

윤성우의 열혈 C++ 프로그래밍



윤성우 저 열혈강의 C++ 프로그래밍 개정판

Chapter 03. 클래스의 기본

윤성우의 열혈 C++ 프로그래밍



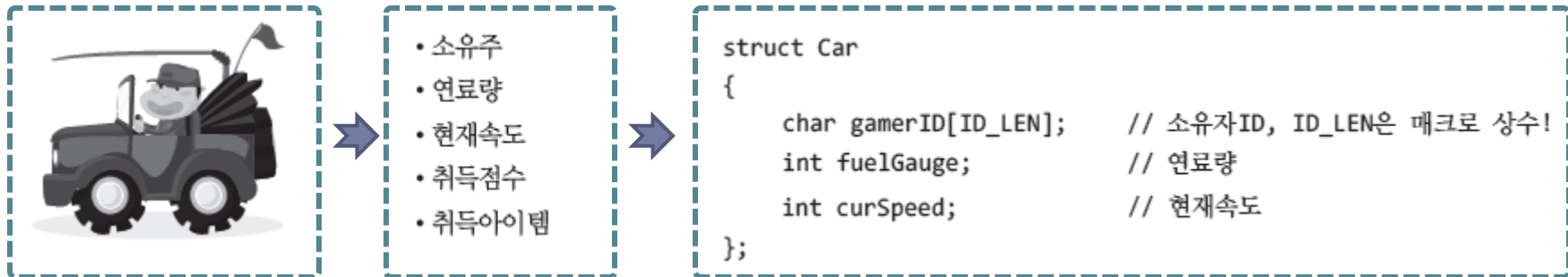
Chapter 03-1. C++에서의 구조체

윤성우 저 열혈강의 C++ 프로그래밍 개정판

C++에서의 구조체

구조체의 등장배경

- ▶ 연관 있는 데이터를 하나로 묶으면 프로그램의 구현 및 관리가 용이하다.
- ▶ 구조체는 연관 있는 데이터를 하나로 묶는 문법적 장치이다.



연관 있는 데이터들은 생성 및 소멸의 시점이 일치하고, 이동 및 전달의 시점 및 방법이 일치하기 때문에 하나의 자료형으로 묶어서 관리하는 것이 용이하다.

C++에서의 구조체 변수 선언

```
struct Car basicCar;
struct Car simpleCar;
```



```
Car basicCar;
Car simpleCar;
```

따라서 C++에서는 구조체 변수 선언시 struct 키워드의 생략을 위한 typedef 선언이 불필요하다.

C 스타일 구조체 변수 초기화

C++ 스타일 구조체 변수 초기화

```
struct Car
{
    char gamerID[ID_LEN]; // 소유자ID
    int fuelGauge;         // 연료량
    int curSpeed;          // 현재속도
};
```

Car와 관련된 연관된 데이터들의 모임

데이터 뿐만 아니라, 해당 데이터와 연관된 함수들도 함께 그룹을 형성하기 때문에 함수도 하나로 묶는 것에 대해 나름의 가치를 부여할 수 있다.

```
void ShowCarState(const Car &car)
{
    . . . .
}

void Accel(Car &car)
{
    . . . .
}

void Break(Car &car)
{
    . . . .
}
```

Car와 관련된 연관된 함수들의 모임



구조체 안에 함수 삽입하기

```
struct Car
{
    char gamerID[ID_LEN];
    int fuelGauge;
    int curSpeed;

    void ShowCarState()
    {
        . . . . .
    }

    void Accel()
    {
        . . . . .
    }

    void Break()
    {
        . . . . .
    }
};
```

C++에서는 구조체 안에 함수를 삽입하는 것이 가능하다.
따라서 C++에서는 구조체가 아닌, 클래스라 한다.

```
void ShowCarState()
{
    cout<<"소유자ID: "<<gamerID<<endl;    // 위에 선언된 gamerID에 접근
    cout<<"연료량: "<<fuelGauge<<"%"<<endl;
    cout<<"현재속도: "<<curSpeed<<"km/s"<<endl<<endl;
}
```

```
void Break()
{
    if(curSpeed<BRK_STEP)
    {
        curSpeed=0;    // 위에 선언된 curSpeed에 접근
        return;
    }

    curSpeed-=BRK_STEP;
}
```

함께 선언된 변수에는 직접 접근
이 가능하다.

C++에서의 구조체 변수 선언

변수의 생성

```
Car run99={"run99", 100, 0};
Car sped77={"sped77", 100, 0};
```



실제로는 구조체 변수마다 함수가 독립적으로 존재하는 구조는 아니다. 그러나 논리적으로는 독립적으로 존재하는 형태로 보아도 문제가 없으니, 위의 그림의 형태로 변수(객체)를 이해하자!

구조체 안에 enum 상수의 선언

Car 클래스를 위해 정의된 상수!

```
#define ID_LEN      20
#define MAX_SPD     200
#define FUEL_STEP   2
#define ACC_STEP    10
#define BRK_STEP    10
```

```
namespace CAR_CONST
{
    enum
    {
        ID_LEN      =20,
        MAX_SPD     =200,
        FUEL_STEP    =2,
        ACC_STEP     =10,
        BRK_STEP     =10
    };
}
```

이렇듯 연관 있는 상수들을 하나의 이름공간에 별도로 묶기도 한다!

```
struct Car
{
    enum
    {
        ID_LEN      =20,
        MAX_SPD     =200,
        FUEL_STEP    =2,
        ACC_STEP     =10,
        BRK_STEP     =10
    };

    char gamerID[ID_LEN];
    int fuelGauge;
    int curSpeed;

    void ShowCarState() { . . . . }
    void Accel() { . . . . }
    void Break() { . . . . }
};
```

이렇듯 구조체 안에 enum 선언을 함으로써 잘못된 외부의 접근을 제한할 수 있다.

함수는 외부로 뺄 수 있다.

```
struct Car
{
    . . . . .
    void ShowCarState();
    void Accel();
    . . . . .
};
```

구조체 안에 삽입된 함수의 선언!

```
void Car::ShowCarState()
{
    . . . . .
}
void Car::Accel()
{
    . . . . .
}
```

구조체 안에 선언된 함수의 정의!

구조체 안에 정의된 함수는 **inline** 선언된 것으로 간주한다.

따라서 필요하다면, 함수의 정의를 외부로 뺄 때에는 다음과 같이 명시적으로 **inline** 선언을 해야 한다.

```
inline void Car::ShowCarState() { . . . . . }
inline void Car::Accel() { . . . . . }
inline void Car::Break() { . . . . . }
```



윤성우의 열혈 C++ 프로그래밍



Chapter 03-2. 클래스(Class)와 객체 (Object)

윤성우 저 열혈강의 C++ 프로그래밍 개정판

클래스와 구조체의 유일한 차이점

```
class Car
{
    char gamerID[CAR_CONST::ID_LEN];
    int fuelGauge;
    int curSpeed;

    void ShowCarState() { . . . . }
    void Accel() { . . . . }
    void Break() { . . . . }
};
```

키워드 **struct**를 대신해서 **class**를 사용한
것이 유일한 외형적 차이점이다.

```
int main(void)
{
    Car run99;
    strcpy(run99.gamerID, "run99");      (×)
    run99.fuelGauge=100;                  (×)
    run99.curSpeed=0;                     (×)
    . . . .
```

왼쪽과 같이 단순히 키워드만 **class**로 바꾸면
선언된 멤버의 접근이 불가능하다.
따라서 별도의 접근제어와 관련된 선언을 추가
해야 한다.

접근제어 지시자

접근제어 지시자

- ▶ **public** 어디서든 접근허용
- ▶ **protected** 상속관계에 놓여있을 때, 유도 클래스에서의 접근허용
- ▶ **private** 클래스 내(클래스 내에 정의된 함수)에서만 접근허용

```
class Car
{
private:
    char gamerID[CAR_CONST::ID_LEN];
    int fuelGauge;
    int curSpeed;
public:
    void InitMembers(char * ID, int fuel);
    void ShowCarState();
    void Accel();
    void Break();
};
```

private! [char gamerID[CAR_CONST::ID_LEN];
int fuelGauge;
int curSpeed;

public! [void InitMembers(char * ID, int fuel);
void ShowCarState();
void Accel();
void Break();

```
int main(void)
{
    Car run99;
    run99.InitMembers("run99", 100);
    run99.Accel();
    run99.Accel();
    run99.Accel();
    run99.ShowCarState();
    run99.Break();
    run99.ShowCarState();
    return 0;
}
```

Car의 멤버함수는 모두 public이므로 클래스의 외부에 해당하는 main 함수에서 접근가능!

용어정리: 객체(Object), 멤버변수, 멤버함수

```
class Car
{
private:
    char gamerID[CAR_CONST::ID_LEN];
    int fuelGauge;
    int curSpeed;
public:
    void InitMembers(char * ID, int fuel);
    void ShowCarState();
    void Accel();
    void Break();
};
```

왼쪽의 Car 클래스를 대상으로 생성된 변수를
가리켜 '객체'라 한다.

왼쪽의 Car 클래스 내에 선언된 변수를 가리켜
'멤버변수'라 한다.

왼쪽의 Car 클래스 내에 정의된 함수를 가리켜
'멤버함수'라 한다.



C++에서의 파일 분할

```
class Car
{
private:
    char gamerID[CAR_CONST::ID_LEN];
    int fuelGauge;
    int curSpeed;
public:
    void InitMembers(char * ID, int fuel);
    void ShowCarState();
    void Accel();
    void Break();
};
```

클래스의 선언은 일반적으로 헤더파일에 삽입한다. 객체생성문 및 멤버의 접근문장을 컴파일하기 위해서 필요하다.

클래스의 이름을 따서 Car.h로 헤더파일의 이름을 정의하기도 한다.

단! 인라인 함수는 컴파일 과정에서 함수의 호출문을 대체해야 하기 때문에 헤더파일에 함께 정의되어야 한다

```
void Car::InitMembers(char * ID, int fuel) { . . . . }
void Car::ShowCarState() { . . . . }
void Car::Accel() { . . . . }
void Car::Break() { . . . . }
```

Car 클래스의 멤버함수의 몸체는 다른 코드의 컴파일 과정에서 필요한 게 아니다. 링크의 과정을 통해서 하나의 바이너리로 구성만 되면 된다. 따라서 cpp 파일에 정의하는 것이 일반적이다.

클래스의 이름을 따서 Car.cpp로 소스파일의 이름을 정의하기도 한다.



윤성우의 열혈 C++ 프로그래밍



Chapter 03-3. 객체지향 프로그래밍의 이해

윤성우 저 열혈강의 C++ 프로그래밍 개정판

객체지향 프로그래밍의 이해

객체에 대한 간단한 정의

- ▶ 사전적 의미 물건 또는 대상
- ▶ 객체지향 프로그래밍 객체 중심의 프로그래밍

객체 객체 객체

“ 나 는 과 일 장 수 에 게 두 개 의 사 과 를 구 매 했 다 ! ”

데이터 행위, 기능

객체지향 프로그래밍에서는 나, 과일장수, 사과라는 객체를 등장시켜서 두 개의 사과 구매라는 행위를 실체화한다.

객체지향 프로그래밍은 현실에 존재하는 사물과 대상, 그리고 그에 따른 행동을 있는 그대로 실체화시키는 형태의 프로그래밍이다.

객체를 이루는 것은 데이터와 기능입니다.

과일장수 객체의 표현

- 과일장수는 과일을 팝니다. **행위**
- 과일장수는 사과 20개, 오렌지 10개를 보유하고 있습니다. **상태**
- 과일장수의 과일판매 수익은 현재까지 50,000원입니다. **상태**

과일장수의 데이터 표현

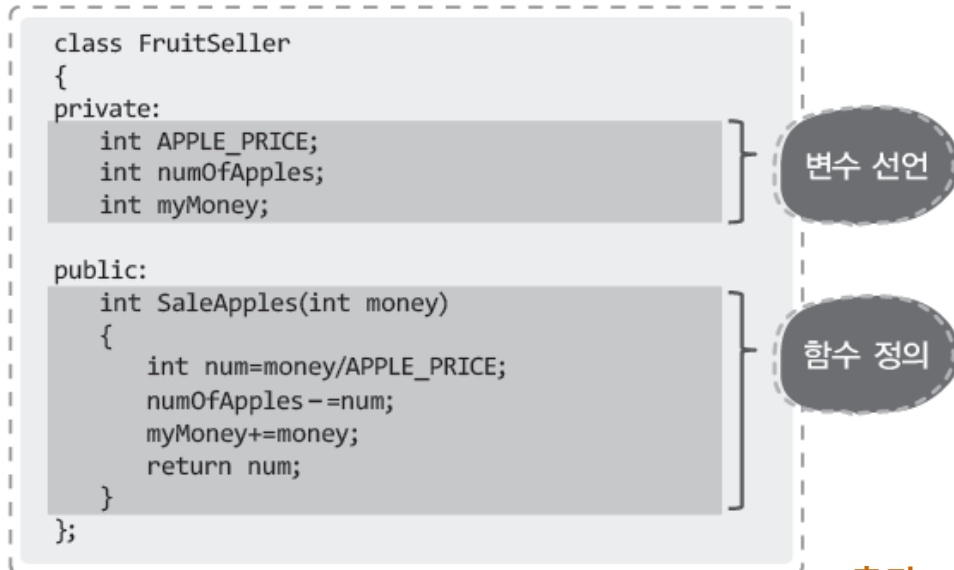
- 보유하고 있는 사과의 수 → `int numOfApples;`
- 판매 수익 → `int myMoney;`

과일장수의 행위 표현

```
int SaleApples(int money)    // 사과 구매액이 함수의 인자로 전달
{
    int num = money/1000;    // 사과가 개당 1000원이라고 가정
    numOfApples -= num;      // 사과의 수가 줄어들고,
    myMoney += money;        // 판매 수익이 발생한다.
    return num;              // 실제 구매가 발생한 사과의 수를 반환
}
```

이제 남은 것은 데이터와 행위를
한데 묶는 것!

'과일장수'의 정의와 멤버변수의 상수화



과일 값은 변하지 않는다고 가정할 때
APPLE_PRICE는 다음과 같이 선언하는 것이 좋다!
const int APPLE_PRICE;

그러나 상수는 선언과 동시에 초기화 되어야 하기
때문에 이는 불가능하다. 물론 클래스를 정의하는
과정에서 선언과 동시에 초기화는 불가능하다.



추가



추가

```
void InitMembers(int price, int num, int money)
{
    APPLE_PRICE=price;
    numOfApples=num;
    myMoney=money;
}
```

초기화를 위한 추가

```
void ShowSalesResult()
{
    cout<<"남은 사과: "<<numOfApples<<endl;
    cout<<"판매 수익: "<<myMoney<<endl;
}
```

얼마나 파셨어요? 라는 질문과 답변을 위한 함수

'나(me)'를 표현하는 클래스의 정의와 객체생성

'나'의 클래스 정의

```
class FruitBuyer
{
    int myMoney;      // private:
    int numOfApples;  // private:
public:
    void InitMembers(int money)
    {
        myMoney=money;
        numOfApples=0;    // 사과구매 이전이므로!
    }
    void BuyApples(FruitSeller &seller, int money)
    {
        numOfApples+=seller.SaleApples(money);
        myMoney-=money;
    }
    void ShowBuyResult()
    {
        cout<<"현재 잔액: "<<myMoney<<endl;
        cout<<"사과 개수: "<<numOfApples<<endl;
    }
};
```

상태

행위

일반적인 변수 선언 방식의 객체생성

```
FruitSeller seller;
FruitBuyer buyer;
```

동적 할당 방식의 객체생성

```
FruitSeller * objPtr1=new FruitSeller;
FruitBuyer * objPtr2=new FruitBuyer;
```

사과장수 시뮬레이션 완료

```
int main(void)
{
    FruitSeller seller;
    seller.InitMembers(1000, 20, 0);
    FruitBuyer buyer;
    buyer.InitMembers(5000);
    buyer.BuyApples(seller, 2000);

    cout<<"과일 판매자의 현황"<<endl;
    seller.ShowSalesResult();
    cout<<"과일 구매자의 현황"<<endl;
    buyer.ShowBuyResult();
    return 0;
}
```

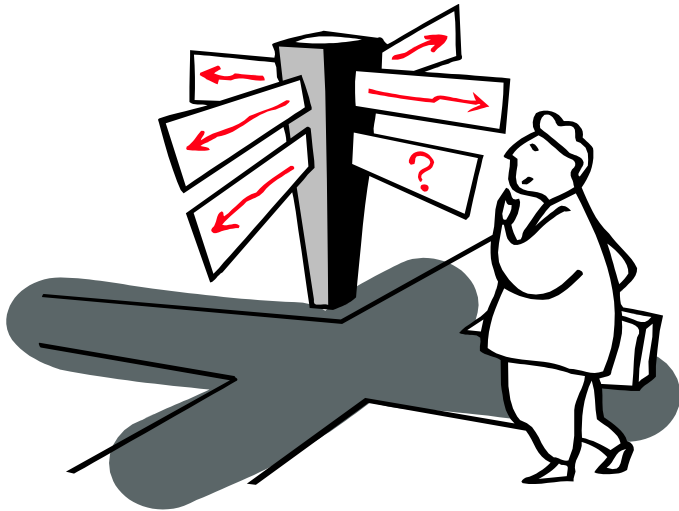
아저씨 사과 2000원어치 주세요.

아저씨 오늘 얼마나 파셨어요.. 라는 질문의 대답

너 사과 심부름 하고 나머지 잔돈이 얼마야.. 라는 질문의 대답

```
void BuyApples(FruitSeller &seller, int money)
{
    numOfApples+=seller.SaleApples(money);
    myMoney-=money;
}
```

FruitBuyer 객체가 FruitSeller 객체의 SaleApples 함수를 호출하고 있다. 그리고 객체지향에서는 이것을 '두 객체가 대화하는 것'으로 본다. 따라서 이러한 형태의 함수호출을 가리켜 '메시지 전달'이라 한다.



Chapter 03이 끝났습니다. 질문 있으신지요?