 전달

### \* 함수를 호출하는 쪽에서 객체 전달

- \* 객체 이름만 사용

### \* 함수의 매개 변수 객체 생성

- \* 매개 변수 객체의 공간이 스택에 할당
- \* 호출하는 쪽의 객체가 매개 변수 객체에 그대로 복사됨
- \* 매개 변수 객체의 생성자는 호출되지 않음 (복사 생성자가 수행)

### \* 함수 종료

- \* 매개 변수 객체의 소멸자 호출

### \* 값에 의한 호출 시 매개 변수 객체의 생성자가 실행되지 않는 이유?

- \* 호출되는 순간의 실 인자 객체 상태를 매개 변수 객체에 그대로 전달하기 위함

## ○ “주소에 의한 호출”로 객체 전달

### \* 함수를 호출 시 객체의 주소만 전달

- \* 함수의 매개 변수는 객체에 대한 포인터 변수로 선언
- \* 함수 호출 시 생성자 소멸자가 실행되지 않는 구조

## ○ 객체 치환 및 객체 리턴

### \* 객체 치환

- \* 동일한 클래스 타입의 객체끼리 치환 가능
- \* 객체의 모든 데이터가 비트 단위로 복사

#### 객체 치환

```
Circle c1(5);  
Circle c2(30);  
c1 = c2; // c2 객체를 c1 객체에 비트 단위로 복사
```

### \* 객체 리턴 (복사 생성자 수행)

#### 객체 리턴

```
Circle getCircle(){  
    Circle tmp(30);  
    return tmp; // 객체 tmp를 리턴  
}
```

```
Circle c; // c의 반지름 1  
c = getCircle(); // tmp 객체의 복사본이 c에 치환됨, c의 반지름은 30이 됨
```

## ○ 참조(reference)

### \* 참조란?

- \* 별명과 같음.
- \* 참조의 활용 : 참조 변수 / 참조에 의한 호출 / 참조 리턴

### \* 참조 변수

#### \* 참조자 &의 도입

- \* 이미 존재하는 변수에 대한 다른 이름(별명)을 선언
  - 참조 변수는 이름만 생김
  - 참조 변수에 새로운 공간을 할당하지 않는다.
  - 초기화로 지정된 기존 변수를 공유한다.
  - 포인터를 사용할 때 &별명을 사용해도 문제 없다.

#### 참조 예제

```
#include <iostream>
#include <string>

using namespace std;

void main(void)
{
    int value = 100;
    int &ivalue = value;
    //int *pvalue = &value;
    int *pvalue = &ivalue; // 별명을 사용해도 상관없음

    cout << (ivalue+10) << " " << value << endl;

    double dvalue;
    double &dd = dvalue;
    dd = 12.34567;
    cout.precision(9);
    cout << dd << endl;

    string name("조광민");
    string &cho = name;
    cout << "name >>> " << cho << endl;
}
```

9주차-2 (16. 11. 01)

## ○ “참조에 의한 호출”로 객체 전달 // call by reference

### \* 함수 형식

- \* 함수의 매개 변수를 참조 타입으로 선언
  - 참조 매개 변수(reference parameter)라고 부름
    - 참조 매개 변수는 실 인자 변수를 참조함
  - 참조매개 변수의 이름만 생기고 공간이 생기지 않음
  - 참조매개 변수는 실 인자 변수 공간 공유
  - 참조 매개 변수에 대한 조작은 실 인자 변수 조작 효과

## 참조 예제2

```
#ifndef CIRCLE_H
#define CIRCLE_H
#include <iostream>

using namespace std;

class Circle{
    double radius;

public:
    Circle();
    Circle( int r ){ cout << "<생성자 수행>" << endl; radius = r;};
    void setRadius( int r ) {radius = r;};
    double getRadius(){ return radius; };
    ~Circle(){cout << "<소멸자 수행>"<< endl << "radius >>>" << radius << endl;};
};

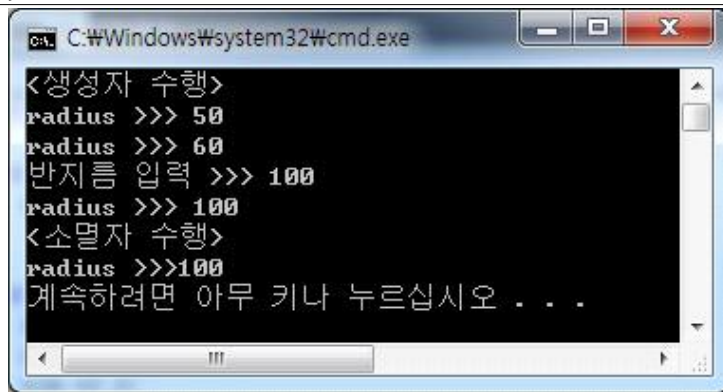
#endif

#include "Circle.h"

void increaseRadius( Circle &c ){
    c.setRadius( c.getRadius()+10 );
}

void readRadius( Circle &d ){
    double r;
    cout << "반지름 입력 >>> ";
    cin >> r;
    d.setRadius( r );
}

void main(){
    Circle waffle(50);
    cout << "radius >>> " << waffle.getRadius() << endl;
    increaseRadius( waffle );
    cout << "radius >>> " << waffle.getRadius() << endl;
    readRadius( waffle );
    cout << "radius >>> " << waffle.getRadius() << endl;
}
```



## \* 참조 리턴

### \* C언어의 함수 리턴

- \* 함수는 반드시 값만 리턴
  - 기본 타입 값 : int, char, double...
  - 포인터 값

### \* C++언어의 함수 리턴 // 공간을 리턴

- \* 함수는 값 외에 참조 리턴 가능
- \* 참조 리턴
  - 변수 등과 같이 현존하는 공간에 대한 참조 리턴 (변수의 값을 리턴하는 것이 아님)
- \* 참조 리턴된 값은 경우에 따라 lvalue가 되기도 하고 rvalue가 되기도 한다.



\* 값을 리턴하는 함수 vs 참조를 리턴하는 함수

**문자 리턴**

```
char c = 'a';

char get() { // char 리턴
    return c; // 변수 c의 문자('a') 리턴
}

char a = get(); // a = 'a'가 됨
get() = 'b'; // 컴파일 오류
```

(a) 문자 값을 리턴하는 get()

**char 타입의 공간에 대한 참조 리턴**

```
char c = 'a';

char& find() { // char 타입의 참조 리턴
    return c; // 변수 c에 대한 참조 리턴
}

char a = find(); // a = 'a'가 됨

char &ref = find(); // ref는 c에 대한 참조
ref = 'M'; // c = 'M'

find() = 'b'; // c = 'b'가 됨
```

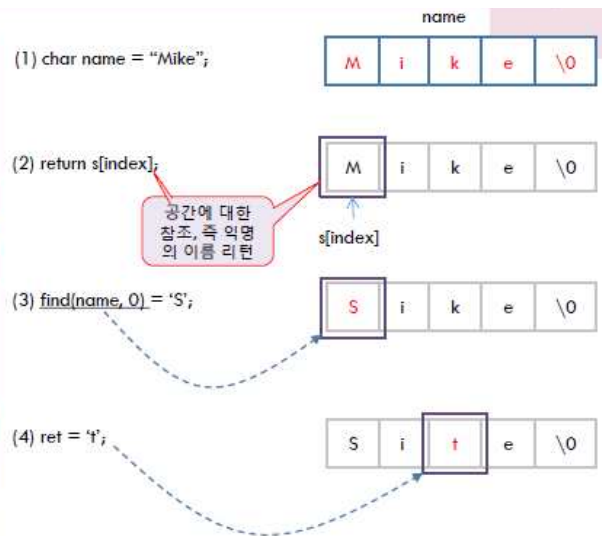
(b) char 타입의 참조(공간)을 리턴하는 find()

참조 리턴 예제

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

char& find(char s[], int index) {
    return s[index]; // 참조 리턴
}

int main() {
    char name[] = "Mike";
    cout << name << endl;
    find(name, 0) = 'S';
    // name[0]='S'로 변경
    cout << name << endl;
    char& ref = find(name, 2);
    ref = 't'; // name = "Site"
    cout << name << endl;
}
```



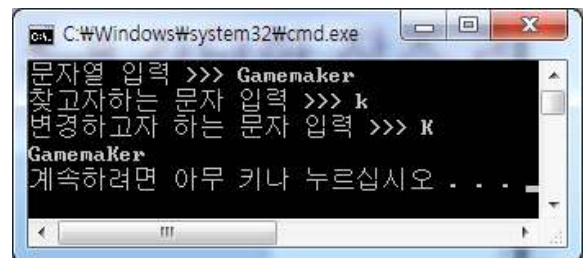
find 함수 구현 - 연습문제2

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

#define N 100

char& find(char str[], char ch, bool& success){
    int n = strlen( str );
    for (int i=0; i< n; i++){
        if (str[i] == ch){
            success = true;
            return str[i];
        }
    }
}

int main(){
    char s[N];
    bool b = false;
    char ch, newch;
    cout << "문자열 입력 >>> ";
    cin.getline(s, N, '\n');
    cout << "찾고자하는 문자 입력 >>> ";
    cin >> ch;
    char& loc = find(s, ch, b);
    if(b == false){
        cout << "M을 발견할 수 없다" << endl;
        return 0;
    }
    cout << "변경하고자 하는 문자 입력 >>> ";
    cin >> newch;
    loc = newch;
    cout << s << endl;
}
```



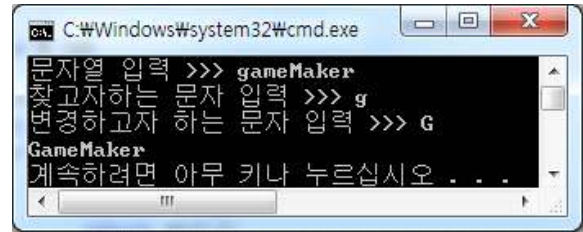
## find 함수 구현(string) - 연습문제2

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

#define N 100

char& find(string& str, char ch, bool&
sucess){
    int n = str.size();
    for (int i=0; i< n; i++){
        if (str[i] == ch){
            sucess = true;
            return str[i];
        }
    }
}

int main(){
    string s;
    bool b = false;
    char ch, newch;
    cout << "문자열 입력 >>> ";
    cin.getline(s, N, '\n');
    cout << "찾고자하는 문자 입력 >>> ";
    cin >> ch;
    char& loc = find(s, ch, b);
    if(b == false){
        cout << "M을 발견할 수 없다" << endl;
        return 0;
    }
    cout << "변경하고자 하는 문자 입력 >>> ";
    cin >> newch;
    loc = newch;
    cout << s << endl;
}
```



10주차-1 (16. 11. 07)

## □ 복사 생성자(copy constructor)

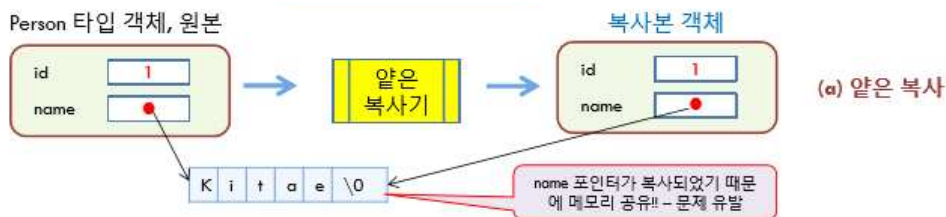
### ○ 얕은 복사와 깊은 복사

#### \* 얕은 복사 : 포인터가 복사

\* 객체 복사 시, 객체의 멤버를 1:1로 복사

\* 객체의 멤버 변수에 동적 메모리가 할당된 경우

- 사본은 원본 객체가 할당 받은 메모리를 공유하는 문제 발생



#### \* 깊은 복사 : 포인터와 포인터의 메모리도 복사 (또 다른 객체가 생성)

\* 객체 복사 시, 객체의 멤버를 1:1로 복사

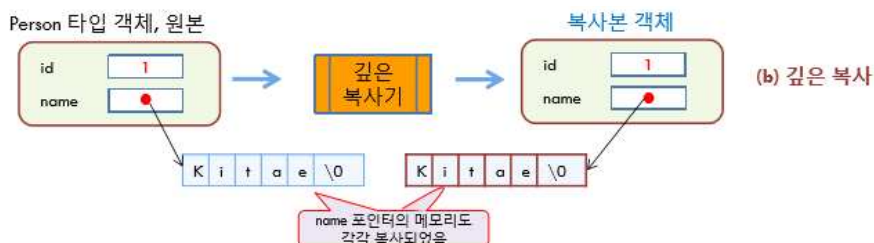
\* 객체의 멤버 변수에 동적 메모리가 할당된 경우

- 사본은 원본이 가진 메모리 크기 만큼 별도로 동적 할당

- 원본의 동적 메모리에 있는 내용을 사본에 복사

\* 완전한 형태의 복사

- 사본과 원본은 메모리를 공유하는 문제 없음



## ○ 복사 생성자 : 객체의 복사 생성시 호출되는 특별한 생성자

### \* 특징

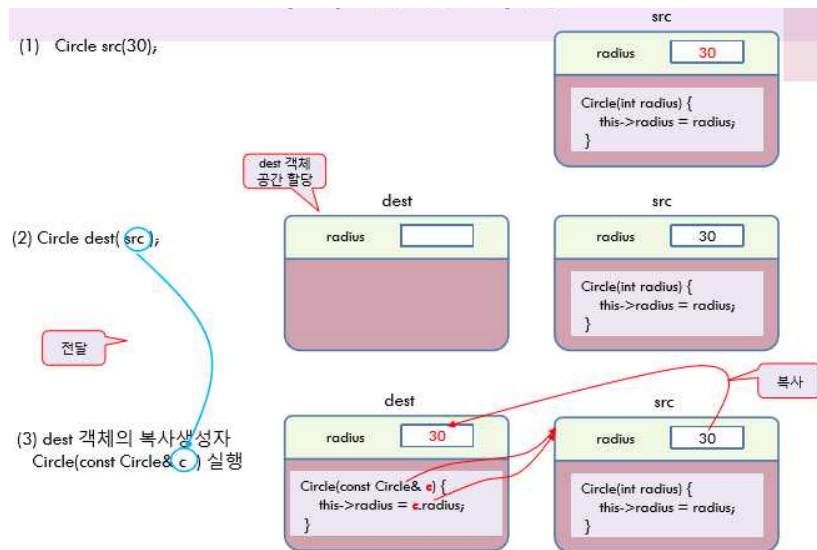
- \* 한 클래스에 오직 한 개만 선언 가능
- \* 모양 : 클래스에 대한 참조 매개 변수를 가지는 독특한 생성자
- \* 포인터가 있는 경우 반드시 깊은 복사로 한다.

### \* 복사 생성자 선언

```
class Circle {  
    .....  
    Circle(const Circle& c); // 복사 생성자 선언  
    .....  
}  
Circle::Circle(const Circle& c) { // 복사 생성자 구현  
    .....  
}
```

자기 클래스에 대한 참조 매개 변수

### \* 복사 생성 과정



### \* 디폴트 복사 생성자 (얕은 복사를 수행)

- \* 복사 생성자가 선언되어 있는 없는 클래스
  - 컴파일러가 자동으로 디폴트 복사 생성자를 생성

// 정 훈이라는 외자 이름에 최배성 이름을 넣으면 오류가 발생하기 때문에, 생성자에서 미리 공간을 할당

```
class Circle {  
    int radius;  
public:  
    Circle(int r);  
    double getArea();  
};  
  
Circle dest(src); // 복사 생성. Circle(Circle&) 호출  
  
Circle::Circle(const Circle& c) {  
    this->radius = c.radius;  
    // 원본 객체 c의 각 멤버를 사본(this)에 복사한다.  
}
```

복사 생성자 없음

복사 생성자 없는데 컴파일 오류?

디폴트 복사 생성자

## find 함수 구현(string) - 연습문제2

```
#ifndef COPY_CONSTRUCTOR_H
#define COPY_CONSTRUCTOR_H
#include <iostream>
using namespace std;

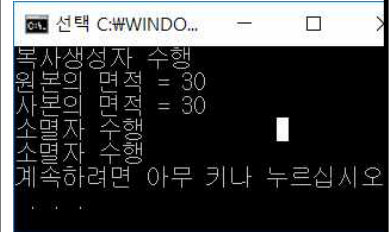
class Circle{
    double radius;

public:
    Circle();
    Circle(double r){ cout << "생성자 수행\n"; radius = r; };

    // 복사 생성자 const
    Circle(const Circle &c){
        cout << "복사생성자 수행\n";
        radius = c.radius;
    };

    double getRadius() { return radius; };
    void setRadius(double r){ radius = r; };
    ~Circle(){ cout << "소멸자 수행\n"; };
};

#endif
```



```
#include "Copy_Constructor.h"

int main() {
    Circle src(30); // src 객체의 보통 생성자 호출
    Circle dest(src); // dest 객체의 복사 생성자 호출

    cout << "원본의 면적 = " << src.getRadius() << endl;
    cout << "사본의 면적 = " << dest.getRadius() << endl;
}
```

## 얕은 복사 생성자

```
// 얕은 복사 생성자 const
Circle(const Circle &c){
    radius = c.radius;
}
```

**const** : 복사 생성자 명령어  
**Circle** : 해당 클래스 이름  
**&c** : 포인터로 받기 때문에 &를 쓴다.

## 깊은 복사 생성자

```
// 깊은 복사 생성자

// 생성자
Person::Person(int id, char* name){
    this->id = id;
    // 잘못된 예 : this->name = name;
    // 올바른 예
    int length = strlen( name );
    this->name = new char[length + 1]; // 마지막 \0
    // 때문에 +1
    strcpy(this->name, name);
}

// 복사 생성자
Person::Person(const Person& person) {
    this->id = person.id; // id 값 복사
    int len = strlen(person.name); // name의 문자 개수
    this->name = new char [len+1]; // name을 위한 공간 할당
    strcpy(this->name, person.name); // name의 문자열 복사
    cout << "복사 생성자 실행. 원본 객체의 이름 " <<
    this->name << endl; }
```

생성자에서 메모리 영역에 할당받는 코드들이 나와야함  
 //int len = strlen(person.name);  
 //this->name = new char[len+1];  
 //strcpy(this->name, person.name);

## 깊은 복사 예제

```
#ifndef PERSON_H
#define PERSON_H
#include <iostream>
#include <string>
using namespace std;

class Person{
    int id;
    char* name;
public:
    Person(int id, char* name);
    Person(const Person& p); // 복사 생성자
    void changeName(char* newname);
    void changeName( Person p ); // 오버로딩
    void show( string s ) { cout << s << "이름 >>> " <<
name << endl << endl; };
    ~Person();
};

#endif
```

```
#include "Person.h"

// 생성자
Person::Person(int id, char* name){
    this->id = id;

    // 잘못된 예 : this->name = name;
    // 올바른 예
    int length = strlen( name );
    this->name = new char[length + 1]; // 마지막 \0 때문에 +1
    strcpy(this->name, name);
}

Person::~Person(){
    cout << "소멸자 실행\n";
    delete[] this->name;
}

// 복사 생성자
Person::Person(const Person& p){
    cout << "복사 생성자 수행\n";
    this->id = p.id;
    this->name = new char[strlen(p.name) + 1];
    strcpy(this->name, p.name);
}

void Person::changeName(char* newname){
    if (strlen(this->name) < strlen(newname)){
        delete[] this->name;
        this->name = new char[strlen(newname) + 1];
        strcpy(this->name, newname);
    }
    else {
        strcpy(this->name, newname);
    }
}

// 오버로딩
void Person::changeName( Person p ){
    if (strlen(this->name) < strlen(p.name)){
        delete[] this->name;
        this->name = new char[strlen(p.name) + 1];
        strcpy(this->name, p.name);
    }
    else {
        strcpy(this->name, p.name);
    }
}
```

```
#include "Person.h"

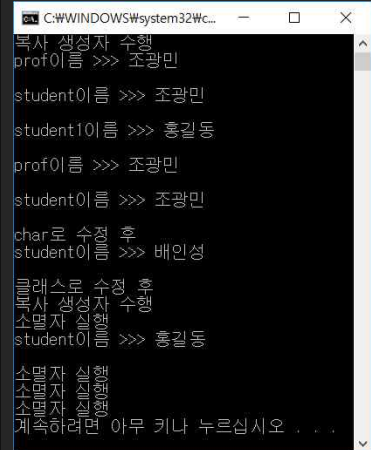
void main(){
    Person prof(100, "조광민");
    Person student(prof);
    Person student1(200, "홍길동");

    prof.show("prof");
    student.show("student");
    student1.show("student1");

    prof.show("prof");
    student.show("student");

    cout << "char로 수정 후 " << endl;
    student.changeName("배인성");
    student.show("student");

    cout << "클래스로 수정 후 " << endl;
    student.changeName(student1);
    student.show("student");
}
```



```
C:\WINDOWS\system32\cmd.exe
복사 생성자 수행
prof이름 >>> 조광민

student이름 >>> 조광민

student1이름 >>> 홍길동

prof이름 >>> 조광민

student이름 >>> 조광민

char로 수정 후
student이름 >>> 배인성

클래스로 수정 후
복사 생성자 수행
소멸자 실행
student이름 >>> 홍길동

소멸자 실행
소멸자 실행
소멸자 실행
계속하려면 아무 키나 누르십시오 . . .
```

## \* 목시적 복사 생성자

\* 컴파일러가 복사 생성자를 자동으로 호출하는 경우

- 객체로 초기화하여 객체가 생성될 때

```
Person father(1, "Hongkilsoo");
Person son1 = father; //복사 생성자 호출
Person son2(father); //복사 생성자 호출
```

- “값에 의한 호출”로 객체가 전달될 때

```
void changeName(Person person)
{person.changeName("Hongkildong"); }
```

```
changeName ( son ); //함수 호출 시 객체를 매개변수로 전달
```

- 함수가 객체를 리턴할 때

```
Person getPerson() {
    Person tmp(2, "HongYounghee");
    return tmp;
}
```

```
getPerson(); //함수 호출 결과 반환된 결과를 m에 치환
```

\* 목시적 복사 생성에 의해 복사 생성자가 호출되는 경우 예제

### 목시적 복사 생성자 예제

```
void chageName( Person person){
    person.changeName("Hongkildong");
}

Person getPerson() {
    Person tmp(2, "HongYounghee");
    return tmp;
}

int main() {
    Person father(1, "Kimkisoo");
    Person son1 = father; //복사 생성자 호출
    Person son2(father); //복사 생성자 호출
    chageName( son2 ); //복사생성자 호출
    getPerson(); //복사생성자 호출

    return 0;
}
```





○ **Stack 후입선출** : 나중에 들어온 것이 먼저 나감 (인터넷 뒤로가기)

Stack 예제

```

C:\WINDOWS\system32\cmd.exe
정수 값 입력 >>> 1
정수 값 입력 >>> 7
정수 값 입력 >>> 4
정수 값 입력 >>> 2
정수 값 입력 >>> 8
정수 값 입력 >>> 5
정수 값 입력 >>> 3
정수 값 입력 >>> 9
정수 값 입력 >>> 6
정수 값 입력 >>> 0
정수 값 입력 >>> 1
11번째 stack full
0 6 9 3 5 8 2 4 7 1
11 번째 stack empty
계속하려면 아무 키나 누르십시오

```

```

#include "MyIntStack.h"

MvIntStack::MvIntStack(){
    top = -1;
}

bool MvIntStack::push( int n ){
    if (this->top == 9){
        return false;
    }
    this->top++;
    this->p[top] = n;
    return true;
}

bool MvIntStack::pop( int &n ){
    if (this->top == -1){
        return false;
    }
    n = this->p[top];
    this->top--;
    return true;
}

#ifdef MYINTSTACK
#define MYINTSTACK

#include <iostream>
using namespace std;

class MvIntStack{
    int top;
    int p[10];
public:
    MvIntStack():
        bool push( int n );
        bool pop( int &n );
};

#endif

#include "MyIntStack.h"

void main(){
    MvIntStack a;
    int n;
    for (int i = 0; i < 11; i++){ // 11개를 푸쉬
        cout << "정수 값 입력 >>> ";
        cin >> n;
        if (a.push(n)) cout << n << endl; // 푸시된 값 에코
        else cout << endl << i + 1 << "번째 stack full" << endl;
    }
    cout << endl;

    for (int i = 0; i < 11; i++) { // 11개를 팝한다
        if(a.pop(n)) cout << n << " "; // 팝 한 값 출력
        else cout << endl << i+1 << " 번째 stack empty" << endl;
    }
    cout << endl;
}

```

○ **Accumulator 클래스(확인문제)** - 누적

Stack 예제

```

#ifdef ACCUMULATOR
#define ACCUMULATOR

#include <iostream>
using namespace std;

class Accumulator{
    int value;
public:
    Accumulator(int v) { value = v; };
    Accumulator& add( int v );
    int get() { return value; };
    ~Accumulator(){};
};

#endif

#include "Accumulator.h"

Accumulator& Accumulator::add(int v){
    this->value += v;
    return *this;
}

void main(){
    Accumulator acc(20);

    acc.add(30).add(50).add(100);
    cout << "누적된 값 >>> " << acc.get() << endl;
}

```

```

선택 C:\WINDOWS\system32\cmd.exe
누적된 값 >>> 200
계속하려면 아무 키나 누르십시오

```

## □ 함수 중복

### ○ 함수 중복(오버로딩)

#### \* 함수 중복

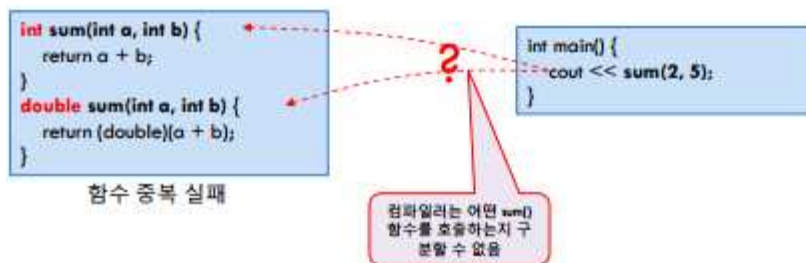
- \* 동일한 이름의 함수가 공존
  - 다형성의 한 형태
  - C언어에서는 불가능
- \* 함수 중복이 가능한 범위
  - 보통 함수들 사이
  - 클래스의 멤버 함수들 사이
  - 상속 관계에 있는 기본 클래스와 파생 클래스의 멤버 함수들 사이

#### \* 함수 중복 성공 조건

- \* 중복된 함수들의 이름 동일
- \* 중복된 함수들의 매개 변수 타입이 다르거나 개수가 달라야함
- \* 리턴 타입은 함수 중복과 무관

#### \* 함수 중복 실패 사례

- \* 리턴 타입이 다르다고 함수 중복이 성공하지 않음



### ○ 생성자 함수 중복

#### \* 생성자 함수 중복 가능

- \* 생성자 함수 중복 목적
  - 객체 생성시, 매개 변수를 통해 다양한 형태의 초기 값 전달

```
class string {
    ....
public:
    string(); // 빈 문자열을 가진 스트링 객체 생성
    string(string& str); // str을 복사한 새로운 스트링 객체 생성
    string(char* s); // '\0'로 끝나는 C-스트링 s를 스트링 객체로 생성
    ....
};

string str; // 빈 문자열을 가진 스트링 객체
string copyAddress(address); // address의 문자열을 복사한 별도의 copyAddress 생성
string address("서울시 성북구 삼선동 389");
```

### ○ 소멸자 함수 중복

#### \* 소멸자 함수 중복 불가

- \* 소멸자는 매개 변수를 가지지 않음
- \* 한 클래스 내에서 소멸자는 오직 하나만 존재

### ○ 디폴트 매개 변수

#### \* 디폴트 매개 변수 (default parameter)

- \* 매개 변수에 값이 넘어오지 않는 경우, 디폴트 값을 받도록 선언된 매개 변수
  - “매개 변수 = 디폴트 값” 형태로 선언

\* 디폴트 매개 변수 선언 사례

```
void star(int a=5); // a의 디폴트 값은 5
```

\* 디폴트 매개 변수를 가진 함수 호출

```
star(); // 매개 변수 a에 디폴트 값 5가 전달됨. star(5)와 동일
star(10); // 매개 변수 a에 10을 넘겨줌
```

\* 디폴트 매개 변수 사례

```
void msg(int id, string text="Hello"); // text의 디폴트 값은 "Hello"
```

```
msg(10); // id에 10, text에 "Hello" 전달
msg(20, "Good Morning"); // id에 20, text에 "Good Morning" 전달
```

호출 오류  
msg(); // 컴파일 오류. 첫 번째 매개 변수 id에 반드시 값을 전달하여야 함  
msg("Hello"); // 컴파일 오류. 첫 번째 매개 변수 id에 값이 전달되지 않았음

\* 디폴트 매개 변수에 관한 제약 조건

- \* 디폴트 매개 변수는 보통 매개 변수 앞에 선언될 수 없음 // 오른쪽에 몰아서 써야함
- 디폴트 매개 변수는 끝 쪽에 몰려 선언되어야 함

컴파일 오류  
void calc(int a, int b=5, int c, int d=0); // 컴파일 오류  
void sum(int a=0, int b, int c); // 컴파일 오류  
void calc(int a, int b=5, int c=0, int d=0); // 컴파일 성공

\* 매개 변수에 값을 정하는 규칙

void square(int width=1, int height=1);

square( ); → square( 1, 1 ); → square( 1, 1 ); g(10);  
square(5); → square( 5, 1 ); → square( 5, 1 ); g(10, 5);  
square(3, 8); → square( 3, 8 ); → square( 3, 8 ); g(10, 5, 20);  
g(10, 5, 20, 30);

컴파일러에 의해 변환되는 과정

void g(int a, int b=0, int c=0, int d=0);

g( 10, 1, 1, 1 ); → g( 10, 0, 0, 0 );  
g( 10, 5, 1, 1 ); → g( 10, 5, 0, 0 );  
g( 10, 5, 20, 1 ); → g( 10, 5, 20, 0 );  
g( 10, 5, 20, 30 ); → g( 10, 5, 20, 30 );

컴파일러에 의해 변환되는 과정

\* 디폴트 매개 변수를 가진 함수 선언 및 호출

\* 원형 선언을 통해 디폴트 매개 변수 설정

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

// 원형 선언
void star(int a=5);
void msg(int id, string text="");

// 함수 구현
void star(int a) {
    for(int i=0; i<a; i++)
        cout << "x";
    cout << endl;
}

void msg(int id, string text) {
    cout << id << " " << text << endl;
}

int main() {
    // star() 호출
    star();
    star(10);

    // msg() 호출
    msg(10);
    msg(10, "Hello");
}
```

디폴트 매개 변수 선언

동일한 코드

```
void star(int a=5) {
    for(int i=0; i<a; i++)
        cout << "x";
    cout << endl;
}

void msg(int id, string text="") {
    cout << id << " " << text << endl;
}
```

\*\*\*\*\*  
10  
10 Hello

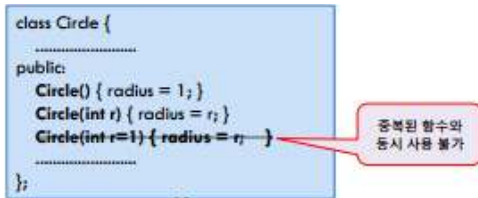
C++ 프로그래밍

## \* 함수 중복의 간소화

### \* 디폴트 매개 변수의 장점 - 함수 중복 간소화



### \* 중복 함수들과 디폴트 매개 변수를 가진 함수를 함께 사용 불가

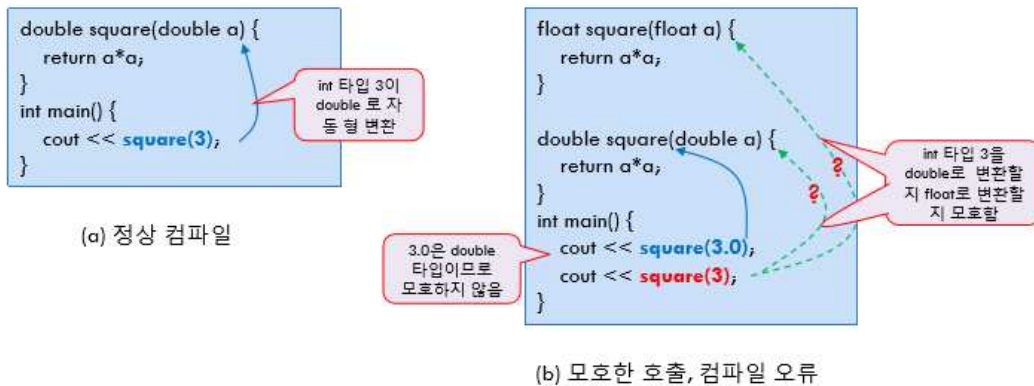


12주차 (16. 11. 21)

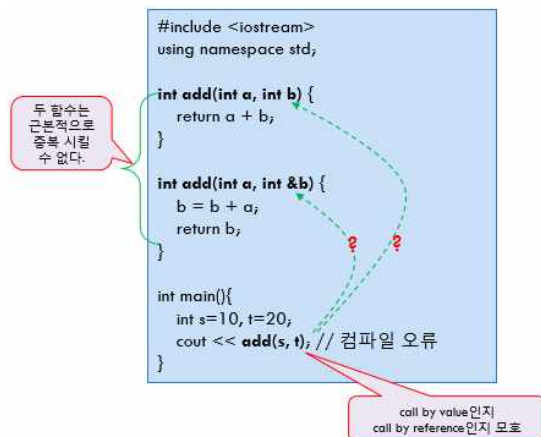
## \* 함수 중복의 모호성

### \* 함수 중복이 모호하여 컴파일러가 어떤 함수를 호출하는지 판단하지 못함

#### - 형 변환으로 인한 모호성



#### - 참조 매개 변수로 인한 모호성



#### - 디폴트 매개 변수로 인한 모호성



## □ Static 멤버

### ○ Static 멤버의 특성

#### \* Static

- \* 변수와 함수에 대한 기억 부류의 한 종류
  - 생명 주기 : **프로그램이 시작될 때 생성, 프로그램 종료 시 소멸**
  - 사용 범위 : 선언된 범위, 접근 지정에 따른

#### \* 클래스의 멤버

- \* static 멤버
  - 프로그램이 시작할 때 생성
  - 클래스 당 하나만 생성, 클래스 멤버라고 불림
  - 클래스의 모든 인스턴스(객체)들이 공유하는 멤버
- \* non-static 멤버
  - 객체가 생성될 때 함께 생성
  - 객체마다 객체 내에 생성
  - 인스턴스 멤버라고 불림

### ○ static 멤버 선언

#### \* static 멤버 선언

- \* static 멤버 변수 선언 : static 인자타입 변수명;
- \* static 멤버 함수 선언 : static 함수타입 함수명();

#### \* static 멤버와 non-static 멤버 비교

항목	non-static 멤버	static 멤버
선언 사례	<pre>class Sample {     int n;     void f(); };</pre>	<pre>class Sample {     static int n;     static void f(); };</pre>
공간 특성	멤버는 객체마다 별도 생성 • 인스턴스 멤버라고 부름	멤버는 클래스 당 하나 생성 • 멤버는 객체 내부가 아닌 별도의 공간에 생성 • 클래스 멤버라고 부름
시간적 특성	객체와 생명을 같이 함 • 객체 생성 시에 멤버 생성 • 객체 소멸 시 함께 소멸 • 객체 생성 후 객체 사용 가능	프로그램과 생명을 같이 함 • 프로그램 시작 시 멤버 생성 • 객체가 생기기 전에 이미 존재 • 객체가 사라져도 여전히 존재 • 프로그램이 종료될 때 함께 소멸
공유의 특성	공유되지 않음 • 멤버는 객체 별로 따로 공간 유지	동일한 클래스의 모든 객체들에 의해 공유됨

#### \* static 멤버는 객체 이름이나 객체 포인터로 접근

- \* 보통 멤버처럼 접근할 수 있음

```
객체.static멤버  
객체포인터->static멤버
```

- \* Person 타입의 객체 lee와 포인터 p를 이용하여 static 멤버를 접근하는 예

```
Person lee;  
lee.sharedMoney = 500; // 객체.static멤버 방식  
  
Person *p;  
p = &lee;  
p->addShared(200); // 객체포인터->static멤버 방식
```



## \* 클래스명과 범위 지정 연산자 :: 로 접근 가능

- \* static 멤버는 클래스마다 오직 한 개만 생성되지 때문

클래스명::static멤버

han.sharedMoney = 200;	<->	Person::sharedMoney = 200;
lee.addShared(200);	<->	Person::addShared(200);

- \* non-static 멤버는 클래스 이름을 접근 불가 // 아직 생성이 이루어지지 않음

Person::money = 100; // 컴파일 오류. non-static 멤버는 클래스 명으로 접근불가  
Person::addMoney(200); // 컴파일 오류. non-static 멤버는 클래스 명으로 접근불가

## \* static 멤버 특징

- \* 전역 변수나 전역 함수를 클래스에 캡슐화
- \* 객체 사이에 공유 변수를 만들고자 할 때 사용
- \* non-Static 멤버 함수에서는 non-static, static 모두 사용 가능
- \* Static 멤버 함수에서는 static 멤버만 사용 가능
  - static 멤버 함수
  - static 멤버 변수
  - 함수 내에 지역 변수
- \* static 멤버 함수는 non-static 멤버에 접근 불가
  - 객체가 생성되지 않은 시점에서 static 멤버 함수가 호출될 수 있기 때문
- \* static 멤버 함수는 this 사용 불가
  - static 멤버 함수는 객체가 생기기 전부터 호출 가능하기 때문

## □ 프렌드

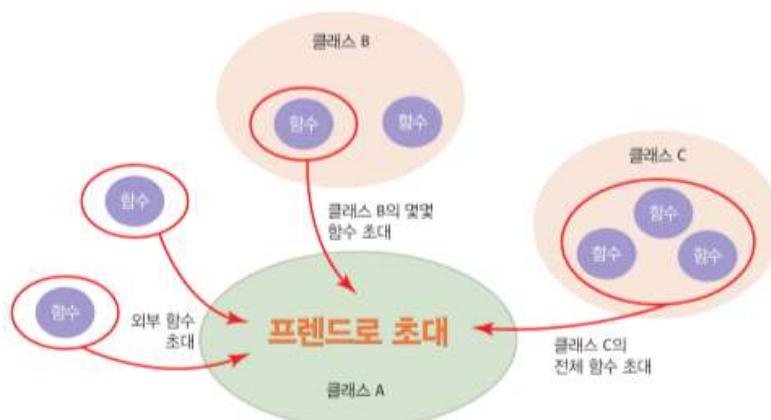
### ○ C++ 프렌드

#### \* 프렌드란?

- \* 클래스 외부에 작성된 함수에 대해 클래스 멤버 함수와 동일한 접근 자격을 부여하는 명령
- \* 클래스의 멤버가 아니지만 멤버의 권한을 가지도록 함
- \* friend 키워드 사용
- \* 클래스의 멤버로 선언하기에는 무리가 있고, 클래스의 모든 멤버를 자유롭게 접근할 수 있는 일부 외부 함수 작성이 필요한 경우에 사용

#### \* 프렌드 함수가 되는 3가지 유형

- \* 전역 함수 : 클래스 외부에 선언된 전역 함수
- \* 다른 클래스의 멤버 함수 : 다른 클래스의 특정 멤버 함수
- \* 다른 클래스 전체 : 다른 클래스의 모든 멤버 함수



## \* 프렌드 선언 3종류

- \* 외부 함수 equals()를 Rect 클래스에 프렌드로 선언

```
class Rect { // Rect 클래스 선언
...
    friend bool equals(Rect r, Rect s);
};
```

- \* RectManager 클래스의 equals() 멤버 함수를 Rect 클래스에 프렌드로 선언

```
class Rect {
...
    friend bool RectManager::equals(Rect r, Rect s);
};
```

- \* RectManager 클래스의 모든 멤버 함수를 Rect 클래스에 프렌드로 선언

```
class Rect {
...
    friend RectManager;
};
```

## □ 연산자 중복

13주차 (16. 11. 28)

### ○ 연산자 중복

- \* 연산자 중복은 연산자 오버로딩 => 다형성의 구현

- \* 함수의 중복(오버로딩)과 같이 연산자도 하나의 함수라는 개념을 사용하여 중복 정의
- \* C++ 언어에서의 연산자 중복 : 새로운 의미를 재정의하여 사용

### \* 연산자 중복의 특징

- \* C++에 본래 있는 연산자만 중복 가능
- \* 피 연산자 타입이 다른 새로운 연산을 정의하는 것임
  - 수+객체, 객체+수, 객체+객체
- \* 연산자는 함수 형태로 구현
  - 연산자 함수
- \* 반드시 클래스와의 관계를 가짐
  - 피연산자에 객체를 동반하기 때문에 반드시 클래스의 멤버함수로 구현하든지, 전역 변수로 구현할 경우 클래스의 프렌드 함수로 구현
- \* 피연산자의 개수를 바꿀 수 없음
  - 단항 연산자(!, ++, -- 등)은 단항 연산자로 구현해야 하고
  - 이항 연산자(+, -, \*, / 등)은 이항 연산자로 구현해야 한다.
- \* 연산의 우선 순위 변경 안됨
- \* 모든 연산자가 중복 가능하지 않음
- \* 중복 가능한 연산자들

+	-	*	/	%	^	&
	~	!	=	<	>	+=
-=	*=	/=	%=	^=	&=	=
<<	>>	>>=	<<=	==	!=	>=
<=	&&		++	--	->*	.
->	[]	()	new	delete	new[]	delete[]

- \* 중복 불가능한 연산자들

.	*	:: (범위지정 연산자)	? : (3항 연산자)
---	---	---------------	--------------



## \* 연산자 중복을 위한 2가지 방법

1. 클래스의 멤버 함수로 구현하는 방법
2. 외부 함수로 구현하고 클래스에 프렌드 함수로 선언하는 방법

## \* 연산자 함수 형식

리턴타입 **operator** 연산자(매개변수리스트);

```
Color a(BLUE), b(RED), c;
```

```
c = a + b; // a와 b를 더하기 위한 + 연산자 작성 필요
if(a == b) { // a와 b를 비교하기 위한 == 연산자 작성 필요
    ...
}
```

## \* +와 == 연산자의 작성 사례

1. 클래스의 멤버 함수로 작성되는 경우

```
class Color {
    ...
    Color operator+ (Color op2); //왼쪽 피연산자가 객체 자신이고 오른쪽 피연산자가 op2에 전달
    bool operator== (Color op2); //왼쪽 피연산자가 객체 자신이고 오른쪽 피연산자가 op2에 전달
};
```

2. 외부 함수로 구현되고 클래스에 프렌드로 선언되는 경우

```
Color operator+ (Color op1, Color op2) {
    ...
}
bool operator== (Color op1, Color op2) {
    ...
}

class Color {
    ...
    friend Color operator+ (Color op1, Color op2);
    friend bool operator== (Color op1, Color op2);
};
```

## \* 연산자 함수 구현 방법 - 클래스의 멤버 함수로 구현

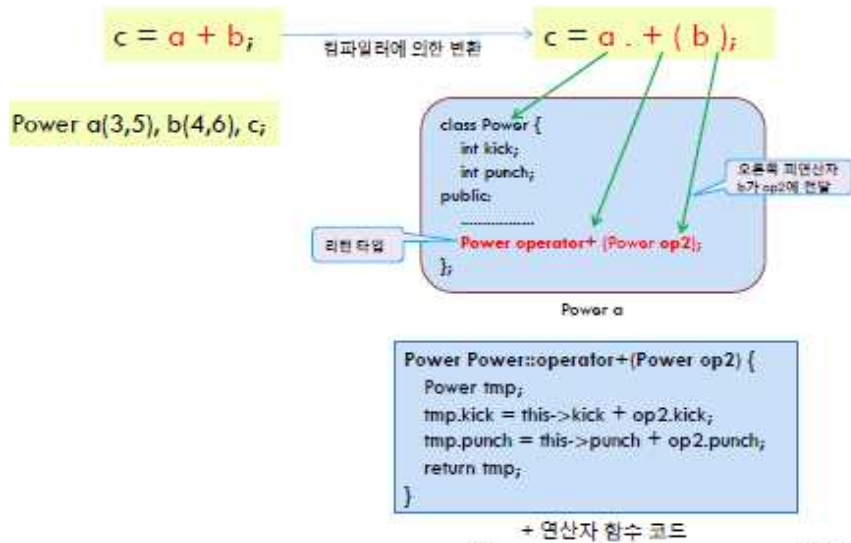
- 연산자 중복 설명에 사용할 클래스

```
class Power { // 에너지를 표현하는 파워 클래스
    int kick; // 발로 차는 힘
    int punch; // 주먹으로 치는 힘

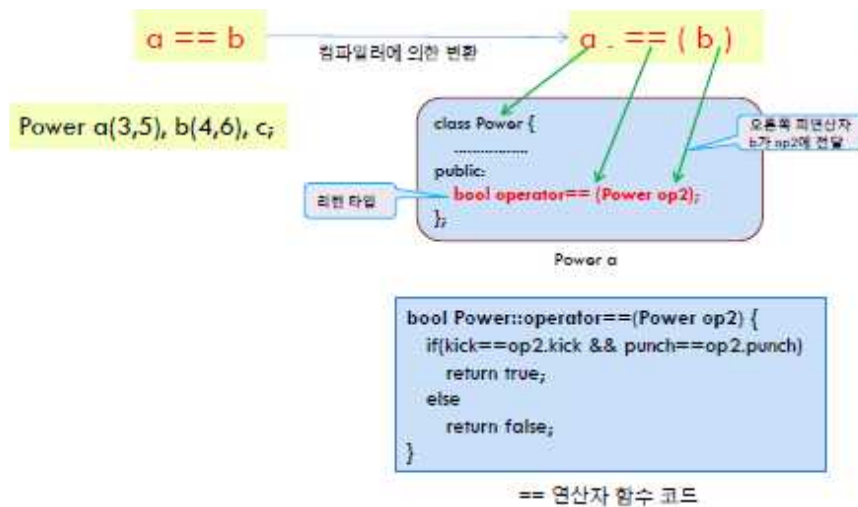
public:
    Power(int kick=0, int punch=0) {
        this->kick = kick;
        this->punch = punch;
    }
};
```

## \* 이항 연산자 중복

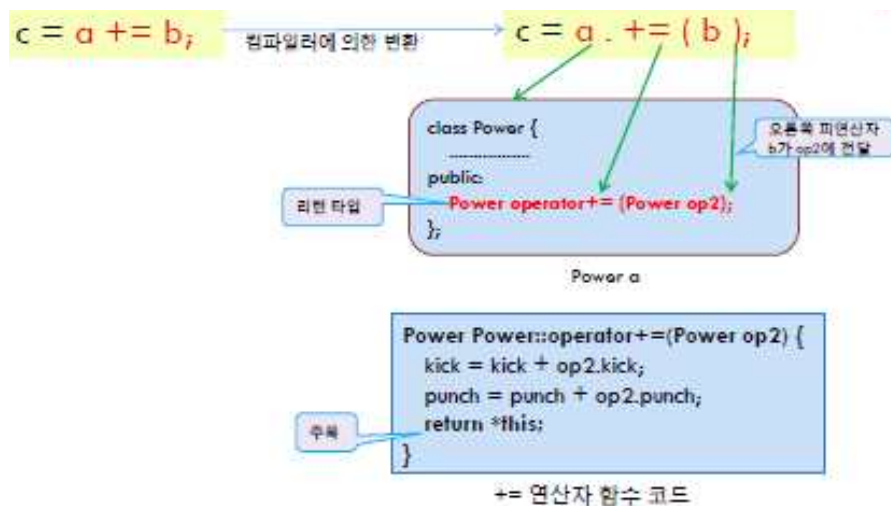
### \* + 연산자 중복



### \* == 연산자 중복



### \* += 연산자 중복



## \* 단항 연산자 중복

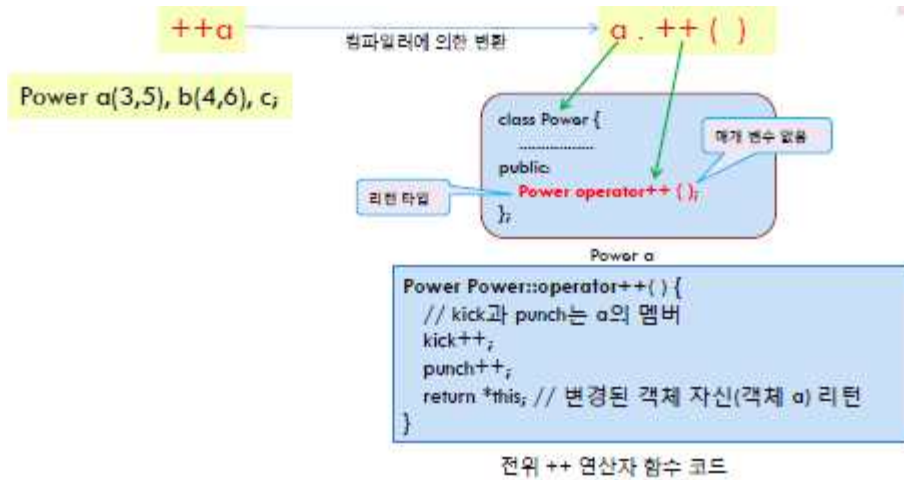
### \* 피연산자가 하나 뿐인 연산자

- 연산자 중복 방식은 이항 연산자의 경우와 거의 유사함

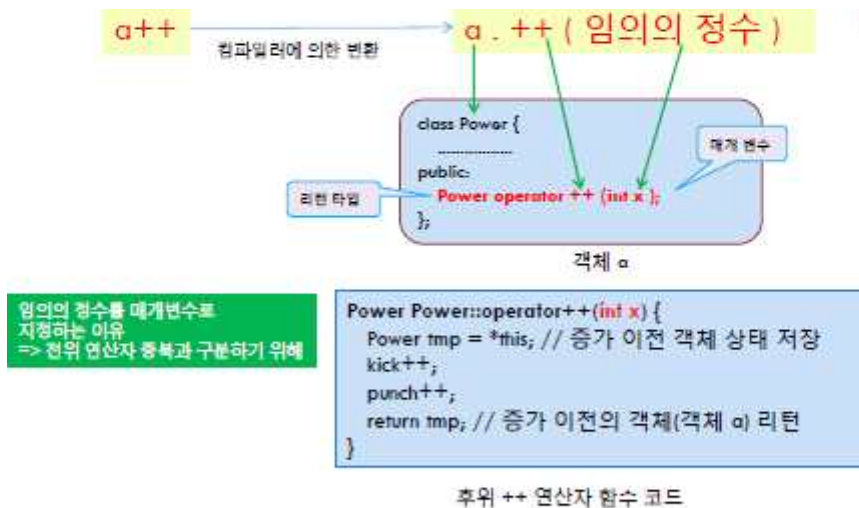
### \* 단항 연산자 종류

- 전위 연산자 : !op, ~op, ++op, --op
- 후위 연산자 : op++, op--

### \* 전위 ++ 연산자 중복



### \* 후위 연산자 중복, ++ 연산자



## \* 연산자 함수 구현 방법

### \* 외부 함수로 구현

- 외부 함수로 구현하고 클래스에 프렌드 함수로 선언하는 방법

#### ▶ 상수 + 객체

```
Power a(3,4), b;
b = 2 + a;
```

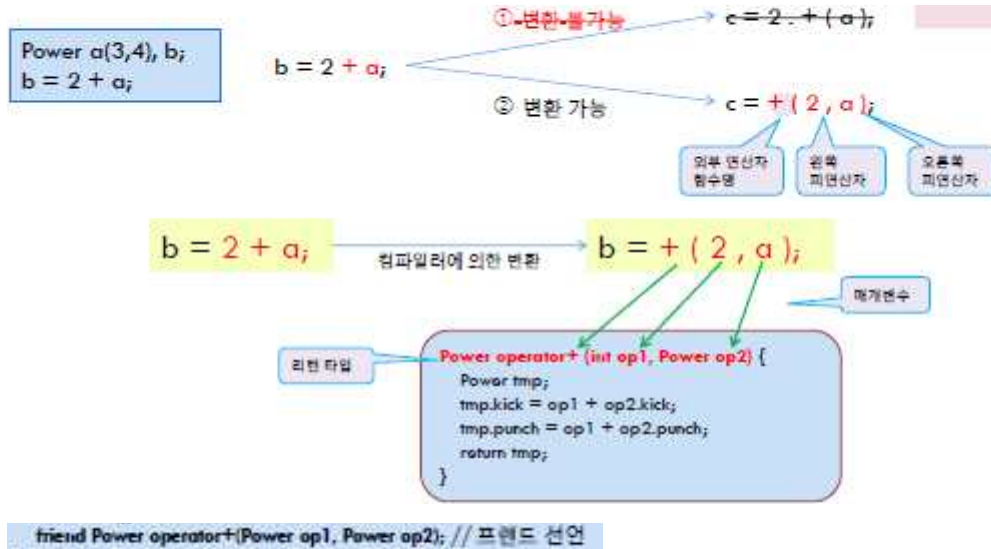
#### ▶ 객체 + 객체

```
c = a+b;
```

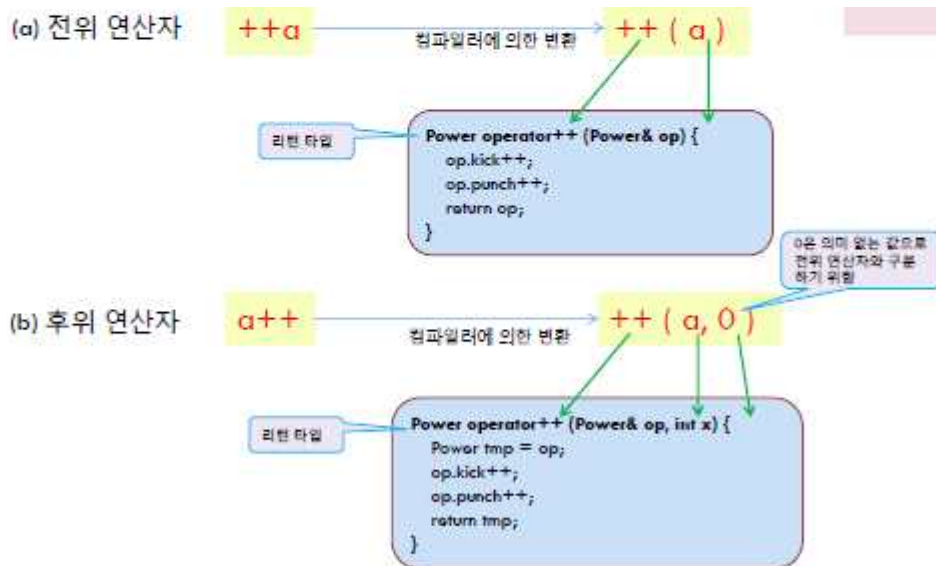
#### ▶ 단항 연산자를 프렌드 외부 함수로 구현

```
a++;
```

\* 2 + a 덧셈을 위한 + 연산자 함수 작성



\* 단항 연산자 ++를 프렌드로 작성



\* 사용자 정의 객체 입출력 연산자 <<, >> 구현

- \* C++의 입출력 연산자 <<, >> 도 연산자 오버로딩에 의해 구현된 것
  - 이름 영역 std의 클래스 ostream, istream 클래스에서 정의

\* 사용자 정의 객체에 <<, >> 연산자를 적용하려면 friend 함수 정의

```
ostream& operator<<(ostream& os, class ob)
{
    ...
    return os;
}

istream& operator>>(istream& is, class ob)
{
    ...
    return is;
}
```

friend ostream& operator<<(ostream& os, const Power& p);  
friend istream& operator>>(istream& is, Power& p);

```
#include "op.h"
ostream& operator<<(ostream& os, const Power& p)
{
    os<<"["<<p.kick<<" "<<p.punch<<"<<endl;
    return os;
}
istream& operator>>(istream& is, Power& p)
{
    is>>p.kick>>p.punch;
    return is;
}
```