Ch09 클래스와 객체

C ++ 언어에서 객체 지향을 도입한 목적

- 소프트웨어 생산성 향상
  - 소프트웨어의 생명 주기 단축 문제 해결 필요
  - 기 작성된 코드의 재사용 필요
  - C++ 클래스 상속 및 객체 재사용으로 해결
- 실세계에 대한 쉬운 모델링
  - 과거의 소프트웨어
    - 수학 계산이나 통계 처리에 편리한 절차 지향 언어가 적합
  - 현대의 소프트웨어
    - 물체 혹은 객체의 상호 작용에 대한 묘사가 필요
    - 실세계는 객체로 구성된 세계
    - 객체를 중심으로 하는 객체 지향 언어 적합

C++ 언어의 주요한 설계 목적

- C 언어와의 호환성
  - C 언어의 문법 체계 계승
    - 소스 레벨 호환성 – 기존에 작성된 C 프로그램을 그대로 가져다 사용
    - 링크 레벨 호환성 – C 목적 파일과 라이브러리를 C++ 프로그램에서 링크
- 객체 지향 개념 도입
  - 캡슐화, 상속, 다형성
  - 소프트웨어의 재사용을 통해 생산성 향상
  - 복잡하고 큰 규모의 소프트웨어의 작성, 관리, 유지보수 용이
- 엄격한 타입 체크
  - 실행 시간 오류의 가능성을 줄임
  - 디버깅 편리
- 실행 시간의 효율성 저하 최소화
  - 실행 시간을 저하시키는 요소와 해결
    - 작은 크기의 멤버 함수 잦은 호출 가능성 -> 인라인 함수로 실행 시간 저하 해소

#include <iostream>

- <iostream> 헤더 파일
  - 표준 입출력을 위한 클래스와 객체, 변수 등이 선언됨
    - ios, istream, ostream, iostream 클래스 선언
    - cout, cin, <<, >> 등 연산자 선언

```
#include <iostream>
....
    std::cout << "Hello₩n";
    std::cout << "첫 번째 맛보기입니다.";
```

이름 충돌 사례

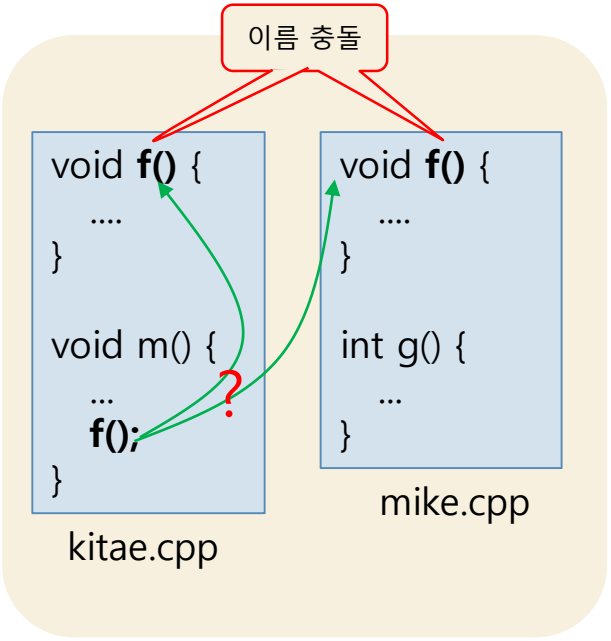
우리 아파트에 여러 명의 마이클이 산다.
마이클을 부를 때, 1동::마이클, 2동::마이클로 부른다.

namespace 개념

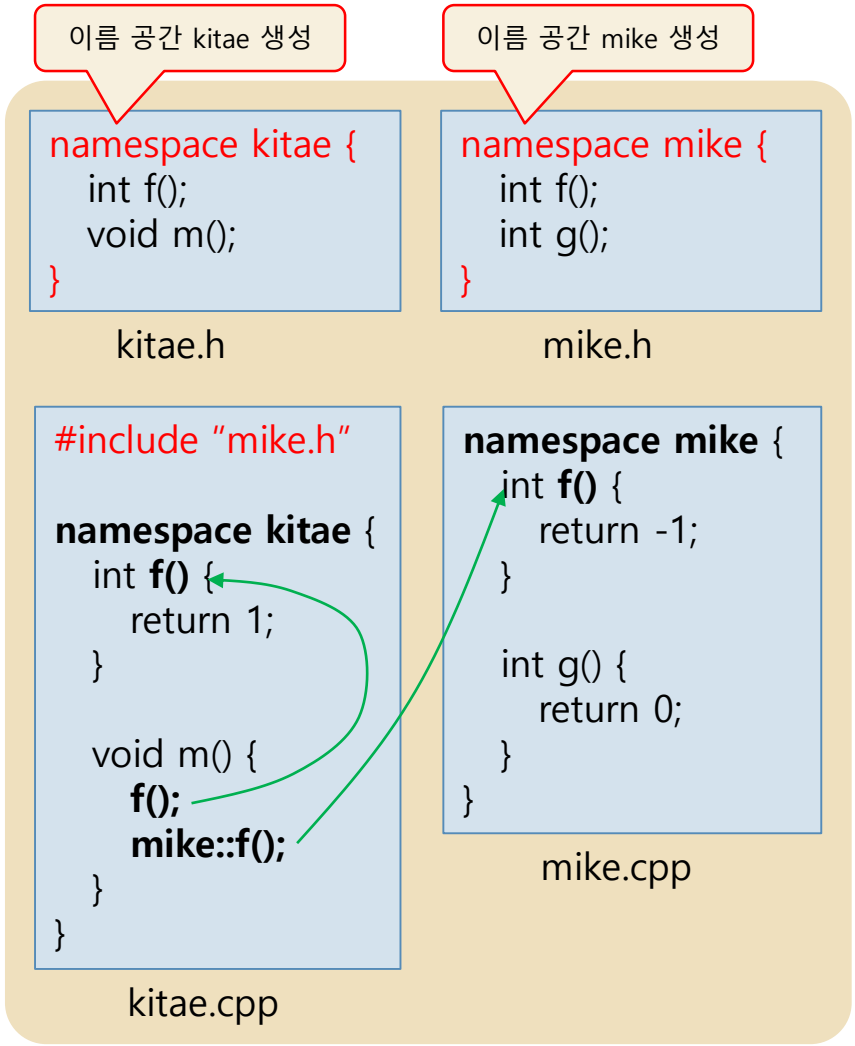
- 이름(identifier) 충돌이 발생하는 경우
  - 여러 명이 서로 나누어 프로젝트를 개발하는 경우
  - 오픈 소스 혹은 다른 사람이 작성한 소스나 목적 파일을 가져와서 컴파일하거나 링크하는 경우
  - 해결하는데 많은 시간과 노력이 필요
- namespace 키워드
  - 이름 충돌 해결
    - 2003년 새로운 C++ 표준에서 도입
  - 개발자가 자신만의 이름 공간을 생성할 수 있도록 함
    - 이름 공간 안에 선언된 이름은 다른 이름공간과 별도 구분
- 이름 공간 생성 및 사용(자세한 것은 부록 B 참고)

```
namespace kitae { // kitae 라는 이름 공간 생성
   ..... // 이 곳에 선언된 모든 이름은 kitae 이름 공간에 생성된 이름
}
```

  - 이름 공간 사용
    - 이름 공간 :: 이름



(a) kitae와 mike에 의해 작성된 소스를 합치면 f() 함수의 이름 충돌. 컴파일 오류 발생



(b) 이름 공간을 사용하여 f() 함수 이름의 충돌 문제 해결

std:: 란?

- std
  - C++ 표준에서 정의한 **이름 공간(namespace)** 중 하나
    - <iostream> 헤더 파일에 선언된 모든 이름: std 이름 공간 안에 있음
    - cout, cin, endl 등
  - std 이름 공간에 선언된 이름을 접근하기 위해 std:: 접두어 사용
    - std::cout, std::cin, std::endl
- std:: 생략
  - **using 지시어 사용**

```
using std::cout; // cout에 대해서만 std:: 생략
.....
cout << "Hello" << std::endl; // std::cout에서 std:: 생략
```

std:: 생략

```
using namespace std; // std 이름 공간에 선언된 모든 이름에 std:: 생략
.....
cout << "Hello" << endl; // std:: 생략
```

std:: 생략

std:: 생략

#include <iostream>과 std

- <iostream>이 통째로 std 이름 공간 내에 선언
  - <iostream> 헤더 파일을 사용하려면 다음 코드 필요

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

화면 출력

- cout과 << 연산자 이용

```
cout << "Hello₩n"; // 화면에 Hello를 출력하고 다음 줄로 넘어감
cout << "첫 번째 맛보기입니다.";
```

- cout 객체
  - **스크린** 출력 장치에 연결된 **표준** C++ 출력 스트림 객체
  - <iostream> 헤더 파일에 선언
  - std 이름 공간에 선언: **cout**으로 사용
- << 연산자
  - 스트림 삽입 연산자(stream insertion operator)
    - iostream 클래스에 구현됨
    - cout 객체에 연결된 화면에 출력
  - 여러 개의 << 연산자로 여러 값 출력

```
cout << "Hello₩n" << "첫 번째 맛보기입니다.";
```

<< 연산자 활용

- 문자열 및 기본 타입의 데이타 출력
  - bool, char, short, int, long, float, double 타입 값 출력

```
int n=3;
char c='#';
std::cout << c << 5.5 << '-' << n << "hello" << true;
```

#5.5-3hello1

  - 연산식뿐 아니라 함수 호출도 가능

```
std::cout << "n + 5 =" << n + 5;
std::cout << f(); // 함수 f()의 리턴값을 출력한다.
```

- 다음 줄로 넘어가기
  - '₩n'이나 endl 조작자 사용

```
std::cout << "Hello" << '₩n';
std::cout << "Hello" << std::endl;
```

예제 C++에서 출력

ch09_ex_01-1.cpp

```
#include <iostream>

int main()
{
   int n = 3;
   char c = '#';
   cout << c << 5.5 << '-' << n << "hello" << endl;
   cout << "n + 5 = " << n + 5 << '₩n';

   return 0;
}
```

ch09_ex_01-2.c

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void)
{
   int n = 3;
   char c = '#';
   printf("%c5.5-%dhello₩n", c, n );
   printf("n + 5 = %d₩n", n + 5);

   return 0;
}
```

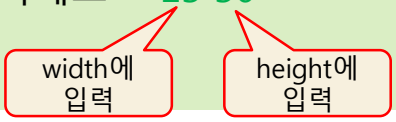
```
#5.5-3hello
n + 5 = 8
```

cin과 >> 연산자를 이용한 키 입력

- cin
  - 표준 입력 장치인 키보드를 연결하는 C++ 입력 스트림 객체
- >> 연산자
  - 스트림 추출 연산자(stream extraction operator)
    - C++ 산술 시프트 연산자(>>)가 <iostream> 헤더 파일에 스트림 추출 연산자로 재정의됨
    - 입력 스트림에서 값을 읽어 변수에 저장
  - 연속된 >> 연산자를 사용하여 여러 값 입력 가능

```
cout << "너비와 높이를 입력하세요>>";
cin >> width >> height;
cout << width << '₩n' << height << '₩n';
```

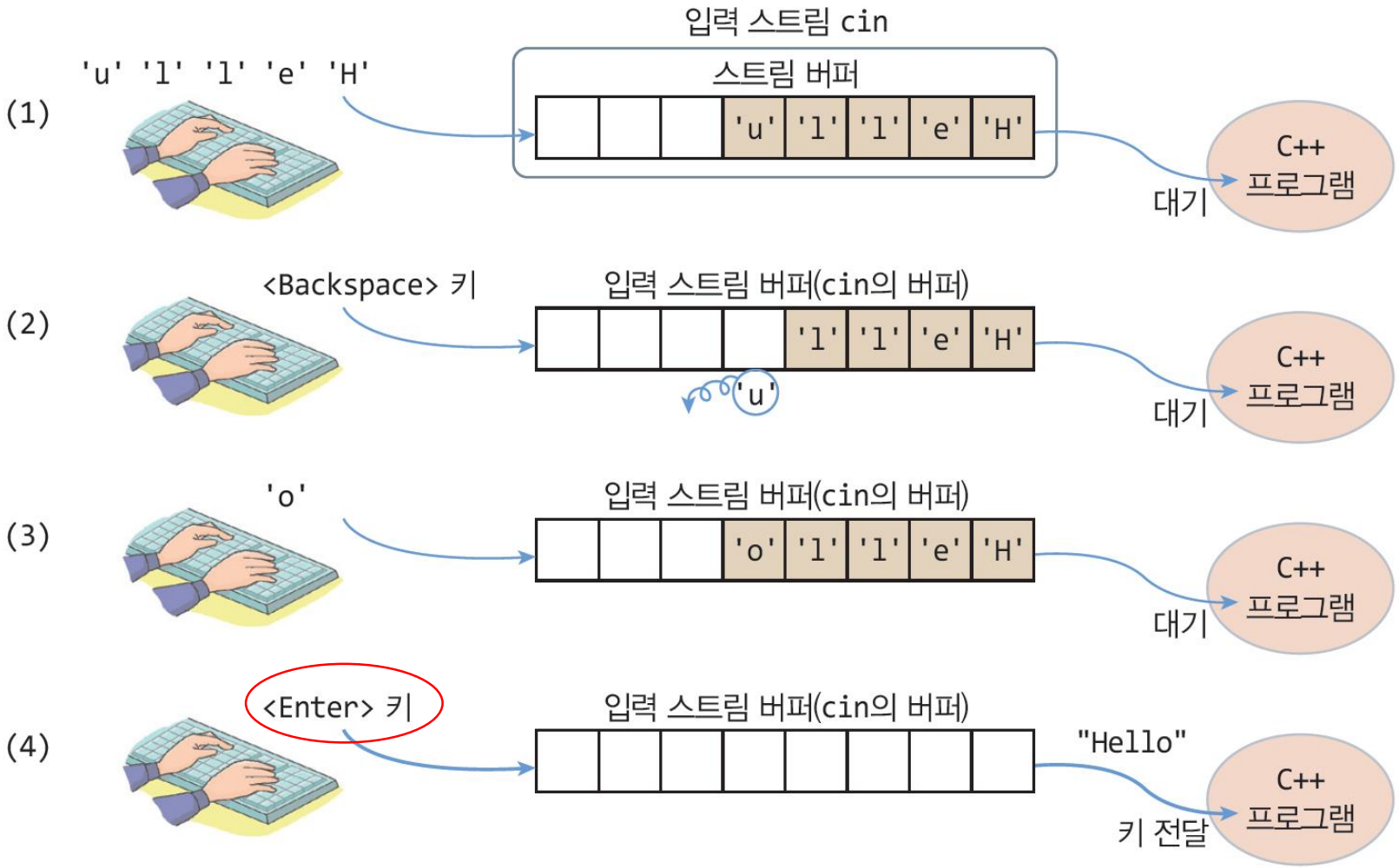
```
너비와 높이를 입력하세요>>23 36
23
36
```



<Enter> 키를 칠 때 변수에 값 전달

- cin의 특징
  - 입력 버퍼를 내장하고 있음
  - <Enter>키가 입력될 때까지 입력된 키를 입력 버퍼에 저장
    - 도중에 <Backspace> 키를 입력하면 입력된 키 삭제
- >> 연산자
  - <Enter>키가 입력되면 비로소 cin의 입력 버퍼에서 키 값을 읽어 변수에 전달

cin으로부터 키 입력 받는 과정(11.1절)



예제 C++에서 입력


```
#include <iostream>

int main()
{
   int n = 3;
   char c = '#';
   cout << c << 5.5 << '-' << n << "hello" << endl;
   cout << "n + 5 = " << n + 5 << '₩n';

   return 0;
}
```

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void)
{
   int n = 3;
   char c = '#';
   printf("%c5.5-%dhello₩n", c, n);
   printf("n + 5 = %d₩n", n + 5);

   return 0;
}
```

```
#5.5-3hello
n + 5 = 8
```

실행문 중간에 변수 선언

- C++의 변수 선언
  - 변수 선언은 아무 곳이나 가능

실행문 중간에 변수 선언

```
int width;
cin >> width; // 키보드로부터 너비를 읽는다.

cout << "높이를 입력하세요>>";

int height;
cin >> height; // 키보드로부터 높이를 읽는다.

// 너비와 높이로 구성되는 사각형의 면적을 계산한다.
int area = width*height;
cout << "면적은 " << area << "₩n"; // 면적을 출력하고 한 줄 띈다.
```

  - C++ 변수 선언 방식의 장점
    - 변수를 사용하기 직전 선언함으로써 변수 이름에 대한 타이핑 오류 줄임
  - C++ 변수 선언 방식의 단점
    - 선언된 변수를 일괄적으로 보기 힘듦
    - 코드 사이에 있는 변수 찾기 어려움

예제 - 입출력

ch09_ex_01.cpp

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

int main() {

   int n = 3;
   char c = '#';
   cout << c << 5.5 << '-' << n << "hello" << true << endl;
   cout << "n + 5 = " << n + 5 << endl;

   cout << "너비 : ";
   int width;
   cin >> width; // 키보드로부터 너비를 읽어 width 변수에 저장

   cout << "높이 : ";
   int height;
   cin >> height; // 키보드로부터 높이를 읽어 height 변수에 저장

   int area = width * height; // 사각형의 면적 계산
   cout << "면적은 " << area << endl; // 면적을 출력하고 다음 줄로 넘어감
}
```

C++ 문자열

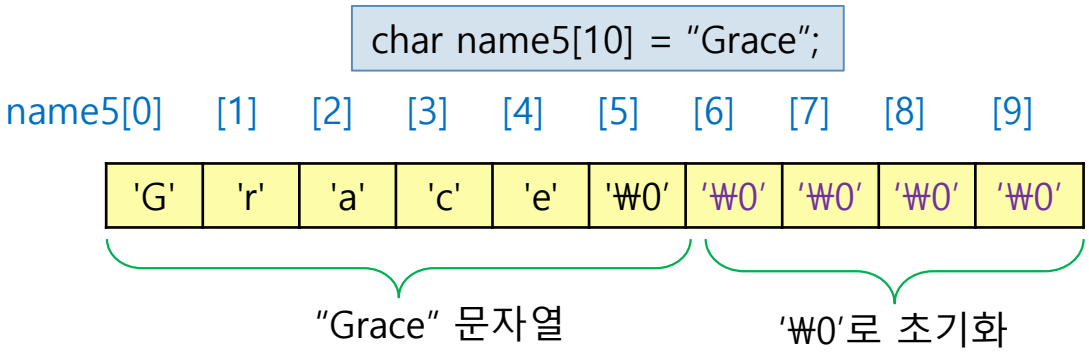
- C++의 문자열 표현 방식 : 2가지
  - **C-스트링 방식** – '₩0'로 끝나는 문자 배열

C-스트링 문자열

```
char name1[6] = {'G', 'r', 'a', 'c', 'e', '₩0'}; // name1은 문자열 "Grace"
```

단순 문자 배열

```
char name2[5] = {'G', 'r', 'a', 'c', 'e'}; // name2는 문자열이 아니고 단순 문자 배열
```



  - **string 클래스 이용**
    - <string> 헤더 파일에 선언됨
    - 다양한 멤버 함수 제공, 문자열 비교, 복사, 수정 등

C-스트링 방식으로 문자열 다루기

- C-스트링으로 문자열 다루기
  - C 언어에서 사용한 함수 사용 가능
    - strcmp(), strlen(), strcpy() 등
  - <cstring>이나 <string.h> 헤더 파일 include

```
#include <cstring> 또는
#include <string.h>
...
int n = strlen("hello");
```

  - <cstring> 헤더 파일을 사용하는 것이 바람직함
    - <cstring>이 C++ 표준 방식

cin을 이용한 문자열 입력

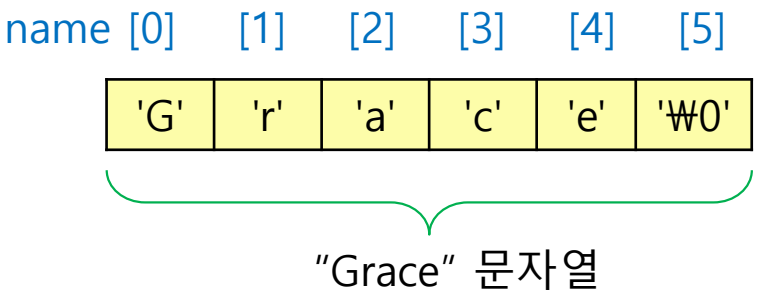
- 문자열 입력

입력된 값이 배열 크기를 넘어가면 안됨

```
char name[6]; // 5 개의 문자를 저장할 수 있는 char 배열
cin >> name; // 키보드로부터 문자열을 읽어 name 배열에 저장한다.
```

Grace

키 입력



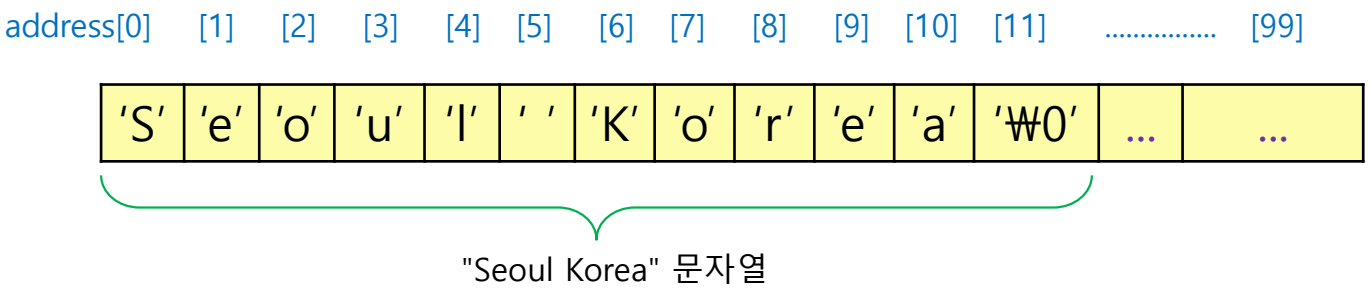
cin.getline()으로 공백이 낀 문자열 입력

- **공백이 낀 문자열**을 입력 받는 방법
- cin.getline(char buf[], int size, char delimitChar)
  - buf에 최대 size-1개의 문자 입력. 끝에 '₩0' 붙임
  - delimitChar를 만나면 입력 중단. 끝에 '₩0' 붙임
    - delimitChar의 디폴트 값은 '₩n'(<Enter>키)

```
char address[100];
cin.getline(address, 100, '₩n');
```

최대 99개의 문자를 읽어 address 배열에 저장. 도중에 <Enter> 키를 만나면 입력 중단

사용자가 'Seoul Korea<Enter>'를 입력할 때,



예제 - 키보드에서 문자열 입력 받고 출력

ch09_ex_02.cpp

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

int main() {

   char sname[6]; // 5 개의 문자를 저장할 수 있는 char 배열
   cout << " 이름1 : >>";
   cin >> sname;
   cout << "이름은 " << sname << "입니다." << endl;

   cout << "이름2 : ";
   char name[11]; // 한글은 5개 글자, 영문은 10까지 저장할 수 있다.
   cin >> name;
   cout << "이름은 " << name << "입니다."<<endl;

   cout << "주소 :";
   char address[100];
   cin.getline(address, 100, '₩n');
   cout << "주소는 " << address << "입니다₩n";

   return 0;
```

빈 칸 없이 키 입력해야 함

```
이름을 입력하세요>>마이클
이름은 마이클입니다
```

```
이름을 입력하세요>>마 이 클
이름은 마입니다
```

빈 칸을 만나면 문자열 입력 종료

C++에서 문자열을 다루는 string 클래스

- string 클래스
  - C++에서 강력 추천
  - C++ 표준 클래스
  - 문자열의 크기에 따른 제약 없음
    - string 클래스가 스스로 문자열 크기게 맞게 내부 버퍼 조절
  - 문자열 복사, 비교, 수정 등을 위한 다양한 함수와 연산자 제공
  - 객체 지향적
  - <string> 헤더 파일에 선언
    - #include <string> 필요
  - C-스트링보다 다루기 쉬움

표준 C++ 헤더 파일은 확장자가 없다

- 표준 C++에서 헤더 파일 확장자 없고, std 이름 공간 적시
  - #include <iostream>
  - using namespace std;

- 헤더 파일의 확장자 비교

언어	헤더 파일 확장자	사례	설명
C	.h	〈string.h〉	C/C++ 프로그램에서 사용 가능
C++	확장자 없음	〈cstring〉	using namespace std;와 함께 사용해야 함

예제 - 로그인

ch09_ex_03.cpp

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

int main() {
    char password[11];
    cout << "프로그램을 종료하려면 암호를 입력하세요." << endl;
    while(true) {
        cout << "암호>>";
        cin >> password;
        if(strcmp(password, "C++") == 0) {
            cout << "프로그램을 정상 종료합니다." << endl;
            break;
        }
        else
            cout << "암호가 틀립니다~~" << endl;
    }
}
```

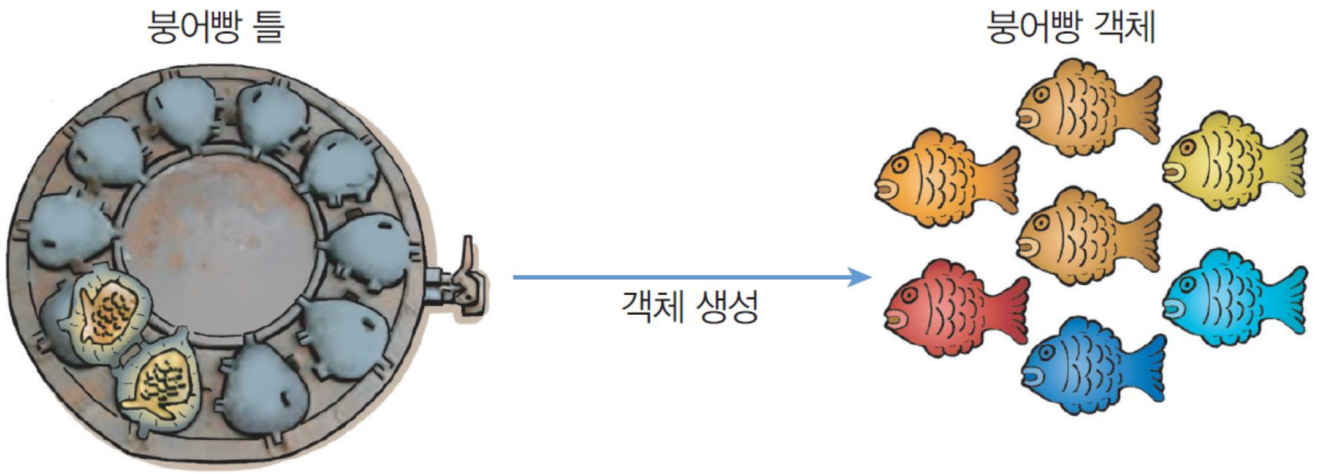
strcmp() 함수를 사용하기 위한 헤더 파일

```
프로그램을 종료하려면 암호를 입력하세요.
암호>>Java
암호가 틀립니다~~
암호>>C
암호가 틀립니다~~
암호>>C++
프로그램을 정상 종료합니다.
```

빈 칸 없이 키 입력해야 함

클래스와 객체

- 클래스 : 객체를 생성하기 위한 틀 또는 설계도
- 객체 : 클래스를 기반으로 생성된 실체



(a) 붕어빵 틀과 붕어빵 객체들

C++ 객체는 멤버 함수와 멤버 변수로 구성된다.

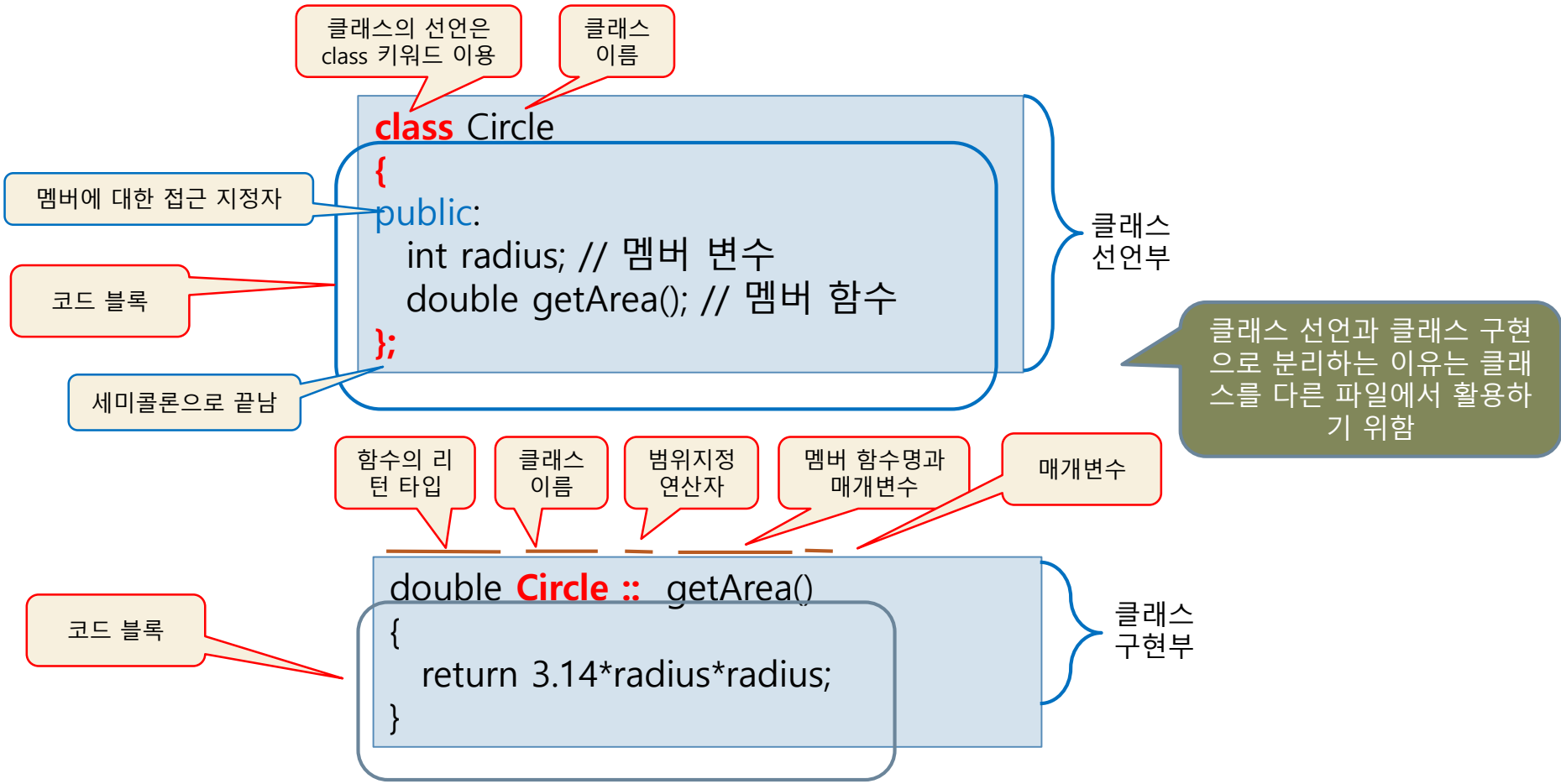
- 객체는 상태(state)와 행동(behavior)으로 구성
- TV 객체 사례
  - 상태
    - on/off 속성 – 현재 작동 중인지 표시
    - 채널(channel) – 현재 방송중인 채널
    - 음량(volume) – 현재 출력되는 소리 크기
  - 행동
    - 켜기(power on)
    - 끄기(power off)
    - 채널 증가(increase channel)
    - 채널 감소(decrease channel)
    - 음량 증가(increase volume)
    - 음량 줄이기(decrease volume)

C++ 클래스 만들기

- 클래스 작성
  - 멤버 변수와 멤버 함수로 구성
  - 클래스 선언부와 클래스 구현부로 구성
- 클래스 선언부(class declaration)
  - class 키워드를 이용하여 클래스 선언
  - 멤버 변수와 멤버 함수 선언
    - 멤버 함수는 원형(prototype) 형태로 선언
  - 멤버에 대한 접근 권한 지정
    - private, public, protected 중의 하나
    - 디폴트는 private
    - public : 다른 모든 클래스나 객체에서 멤버의 접근이 가능함을 표시
- 클래스 구현부(class implementation)
  - 클래스에 정의된 모든 멤버 함수 구현

클래스 만들기 설명

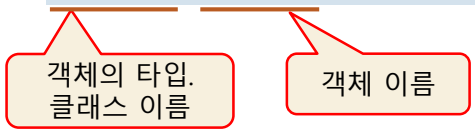
클래스 선언 형식



객체 생성 및 활용 설명

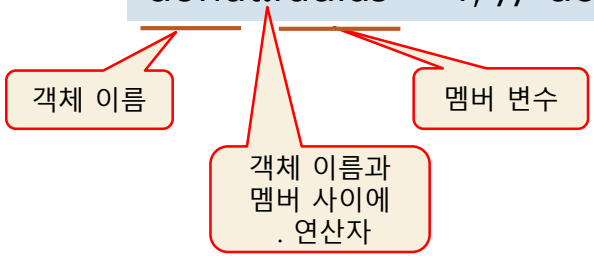
- 객체 이름 및 객체 생성

```
Circle  donut; // 이름이 donut 인 Circle 타입의 객체 생성
```



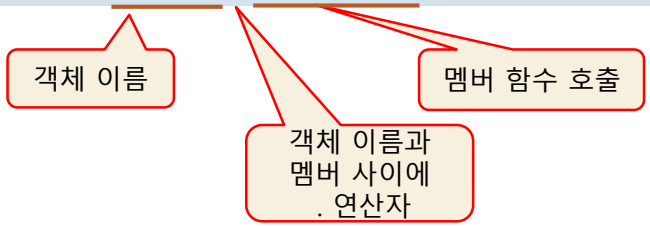
- 객체의 멤버 변수 접근

```
donut.radius = 1; // donut 객체의 radius 멤버 값을 1로 설정
```



- 객체의 멤버 함수 접근

```
double area = donut.getArea(); //donut 객체의 면적 알아내기
```



예제 - Circle 클래스의 객체 생성 및 활용

ch09_ex_04.cpp

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

class Circle {
public:
    int radius;
    double getArea();
};

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

int main() {
    Circle donut;
    donut.radius = 1; // donut 객체의 반지름을 1로 설정
    double area = donut.getArea(); // donut 객체의 면적 알아내기
    cout << "donut 면적은 " << area << endl;

    Circle pizza;
    pizza.radius = 30; // pizza 객체의 반지름을 30으로 설정
    area = pizza.getArea(); // pizza 객체의 면적 알아내기
    cout << "pizza 면적은 " << area << endl;
}
```

Circle 선언부

Circle 구현부

객체 donut 생성

donut의 멤버 변수 접근

donut의 멤버 함수 호출

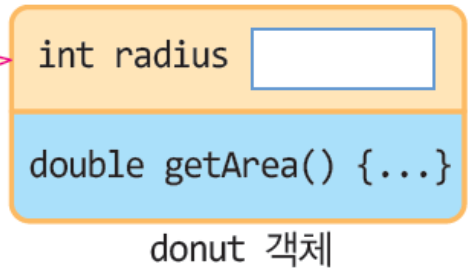
```
donut 면적은 3.14
pizza 면적은 2826
```

객체 이름과 생성, 접근 과정

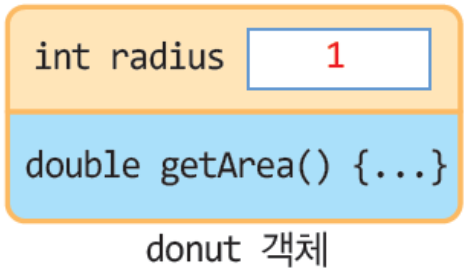

```
(1) Circle donut;
```

객체 이름

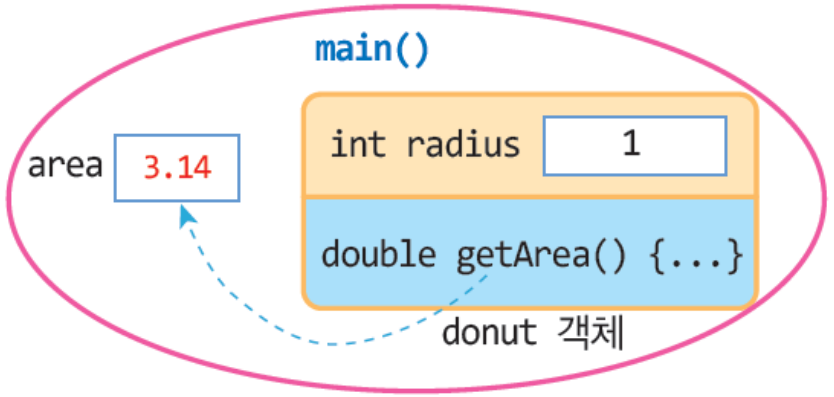
객체가 생성되면
메모리가 할당된다.




```
(2) donut.radius = 1;
```




```
(3) double area = donut.getArea();
```



문제 - Rectangle 클래스 만들기

다음 main() 함수가 잘 작동하도록 너비(width)와 높이(height)를 가지고 면적 계산 기능을 가진Rectangle 클래스를 작성하고 전체 프로그램을 완성하라.

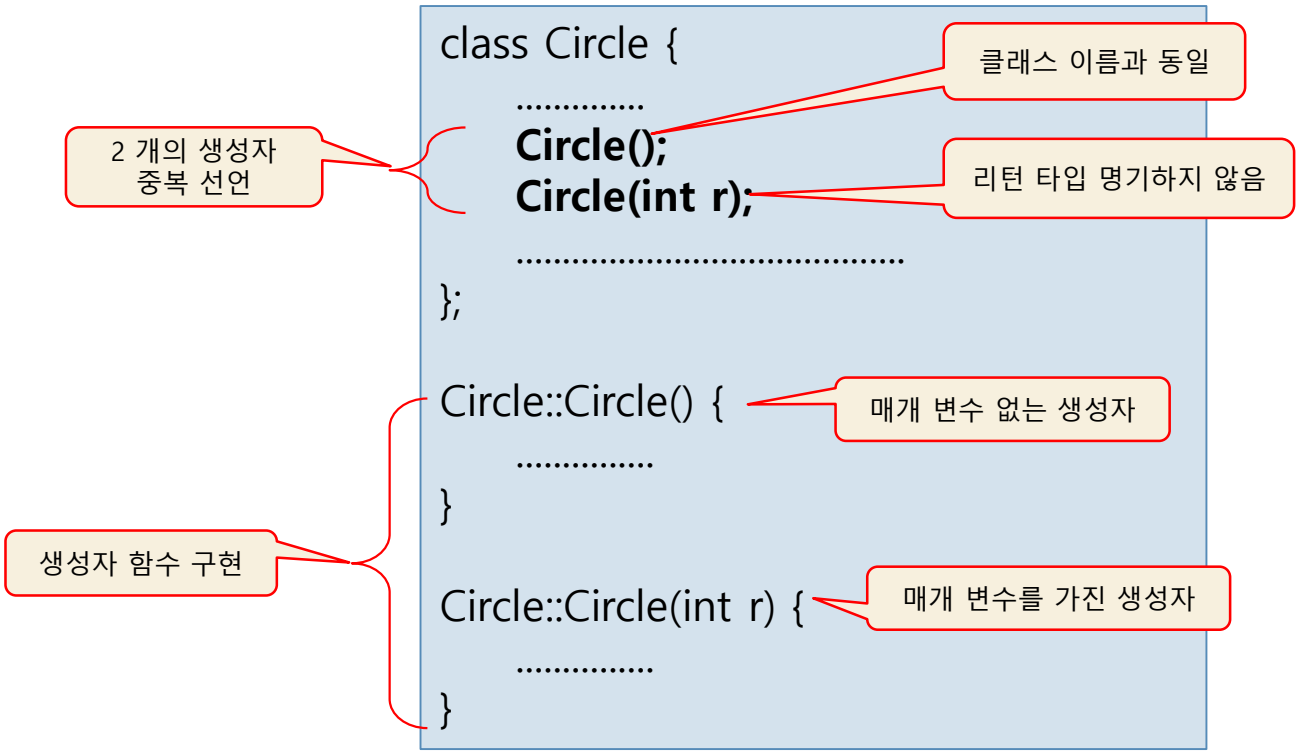
ch09_ex_05.cpp

```
int main() {
    Rectangle rect;
    rect.width = 3;
    rect.height = 5;
    cout << "사각형의 면적은 " << rect.getArea() << endl;
}
```

```
사각형의 면적은 15
```

생성자

- 생성자(constructor)
  - 객체가 생성되는 시점에서 자동으로 호출되는 멤버 함수
  - 클래스 이름과 동일한 멤버 함수



생성자 함수의 특징

- 생성자의 목적
  - 객체가 생성될 때 객체가 필요한 초기화를 위해
    - 멤버 변수 값 초기화, 메모리 할당, 파일 열기, 네트워크 연결 등
- 생성자 이름
  - 반드시 클래스 이름과 동일
- 생성자는 리턴 타입을 선언하지 않는다.
  - 리턴 타입 없음. void 타입도 안됨
- 객체 생성 시 오직 한 번만 호출
  - 자동으로 호출됨. 임의로 호출할 수 없음. 각 객체마다 생성자 실행
- 생성자는 중복 가능
  - 생성자는 한 클래스 내에 여러 개 가능
  - 중복된 생성자 중 하나만 실행
- 생성자가 선언되어 있지 않으면 기본 생성자 자동으로 생성
  - 기본 생성자 – 매개 변수 없는 생성자
  - 컴파일러에 의해 자동 생성

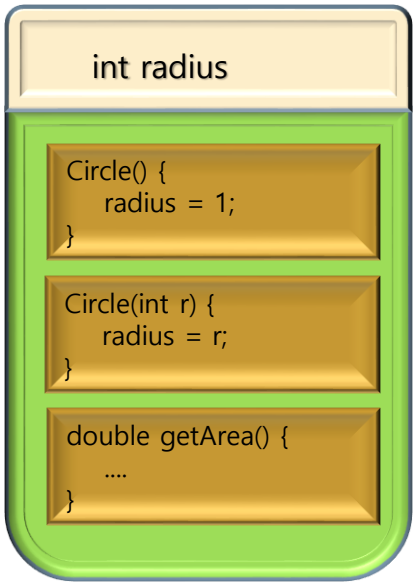
기본 생성자

- 기본 생성자란?
  - 클래스에 생성자가 하나도 선언되어 있지 않은 경우, 컴파일러가 대신 삽입해주는 생성자
  - 매개 변수 없는 생성자
  - 디폴트 생성자라고도 부름

```
class Circle {
   .....
   Circle(); // 기본 생성자
};
```

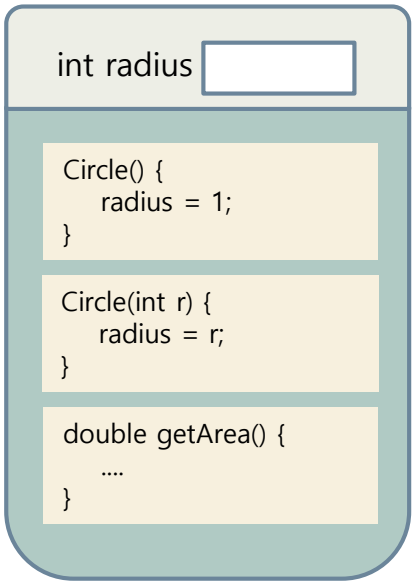
객체 생성 및 생성자 실행 과정

Circle donut;



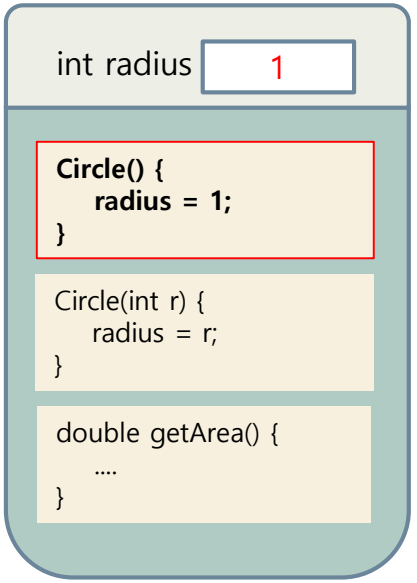
Circle 클래스

❶ 객체 공간 할당



donut 객체

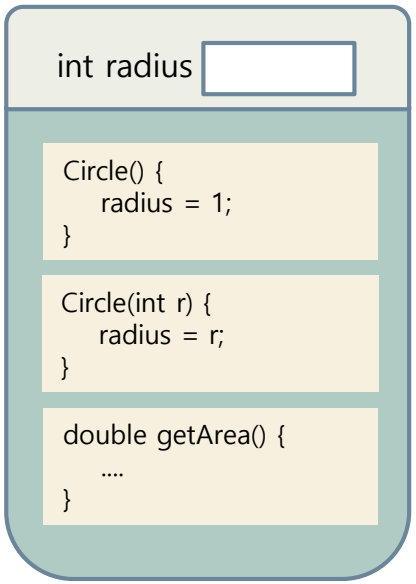
❷ 생성자 실행



donut 객체

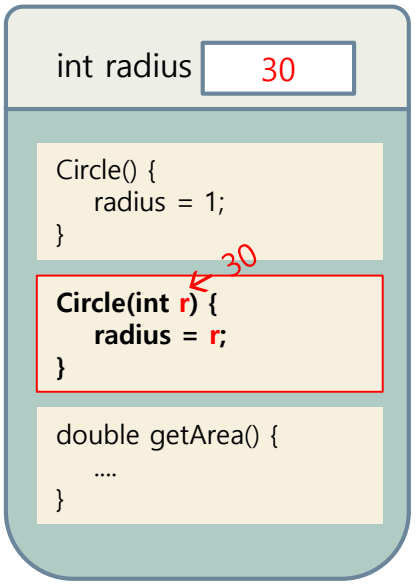
Circle pizza(30);

❶ 객체 공간 할당



pizza 객체

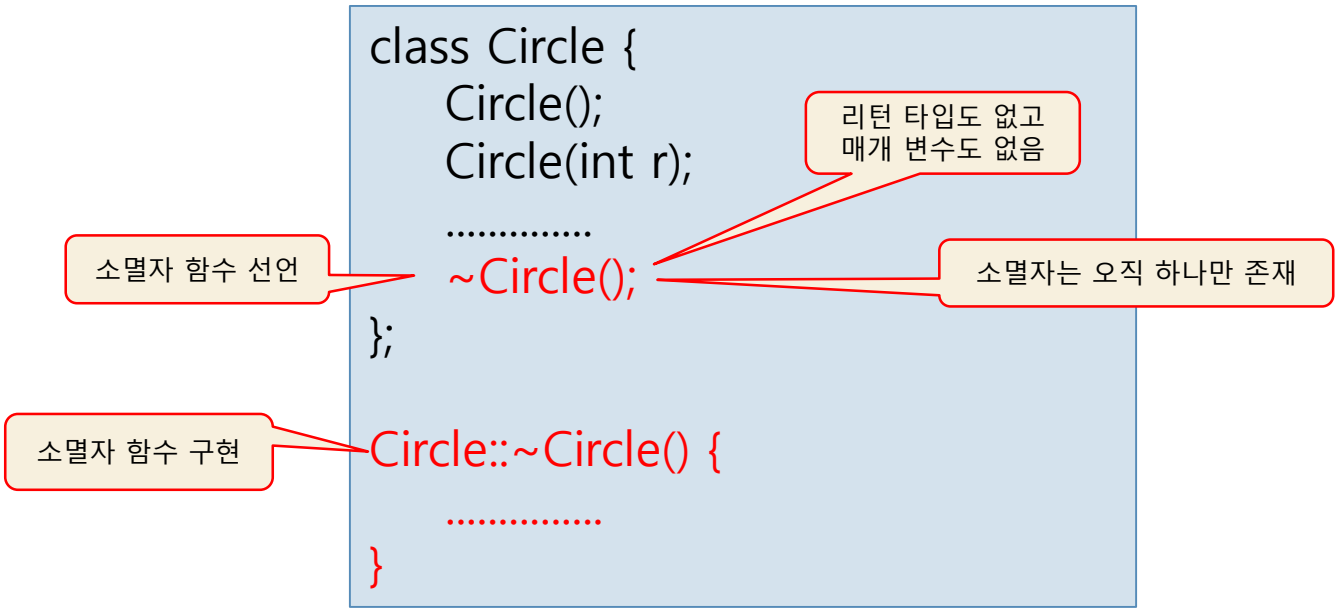
❷ 생성자 실행



pizza 객체

소멸자

- 소멸자
  - 객체가 소멸되는 시점에서 자동으로 호출되는 함수
    - 오직 한번만 자동 호출, 임의로 호출할 수 없음
    - 객체 메모리 소멸 직전 호출됨



소멸자 특징

- 소멸자의 목적
  - 객체가 사라질 때 마무리 작업을 위함
  - 실행 도중 동적으로 할당 받은 메모리 해제, 파일 저장 및 닫기, 네트워크 닫기 등
- 소멸자 함수의 이름은 클래스 이름 앞에 ~를 붙인다.
  - 예) Circle::~Circle() { ... }
- 소멸자는 리턴 타입이 없고, 어떤 값도 리턴하면 안됨
  - 리턴 타입 선언 불가
- 중복 불가능
  - 소멸자는 한 클래스 내에 오직 한 개만 작성 가능
  - 소멸자는 매개 변수 없는 함수
- 소멸자가 선언되어 있지 않으면 기본 소멸자가 자동 생성
  - 컴파일러에 의해 기본 소멸자 코드 생성
  - 컴파일러가 생성한 기본 소멸자 : 아무 것도 하지 않고 단순 리턴

예제 - Circle 클래스에 소멸자 작성 및 실행

ch09_ex_05.cpp

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

class Circle {
public:
    int radius;

    Circle();
    Circle(int r);
    ~Circle(); // 소멸자
    double getArea();
};


Circle::Circle() {
    radius = 1;
    cout << "반지름 " << radius << " 원 생성" << endl;
}

Circle::Circle(int r) {
    radius = r;
    cout << "반지름 " << radius << " 원 생성" << endl;
}

Circle::~Circle() {
    cout << "반지름 " << radius << " 원 소멸" << endl;
}
```

```
double Circle::getArea() {
   return 3.14*radius*radius;
}

int main() {
    Circle donut;
    Circle pizza(30);

    return 0;
}
```

main() 함수가 종료하면 main() 함수의 스택에 생성된 pizza, donut 객체가 소멸된다.

```
반지름 1 원 생성
반지름 30 원 생성
반지름 30 원 소멸
반지름 1 원 소멸
```

객체는 생성의 반대 순으로 소멸된다.

접근 지정자

- 캡슐화의 목적
  - 객체 보호, 보안
  - C++에서 객체의 캡슐화 전략
    - 객체의 상태를 나타내는 데이터 멤버(멤버 변수)에 대한 보호
    - 중요한 멤버는 다른 클래스나 객체에서 접근할 수 없도록 보호
    - 외부와의 인터페이스를 위해서 일부 멤버는 외부에 접근 허용

- 멤버에 대한 3 가지 접근 지정자
  - private
    - 동일한 클래스의 멤버 함수에만 제한함
  - public
    - 모든 다른 클래스에 허용
  - protected
    - 클래스 자신과 상속받은 자식 클래스에만 허용

```
class Sample {
private:
   // private 멤버 선언
public:
   // public 멤버 선언
protected:
   // protected 멤버 선언
};
```

접근 지정자


```
class Circle {
public:
    int radius;
    Circle();
    Circle(int r);
    double getArea();
};


Circle::Circle() {
    radius = 1;
}
Circle::Circle(int r) {
    radius = r;
}
```

멤버 변수 보호받지 못함

```
int main() {
    Circle waffle;
    waffle.radius = 5;
}
```

노출된 멤버는 마음대로 접근. 나쁜 사례

(a) 멤버 변수를 public으로 선언한 나쁜 사례

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

Circle::Circle() {
    radius = 1;
}
Circle::Circle(int r) {
    radius = r;
}
```

멤버 변수 보호받고 있음

```
int main() {
    Circle waffle(5); // 생성자에서 radius 설정
    waffle.radius = 5; // private 멤버 접근 불가
}
```

(b) 멤버 변수를 private으로 선언한 바람직한 사례

예제 – 접근 지정자

ch09_ex_06.cpp

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

class Circle {
private:
    int radius;
public:
    double getArea();
    int getRadius();
    void setRadius(int r);
};

double Circle::getArea() {
    return 3.14*radius*radius;
}
void Circle::setRadius(int r) {
    radius = r;
}
int Circle::getRadius() {
    return radius;
}
```

```
int main() {
    int d_r = 1;
    Circle donut;
    donut.setRadius(d_r);
    double area = donut.getArea();
    cout << "donut 면적은 " << area << endl;

    Circle pizza;
    int p_r = 30;
    pizza.setRadius(p_r);
    area = pizza.getArea();
    cout << "pizza 면적은 " << area << endl;
}
```

바람직한 C++ 프로그램 작성법

- 클래스를 헤더 파일과 cpp 파일로 분리하여 작성
  - 클래스마다 분리 저장
  - 클래스 선언 부
    - 헤더 파일(.h)에 저장
  - 클래스 구현 부
    - cpp 파일에 저장
    - 클래스가 선언된 헤더 파일 include
  - main() 등 전역 함수나 변수는 다른 cpp 파일에 분산 저장
    - 필요하면 클래스가 선언된 헤더 파일 include
- 목적
  - 클래스 재사용

예제 3-3의 소스를 헤더 파일과 cpp 파일로 분리하여 작성한 사례

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

Circle.h

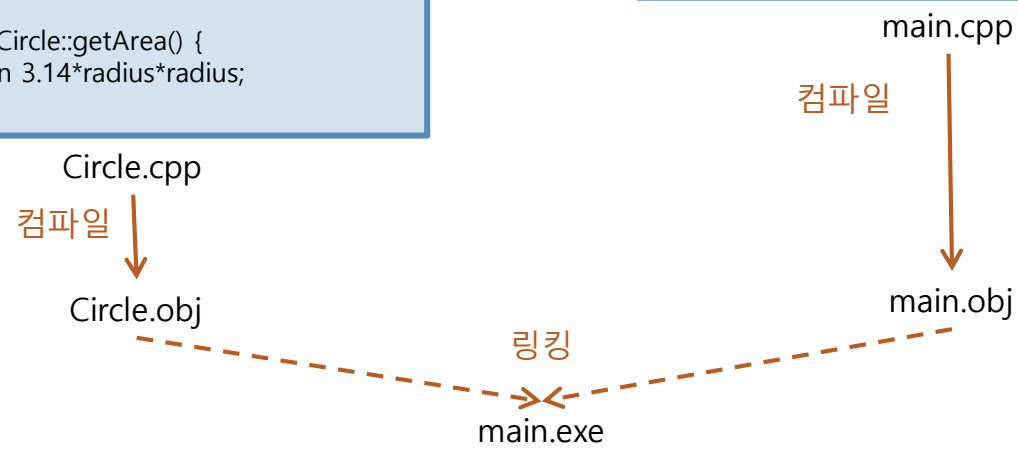
```
#include <iostream>
using namespace std;

#include "Circle.h"

Circle::Circle() {
    radius = 1;
    cout << "반지름 " << radius;
    cout << " 원 생성" << endl;
}

Circle::Circle(int r) {
    radius = r;
    cout << "반지름 " << radius;
    cout << " 원 생성" << endl;
}

double Circle::getArea() {
    return 3.14*radius*radius;
}
```

Circle.cpp

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

#include "Circle.h"

int main() {
    Circle donut;
    double area = donut.getArea();
    cout << "donut 면적은 ";
    cout << area << endl;

    Circle pizza(30);
    area = pizza.getArea();
    cout << "pizza 면적은 ";
    cout << area << endl;
}
```

main.cpp

```
반지름 1 원 생성
donut 면적은 3.14
반지름 30 원 생성
pizza 면적은 2826
```



문제 - 헤더 파일과 cpp 파일로 분리하기

아래의 소스를 헤더 파일과 cpp 파일로 분리하여 재작성하라.

ch09_ex_07

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

class Adder { // 덧셈 모듈 클래스
   int op1, op2;
public:
   Adder(int a, int b);
   int process();
};

Adder::Adder(int a, int b) {
   op1 = a; op2 = b;
}

int Adder::process() {
   return op1 + op2;
}
```

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

void Calculator::run() {
   cout << "두 개의 수를 입력하세요>>";
   int a, b;
   cin >> a >> b; // 정수 두 개 입력
   Adder adder(a, b); // 덧셈기 생성
   cout << adder.process(); // 덧셈 계산
}

int main() {
   Calculator calc; // calc 객체 생성
   calc.run(); // 계산기 시작
}
```

```
두 개의 수를 입력하세요>>5 -20
-15
```

문제 - Circle 클래스의 객체 생성 및 활용

ch09_ex_04.cpp

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

class Rectangle { // Rectangle 클래스 선언부
public:
   int width;
   int height;
   int getArea(); // 면적을 계산하여 리턴하는 함수
};

int Rectangle::getArea() { // Rectangle 클래스 구현부
   return width*height;
}

int main() {
   Rectangle rect;
   rect.width = 3;
   rect.height = 5;
   cout << "사각형의 면적은 " << rect.getArea() << endl;
}
```

```
사각형의 면적은 15
```

문제 - Rectangle 클래스 만들기

ch09_ex_06.cpp

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

class Rectangle {
public:
    int width, height;

    Rectangle();
    Rectangle(int w, int h);
    Rectangle(int length);
    bool isSquare();
};

Rectangle::Rectangle() {
    width = height = 1;
}

Rectangle::Rectangle(int w, int h) {
    width = w; height = h;
}

Rectangle::Rectangle(int length) {
    width = height = length;
}

// 정사각형이면 true를 리턴하는 멤버 함수
bool Rectangle::isSquare() {
    if(width == height) return true;
    else return false;
}
```

```
int main() {
    Rectangle rect1;
    Rectangle rect2(3, 5);
    Rectangle rect3(3);

    if(rect1.isSquare()) cout << "rect1은 정사각형이다." << endl ;
    if(rect2.isSquare()) cout << "rect2는 정사각형이다." << endl;
    if(rect3.isSquare()) cout << "rect3는 정사각형이다." << endl;
}
```

3 개의 생성자가 필요함

```
rect1은 정사각형이다.
rect3는 정사각형이다.
```

문제 - 헤더 파일과 cpp 파일로 분리하기

Adder.h

```
#ifndef ADDER_H
#define ADDER_H

class Adder { // 덧셈 모듈 클래스
    int op1, op2;
public:
    Adder(int a, int b);
    int process();
};

#endif
```

Calculator.h

```
#ifndef CALCULATOR_H
#define CALCULATOR_H

class Calculator { // 계산기 클래스
public:
    void run();
};

#endif
```

ch09_ex_07

Adder.cpp

```
#include "Adder.h"

Adder::Adder(int a, int b) {
    op1 = a; op2 = b;
}

int Adder::process() {
    return op1 + op2;
}
```

Calculator.cpp

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

#include "Calculator.h"
#include "Adder.h"

void Calculator::run() {
    cout << "두 개의 수를 입력하세요>>";
    int a, b;
    cin >> a >> b; // 정수 두 개 입력
    Adder adder(a, b); // 덧셈기 생성
    cout << adder.process(); // 덧셈 계산
}
```

main.cpp

```
#include "Calculator.h"

int main() {
    Calculator calc; // calc 객체 생성
    calc.run(); // 계산기 시작
}
```

```
두 개의 수를 입력하세요>>5 -20
-15
```