

---

## **3장. 클래스의 기본**

**담당교수 : 김미경**  
**부산대학교**

## 3-1 구조체와 클래스

### ■ 구조체의 유용성

- 관련 있는 데이터를 하나의 자료형으로 묶을 수 있다.
- 따라서, 프로그램의 구현 및 관리가 용이해진다.
- 함께 움직이는 데이터들을 묶어주는 효과!

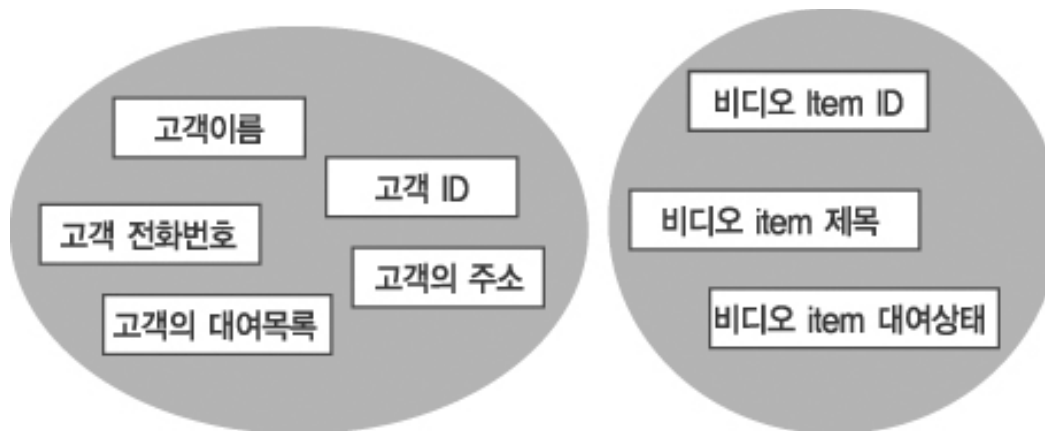


그림 3-1

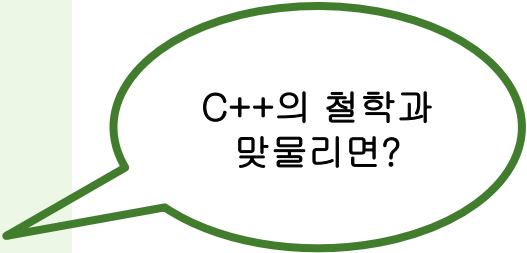
## 3-1 구조체와 클래스

- C 언어의 구조체에 대한 불만
  - 모든 사용자 정의 자료형에 대한 불만
  - 기본 자료형으로 인식해 주지 않는다.

```
struct Person{
    int age;
    char name[10];
};

int main()
{
    int a=10;
    Person p;    // struct Person p;

    return 0;
}
```



C++의 철학과  
맞물리면?

## 3-1 구조체와 클래스

### ■ 함수를 넣으면 좋은 구조체

- 프로그램=데이터+데이터 조작 루틴(함수)
- 잘 구성된 프로그램은 데이터와 더불어 함수들도 부류를 형성
- oo1.cpp, oo2.cpp, oo3.cpp

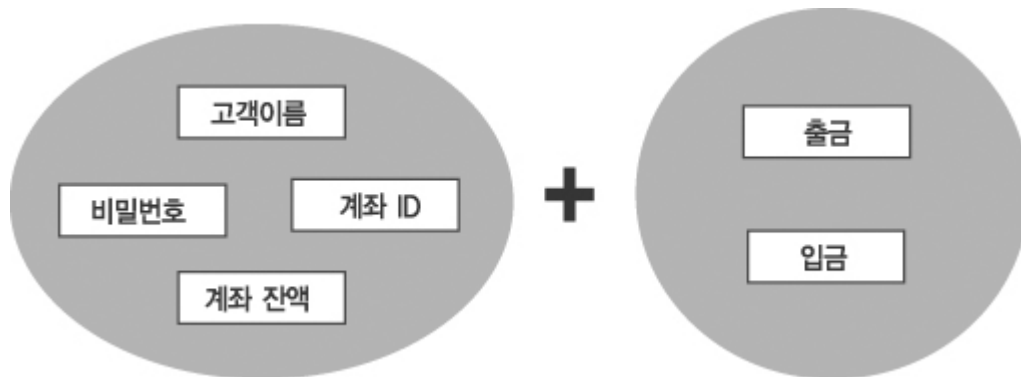


그림 3-4

```
1 /* oo1.cpp */
2 #include <iostream>
3 using std::cout; using std::endl;
4
5 struct Account {
6     char accID[20];    // 계좌 번호
7     char secID[20];    // 비밀번호
8     char name[20];     // 이름
9     int balance;       // 잔액
10 };
11 int main(void)
12 {
13     Account yoon={"1234", "2321", "yoon", 1000};
14     cout<<"계좌번호 : "<<yoon.accID<<endl;
15     cout<<"비밀번호 : "<<yoon.secID<<endl;
16     cout<<"이름 : "<<yoon.name<<endl;
17     cout<<"잔액 : "<<yoon.balance<<endl;
18     return 0;
19 }
```

## 3-1 구조체와 클래스

### ■ oo2.cpp와 oo3.cpp의 비교

```
struct Account {  
    char accID[20];  
    char secID[20];  
    char name[20];  
    int balance;  
};
```

```
void Deposit(Account &acc, int money) {  
    acc.balance+=money;  
}  
void Withdraw(Account &acc, int money) {  
    acc.balance-=money;  
}
```

```
int main(void){  
    Account yoon={"1234", "2321", "yoon", 1000};  
    Deposit(yoon, 100);  
    cout<<"잔액 : "<<yoon.balance<<endl;  
    Withdraw(yoon, 200);  
    cout<<"잔액 : "<<yoon.balance<<endl;  
    return 0;  
}
```



```
struct Account {  
    char accID[20];  
    char secID[20];  
    char name[20];  
    int balance;  
};
```

```
void Deposit(int money){  
    balance+=money;  
}  
void Withdraw(int money){  
    balance-=money;  
}
```

```
};
```

## 3-1 구조체와 클래스

```
oop3.pp
struct Account {
    char accID[20];
    char secID[20];
    char name[20];
    int balance;

    void Deposit(int money){
        balance+=money;
    }
    void Withdraw(int money){
        balance-=money;
    }
};

int main(void)
{
    Account yoon={"1234", "2321", "yoon", 1000};
    yoon.Deposit(100);
    cout<<"잔 액 : "<<yoon.balance<<endl;
    yoon.Withdraw(200);
    cout<<"잔 액 : "<<yoon.balance<<endl;
    return 0;
}
```

Account yoon={"1234", "2321", "yoon", 1000};

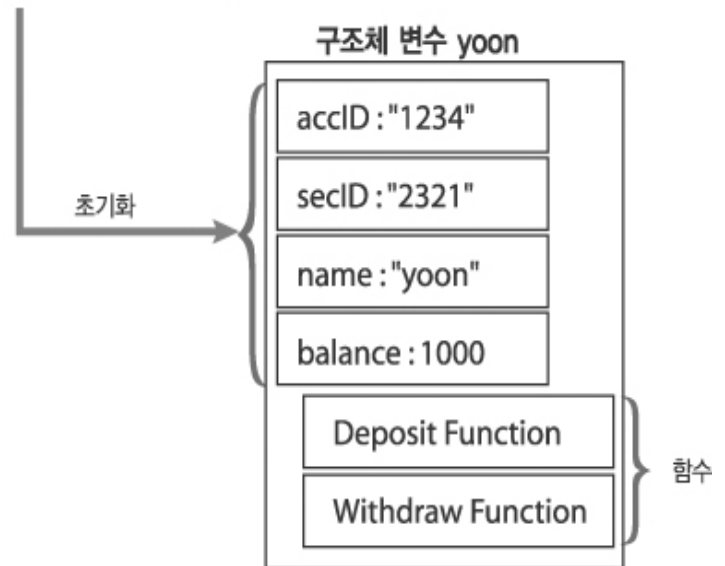


그림 3-5

.c일 경우 구조체에 함수 포함할 수 없다

## 3-1 구조체와 클래스

- 구조체가 아니라 클래스(Class)
  - 클래스 = 멤버 변수 + 멤버 함수
  - 변수가 아니라 객체(Object: 완전한 대상체)
  - 004.cpp

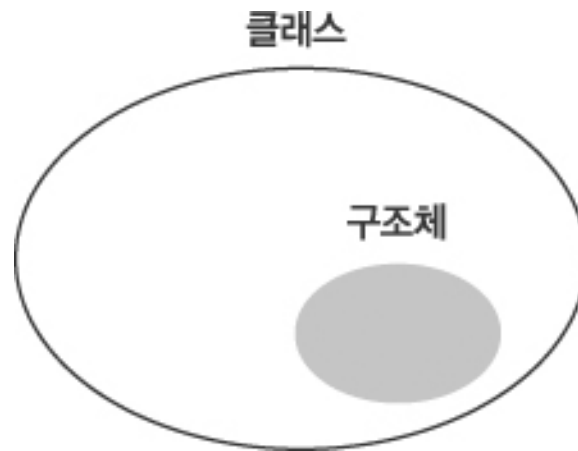


그림 3-6

## 3-1 구조체와 클래스

Oop3.cpp 에서 struct 를 class로 바꾸어 해보면?

→error

해결 방안 : class { 다음에 public: 삽입

public: 이 중요한 포인트

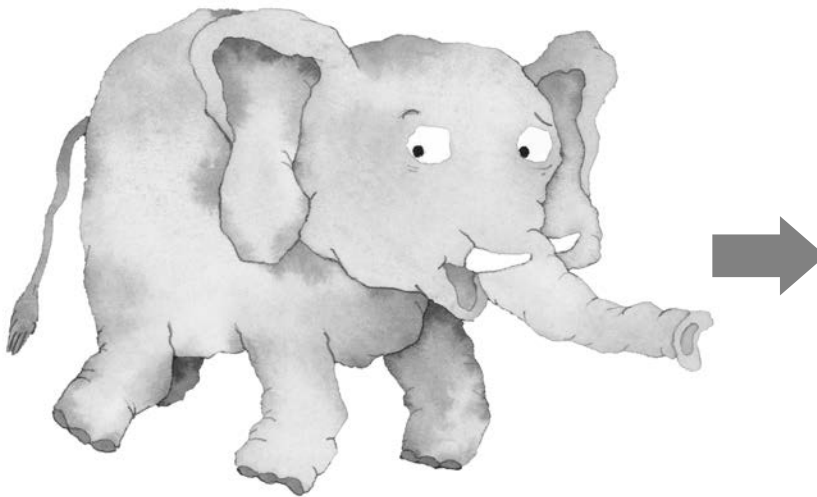
```
1  /* oo4.cpp */
2  #include <iostream>
3  using std::cout; using std::endl;
4
5  class Account {
6      char accID[20];    // 계좌 번호
7      char secID[20];    // 비밀번호
8      char name[20];     // 이름
9      int balance;       // 잔액
10 void Deposit( int money) { // 입금
11     balance+=money;
12 }
13 void Withdraw( int money){ // 출금
14     balance-=money;
15 }
16 void ShowData() {
17     cout << "계좌번호 : " << accID << endl;
18     cout << "비밀번호 : " << secID << endl;
19     cout << "이름 : " << name << endl;
20     cout << "잔액 : " << balance << endl;
21 }
22 };
```

```
24 void Deposit(Account &acc, int money) { // 입금
25     acc.balance+=money;
26 }
27 void Withdraw(Account &acc, int money){ // 출금
28     acc.balance-=money;
29 }
30
31 int main(void){
32     Account yoon={"1234", "2321", "yoon", 1000};
33
34     Deposit(yoon, 100);
35     cout<<"잔액 : "<<yoon.balance<<endl;
36
37     Withdraw(yoon, 200);
38     cout<<"잔액 : "<<yoon.balance<<endl;
39
40     cout << "===클래스 함수 이용한 입출금" <<endl;
41     yoon.Deposit(5000);
42     cout<<"잔액 : "<<yoon.balance<<endl;
43     yoon.Withdraw(2000);
44     cout<<"잔액 : "<<yoon.balance<<endl;
45     yoon.ShowData();
46 }
```



## 3-2 클래스와 객체

- 사물의 관찰 이후의 데이터 추상화
  - 현실 세계의 사물을 데이터적인 측면과 기능적인 측면을 통해서 정의하는 것



특징 1. 발이 네 개

특징 2. 코의 길이가 5미터 내외

특징 3. 몸무게는 1톤 이상

특징 4. 코를 이용해서 목욕을 함

특징 5. 코를 이용해서 물건을 집기도 함

## 3-2 클래스와 객체

- 데이터 추상화 이후의 클래스화
  - 추상화된 데이터를 가지고 사용자 정의 자료형을 정의하는 것

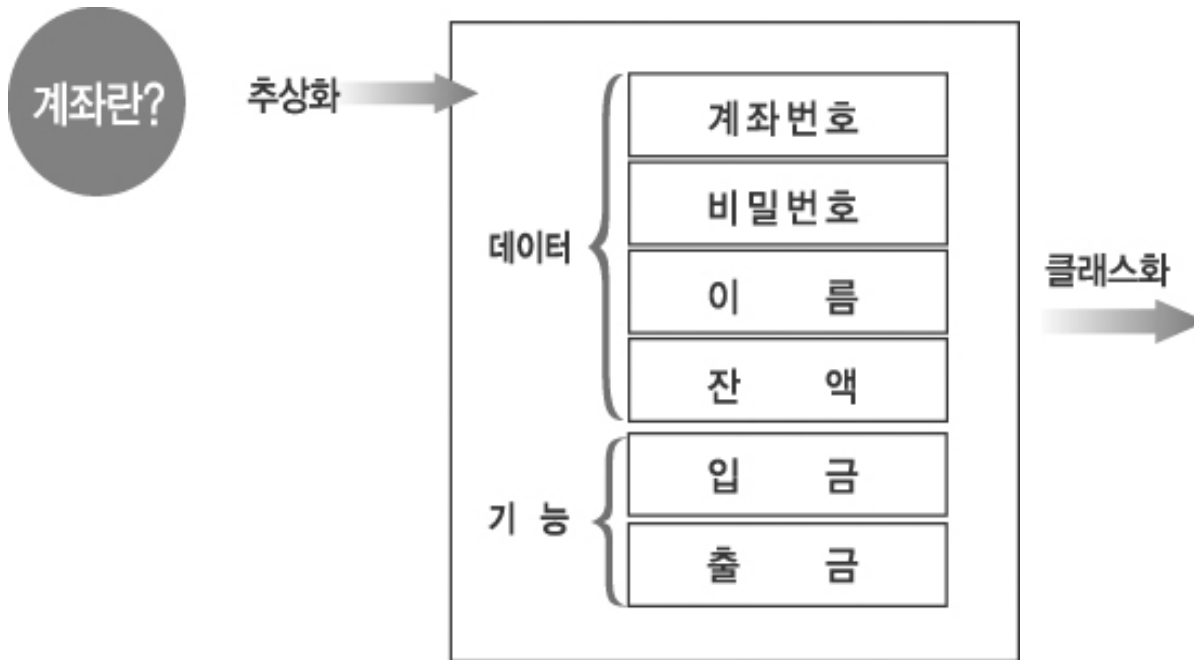


그림 3-7

```
class Account {  
    public:  
        char accID[20];  
        char secID[20];  
        char name[20];  
        int balance;  
  
        void Deposit(int money){  
            balance+=money;  
        }  
        void Withdraw(int money){  
            balance-=money;  
        }  
};
```

그림 3-8

## 3-2 클래스와 객체

- 클래스화 이후의 인스턴스화
  - 클래스 기반의 객체(Object) 생성

```
class Account {  
public:  
    char accID[20];  
    char secID[20];  
    char name[20];  
    int balance;  
  
    void Deposit(int money){  
        balance+=money;  
    }  
    void Withdraw(int money){  
        balance-=money;  
    }  
};
```

인스턴스화



```
int main(void  
{  
    Account yoon={"1234", "2321", "yoon", 1000};  
    .....  
}
```

그림 3-9

## 3-3 클래스 멤버의 접근 제어

- 클래스의 내부 접근과 외부 접근

```
class Counter {  
public:  
    int val;  
    void Increment(void)  
    {  
        val++;                //내부 접근  
    }  
};  
  
int main(void)  
{  
    Counter cnt;  
    cnt.val=0;                //외부 접근  
    cnt.Increment();          //외부 접근  
    cout<<cnt.val<<endl;      //외부 접근  
    return 0;  
}
```

## 3-3 클래스 멤버의 접근 제어

```
const int OPEN=1;
const int CLOSE=2;

class Door{
private:
    int state;
public:
    void Open(){ state=OPEN; }
    void Close(){ state=CLOSE; }
    void ShowState(){
        cout<<"현재 문의 상태 : ";
        cout<<((state==OPEN)? "OPEN" : "CLOSE")<<endl;
    }
};

int main()
{
    Door d;
    //d.state=OPEN;
    d.Open();
    d.ShowState();
    return 0;
}
```

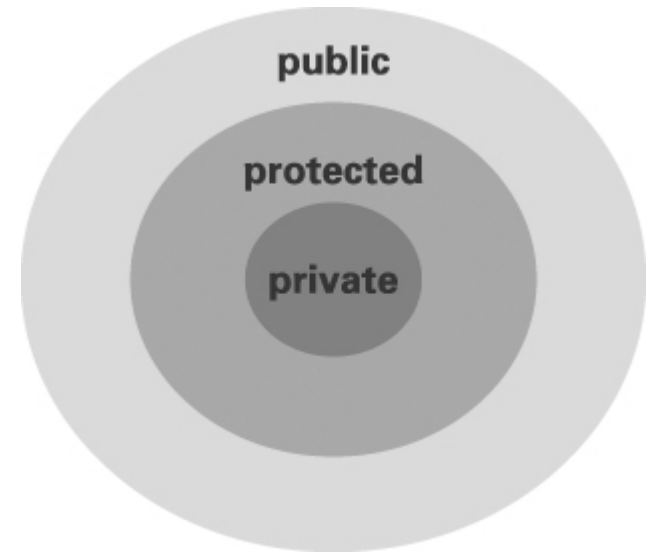


그림 3-10

## 3-4 멤버 함수의 외부정의

```
class Door{  
private:  
    int state;  
  
public:  
    void Open();  
    void Close();  
    void ShowState();  
};  
  
void Door::Open(){  
    state=OPEN;  
}  
void Door::Close(){  
    state=CLOSE;  
}  
void Door::ShowState(){  
    ... 생략 ...  
}
```

```
void Door::Open() {  
    state=OPEN;  
}
```

Door 클래스 내에  
선언되어 있는  
Open 함수의 정의

그림 3-11

## 3-4 멤버 함수의 외부정의

### ■ 멤버 함수의 인-라인화

```
const int OPEN=1;
const int CLOSE=2;

class
Door{

private:
    int state;
public:
    void Open(){
        state=OPEN;
    }
    void Close(){
        state=CLOSE;
    }
    void ShowState() {
        ...생략...
    }
};
```

equal!



```
1: class Door{
2: private:
3:     int state;
4: public:
5:     void Open();
6:     void Close();
7:     void ShowState();
8: };

inline void Door::Open(){
    state=OPEN;
}

inline void Door::Close(){
    state=CLOSE;
}

inline void Door::ShowState(){
    ... 생 략 ...
}
```

## 3-4 멤버 함수의 외부정의

- 헤더 파일을 이용한 파일의 분할

### Door.h

```
#include <iostream>
using std::cout;
using std::endl;

const int OPEN=1;
const int CLOSE=2;

class Door{
private:
    int state;

public:
    void Open();
    void Close();
    void ShowState();
};
```

### Door.cpp

```
#include "Door.h"

void Door::Open(){
    state=OPEN;
}

void Door::Close(){
    state=CLOSE;
}

void Door::ShowState(){
    cout<<"현재 문의 상태 :";
    ... 생략...
}
```

### Main.cpp

```
#include "Door.h"

int main()
{
    Door d;

    d.Open();
    d.ShowState();

    return 0;
}
```

그림 3-12



- **한개의 CPP파일에 여러개의 class존재 가능**
- **다른 파일로 작성된 것을 include를 통하여 사용하는 것이 가능.**
- **class의 주요 골격만 `***.h`로 만들고 세부 내용은 `***.cpp`로 만든다.**
- **다른 문서에 삽입 시에는 `#include "***.h"`로 불러 들인다.**
- **class 의 default접근 제어는 private이며 struct 의 default 접근제어는 public이다.**