C++_구조체, 클래스와 객체

Visual Studio 2022

C++ 구조체

❖ 구조체(structure)란?

다양한 자료형을 그룹화하여 하나의 변수로 처리할 수 있게 만든 자료형이다.

개발자가 다양한 정보를 저장하기 위해 필요에 따라 생성하는 자료형을 사용자 정의 자료형 또는 구조체라 한다.

- 구조체 정의

```
struct 구조체이름{
    자료형 멤버이름;
};
```

- 객체 생성

```
구조체이름 변수이름;
```

C++ 구조체

❖ 구조체(structure)의 정의와 사용

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

struct Student {
    string name;      //이름
    int grade;        //학년
    string address;   //주소
};
```

```
int main() {
    Student st1 = Student(); //기본 생성자로 객체 생성
    //struct는 public 이므로 멤버 접근 허용됨.
    st1.name = "이우주";
    st1.grade = 3;
    st1.address = "서울시 노원구 상계동";

    cout << "학생 이름 : " << st1.name << endl;
    cout << "학년 : " << st1.grade << endl;
    cout << "주소 : " << st1.address << endl;

    return 0;
}
```

구조체 배열

❖ 구조체 배열 – 객체를 여러 개 생성

```
//구조체 배열
Student studentArr[3] = {
    {"김지구", 1, "서울시 종로구"},
    {"박화성", 2, "서울시 은평구"},
    {"최목성", 3, "서울시 강남구"}
};

for (int i = 0; i < 3; i++) {
    cout << studentArr[i].name << " 학생은 "
         << studentArr[i].grade << "학년 입니다." << endl;
}
```

열거형 자료형 enum

❖ **enum 자료형**

- enumeration(열거하다)의 영문 약자 키워드로 , 사용자가 직접 정의하여 사용할 수 있는 자료형이다.
- 열거형은 정수형 상수에 이름을 붙여서 코드를 이해하기 쉽게 해줌
- 열거형은 상수를 편리하게 정의할 수 있게 해줌

```
const int VALUE_A = 1;
const int VALUE_B = 2;
const int VALUE_C = 3;
```

```
enum VALUE{
    VALUE_A = 1,
    VALUE_B
    VALUE_C
}
```

※ 상수의 개수가 많아지면 선언하기에 복잡해짐

열거형 자료형 enum

❖ enum 자료형

```
enum VALUE {
    //기본 인덱스는 0부터 시작함
    VALUE_A = 1,
    VALUE_B,
    VALUE_C
};
```

```
int main()
{
    //상수 선언
    /*const int VALUE_A = 1;
    const int VALUE_B = 2;
    const int VALUE_C = 3;

    cout << VALUE_A << endl;
    cout << VALUE_B << endl;
    cout << VALUE_C << endl;*/

    // enum 자료형 사용
    enum VALUE value;
    value = VALUE_C;

    cout << value << endl;

    return 0;
}
```

열거형 자료형 enum

❖ **enum 자료형**

enum **열거형 이름**{
  값1 = 초기값,
  값2,
  값3
}

→ enum 열거형 이름 **변수 이름**

```
enum WEEK {
    SUN = 1,  //초기값
    MON,
    TUE,
    WED,
    THR,
    FRI,
    SAT
};
```

```
int main() {
    enum WEEK week;    //선언
    week = WED;
    //int week = WED;  사용 가능

    cout << week << endl;

    return 0;
}
```

열거형 자료형 enum

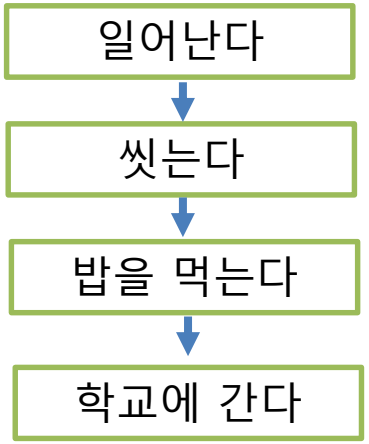
❖ switch ~ case 문에서 사용하기

```
//열거형 상수 정의
enum MEDAL {
    GOLD = 1,
    SILVER,
    BRONZE
};
```

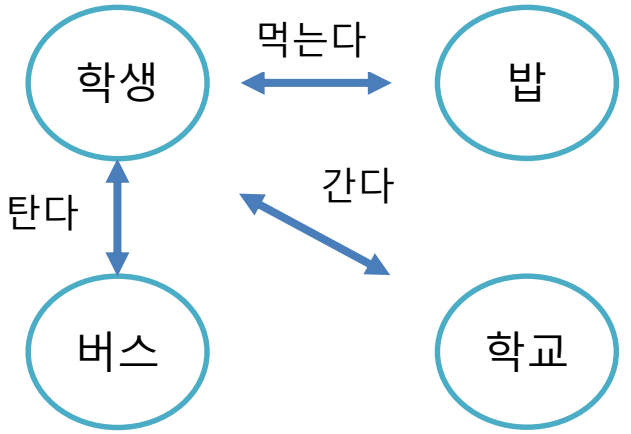
```
//enum MEDAL medal; //선언
//medal = SILVER;   //사용
//int medal = SILVER; //사용 가능

int medal;
cout << "메달 선택(1 ~ 3 입력): ";
cin >> medal;

switch (medal)
{
case GOLD:
    cout << "금메달" << endl;
    break;
case SILVER:
    cout << "은메달" << endl;
    break;
case BRONZE:
    cout << "동메달" << endl;
    break;
default:
    cout << "메달이 없습니다. 다시 입력하세요" << endl;
    break;
}
```

객체 지향 프로그래밍

- **객체(Object)란?**
  - 의사나 행위가 미치는 대상 -> 사전적 의미
  - 구체적, 추상적 데이터 단위 (구체적- 책상, 추상적-회사)
- **객체지향 프로그래밍(Objected Oriented Programming, OOP)**
  - 객체를 기반으로 하는 프로그래밍
  - 먼저 객체를 만들고, 객체 사이에 일어나는 일을 구현함.



<절차지향 –C언어>



<객체지향 –C++,Java>

객체지향 프로그래밍이란?

절차지향 프로그래밍

```
작업 1();
작업 2();
작업 3();
.
.
작업 99();
작업 100();
```

작업(함수) 100개가 동등한 위치에서 나열되어 있다.

객체지향 프로그래밍

```
작업 1();
작업 2();
작업 3();
.
.
작업 10();
```

```
작업 1();
작업 2();
작업 3();
.
.
작업 10();
```

```
작업 1();
작업 2();
작업 3();
.
.
작업 10();
```

연관있는 작업을 객체로 묶어서 처리하기때문에 보다 효율적으로 관리할 수 있다.

C++의 객체 지향 특성

- 객체와 캡슐화(Encapsulation)

  캡슐화는 데이터를 캡슐로 싸서 외부의 접근으로부터 데이터를 보호하는 객체 지향 특성이다.

  C++에서는 캡슐의 역할을 하는 것이 클래스이며 **class 키워드**를 이용하여 작성한다.

  객체는 클래스라는 틀에서 생겨난 실체(instance)이다.

  C++ 클래스는 멤버변수들과 멤버 함수들로 이루어지며, 멤버들은 캡슐 외부에 공개하거나(public), 보이지 않게(private) 선언할 수 있다.

- 상속성

  자식 클래스의 객체가 생성될때 부모 클래스의 멤버나 함수를 사용할수 있다.

구조체의 진화 -〉 클래스

상태(State)

```
struct Dog{
  char name[20];
  int age;
  char type[20];
};
```

(구조체)

객체(Object)

행위(Behaviour)

```
void bark();
void eat();
void sleep();
```

(일반함수)

★ 절차지향적 접근 방법

구조체의 진화 -〉 클래스

객체(Object)

```
class Dog{
    string name;
    int age;
    string type;

    void bark();
    void eat();
    void sleep();
};
```

클래스

★ 클래스는 구조체에 함수를 포함시킨 틀이다.

클래스 정의 및 사용

- 클래스(class) 정의
  - 클래스란 객체(사물)를 추상화한 자료형이다.
  - 클래스란 객체를 정의하는 틀 혹은 설계도이다.
  - 클래스에 멤버 변수와 멤버 함수를 선언한다.
  - 클래스 이름은 대문자로 시작한다.
  - 접근 제어자 private – 접근 불허, public – 접근 허용

```
class 클래스 이름{
private:
        멤버 변수;
public:
        멤버 함수;
}
```

- 인스턴스(instance)
  - 클래스를 사용하기 위해 생성된 객체를 인스턴스(instance)라 한다.
  - 인스턴스로 클래스의 멤버변수에 점(.) 연산자로 접근하여 값을 지정한다.

```
클래스 이름 인스턴스(객체)
인스턴스.멤버변수
```

Dog 클래스 만들기

```
//Dog 클래스 정의
class Dog {
public:  //접근 제어자 - 멤버변수 및 함수에 접근 허용
    string type;  //종류
    int age;      //나이

    void dogInfo()
    {
        cout << "강아지 종류 : " << type << endl;
        cout << "강아지 나이 : " << age << "세" << endl;

    }

    void bark()
    {
        cout << "멍~ 멍~\n";
    }
};
```

멤버 변수

멤버 함수

Dog 클래스 만들기

```
강아지 종류 : 푸들
강아지 나이 : 2세
멍~ 멍~
강아지 종류 : 진돗개
강아지 나이 : 3세
멍~ 멍~
```

```
int main()
{
    Dog dog1;  //객체(인스턴스) 생성

    dog1.type = "푸들";  //멤버 변수 초기화
    dog1.age = 2;
    dog1.dogInfo();
    dog1.bark();

    Dog dog2; //객체(인스턴스) 생성

    dog2.type = "진돗개";
    dog2.age = 3;
    dog2.dogInfo();
    dog2.bark();

    return 0;
}
```

클래스 선언과 구현부 분리

```
//Dog 클래스 정의 - 함수 구현부 분리
class Dog {
public:  //접근 제어자 - 멤버변수 및 함수에 접근 허용
    string type;  //종류
    int age;      //나이

    void dogInfo();
    void bark();
};

void Dog::dogInfo()
{
    cout << "강아지 종류 : " << type << endl;
    cout << "강아지 나이 : " << age << "세" << endl;
}

void Dog::bark()
{
    cout << "멍~ 멍~\n";
}
```

멤버함수 선언부

분리하는 이유-클래스의 재사용을 위해서...
클래스를 사용하는 다른 C++파일에서는 컴파일 시 클래스 선언부만 필요하기 때문이다.

멤버함수 구현부

생성자(Constructor) – 기본 생성자

```
//Dog 클래스 정의 - 기본 생성자 사용
class Dog {
public:
    //멤버 변수
    string type;
    int age;

    //기본 생성자
    Dog();

    //멤버 함수
    void dogInfo();
    void bark();
};

//기본 생성자 초기화
Dog::Dog()
{
    type = "푸들";
    age = 2;
}
```

생성자 구현부

★ 생성자는 객체가 만들어질때 자동으로 호출된다.
- 이름이 클래스와 동일하다.
- 생성자는 반환형이 없다.
- 생성자가 정의되어 있지 않으면 컴파일러가 자동으로 기본생성자(default constructor)를 제공한다.
- 기본생성자는 매개변수가 없는 생성자이다.

```
int main()
{
    Dog dog;  //객체(기본 생성자)
    dog.dogInfo();

    return 0;
}
```

매개 변수가 있는 생성자

```
class Dog {
public:
    //멤버 변수
    string type;
    int age;

    Dog(); //기본 생성자
    Dog(string t, int a); //매개변수 있는 생성자

    //멤버 함수
    void dogInfo();
    void bark();
};

Dog::Dog(){
    type = "강아지";
    age = 1;
}

Dog::Dog(string t, int a){
    type = t;
    age = a;
}
```

★ 생성자 오버로딩 – 생성자를 여러 개 사용할 수 있다
- 이름이 같고 매개변수가 다르다.

```
int main()
{
    Dog dog;    //기본 생성
    dog.dogInfo();

    Dog dog2("진돗개", 3); //생성자에 입력
    dog2.dogInfo();

    return 0;
}
```

소멸자(destructor)

```
class Dog {
public:
    //멤버 변수
    string type;
    int age;

    Dog(); //기본 생성자
    Dog(string t, int a); //매개변수 있는 생성자
    ~Dog();  //소멸자 - 생략 가능

    //멤버 함수
    void dogInfo();
    void bark();
};

Dog::Dog(){
    type = "강아지";
    age = 1;
}

Dog::~Dog() {
    cout << "객체가 소멸됩니다..\n";
}
```

★ 소멸자는 객체가 생성된후 자동으로 호출된다.
- 이름이 클래스와 동일하다.
- 이름 앞에 '~'을 붙인다.
- 소멸자가 정의되어 있지 않으면 컴파일러가 자동으로 기본소멸자를 제공한다.

헤더 파일(.h), cpp 파일로 분할하기

✓ 분할 컴파일

1. Dog.h – 헤더 파일(클래스 포함)
2. Dog.cpp – 함수 포함
3. Main.cpp – 실행 파일(객체 생성)

Dog.h

```
//조건부 컴파일 블록
//헤더파일이 중복정의되지 않도록 해줌
#ifndef DOG_H
#define DOG_H  //매크로 정의

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

class Dog {
public:
    string type;
    int age;

    Dog(string t, int a);

    void dogInfo();
    void bark();
};

#endif  //조건부 컴파일 블록 종료
```

헤더 파일(.h), cpp 파일로 분할하기

Dog.cpp

```cpp
#include "Dog.h"

//생성자 초기화
Dog::Dog(string t, int a)
{
    type = t;
    age = a;
}

void Dog::dogInfo()
{
    cout << "강아지 종류 : " << type << endl;
    cout << "강아지 나이 : " << age << "세" << endl;
}
```

Main.cpp

```cpp
#include "Dog.h"

void Dog::bark()
{
    cout << "멍~ 멍~\n";
}

int main()
{
    Dog dog1("푸들", 2);
    dog1.dogInfo();

    Dog dog2("진돗개", 3);
    dog2.dogInfo();

    return 0;
}
```

정보은닉(Information Hiding)

- **정보 은닉**
  - **접근 제어자 : 접근 권한 지정**
    - **public : 외부 클래스에서 접근 가능**
    - **private : 클래스의 외부에서 클래스 내부의 멤버 변수나 메서드에 접근 못하게 하는 경우 사용**
  - **변수 접근하기 위해 get(), set() 함수를 만들어 사용한다.**

접근 지정자	설 명
public	외부 클래스 어디에서나 접근 할수 있다.
protected	같은 클래스와 상속관계의 모든 자식클레스에서 접근 가능
private	같은 클래스 내부 가능, 그 외 접근 불가

정보은닉(Information Hiding)

- Book 클래스

```
//Book 클래스 정의 - 정보 은닉
class Book {
private:
    int number;    //책 번호
    string title;  //책 제목
    string author; //저자

public:
    //Book();   //기본 생성자(생략)

    //get(), set() 함수로 private 멤버에 접근
    void setNumber(int n);
    int getNumber();
    void setTitle(string t);
    string getTitle();
    void setAuthor(string a);
    string getAuthor();
};
```

정보은닉(Information Hiding)

- Book 클래스

```
void Book::setNumber(int n){
    number = n;
}

int Book::getNumber(){
    return number;
}

void Book::setTitle(string t) {
    title = t;
}

string Book::getTitle() {
    return title;
}

void Book::setAuthor(string a) {
    author = a;
}

string Book::getAuthor() {
    return author;
}
```

```
int main()
{
    Book book1;

    book1.setNumber(100);
    book1.setTitle("채식주의자");
    book1.setAuthor("한강");

    cout << "********** 책의 정보 **********" << endl;
    cout << "책 번호 : " << book1.getNumber() << endl;
    cout << "책 제목 : " << book1.getTitle() << endl;
    cout << "책 저자 : " << book1.getAuthor() << endl;

    return 0;
}
```

정보은닉(Information Hiding)

- Book 클래스 – 객체 배열

```
class Book {
private:
    int number;     //책 번호
    string title;   //책 제목
    string author; //저자
public:
    Book(int n, string t, string a);

    //get() 함수만 사용
    int getNumber();
    string getTitle();
    string getAuthor();
};
```

```
// 생성자 초기화
Book::Book(int n, string t, string a) {
    number = n;
    title = t;
    author = a;
}

int Book::getNumber() {
    return number;
}

string Book::getTitle() {
    return title;
}

string Book::getAuthor() {
    return author;
}
```

정보은닉(Information Hiding)

- Book 클래스

```
int main()
{
    //객체 배열
    Book book[3] = {
        Book(100, "채식주의자", "한강"),
        Book(101, "C++ 완전정복", "조규남"),
        Book(102, "모두의 C언어", "이형우"),
    };

    cout << "********** 책의 정보 **********" << endl;
    for (int i = 0; i < 3; i++)
    {
        cout << "책 번호 : " << book[i].getNumber() << endl;
        cout << "책 제목 : " << book[i].getTitle() << endl;
        cout << "책 저자 : " << book[i].getAuthor() << endl;
    }

    return 0;
}
```

this 예약어

- **자신의 메모리를 가리키는 this**
  - 생성된 인스턴스 스스로를 가리키는 예약어
  - 객체 자신의 메모리상의 주소를 나타내는 포인터이다.

```
//Car 클래스 정의
class Car{
private:
    string model;
    int year;

public:
    /*
        생성자
        - 외부 입력을 this로 초기화(저장)
        - 변수이름이 같아야 함
    */
    Car(string model, int year) {
        this->model = model;
        this->year = year;
    }
    void drive();
    void carInfo();
};
```

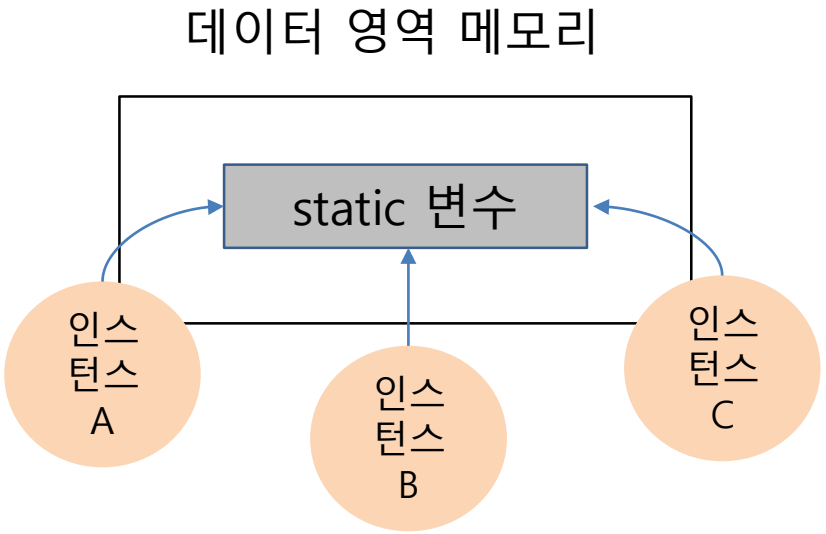
this 예약어

- 자신의 메모리를 가리키는 this

```
void Car::drive(){
    cout << "차가 달립니다.\n";
}

void Car::carInfo() {
    cout << "모델명: " << this->model << endl;
    cout << "년식: " << this->year << endl;
}

int main()
{
    Car car1("Avante", 2016);  //car1 객체 생성

    car1.drive();
    car1.carInfo();
    cout << "====================\n";

    Car car2("Ionic6", 2023);  //car2 객체 생성
    car2.drive();
    car2.carInfo();

    return 0;
}
```

static 변수

- **static 변수의 정의와 사용 방법**
  - 다른 멤버변수처럼 인스턴스가 생성될 때마다 새로 생성되는 변수가 아니다.
  - 프로그램이 실행되어 메모리에 적재(load)될때 메모리 공간이 할당된다.
  - 여러 개의 인스턴스가 같은 메모리의 값을 공유하기 위해 사용



Static 예약어

```
static int serialNum=1000;
```

static 변수

- **카드번호 자동 발급**

```
/*
    카드 번호 자동 발급
*/
class Card {
private:
    static int serialNum; //static 변수
    string name;     //고객 이름
    int cardNumber;  //카드 번호
```

```
public:
    Card(string name) {
        serialNum++;  //1증가
        cardNumber = serialNum;
        this->name = name;
    }

    string getName(){
        return name;
    }
    int getCardNumber() {
        return cardNumber;
    }
};
```

static 변수

- **카드번호 자동 발급**

```
int Card::serialNum = 1000;  //전역 변수

int main()
{
    Card card1("신유빈");
    cout << "고객 이름: " << card1.getName() << endl;
    cout << "카드 번호: " << card1.getCardNumber() << endl;

    Card card2("이정후");
    cout << "고객 이름: " << card2.getName() << endl;
    cout << "카드 번호: " << card2.getCardNumber() << endl;

    Card card3("한강");
    cout << "고객 이름: " << card3.getName() << endl;
    cout << "카드 번호: " << card3.getCardNumber() << endl;

    return 0;
}
```

static 변수

- 개인 돈과 회비 관리

```
/*
    개인 돈과 회비 관리하기
*/
class Person {
public:
    int money;  //개인 돈
    static int sharedMoney; //회비

    void addMoney(int money) {
        this->money = money;
    }

    static void addSharedMoney(int money) {
        sharedMoney += money;
    }
};
```

static 변수

- 개인 돈과 회비 관리

```
int Person::sharedMoney = 0; //전역 변수 0으로 초기화

int main()
{
    Person kim;
    kim.money = 20000;
    kim.sharedMoney = 10000;

    Person lee;
    lee.money = 30000;
    lee.sharedMoney = 10000;

    //개인 돈 출력
    cout << kim.money << ' ' << lee.money << endl;

    //회비 출력
    cout << kim.sharedMoney << ' ' << lee.sharedMoney << endl;

    return 0;
}
```

객체의 동적 생성 및 반환

- **동적 객체 생성**

```
Car* car1 = new Car()
```

- **동적 객체 반환**

```
delete car1;
```

객체의 동적 생성 및 반환

- **기본 생성자로 생성**

```
//Car 클래스 정의
class Car{
private:
    string model;
    int year;

public:
    //기본 생성자
    Car() {};

    //멤버 함수
    void setModel(string model);
    void setYear(int year);
    void carInfo();
    void drive();
};
```

객체의 동적 생성 및 반환

- 기본 생성자로 생성

```
void Car::setModel(string model) {
    this->model = model;
}

void Car::setYear(int year) {
    this->year = year;
}

void Car::carInfo() {
    cout << "모델명: " << this->model << endl;
    cout << "년식: " << this->year << endl;
}

void Car::drive() {
    cout << "차가 달립니다.\n";
}
```

객체의 동적 생성 및 반환

- **기본 생성자로 생성**

```
int main()
{
    //동적 객체 생성
    Car* car1 = new Car();
    car1->setModel("Sonata");
    car1->setYear(2017);
    car1->carInfo();
    car1->drive();

    cout << "--------------------\n";

    Car* car2 = new Car();
    car2->setModel("EV3");
    car2->setYear(2024);
    car2->carInfo();
    car2->drive();

    delete car1;  //객체 반환
    delete car2;

    return 0;
}
```

객체의 동적 생성 및 반환

- 매개변수가 있는 생성자로 생성

```
class Car{
private:
    string model;
    int year;

public:
    //생성자 - this로 초기화
    Car(string model, int year) {
        this->model = model;
        this->year = year;
    }
    //멤버 함수
    void carInfo();
    void drive();
};

void Car::carInfo() {
    cout << "모델명: " << this->model << endl;
    cout << "년식: " << this->year << endl;
}

void Car::drive() {
    cout << "차가 달립니다.\n";
}
```

객체의 동적 생성 및 반환

- 매개변수가 있는 생성자로 생성

```
int main()
{
    //동적 객체 생성
    Car* car1 = new Car("Sonata", 2017);
    car1->carInfo();
    car1->drive();

    cout << "-------------------\n";

    Car* car2 = new Car("EV3", 2024);
    car2->carInfo();
    car2->drive();

    delete car1;  //객체 반환
    delete car2;

    return 0;
}
```

객체 배열의 동적 생성 및 반환

- **동적 객체 배열 생성**

```
Car* cars = new Car[3]
```

- **동적 객체 배열 반환**

```
Delete [ ] cars;
```

객체 배열의 동적 생성 및 반환

- 기본 생성자로 생성

```
int main()
{
    //동적 객체 배열 생성
    Car* cars = new Car[3];
    //인덱스로 저장
    cars[0].setModel("Sonata");
    cars[0].setYear(2017);
    cars[1].setModel("Ionic6");
    cars[1].setYear(2023);
    cars[2].setModel("EV3");
    cars[2].setYear(2024);
```

```
    //전체 출력
    for (int i = 0; i < 3; i++) {
        cars[i].carInfo();
        cars[i].drive();
        cout << "-------------------\n";
    }

    delete [] cars;  //동적 객체 배열 반환

    return 0;
}
```

객체 배열의 동적 생성 및 반환

- 매개 변수가 있는 생성자로 생성

```
int main()
{
    //객체 배열 생성
    Car* cars = new Car[3]{
        Car("Sonata", 2017),
        Car("Ionic6", 2023),
        Car("EV3", 2024)
    };

    //인덱싱 조회
    /*cars[0].carInfo();
    cars[1].carInfo();
    cars[2].carInfo();*/
```

```
    //전체 출력
    for (int i = 0; i < 3; i++) {
        cars[i].carInfo();
        cars[i].drive();
        cout << "-------------------\n";
    }

    delete [] cars;  //객체 배열 반환

    return 0;
}
```

실습 문제 1 – 클래스

회원(Member) 클래스를 정의하고 아래와 같이 구현하세요.

[파일이름: MemberTest.cpp]

데이터 이름	필드 이름	타입	접근 제어
아이디	id	문자열	private
패스워드	password	문자열	private

☞ 실행 결과

```
********** 회원 현황 **********
아이디: flower, 패스워드: f1234
아이디: tree, 패스워드: t1234
아이디: bird, 패스워드: b1234
```