8장
멤버함수 자세히 살펴보기

변영철 교수
(ycb@jejunu.ac.kr)

제1절 콜론 초기화 이야기

멤버 변수 초기화
(생성자 함수에서)

```
class My {
public:
   int a;
   int b;

   My() {
      a=0;
      b=0;
   }
   ~My() {
   }
};
```

멤버 변수 초기화
(선언할 때 바로)

```
class My {
public:
   int a=0;
   int b=0;

   My() {
   }
   ~My() {
   }
};
```

멤버 **별명** 초기화
(선언할 때 바로)

```
class My {
public:
   int a=0;
   int b=0;
   int& babo=b;

   My() {
   }
   ~My() {
   }
};
```

멤버 **별명** 초기화
(생성자 함수에서 ???)

```
class My {
public:
   int a;
   int b;
   int& babo;

   My() {
      a=0;
      b=0;
      babo = b;
   }
   ~My() {
   }
};
```

멤버 **별명** 초기화
(콜론 초기화)

```
class My {
public:
   int a;
   int b;
   int& babo;

   My():babo(b){
      a=0;
      b=0;
   }
   ~My() {
   }
};
```

제1절 콜론 초기화 이야기

- 왜 콜론 초기화를 사용할까?
  - main 함수의 지역 변수를 클래스 멤버 레퍼런스로 연결하는 방법

제2절 친구 이야기(친구 함수)

```
class Base
{
//(가)
private:
    int a;
};

void main() {
    Base gildong;
    gildong.a = 7;
}
```

(정답) friend void main();

- private 멤버변수 a를 갖는 Base 클래스
- main 함수에서 객체 gildong을 만든 후 a 접근 가능?
- main 함수는 내 친구
- Base 클래스 파생 클래스 Derived 에서 a 접근 가능?
- 피는 물보다 진하지 않다?

제2절 친구 이야기(친구 클래스)

- set 멤버 함수를 갖는 My 클래스 선언
- My 클래스에서 Base 객체 gildong을 만든 후 멤버변수 a 접근 가능?
- Base 클래스에서 My 클래스가 친구라고 선언하면?

```
#include <stdio.h>

class Base
{
friend class My;
private:
   int a;
};

class My
{
public:
   void Set() {
      Base cheolsu;
      cheolsu.a = 7;
      printf("%d ₩n", cheolsu.a);
   }
};

void main()
{
   My gildong;
   gildong.Set();
}
```

제3절 연산자 중복 정의 이야기

```
void main()
{
    Database gildong;
    gindong.Set(3);
    gindong << 3; // 더 직관적인가?
    gildong.Draw();
}
```

- C++ 언어에서 << 연산자는 쉬프트 (shift) 연산자
- 이런 연산자를 다른 의미로 (중복) 정의하여 사용 가능
- 멤버함수 Set 이름을 operator << 로 바꾸면?

```
class Database
{
private:
    int data;

public:
    void Set(int x) {
        data = x;
    }

    void Draw() {
        printf("%d ₩n", data);
    }
};

void main()
{
    Database gildong;
    gindong.Set(3);
    gildong.Draw();
}
```

제3절 연산자 중복 정의 이야기

- << 연산자를 값을 할당하는 것으로 정의할 수 있음
- 원래는 쉬프트 연산자인데 이를 값을 할당하는 연산자로 '중복으로' 정의(operator overloading)

마음에 들지 않아서 재정의(override)
다른 의미로도 쓰려고 중복정의(overloading)

```
class Database
{
private:
   int data;

public:
   void operator<<(int x) {
      data = x;
   }
   void Draw() {
      printf("%d ₩n", data);
   }
};

void main()
{
   Database gildong;
   gildong.operator<<(3);
   gildong << 3; //이렇게 해도 됨!!
   gildong.Draw();
}
```

제3절 연산자 중복 정의 이야기

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

void main( )
{
    cout << "Hello,: " << "World!" << endl;
}
```

```
#include <stdio.h>

class Database
{
    friend void operator<<(Database& db, int x);
private:
    int data;

public:

    void Draw() {
        printf("%d ₩n", data);
    }
};

void operator<<(Database& db, int x) {
    db.data = x;
}

void main()
{
    Database gildong;
    operator<<(gildong, 3);
    gildong << 3; //이렇게 해도 됨!!
    gildong.Draw();
}
```

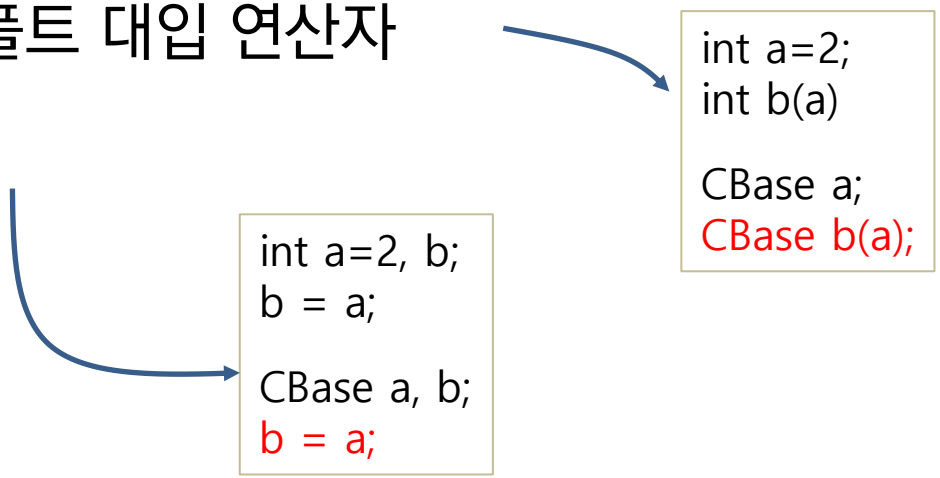
제3절 연산자 중복 정의 이야기

```
#include <stdio.h>

class Database
{
    friend Database& operator<<(Database& db, int x);
private:
    int data;

public:

    void Draw() {
        printf("%d ₩n", data);
    }
};

Database& operator<<(Database& db, int x) {
    db.data = x;
    return db;
}

void main()
{
    Database gildong;
    operator<<(gildong, 3);
    gildong << 3 << 4 << 7; //이렇게 해도 됨!!
    gildong.Draw();
}
```

```
#include <stdio.h>

class Database
{
private:
    int data;

public:
    Database& operator<<(int x) {
        data = x;
        return *this;
    }

    void Draw() {
        printf("%d ₩n", data);
    }
};

void main()
{
    Database gildong;
    gildong.operator<<(3);
    gildong << 3 << 4 << 7; //이렇게 해도 됨!!
    gildong.Draw();
}
```

제4절 디폴트 멤버 함수 이야기

- 우리가 작성하지 않아도 컴파일러에 의해 기본적으로(default, not option), 자동으로 만들어지는 멤버 함수
  - 디폴트 생성자 (p307)
  - 디폴트 소멸자 (p307)
  - 디폴트 복사 생성자
  - 디폴트 대입 연산자



제5절 const 지시어 이야기

- const 지시어는 변수, 레퍼런스, 포인터 변수 등과 함께 사용되어 값을 변경할 수 없도록 함
- 왜 하지?
  - (답1)혹시나 실수로 변경할 수 있어서
  - (답2)여러 번 반복되는 것을 한번에 수정하기 위하여

```
#include <stdio.h>

void main()
{
    int a = 0;
    a = 3;

    const int b = 0;
    // 이후 b 변경 불가)
    b = 3;  //error!
```

```
#include <stdio.h>

void main()
{
    const int MAX = 0;

    for(int i=0; i<MAX; i++) {

    }
}
```

제6절 정적 멤버 이야기

- 멤버 변수 앞에 static을 넣으면 정적 멤버 변수가 됨
- 해당 클래스로 정의한 모든 객체들은 정적 변수를 공유
- 현재 생성된 객체의 수를 저장할 수 있음 : 스마트 객체
- 참고로, 멤버 함수 앞에 static을 넣으면? 객체를 정의하지 않아도 호출할 수 있는 함수가 됨.

```
class My //객체 x, y, z
{
public:
   static int a; //선언!

   My() {
      a = a + 1;
   }

   ~My() {
      a = a - 1;
   }
};

int My::a = 0; //정의
```

제6절 정적 멤버 이야기

- 철수야, 볼펜 하나만 갖다 줄래? 없으면 사다 줘.
- 프로그램을 짜다 보니 '있으면 있는 것을, 없으면 새로 만들어 주는 객체'가 자주 사용되더라.
- 소프트웨어 디자인 패턴(design pattern)
  - 원래는 건축에서 쓰는 말
  - '있으면 있는 것을, 없으면 새로 만들어서 주는 객체'를 팩토리(factory) 객체라고 함.
  - 이렇게 자주 사용되는 객체들은 어떤 것이 있을까? → 소프트웨어 디자인 패턴