원리를 알면 IT가 맛있다
IT COOKBOOK

IT CookBook, C++ 하이킹 객체지향과 만나는 여행

[강의교안 이용 안내]

C++ 하이킹

객체지향과 만나는 여행

Chapter 10. 클래스와 객체

한빛아카데미 Hanbit Academy, Inc.

목차

학습목표

- 객체지향 프로그래밍에 대해서 이해한다.
- 클래스를 정의하고 객체를 생성해서 사용하는 방법을 익힌다.
- 클래스 멤버인 멤버변수와 멤버함수에 대해서 살펴본다.
- 캡슐화를 위한 접근지정자에 대해서 학습한다.

클래스의 선언

- C++에서 클래스를 선언하는 예약어는 class다. class는 자료를 추상화해서 사용자 정의 자료형으로 구현할 수 있게 하는 C++의 도구다.

클래스 선언	클래스 멤버함수 정의

class 클래스명 {
  접근 지정자 :
  자료형 멤버변수; → 멤버변수 선언
 	
  접근 지정자 :
  자료형 멤버함수(); → 멤버변수 원형 정의
};
	자료형 클래스명::멤버함수()
	{
	  …
	  …
	}

- 클래스는 크게 클래스 선언과 클래스 멤버함수 정의로 구성되어 있다. 클래스 선언에는 멤버변수와 멤버함수 원형을 정의한다. 멤버함수 정의는 클래스 선언 밖에서 따로 이루어진다.

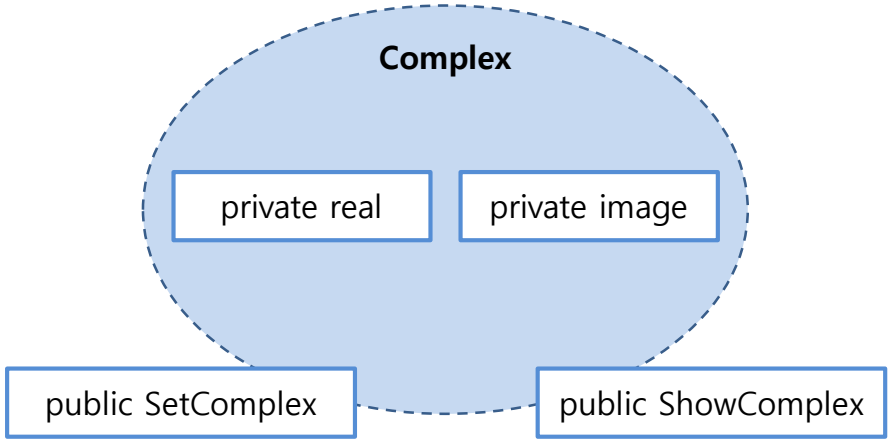
[표 10-1] 접근 지정자

구분	현재 클래스 내	현재 클래스 밖
private	o	x
public	o	o

- private : 해당 멤버가 속한 클래스의 멤버함수에서만 사용 가능하며, 캡슐화(데이터 은닉)된다.
- public : 객체를 사용할 수 있는 범위라면 어디서나 접근 가능한 공개된 멤버로, 주로 private 멤버를 해당 클래스 외부에서 사용하도록 하기 위한 멤버함수를 정의할 때 사용한다.

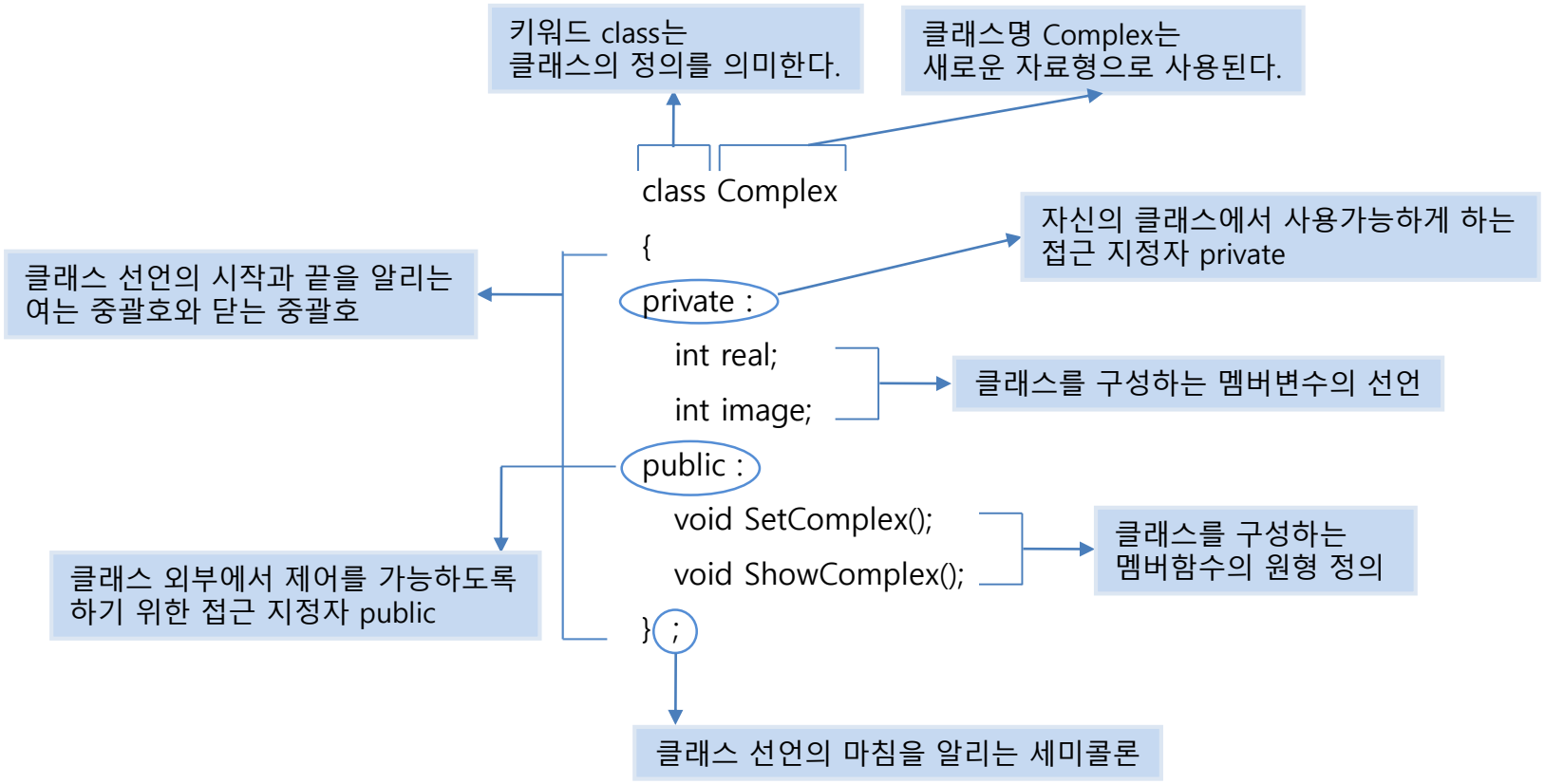
01 클래스의 이해

- 복소수를 Complex라는 이름의 클래스로 설계해 보자.
- 클래스를 새로운 자료형으로 설계할 때에는 단순히 데이터의 저장이라는 측면뿐만 아니라 데이터를 처리할 함수(멤버함수) 제공도 고려해야 한다. 그러므로 다음과 같이 클래스는 새로운 자료형이 되고 데이터는 멤버변수가, 데이터를 처리할 함수는 멤버함수가 된다.
- 새로운 자료형(클래스) = 데이터의 저장(멤버변수) + 데이터를 처리할 함수(멤버함수)
- 실수를 저장하기 위한 멤버변수를 real, 허수를 저장하기 위한 멤버변수를 image라고 하자. 데이터 은닉에 입각해서 멤버변수의 접근 지정자는 private로 선언하는 것이 일반적이다.
- 두 멤버변수에 값을 설정(SetComplex)하거나 알아내기(ShowComplex) 위한 멤버함수를 클래스 외부에서 접근할 수 있도록 공개한다.



[그림 10-1] Complex 클래스 설계

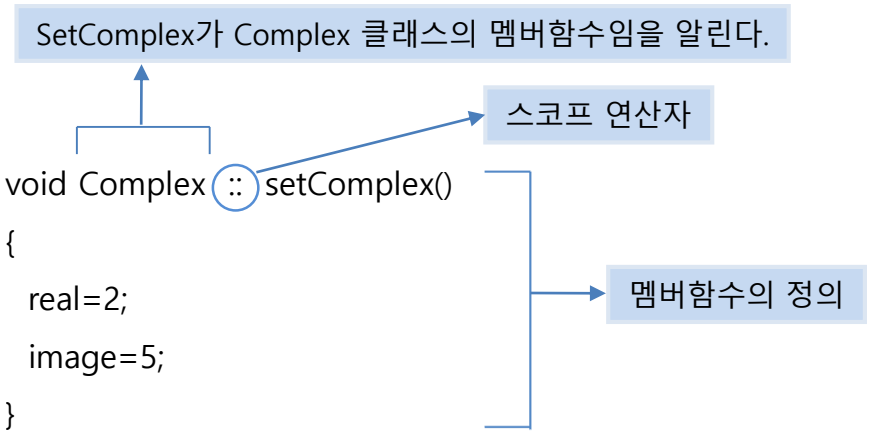
01 클래스의 이해



[그림 10-2] Complex 클래스 구현

클래스의 멤버함수 구현

- 멤버함수의 특징
  1. 멤버함수를 정의할 때 그 함수가 어느 클래스에 소속되는지 나타내려면 함수명 앞에 클래스명을 명시해야 한다. 이때 클래스명 다음에 스코프 연산자(::)를 사용한다.
  2. 클래스의 멤버함수를 정의하는 목적은 클래스 내에 정의된 private 멤버에 접근해서 이를 다루기 위해서다. 그래서 멤버함수를 '메소드'라고도 한다.



[그림 10-3] SetComplex 에 대한 정의

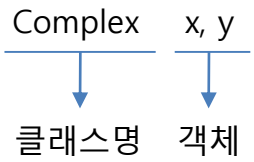
객체 선언과 멤버 참조

- 객체는 클래스를 실체(instance)화한 것이다. 그래서 객체를 '인스턴스'라고도 한다.

객체 선언 기본 형식

```
[class] 클래스명 객체명1, 객체명2, ..., 객체명n;
```

- Complex 클래스를 이용해서 x와 y라는 객체 2개를 생성한 예를 보자. 객체 x, y는 멤버변수 real과 image를 포함하고 있는데, 이 두 멤버변수는 private로 선언되어 있으므로 이를 사용하려고 SetComplex와 ShowComplex를 멤버함수로 제공한다.



01 클래스의 이해

클래스 멤버의 접근 방법

- 멤버를 사용하려면 구조체처럼 .과 -> 같은 멤버참조 연산자를 사용한다.

. 연산자를 이용한 클래스 멤버 접근 방법

```
객체명.멤버변수;
객체명.멤버함수();
```

- 복소수를 구현한 Complex 클래스로 객체를 생성한 후 멤버함수를 호출하는 예를 보자.

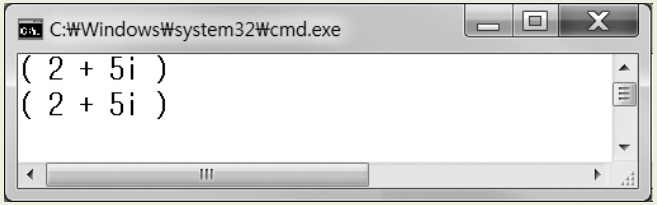
```
Complex x, y;
x.SetComplex();
y.SetComplex();
```

- SetComplex는 멤버함수이므로 함수명만으로 호출할 수 없다. 반드시 함수명 앞에 객체명과 멤버참조 연산자를 함께 기술해야 한다.

```
SetComplex();      // 에러 ------------------ 잘못된 예(X)
x.SetComplex();    // 객체명.멤버함수(); ----올바른 예(O)
```

예제 10-1. 복소수를 클래스로 설계하기(10_01.cpp)


```
01 #include <iostream>
02 using namespace std;
03 class Complex
04 {
05 private :
06 int real;
07 int image;
08 public :
09 void SetComplex();
10 void ShowComplex();
11 };
12
13 void Complex::SetComplex()
14 {
15 real=2;
16 image=5;
17 }
18 void Complex::ShowComplex()
19 {
20 cout<<"( " <<real <<" + " <<image << "i )" <<endl ;
21 }
22 void main()
23 {
24 Complex x, y;
25
26 x.SetComplex();
27 x.ShowComplex();
28 y.SetComplex();
29 y.ShowComplex();
30 }
```

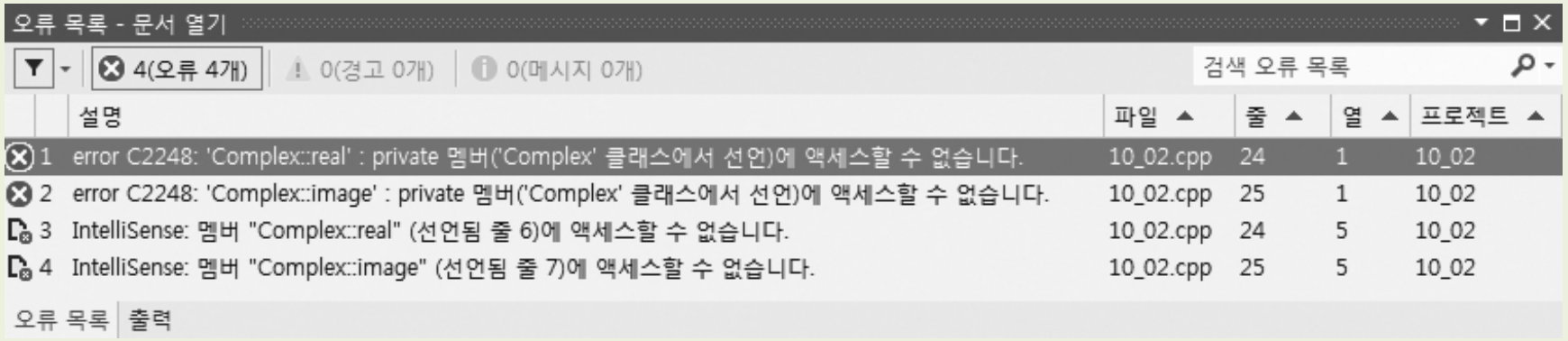


클래스의 접근 지정자, private/public

- private 접근 지정자
  - ❶ 접근 지정자가 생략되면 기본(default)으로 private가 적용된다. 하지만 private:를 명시적으로 기술할 수도 있다.
  - ❷ private 멤버의 사용 범위는 소속된 클래스 내의 멤버함수로 국한된다.
  - ❸ 일반적으로 멤버변수를 private로 설정한다.
- public 접근 지정자
  - ❶ public 멤버로 지정하려면 public:을 명시적으로 기술해야 한다.
  - ❷ 클래스 내의 멤버함수에서는 물론 객체가 선언되어 있는 영역이라면 어디서든지 객체명 다음에 멤버참조 연산자(.)로 연결해서 멤버함수를 사용할 수 있다.
  - ❸ private 멤버변수를 처리하기 위한 목적으로 작성하는 멤버함수는 일반적으로 public 멤버로 설정한다.

예제 10-2. private 멤버 성격 파악하기(10_02.cpp)

책의 소스코드 참고

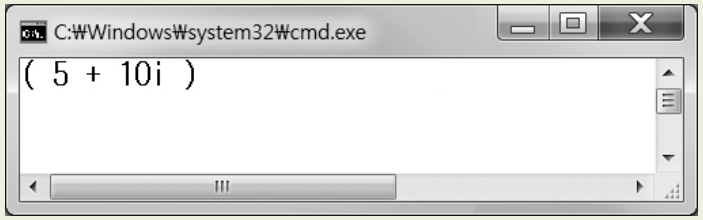


01 클래스의 이해

- 예제[10-2]의 에러를 해결하는 방법을 알아보자.
  - ❶ main 함수에서 real과 image를 사용할 수 있도록 이 두 멤버변수의 접근 지정자를 public으로 변경한다.
  - ❷ 위의 방법은 문제를 간단히 해결하지만 private 접근 지정자를 public으로 변경하면 데이터 은닉이라는 객체지향 개념을 위배하므로 최선의 방법이 아니다. C++가 나오게 된 배경이 객체지향 개념이므로 데이터 은닉이라는 특성을 지키려면 멤버변수의 접근 지정자는 private로 유지해야 한다. 그럼 private 멤버변수의 값을 변경하려면 어떻게 해야할까? 멤버변수에 직접 접근할 수 없으므로 이런 작업을 할 수 있도록 멤버함수를 추가하면 된다. 멤버변수의 값을 변경하는 로직을 멤버함수에 기술하고 특정 멤버변수의 값을 변경하기 위해 멤버함수를 호출하면 된다.

예제 10-3. private 멤버를 다루기 위한 멤버함수 추가하기(10_03.cpp)

책의 소스코드 참고



클래스 내부에 멤버함수 정의하기

- 인라인 함수
  - 인라인 함수는 매크로 함수와 실행 원리가 동일하다, 그래서 함수를 호출할 때 생기는 시간 지연을 줄일 수 있다는 장점이 있지만, 함수가 긴 경우에는 프로그램 코드가 그만큼 길어져서 프로그램이 커진다는 단점이 있다. 그러므로 인라인 함수는 주로 정의가 짧을 때 사용하고, 인라인 함수를 정의할 때는 함수 선언 앞에 inline이라는 예약어를 써 준다.

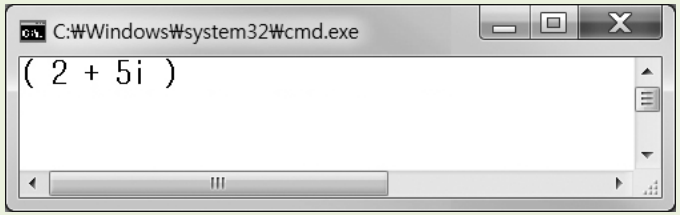
인라인 함수 기본 형식

```
inline 자료형 함수명 (매개변수리스트)
{
변수 선언;
문장;
return (결과값);
}
```

- 자동 인라인 함수
  - 멤버함수의 정의가 아주 짧으면, 클래스 선언 내부에 함수를 직접 정의할 수도 있다. 그리고 클래스 내부에 정의된 함수는 함수 선언 앞에 inline이 없어도 자동으로 인라인 함수가 된다.

예제 10-4. 인라인 함수 사용하기(10_04.cpp)


```
01 #include <iostream>
02 using namespace std;
03 class Complex
04 {
05 private :
06 int real;
07 int image;
08 public :
09 void SetComplex()
10 {
11 real=2;
12 image=5;
13 }
14 void ShowComplex();
15 };
16
17
18 inline void Complex::ShowComplex()
19 {
20 cout<<"( " <<real <<" + " <<image << "i )" <<endl ;
21 }
22 void main()
23 {
24 Complex x;
25
26 x.SetComplex();
27 x.ShowComplex();
28 }
```



const 상수와 const 멤버함수

- #define문으로 정의한 매크로 상수보다 상수를 정의하는 더 쉬운 방법은 예약어 const를 사용하는 것이다. 일반 변수 선언과 유사하지만 반드시 초깃값을 주어야 한다.

const 상수 기본 형식

```
const 자료형 변수명 = 초깃값;
```

- 형식은 변수 선언과 같지만 키워드 const 다음의 변수명은 변수로서의 역할을 못한다. 변수 선언 시 초깃값으로 지정한 값이 영원히 그 변숫값이 되어 상수로 사용된다.
- const 상수는 매크로 상수와 동일한 목적으로 사용된다. 하지만 매크로 상수는 자료형을 명시하지 않지만 const 상수는 자료형을 명시한다는 점이 다르다.

- Set으로 새롭게 값을 설정하는 멤버함수는 매개변수로 넘어 온 값으로 멤버변수 값을 변경해야 한다. 그러므로 SetXXX 함수들은 const를 붙일 수 없다. 만약 const를 붙이면 컴파일 에러가 발생한다.

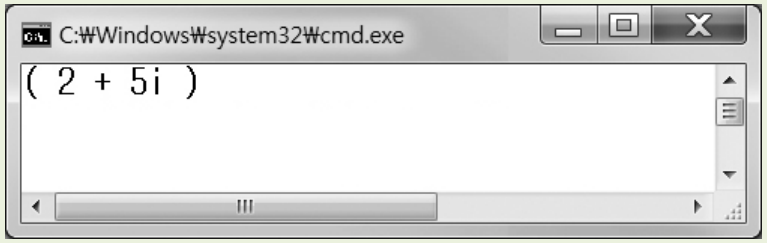
```
void Complex::SetComplex() const
{
  real=2;
  image=5;
}
```

- GetXXX 함수들은 멤버변수 값을 알려주기 위한 용도로 사용된다. GetXXX 함수 내부에서 멤버변수 값을 변경하지 말아야 하므로 const 예약어로 함수를 정의할 때 기술한다.

```
void Complex::ShowComplex() const
{
  cout<<"( " << real << " + " << image << "i )" <<endl;
}
```

예제 10-5. const 멤버함수 사용하기(10_05.cpp)

책의 소스코드 참고



함수의 오버로딩

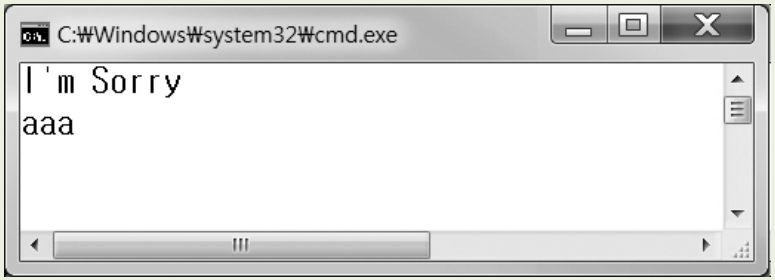
- 함수의 시그너처
  - 함수를 구분하기 위한 구성요소를 시그너처(signature)라 한다.
    ❶ 함수명
    ❷ 매개변수의 개수
    ❸ 매개변수의 자료형
  - 함수의 다형성(polymorphism)은 객체지향 프로그램의 특징 중 하나다. poly는 많다는 뜻이고 morphism은 형(형태)을 의미한다. 함수의 오버로딩도 다형성의 한 형태다. 왜냐하면 함수를 호출할 때에는 동일한 접근 방식으로 호출하지만 다양한 결과를 얻을 수 있기 때문이다.
- 함수의 오버로딩이 필요한 이유
- 같은 의미로 사용하는 함수를 모두 다른 이름으로 정의한다면 프로그램을 작성할 때마다 함수명을 개별적으로 외워야 할 것이다. 예를 들어 절댓값을 구하는 함수는 표준 함수로 다음과 같이 정의되어 있다.

  • int abs(int x);
  • double fabs(double x);
  • long int labs(long int x);

- 만일 정수값에 대해서 절댓값을 구하려면 abs 함수를 사용해야 하고 fabs 함수는 실수형의 절댓값을 구할 때 사용한다. 또 labs 함수는 정수 중에서도 long int형의 절댓값을 구할 때 사용한다. 함수가 이렇게 많다면 프로그램을 작성하는 작업이 더욱 어려울 것이다. 이럴 때 오버로딩을 통해 동일한 목적을 수행하기 위해 사용하는 함수명을 동일하게 사용할 수 있으므로 프로그램을 훨씬 쉽게 작성할 수 있다.

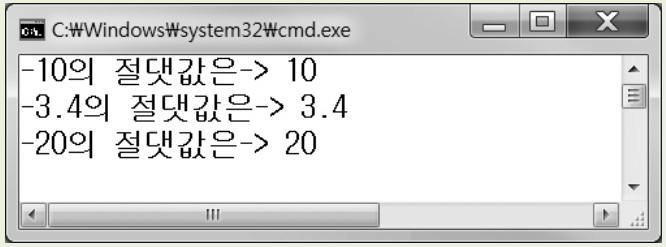
예제 10-6. 함수의 오버로딩 살펴보기(10_06.cpp)

책의 소스코드 참고



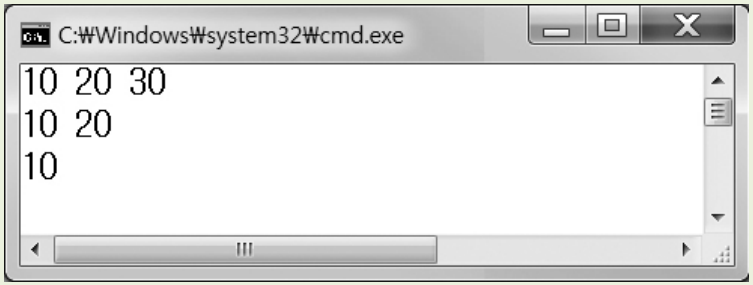
예제 10-8. 매개변수의 자료형이 다른 함수의 오버로딩을 이용해서 절댓값 구하기(10_08.cpp)

책의 소스코드 참고



예제 10-9. 매개변수의 개수가 다른 함수의 오버로딩 살펴보기(10_09.cpp)


```
01 #include <iostream>
02 using namespace std;
03 void print(int x, int y, int z)
04 {
05 cout<<x<<" "<<y<<" "<<z<<endl;
06 }
07 void print(int x, int y)
08 {
09 cout<<x<<" "<<y<<endl;
10 }
11 void print(int x)
12 {
13 cout<<x<<endl;
14 }
15 void main()
16 {
17 print(10, 20, 30);
18 print(10, 20);
19 print(10);
20 }
```



함수의 기본 매개변수

- 함수의 형식 매개변수에 값을 설정할 수 있는데, 이렇게 형식 매개변수에 값을 설정해 놓은 것을 기본 매개변수라 한다.

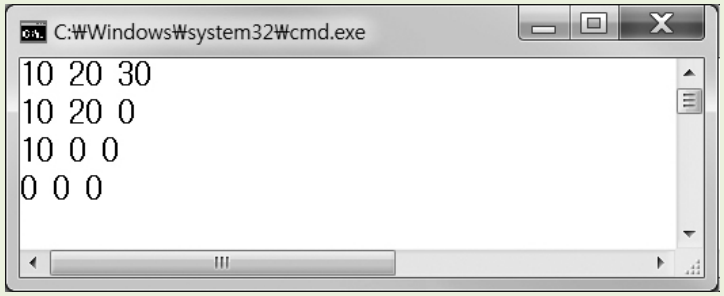
```
void print(int x=10, int y=20, int z=30)
{
  cout<<x<<" "<<y<<" "<<z<<endl;
}
```

- 매개변수에 '기본(default)'이라는 용어를 붙인 이유는 기본값을 설정해 놓은 매개변수라는 의미를 주기 위해서다. 기본 매개변수는 함수를 호출할 때 대응되는 실 매개변수를 모두 기술하지 않아도 자동으로 적용되어 사용할 수 있는 매개변수다. 기본 매개변수를 사용할 때 주의할 점은 함수의 정의와 함수의 선언을 따로 할 경우, 기본 매개변수를 함수의 선언부에서 지정해줘야 한다는 점이다. 만약 함수의 선언 없이 함수를 main 함수 위에 정의했다면 기본 매개변수를 함수의 정의에 설정한다. 기본 매개변수를 갖는 함수는 다양한 형태로 호출할 수 있다.

```
prn(4, 5, 6);
prn(4, 5);
prn(4);
prn();
```

예제 10-10. 함수의 매개변수에 기본값 지정하기(10_10.cpp)


```
01 #include <iostream>
02 using namespace std;
03 void print(int x=0, int y=0, int z=0);
04 void main()
05 {
06 print(10, 20, 30);
07 print(10, 20);
08 print(10);
09 print();
10 }
11 void print(int x, int y, int z)
12 {
13 cout<<x<<" "<<y<<" "<<z<<endl;
14 }
```



02 생성자와 소멸자

생성자의 의미와 특징

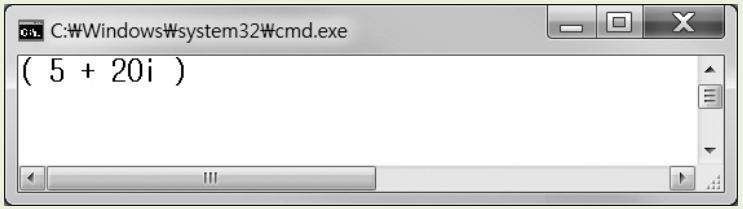
- 기본 자료형으로 변수를 선언할 때 선언과 동시에 값을 주는 것을 초기화라 한다. 클래스도 객체를 생성할 때 기본 자료형처럼 초깃값을 줄 수 있어야 하는데, 이를 가능하게 하는 것이 생성자다.
- 생성자의 특징
  ❶ 생성자는 특별한 멤버함수다.
  ❷ 생성자명은 클래스명과 동일하다.
  ❸ 생성자는 자료형(반환값의 유형)을 지정하지 않는다.
  ❹ 생성자의 호출은 명시적이지 않다.
  ❺ 생성자는 객체를 선언(생성)할 때 컴파일러에 의해 자동으로 호출된다.
  ❻ 객체의 초기화란 멤버변수의 초기화를 의미한다.
- 프로그래머가 생성자를 명시적으로 만들지 않으면 C++ 컴파일러는 매개변수가 없는 생성자를 자동으로 만들어 놓는다. 컴파일러가 만든 매개변수가 없는 생성자는 아무런 일도 하지 않는데, 이러한 생성자를 '기본 생성자(default constructor)'라고 한다. 기본 생성자를 호출하는 예를 보자.

```
Complex x;
```

- 컴파일러에서 자동으로 주는 생성자를 사용하면 아무 일도 하지 않기 때문에 멤버변수에 쓰레기 값이 저장된다. 그러므로 객체가 생성될 때 멤버변수에 특정한 값을 저장하려면 프로그래머가 매개변수가 없는 생성자를 재정의해 주어야 한다.

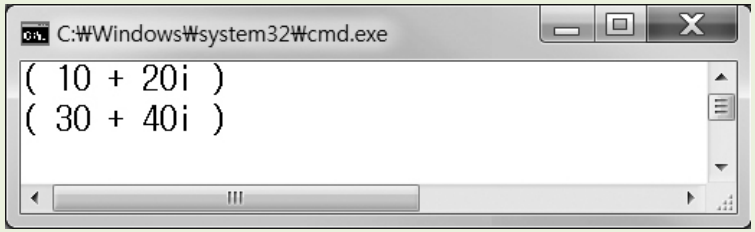
예제 10-11. 매개변수가 없는 생성자 작성하기(10_11.cpp)


```
01 #include <iostream>
02 using namespace std;
03 class Complex
04 {
05 private :
06 int real;
07 int image;
08 public :
09 Complex();
10 void ShowComplex() const;
11 };
12
13 Complex::Complex()
14 {
15 real=5;
16 image=20;
17 }
18
19 void Complex::ShowComplex() const
20 {
21 cout<<"( " <<real <<" + " <<image << "i )" <<endl ;
22 }
23
24 void main()
25 {
26 Complex x;
27 x.ShowComplex();
28 }
```



예제 10-12. 다양한 초깃값의 매개변수를 이용하는 생성자 작성하기(10_12.cpp)

책의 소스코드 참고



02 생성자와 소멸자

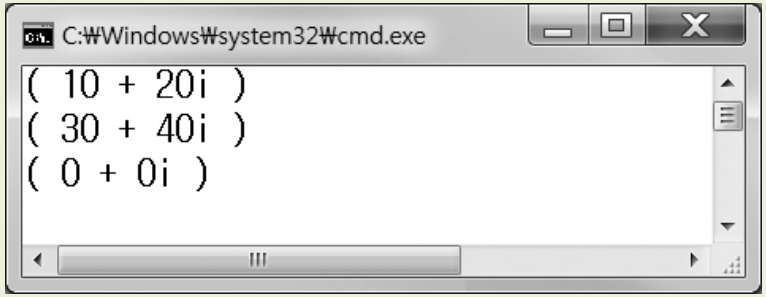
생성자 오버로딩

- 객체를 선언할 때도 초깃값을 주지 않을 수 있다. 매개변수가 없는 기본 생성자는 C++ 컴파일러가 제공해 준다. 그런데 프로그래머가 매개변수를 갖는 기본 생성자를 만들어주면 컴파일러는 더 이상 기본 생성자를 제공하지 않고 프로그래머가 직접 매개변수 없는 생성자를 작성할 것을 떠맡긴다. 예를 들어 [예제10-12]에서 28행의 주석문을 제거하면 다음과 같은 에러가 발생한다.

- 컴파일 에러를 없애려면 기본 생성자를 하나 더 추가해서 정의해야 한다. 즉, 매개변수가 2개인 생성자를 만들어 사용하던 중 매개변수를 갖지않는 생성자가 필요하다면 이를 프로그래머가 직접 정의하고 사용해야 한다는 결론을 내릴 수 있다. 동일한 이름의 함수를 여러 번 정의할 수 있는 함수의 오버로딩은 생성자에도 적용된다. 생성자도 함수의 일종이므로 매개변수의 개수나 자료형을 달리해서 여러 번 정의할 수 있는데, 이를 생성자 오버로딩이라고 한다.

예제 10-13. 생성자 오버로딩하기(10_13.cpp)

책의 소스코드 참고



02 생성자와 소멸자

생성자의 기본 매매변수 값 지정하기

- 생성자의 오버로딩보다 더 편리한 방법이 있다. 바로 생성자에 기본 매개변수 값을 설정하는 것인데, 함수 선언문의 매개변수에 대입 연산자를 사용해서 기본 매개변수 값을 함께 주는 것이다

```
Complex(int r=0 , int i=0);
```

- 함수의 선언문에 기본 매개변수 값을 기술하면 함수를 한 번만 정의해서 기본 매개변수 값을 다양한 형태로 호출할 수 있다. 기본 매개변수 값을 지정한 멤버함수는 실 매개변수 없이 실 매개변수 1개, 실 매개변수 2개 등으로 다양하게 호출할 수 있다는 장점이 있다.

```
Complex x(10, 20);   // 실 매개변수 2개를 설정해서 호출
Complex y(30);       // 실 매개변수 1개만을 설정해서 호출
Complex z;           // 매개변수 없이 호출
```

생성자의 콜론 초기화

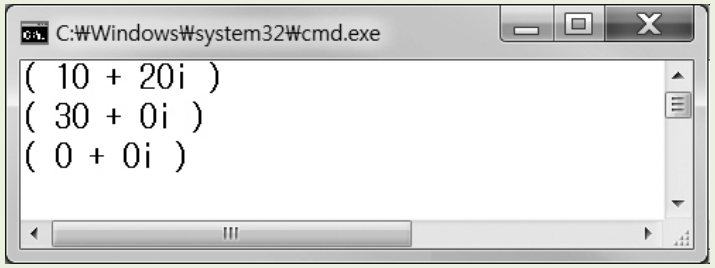
- 생성자의 머리 부분에서 멤버변수에 초깃값을 설정할 수 있는데, 이때 콜론 초기화가 사용된다.

```
Complex::Complex(int r, int i) : real(r), image(i)
{
}
```

- 함수의 몸체에 기술되어야 할 멤버변수의 초깃값 설정을 함수의 머리 부분으로 옮기되 콜론으로 멤버변수를 초기화한다고 알린다. 콜론 다음에는 초깃값을 설정할 멤버변수 다음에 소괄호를 기술한 후 소괄호 안에 초깃값을 기술한다. 초기화할 변수가 여러 개이면 콤마로 연결해서 반복적으로 기술하면 된다.

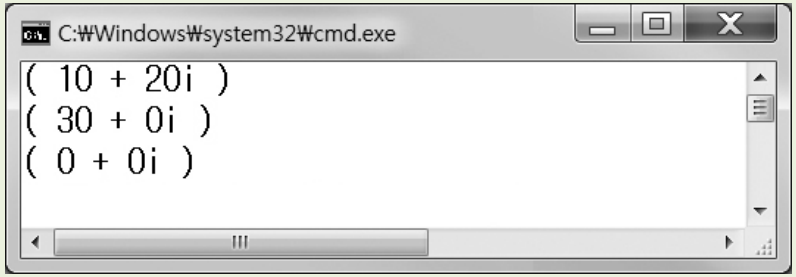
예제 10-14. 생성자의 기본 매개변수 값 설정하기(10_14.cpp)

책의 소스코드 참고



예제 10-15. 생성자 콜론 초기화하기(10_15.cpp)

책의 소스코드 참고



02 생성자와 소멸자

소멸자의 의미와 특징

- 소멸자는 생성자의 반대되는 개념이다. 소멸자와 생성자와 비교해 보자.
  1. 생성자(constructor)는 객체가 생성될 때 자동 호출되고 소멸자(destructor)는 객체가 소멸될 때 자동으로 호출된다.
  2. 생성자가 객체를 초기화하기 위한 멤버함수라면 소멸자는 객체를 정리해 주는(리소스를 해제한다든지 하는 작업) 멤버함수다.
- 소멸자의 특징
  1. 소멸자 함수는 멤버함수다.
  2. 소멸자 함수명도 생성자처럼 클래스명을 사용한다.
  3. 소멸자 함수는 생성자 함수와 구분하려고 함수명 앞에 ~ 기호를 붙인다.
  4. 소멸자는 자료형을 지정하지 않는다(반환값의 유형을 지정하지 않는다).
  5. 소멸자의 호출은 명시적이지 않다.
  6. 소멸자는 객체 소멸 시 자동 호출된다.
  7. 소멸자는 전달 매개변수를 지정할 수 없다.
  8. 소멸자는 전달 매개변수를 지정할 수 없으므로 오버로딩 할 수 없다.

예제 10-16. 소멸자 정의하기(10_16.cpp)

책의 소스코드 참고