# 3. 클래스의 기본

3주차

# C++에서의 구조체

- **구조체의 등장배경**
  - 연관 있는 데이터를 하나로 묶어서 프로그램의 구현 및 관리를 용이하게 함.

- **C++에서의 구조체 변수 선언**

```
struct Car basicCar;

struct Car simpleCar;
```

```
Car basicCar;

Car simpleCar;
```

C 스타일 구조체 변수 초기화

C++ 스타일 구조체 변수 초기화

C++에서는 **struct 키워드 생략**을 위한 typedef 선언이 불필요.

# 구조체를 이용한 프로그램
## (RacingCar.cpp 1/3)

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

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

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

# 구조체를 이용한 프로그램
## (RacingCar.cpp 2/3)

```
void ShowCarState(const Car& car) {
	cout << "소유자ID: " << car.gamerID << endl;
	cout << "연료량: " << car.fuelGauge << "%" << endl;
	cout << "현재속도: " << car.curSpeed << "km/s" << endl << endl;
}
void Accel(Car& car) {
	if (car.fuelGauge <= 0)
		return;
	else
		car.fuelGauge -= FUEL_STEP;

	if (car.curSpeed + ACC_STEP >= MAX_SPD) {
		car.curSpeed = MAX_SPD;
		return;
	}

	car.curSpeed += ACC_STEP;
}
```

# 구조체를 이용한 프로그램
## (RacingCar.cpp 3/3)

```
void Break(Car& car) {
        if (car.curSpeed < BRK_STEP) {
                car.curSpeed = 0;
                return;
        }
        car.curSpeed -= BRK_STEP;
}
int main(void) {
        Car run99 = { "run99", 100, 0 };
        Accel(run99);
        Accel(run99);
        ShowCarState(run99);
        Break(run99);
        ShowCarState(run99);

        Car sped77 = { "sped77", 100, 0 };
        Accel(sped77);
        Break(sped77);
        ShowCarState(sped77);
        return 0;
}
```

# 구조체 안으로 함수 넣기
## (RacingCarFuncAdd.cpp 1/4)

[이전 프로그램과 같음]

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

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

# 구조체 안으로 함수 넣기
## (RacingCarFuncAdd.cpp 2/4)

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

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

        void Break() {
                if (curSpeed < BRK_STEP) {
                        curSpeed = 0;
                        return;
                }

                curSpeed -= BRK_STEP;
        }
```

연산 대상을 인자로 주지 않아도 된다.

같은 구조체의 멤버이므로 어느 구조체인지를 명시할 필요가 없다.

# 구조체 안으로 함수 넣기
## (RacingCarFuncAdd.cpp 3/4)

```
		void Accel()
		{
			if (fuelGauge <= 0)
				return;
			else
				fuelGauge -= FUEL_STEP;

			if (curSpeed + ACC_STEP >= MAX_SPD)
			{
				curSpeed = MAX_SPD;
				return;
			}

			curSpeed += ACC_STEP;
		}
};
```

# 구조체 안으로 함수 넣기
## (RacingCarFuncAdd.cpp 4/4)

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

        Car sped77 = { "sped77", 100, 0 };
        sped77.Accel();
        sped77.Break();
        sped77.ShowCarState();
        return 0;
}
```

구조체 변수마다
"데이터"는 각각 갖고
함수는 공유하게 됨.

# 구조체 안으로 define 값 넣기
## (RacingCarFuncAdd.cpp 수정)

115-117p
변형

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

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

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

해당 구조체 안에서만 의미 있는 define값들은 구조체 안으로 넣기.

118-121p
변형

# 구조체 선언과 정의를 분리

- 함수 정의를 분리하는 이유
  - 구조체가 포함하는 함수, 변수를 한 눈에 확인 가능
  - 소스파일과 헤더파일을 분리하여 배포 가능
- 함수 정의가 구조체 내에 있을 때의 의미
  - inline 함수임을 의미
  - 구조체 밖에 정의된 함수는 inline 이라고 명시해야만 inline으로 간주됨.

# 구조체 선언과 정의를 분리
## (RacingCarOuterFunc.cpp 1/4)

118-121p
변형

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

struct Car {
        enum {
                ID_LEN = 20,
                MAX_SPD = 200,
                FUEL_STEP = 2,
                ACC_STEP = 10,
                BRK_STEP = 10
        };
        char gamerID[ID_LEN];      // 소유자ID
        int fuelGauge;             // 연료량
        int curSpeed;              // 현재속도

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

구조체 안에는 함수 원형 선언만 남기기.

# 구조체 선언과 정의를 분리
## (RacingCarOuterFunc.cpp 2/4)

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

구조체 밖으로 옮긴 함수 이름 앞에
Car:: 를 붙여서 소속을 알려주기.

# 클래스와 구조체의 차이점

- 선언시의 키워드 :struct -> class
- 변수 선언하면서 클래스 외부에서 초기화 할 수 없음.
  - Car run99 = { “run99” , 10, 0 };
    위 코드는 Car가 구조체일 때는 OK, 클래스일 때는 NG.
- 접근 제어 지시자. 레이블(위치표시)이므로 키워드 뒤에 colon 붙여 사용.
  다음 지시자 나올 때까지 효력 있음.
  - public: 어디서든 접근허용
  - protected: 이 클래스를 상속 받은 클래스에서의 접근허용
  - private: 클래스 내(클래스 내에 정의된 함수)에서만 접근허용
- 접근 제어 지시자의 기본값
  - 구조체 : public
  - 클래스 : private

# 클래스와 구조체의 차이점
## (RacingCarClassBase.cpp 변형(소스부분적수정) 1/2)

122-127p

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
using namespace std;
class Car {
        enum {
            ID_LEN = 20,
            MAX_SPD = 200,
            FUEL_STEP = 2,
            ACC_STEP = 10,
            BRK_STEP = 10
        };
        char gamerID[ID_LEN];// 소유자ID
        int fuelGauge;      // 연료량
        int curSpeed;       // 현재속도
public:
        void InitMembers(char* ID, int fuel);
        void ShowCarState();
        void Break();
        void Accel();
};
```

strcpy 함수 사용 위해.

클래스 외부함수에서도 호출할 수 있게 함. 명시하지 않으면 기본으로 private으로 간주됨.

[용어] 멤버변수

[용어] 멤버함수

클래스 멤버 초기화를 위한 멤버함수

# 클래스와 구조체의 차이점
## (RacingCarClassBase.cpp 변형(소스 부분적으로 수정) 2/2)

```
int main(void)
{
        Car run99;
        run99.InitMembers( (char *)"run99", 100 );
        run99.Accel();
        run99.Accel();
        run99.ShowCarState();
        run99.Break();
        run99.ShowCarState();

        Car sped77;
        sped77.InitMembers((char *)"sped77", 100);
        sped77.Accel();
        sped77.Break();
        sped77.ShowCarState();
        return 0;
}
```

선언하면서 초기화하지 않고 초기화 함수 호출.

[용어] 클래스 객체선언 ("변수" 아니고 "객체")

127-131p

# C++에서의 파일분할

- 클래스 별로 클래스 이름을 딴 .h 와 .c 파일 만듬
  - .h 파일 : 클래스 선언. 클래스 사용 시 include.
  - .cpp 파일 : 클래스 정의.
- 앞선 예제에 Car.h, Car.cpp 추가하기

# C++에서의 파일분할
## (Car.h)

```
#ifndef _CAR_H_
#define _CAR_H_
class Car {
        enum {
                ID_LEN = 20,
                MAX_SPD = 200,
                FUEL_STEP = 2,
                ACC_STEP = 10,
                BRK_STEP = 10
        };
        char gamerID[ID_LEN];      // 소유자ID
        int fuelGauge;             // 연료량
        int curSpeed;              // 현재속도
public:
        void InitMembers(char* ID, int fuel);
        void ShowCarState();
        void Break();
        void Accel();
};
#endif // _CAR_H_
```

# C++에서의 파일분할
## (Car.cpp)

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include "Car.h"
using namespace std;

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

# C++에서의 파일분할
## (RacingMain.cpp)

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include "Car.h"
using namespace std;
int main(void) {
	Car run99;
	run99.InitMembers( (char *)"run99", 100 );
	run99.Accel();
	run99.Accel();
	run99.ShowCarState();
	run99.Break();
	run99.ShowCarState();

	Car sped77;
	sped77.InitMembers((char *)"sped77", 100);
	sped77.Accel();
	sped77.Break();
	sped77.ShowCarState();
	return 0;
}
```

# 파일 분할에서 inline 함수의 위치

- 헤더 파일에 있어야 함!
  - 컴파일 과정에서 함수 정의부가 함수의 호출문을 대체해야 하기 때문에 헤더파일에 있어야 함.
  - class 내부에 포함되건 class 외부에 따로 정의되건 상관없음.
    헤더 파일에 위치하면 됨.

136-137p

# 객체지향 프로그래밍의 이해
## (FruitSaleSim1.cpp)

- C : 절차지향, C++ : 객체지향
- 객체 = 상태정보(데이터) + 행동(기능)(함수)
- 사과장수(class FruitSeller)의 데이터와 함수
  - 데이터
    - int APPLE_PRICE; // 차후 상수로 사용하려고 대문자로 정함.
    - int numOfApples;
    - int myMoney;
  - 함수
    - 멤버변수 초기화
      void InitMembers(int price, int num, int money);
    - 사과 판매
      int SaleApples(int money);
    - 현재 상태 보이기
      void ShowSalesResult();
- 클래스의 정의 = 타입 생성 ( != 객체 생성)
- 클래스 타입의 변수 선언 또는 할당 = 객체 생성
  - FruitSeller seller;
  - FruitSeller *pSeller = new FruitSellet;

# 객체지향 프로그래밍의 이해
## (FruitSaleSim1.cpp 1/3)

```
#include <iostream>
using namespace std;
class FruitSeller {
        int APPLE_PRICE;
        int numOfApples;
        int myMoney;
public:
        void InitMembers(int price, int num, int money)       {
                APPLE_PRICE = price;
                numOfApples = num;
                myMoney = money;
        }
        int SaleApples(int money) {
                int num = money / APPLE_PRICE;
                numOfApples -= num;
                myMoney += money;
                return num;
        }
        void ShowSalesResult()     {
                cout << "남은 사과: " << numOfApples << endl;
                cout << "판매 수익: " << myMoney << endl << endl;
        }
};
```

# 객체지향 프로그래밍의 이해
## (FruitSaleSim1.cpp 2/3)

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

다른 객체(seller)에게 메시지 전달(Message Passing).
함수 호출 기반으로 이루어짐.

# 객체지향 프로그래밍의 이해
## (FruitSaleSim1.cpp 3/3)

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

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

	buyer.BuyApples(seller, 2000);

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