




고급프로그래밍

함수와 참조

Professor Jeong, Mun-Ho

Robot Vision & Intelligence Laboratory
Kwangwoon University
(02-940-5625, mhjeong@kw.ac.kr)

Schedule

week	Topics		Homework	Quiz
1	과목소개	교과목 소개 (1), C++ 시작 (2)		
2	C++ 	C++ 프로그래밍의 기본(3, 3/12), 클래스와 객체(4, 3/14)	1	1
3		휴강(3/17), 객체생성과 사용(5, 3/19)	2	
4		함수와 참조(6, 3/26), 복사 생성자와 함수중복, static, friend, 연산자 중복	3	2, 3
5		상속, 가상함수와 추상클래스	4	4
6		템플릿과 STL, 표준 입출력	5	5
7		파일 입출력		
8				
9	C++	예외처리 및 C 사용, 람다식	6	6
10		멀티스레딩	7	7
11		멀티스레딩, 고급문법	8	8
12		고급문법	9	9
13	병렬 프로그래밍	병렬프로그래밍		
14		병렬프로그래밍		
15	기말고사			

오늘의 학습내용

- 객체의 생성과 사용 - 계속
- 함수와 참조

실습 1

- 다음 main() 함수가 실행되도록 class Sample을 완성하시오.

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

class Sample
{
    int* _npData;
    int _nSize;

public:
    Sample() { _npData = 0; }
    void Read(int nSize);
    void Write( );
    int Big( );
    ~Sample( );
};
```

```
int main()
{
    Sample s;
    s.Read(5); //키보드에서 5개의 정수 읽어들이기
    s.Write();// 정수 배열 출력
    cout << "The big No. is " << s.Big( ) << endl;
}
```

실습 1 - 답

- 다음 main() 함수가 실행되도록 class Sample을 완성하시오.

```
class Sample
{
    int*    _npData;
    int     _nSize;
public:
    Sample() {
        _nSize = 0;
        _npData = 0;    // n개 정수 배열의 동적 생성
    }
    void Read(int size); // 사용자로부터 정수를 입력 받음
    void Write();        // 정수 배열을 화면에 출력
    int Big();           // 정수 배열에서 가장 큰 수 리턴
    ~Sample();
};
```

```
void Sample::Read(int size)
{
    if(_npData)
        delete[] _npData;
    _nSize = size;
    _npData = new int[size];

    for(int i=0; i<size; i++)
        cin >> _npData[i];
}

void Sample::Write()
{
    if(_npData == 0)
        return;

    for(int i=0; i<_nSize; i++)
        cout << _npData[i] << ' ';
    cout << endl;
}
```

```
int Sample::Big()
{
    int big = _npData[0];
    for(int i=1; i<_nSize; i++)
        if(big < _npData[i])
            big = _npData[i];

    return big;
}

Sample::~Sample() {
    if(_npData)
        delete[] _npData;
}
```

실습 2

- Family 클래스를 완성하시오(family.h, family.cpp)

```
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(string name, int size); // size 개수만큼 Person 배열 동적 생성
    void setName(int index, string name);
    void show(); // 모든 가족 구성원 출력
    ~Family();
};
```

```
int main() {
    Family *simpson = new Family("Simpson", 3);
    simpson->setName(0, "Mr. Simpson");
    simpson->setName(1, "Mrs. Simpson");
    simpson->setName(2, "Bart Simpson");
    simpson->show();
    delete simpson;
}
```

C:\Qt\Tools\QtCreator\bin\qtcreator_process_stub.exe

```
Simpson 가족은 다음과 같이 3명 입니다.
Mr. Simpson    Mrs. Simpson    Bart Simpson
Press <RETURN> to close this window...
```


실습 2 - 답

- Family 클래스를 완성하시오(family.h, family.cpp)

```
Family::Family(string name, int size) {
    this->name = name;
    this->size = size;
    p = new Person [size];
}

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

void Family::setName(int index, string name) {
    p[index].setName(name);
}

void Family::show() {
    cout << name + " 가족은 다음과 같이 " << size << "명 입니다." << endl;
    for(int i=0; i<size; i++) {
        cout << p[i].getName() << '\t';
    }
    cout << endl;
}
```

this 포인터

■ this

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

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


this와 객체

- 각 객체 속의 this는 다른 객체의 this와 다름

c1

```
radius
...
void setRadius(int radius) {
    this->radius = radius;
}
...
```

c2

```
radius
...
void setRadius(int radius) {
    this->radius = radius;
}
...
```

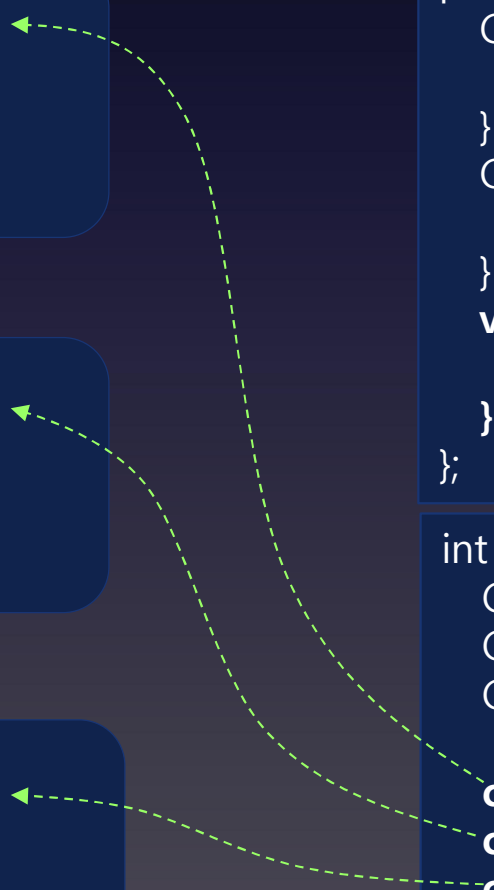
c3

```
radius
...
void setRadius(int radius) {
    this->radius = radius;
}
...
```

```
class Circle {
    int radius;
public:
    Circle() {
        this->radius=1;
    }
    Circle(int radius) {
        this->radius = radius;
    }
    void setRadius(int radius) {
        this->radius = radius;
    }
};
```

```
int main() {
    Circle c1;
    Circle c2(2);
    Circle c3(3);

    c1.setRadius(4);
    c2.setRadius(5);
    c3.setRadius(6);
}
```



this가 필요한 경우

- 매개변수의 이름과 멤버 변수의 이름이 같은 경우

```
Circle(int radius)
{
    this->radius = radius;
}
```



```
Circle(int radius)
{
    radius = radius;
}
```

- 멤버 함수가 객체 자신의 주소를 리턴할 때
 - 연산자 중복 시에 매우 필요

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

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

this의 제약 사항

- 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 때문에

this 포인터의 실체

■ 컴파일러에서 처리

```
class Sample {  
    int a;  
public:  
    void setA(int x) {  
        this->a = x;  
    }  
};
```

개발자가 작성한 클래스

컴파일러에
의해 변환

```
class Sample {  
    ...  
public:  
    void setA(Sample* this, int x) {  
        this->a = x;  
    }  
};
```

컴파일러에 의해 변환된 클래스

Sample ob;

ob.setA(5);

컴파일러에 의해 변환

ob.setA(&ob, 5);

객체의 멤버 함수를 호출하는 코드의 변환

string 객체 생성 및 입출력

■ 문자열 생성

```
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
```

■ 문자열 출력

- cout과 << 연산자

```
cout << address << endl; // "서울시 성북구 삼선동 389" 출력
cout << title << endl; // "Love C++" 출력
```

■ 문자열 입력

- cin과 >> 연산자

```
string name;
cin >> name; // 공백이 입력되면 하나의 문자열로 입력
```

■ 문자열 숫자 변환

- stoi() 함수 이용
 - C++11 부터

```
string s="123";
int n = stoi(s); // n은 정수 123. 비주얼 C++ 2010 이상 버전
```

```
string s="123";
int n = atoi(s.c_str()); // n은 정수 123. 비주얼 C++ 2008 이하
```

string 객체의 동적 생성

- `new/delete`를 이용하여 문자열을 동적 생성/반환 가능

```
string* p = new string("C++"); // 스트링 객체 동적 생성
```

```
cout << *p;           // "C++" 출력  
p->append(" Great!!"); // p가 가리키는 스트링이 "C++ Great!!"이 됨  
cout << *p;           // "C++ Great!!" 출력
```

```
delete p; // 스트링 객체 반환
```

예제 – string 클래스 사용

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

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

    char text[] = {'L', 'o', 'v', 'e', ' ', 'C', '+', '+', '\0'}; // C-스트링
    string title(text); // "Love C++" 문자열을 가진 스트링 객체 생성

    // 스트링 출력
    cout << str << endl; // 빈 스트링. 아무 값도 출력되지 않음
    cout << address << endl;
    cout << copyAddress << endl;
    cout << title << endl;
}
```

```
서울시 성북구 삼선동 389
서울시 성북구 삼선동 389
Love C++
```


예제 - string 배열

- 5 개의 string 배열을 선언하고 getline()을 이용하여 문자열을 입력 받아 사전 순으로 가장 뒤에 나오는 문자열을 출력하라. 문자열 비교는 <, > 연산자를 간단히 이용하면 된다.

```
:  
  
int main()  
{  
    string names[5];    // 문자열 배열 선언  
  
    for(int i=0; i<5; i++) {  
        cout << "이름 >> ";  
        getline(cin, names[i], '\n');  
    }  
  
    string latter = names[0];  
    for(int i=1; i<5; i++) {  
        if(latter < names[i]) { // 사전 순으로 latter 문자열이 앞에 온다면  
            latter = names[i]; // latter 문자열 변경  
        }  
    }  
    cout << "사전에서 가장 뒤에 나오는 문자열은 " << latter << endl;  
}
```

```
이름 >> Kim Nam Yun  
이름 >> Chang Jae Young  
이름 >> Lee Jae Moon  
이름 >> Han Won Sun  
이름 >> Hwang Su hee  
사전에서 가장 뒤에 나오는 문자열은 Lee Jae Moon
```

예제 – string 사용

- 빈칸을 포함하는 문자열을 입력 받고, 한 문자씩 왼쪽으로 회전하도록 문자열을 변경하고 출력하라.

```
:  
  
int main() {  
    string s;  
  
    cout << "문자열을 입력하세요 " << endl;  
    getline(cin, s, '\n'); // 문자열 입력  
    int len = s.length(); // 문자열의 길이  
  
    for(int i=0; i<len; i++)  
    {  
        string first = s.substr(0,1); // 맨 앞의 문자 1개를 문자열로 분리  
        string sub = s.substr(1, len-1); // 나머지 문자들을 문자열로 분리  
        s = sub + first; // 두 문자열을 연결하여 새로운 문자열로 만듦  
        cout << s << endl;  
    }  
}
```

문자열을 입력하세요

I love you
love youI
love youI
ove youI I
ve youI lo
e youI lov
youI love
youI love
ouI love y
uI love yo
I love you

예제 – string 사용

- &가 입력될 때까지 여러 줄의 영문 문자열을 입력 받고, 찾는 문자열과 대치할 문자열을 각각 입력 받아 문자열을 변경하라.

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

```
int main()
{
```

```
    string s;
```

```
    cout << "여러 줄의 문자열을 입력하세요. 입력의 끝은 &문자입니다." << endl;
```

```
    getline(cin, s, '&'); // 문자열 입력
```

```
    cin.ignore();
```

& 뒤에 따라 오는 <Enter>
키를 제거하기 위한 코드!!!

```
    string fd;
```

```
    cout << endl << "find: ";
```

```
    getline(cin, fd, '\n'); // 검색할 문자열 입력
```

```
    string rp;
```

```
    cout << "replace: ";
```

```
    getline(cin, rp, '\n'); // 대치할 문자열 입력
```

```
    int startIndex = 0;
```

```
    while(true) {
```

```
        int fIndex = s.find(fd, startIndex); // startIndex부터 문자열 f 검색
        if(fIndex == -1)
```

```
            break; // 문자열 s의 끝까지 변경하였음
```

```
            s.replace(fIndex, fd.length(), rp); // fIndex부터 문자열 f의 길이만큼
                                                    // 문자열 rp로 변경
```

```
            startIndex = fIndex + rp.length();
```

```
    }
```

```
    cout << s << endl;
```

```
}
```

예제 – string 사용

여러 줄의 문자열을 입력하세요. 입력의 끝은 &문자입니다.

It's **now** or never, come hold me tight. Kiss me my darling, be mine<Enter>
tonight<Enter>

Tomorrow will be too late. It's now or never, my love won't wait&<Enter>

find: **now**

replace: **Right Now**

It's **Right Now** or never, come hold me tight. Kiss me my darling, be mine
tonight

Tomorrow will be too late. It's Right Now or never, my love won't wait

<Enter>이후에도
계속입력

입력완료

함수와 참조

함수의 인자 전달 방식 리뷰

■ 인자 전달 방식

- 값에 의한 호출, call by value

- 함수가 호출되면 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

```
void swap(int a, int b)
```

```
{
    int tmp;
```

```
    tmp = a;
```

```
    a = b;
```

```
    b = tmp;
```

```
}
```

```
int main() {
```

```
    int m=2, n=9;
```

```
    swap(m, n);
```

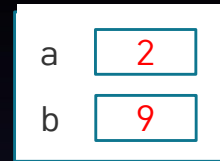
```
    cout << m << " " << n;
```

```
}
```

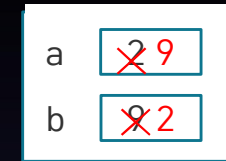
2 9



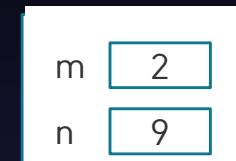
(1) swap() 호출 전



(2) swap() 호출 직후



(3) swap() 실행



(4) swap() 리턴 후

값에 의한 호출

```
void swap(int *a, int *b)
```

```
{
```

```
    int tmp;
```

```
    tmp = *a;
```

```
    *a = *b;
```

```
    *b = tmp;
```

```
}
```

```
int main() {
```

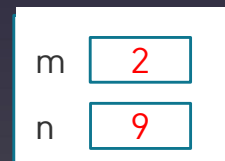
```
    int m=2, n=9;
```

```
    swap(&m, &n);
```

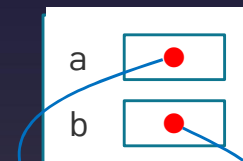
```
    cout << m << " " << n;
```

```
}
```

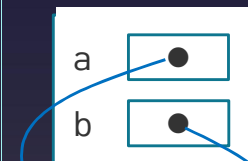
9 2



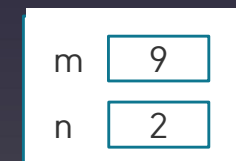
(1) swap() 호출 전



(2) swap() 호출 직후



(3) swap() 실행



(4) swap() 리턴 후

주소에 의한 호출

'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

매개 변수 객체의 생성자 소
멸자의 비대칭 실행 구조

- 함수 종료

- 매개 변수 객체의 소멸자 호출

- 값에 의한 호출 시 매개 변수 객체의 생성자가 실행되지 않는 이유?

- 호출되는 순간의 객체 상태를 매개 변수 객체에 그대로 전달하기 위함

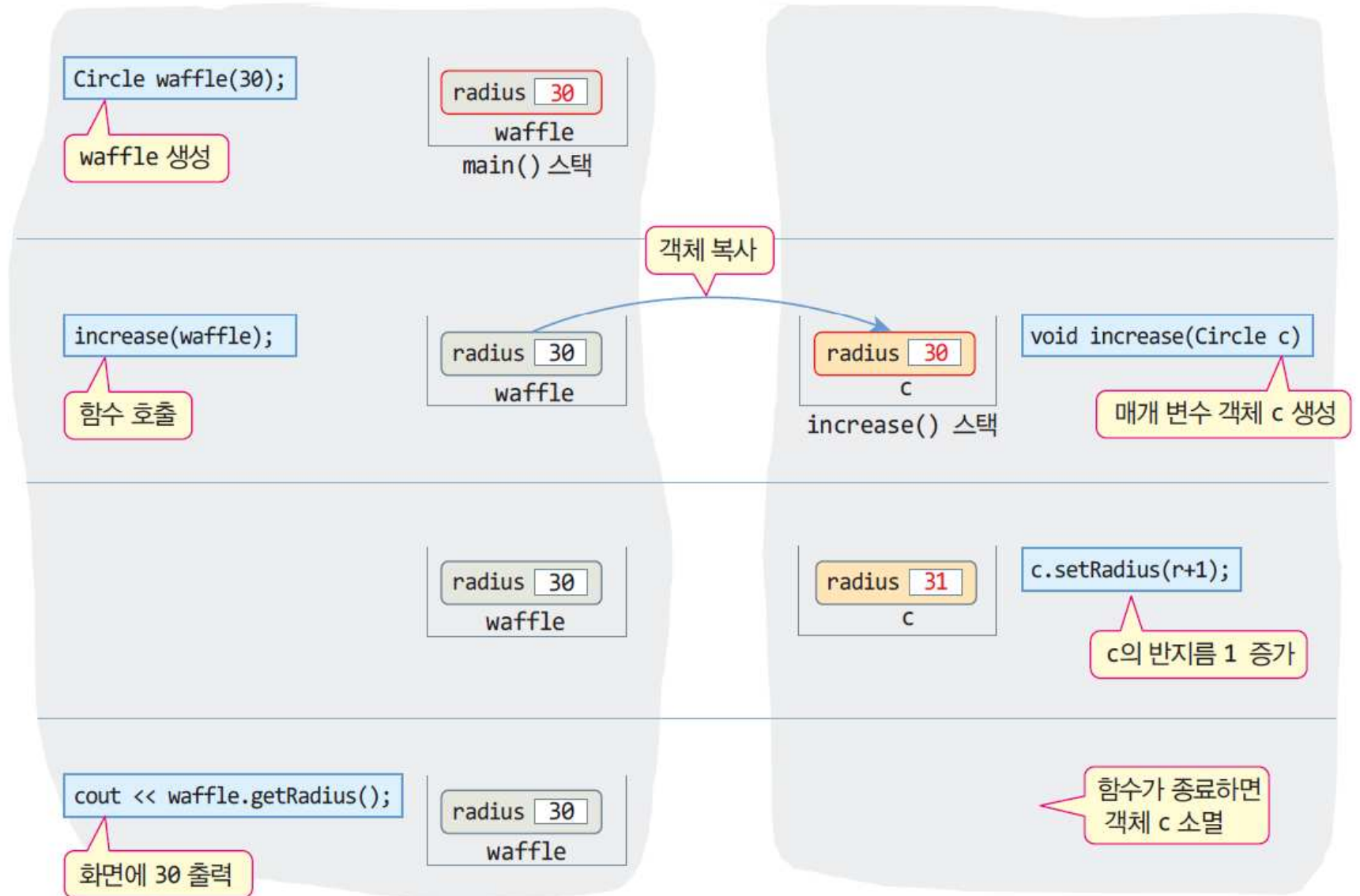
→ 실행 결과

30

```
int main() {  
    Circle waffle(30);  
    increase(waffle);  
    cout << waffle.getRadius() << endl;  
}
```

call by value

```
void increase(Circle c) {  
    int r = c.getRadius();  
    c.setRadius(r+1);  
}
```



예제

```
class Circle {
private:
    int radius;
public:
    Circle();
    Circle(int r);
    ~Circle();
    double getArea() { return 3.14*radius*radius; }
    int getRadius() { return radius; }
    void setRadius(int radius) { this->radius = radius; }
};
```

```
Circle::Circle() {
    radius = 1;
    cout << "생성자 실행 radius = " << radius << endl;
}
```

```
Circle::Circle(int radius) {
    this->radius = radius;
    cout << "생성자 실행 radius = " << radius << endl;
}
```

```
Circle::~~Circle() {
    cout << "소멸자 실행 radius = " << radius << endl;
}
```

```
void increase(Circle c) {
    int r = c.getRadius();
    c.setRadius(r+1);
}
```

```
int main() {
    Circle waffle(30);
    increase(waffle);
    cout << waffle.getRadius() << endl;
}
```

waffle의 내용이
그대로 c에 복사

waffle 생성

waffle 소멸

생성자 실행 radius = 30
소멸자 실행 radius = 31
30
소멸자 실행 radius = 30

c의 생성자
실행되지 않았음

c 소멸

함수에 주소에 의한 객체 전달

- 함수 호출시 객체의 주소만 전달
 - 함수의 매개 변수는 객체에 대한 포인터 변수로 선언
 - 함수 호출 시 생성자 소멸자가 실행되지 않는 구조

```
int main() {
    Circle waffle(30);
    increase(&waffle);
    cout << waffle.getRadius() ;
}
```

call by address

```
void increase(Circle *p) {
    int r = p->getRadius();
    p->setRadius(r+1);
}
```

Circle waffle(30);

waffle 생성

radius 30

waffle
main() 스택

waffle의 주소가
p에 전달

increase(&waffle);

함수호출

radius 30

waffle

p

increase() 스택

void increase(Circle *p)

매개 변수 포인터 p 생성

radius 31

waffle

p

p->setRadius(r+1);

waffle의 반지름 1 증가

cout << waffle.getRadius();

31이 화면에 출력됨

radius 31

waffle

함수가 종료하면
포인터 p 소멸

객체 치환 및 객체 리턴

■ 객체 치환

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

```
Circle c1(5);  
Circle c2(30);  
c1 = c2; // c2 객체를 c1 객체에 비트 단위 복사. c1의 반지름 30됨
```

- 치환된 두 객체는 현재 내용물만 같을 뿐 독립적인 공간 유지

■ 객체 리턴

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

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

예제 - 객체 리턴

```
class Circle
{
    int radius;

public:
    Circle() { radius = 1; }
    Circle(int radius) { this->radius = radius; }
    void setRadius(int radius) { this->radius = radius; }
    double getArea() { return 3.14*radius*radius; }
};

Circle getCircle() {
    Circle tmp(30);
    return tmp; // 객체 tmp을 리턴한다.
}

int main() {
    Circle c; // 객체가 생성된다. radius=1로 초기화된다.
    cout << c.getArea() << endl;

    c = getCircle();
    cout << c.getArea() << endl;
}
```

tmp 객체가 c에 복사된다. c의 radius는 30이 된다.

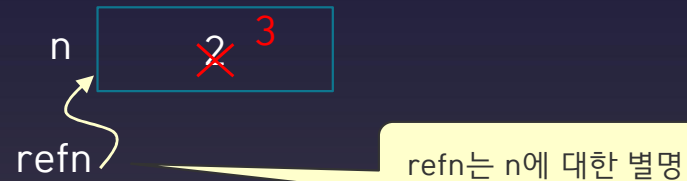
3.14
2826

참조 변수

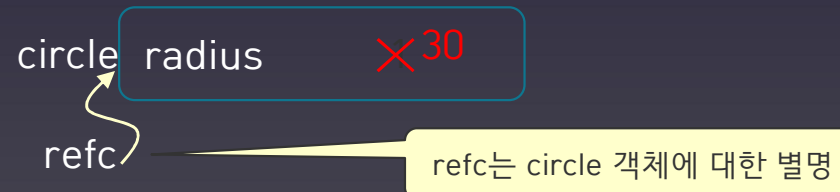
■ 참조 변수 선언

- 참조자 &의 도입
- 이미 존재하는 변수에 대한 다른 이름(별명)을 선언
 - 참조 변수는 이름만 생기며
 - 참조 변수에 새로운 공간을 할당하지 않는다.
 - 초기화로 지정된 기존 변수를 공유한다.

```
int n = 2;  
int &refn = n;  
  
refn = 3;
```



```
Circle circle;  
Circle &refc = circle;  
  
refc.setRadius(30);
```



예제 - 기본 타입 변수에 대한 참조

참조 변수 refn
선언

```
int main() {  
    cout << "i" << '\t' << "n" << '\t' << "refn" << endl;  
    int i = 1;  
    int n = 2;  
    int &refn = n; // 참조 변수 refn 선언. refn은 n에 대한 별명  
    n = 4;  
    refn++; // refn=5, n=5  
    cout << i << '\t' << n << '\t' << refn << endl;  
  
    refn = i; // refn=1, n=1  
    refn++; // refn=2, n=2  
    cout << i << '\t' << n << '\t' << refn << endl;  
  
    int *p = &refn; // p는 n의 주소를 가짐  
    *p = 20; // refn=20, n=20  
    cout << i << '\t' << n << '\t' << refn << endl;  
}
```

참조에 대한
포인터 변수
선언

i	n	refn
1	5	5
1	2	2
1	20	20

예제 - 객체에 대한 참조

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

class Circle {
    int radius;
public:
    Circle() { radius = 1; }
    Circle(int radius) { this->radius = radius; }
    void setRadius(int radius) { this->radius = radius; }
    double getArea() { return 3.14*radius*radius; }
};

int main() {
    Circle circle;
    Circle &refc = circle;
    refc.setRadius(10);
    cout << refc.getArea() << " " << circle.getArea();
}
```

circle 객체에 대한
참조 변수 refc 선언

참조에 의한 호출

- 참조를 가장 많이 활용하는 사례
- call by reference라고 부름
- 함수 형식
 - 함수의 매개 변수를 참조 타입으로 선언
 - 참조 매개 변수(reference parameter)라고 부름
 - 참조 매개 변수는 실인자 변수를 참조함
 - 참조매개 변수의 이름만 생기고 공간이 생기지 않음
 - 참조 매개 변수는 실인자 변수 공간 공유
 - 참조 매개 변수에 대한 조작은 실인자 변수 조작 효과

참조에 의한 호출 사례

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

```
void swap(int &a, int &b) {
    int tmp;

    tmp = a;
    a = b;
    b = tmp;
}
```

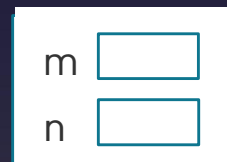
```
int main() {
    int m=2, n=9;
    swap(m, n);
    cout << m << ' ' << n;
}
```

참조 매개 변수를
보통 변수처럼 사용

함수가 호출되면 m, n에
대한 참조 변수 a, b가
생긴다.

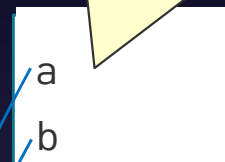
92

a, b는 m, n의 별명.
a, b 이름만 생성.
변수 공간 생기지 않음



main() 스택

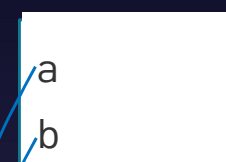
(1) swap() 호출 전



swap() 스택

main() 스택

(2) swap() 호출 직후



swap() 스택

main() 스택

(3) swap() 실행



main() 스택

(4) swap() 리턴 후

m, n이
변경됨

참조 매개변수가 필요한 사례

- 다음 코드에 어떤 문제가 있을까?
 - `average()` 함수의 작동
 - 계산에 오류가 있으면 0 리턴, 아니면 평균 리턴
 - 만일 `average()`가 리턴한 값이 0이라면?
 - 평균이 0인 거야? 아니면 오류가 발생한 거야?

```
int average(int a[], int size) {  
    if(size <= 0) return 0;  
    int sum = 0;  
    for(int i=0; i<size; i++) sum += a[i];  
    return sum/size;  
}
```

```
int x[ ]={1,2,3,4};  
int avg = average(x, 4);
```

// avg는 2

```
int x[ ]={1,2,3,4};  
int avg = average(x, -1);
```

// avg는 0

예제 - 참조 매개 변수로 평균 리턴

참조 매개 변수를 통해 평균을 리턴하고 리턴문을 통해서 함수의 성공 여부를 리턴하도록 average() 함수를 작성하라

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

```
bool average(int a[], int size, int& avg) {
    if(size <= 0)
        return false;
    int sum = 0;
    for(int i=0; i<size; i++)
        sum += a[i];
    avg = sum/size;
    return true;
}
```

참조 매개 변수 avg에 평균 값 전달

```
int main() {
    int x[] = {0,1,2,3,4,5};
    int avg;
    if(average(x, 6, avg)) cout << "평균은 " << avg << endl;
    else cout << "매개 변수 오류" << endl;

    if(average(x, -2, avg)) cout << "평균은 " << avg << endl;
    else cout << "매개 변수 오류 " << endl;
}
```

avg에 평균이 넘어오고,
average()는 true 리턴

avg의 값은 의미없고,
average()는 false 리턴

평균은 2
매개 변수 오류

예제 - 참조 매개 변수를 가진 함수

키보드로부터 반지름 값을 읽어
Circle 객체에 반지름을 설정하는
readRadius() 함수를 작성하라.

```
void readRadius(Circle &c) {  
    int r;  
    cout << "정수 값으로 반지름을 입력하세요>>";  
    cin >> r; // 반지름 값 입력  
    c.setRadius(r); // 객체 c에 반지름 설정  
}
```

```
#include <iostream>  
using namespace std;  
  
class Circle {  
    int radius;  
public:  
    Circle() { radius = 1; }  
    Circle(int radius) { this->radius = radius; }  
    void setRadius(int radius) { this->radius = radius; }  
    double getArea() { return 3.14*radius*radius; }  
};  
  
int main() {  
    Circle donut;  
    readRadius(donut);  
    cout << "donut의 면적 = " << donut.getArea() << endl;  
}
```

```
정수 값으로 반지름을 입력하세요>>3  
donut의 면적 = 28.26
```

참조 리턴

■ C 언어의 함수 리턴

- 함수는 반드시 값만 리턴
 - 기본 타입 값 : int, char, double 등
 - 포인터 값

■ C++의 함수 리턴

- 함수는 값 외에 참조 리턴 가능
- 참조 리턴
 - 변수 등과 같이 현존하는 공간에 대한 참조 리턴
 - 변수의 값을 리턴 하는 것이 아님

참조 리턴 함수

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
```

find()가 리턴한 공간에
'b' 문자 저장

```
char &ref = find();  
ref = 'M';    // c = 'M'
```

```
find() = 'b'; // c = 'b'가 됨
```

예제 - 간단한 참조 리턴

```
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;
}
```

Mike
Sike
Site

(1) `char name[] = "Mike";`

M	i	k	e	\0
---	---	---	---	----

 name

(2) `return s[index];`

공간에 대한
참조, 즉 익명
의 이름 리턴

M	i	k	e	\0
---	---	---	---	----

s[index]

(3) `find(name, 0) = 'S';`

S	i	k	e	\0
---	---	---	---	----

(4) `ref = 't';`

S	i	t	e	\0
---	---	---	---	----

실습 1

- 다음 결과가 나오도록 half() 함수를 작성하시오(단, 참조변수 활용).

```
int main( )  
{  
    double a=20.0;  
    half(a);  
    cout << a;  
}
```

10

실습 2

- 다음 결과가 나오도록 `increaseBy()`를 추가하시오.

```
class Circle
{
private:
    int radius;
public:
    Circle();
    Circle(int r);
    ~Circle() { }
    int getRadius() { return radius; }
    void setRadius(int radius) { this->radius = radius; }
    void show( ) {cout << "radius = "<< radius<< endl; }
};
Circle::Circle(): Circle(1){ }
Circle::Circle(int r) {
    radius = r;
    cout << "Constructor : " << radius << endl;
}
```

```
int main()
{
    Circle x(10), y(5);
    increaseBy(x, y);
    x.show( );
}
```

```
radius = 15;
```

