static 멤버 함수

- 클래스의 모든 객체가 공유
- public으로 선언되면, 클래스 이름을 이용하여 호출 가능.
- 객체에 소속된 것이 아니고 클래스에 소속된 것이다.
- (객체 내에 존재하는 함수가 아니기 때문에) 멤버변수나 멤버함수에 접근 불가능.
- static 변수, static 함수에만 접근 가능.

```
class SoSimple
{
private:
    int num1;
    static int num2;
public:
    SoSimple(int n): num1(n)
    { }
    static void Adder(int n)
    {
        num1+=n;     // 컴파일 에러 발생
        num2+=n;
    }
};
int SoSimple::num2=0;
```

static 멤버 함수에 객체 전달하여 활용

◆ **break point 잡아서 obj 객체와 SoSimple::num2 값을 확인!**

Project1 SoSimpl

```
1  #include <iostream>
2  using namespace std;
3
4  class SoSimple {
5      int num1;
6      static int num2;
7  public:
8      SoSimple(int n) : num1(n) {}
9      static void Adder(SoSimple &item, int n) {
10         item.num1 += n;
11         item.num2 += n;
12     }
13 };
14 int SoSimple::num2 = 0;
15
16 int main(void) {
17     SoSimple obj(7);
18     SoSimple::Adder(obj, 1);
19     return 0;
20 }
```

객체를 인자로 주어 그의 멤버에 접근.

const static 멤버
(ConstStaticMember.cpp)

- const 멤버는 constructor 함수에서 initializer를 이용해 초기화 하지만
- const static 멤버는 선언시에 초기화 가능. (상수이기 때문에)

```
#include <iostream>
using namespace std;
class CountryArea {
public:
        const static int RUSSIA = 1707540;
        const static int CANADA = 998467;
        const static int CHINA = 957290;
        const static int SOUTH_KOREA = 9922;
};
int main(void) {
    cout << "러시아 면적: " << CountryArea::RUSSIA << "㎢" << endl;
    cout << "캐나다 면적: " << CountryArea::CANADA << "㎢" << endl;
    cout << "중국 면적: " << CountryArea::CHINA << "㎢" << endl;
    cout << "한국 면적: " << CountryArea::SOUTH_KOREA << "㎢" << endl;
    return 0;
}
```

7. 상속의 이해

7주차

272-278p

클래스의 상속

- 기존의 개념
  - 정의해 놓은 클래스의 재활용을 목적으로 만들어진 문법적 요소
- 새로운 개념
  - 프로그램 확장성과 유연성 해결의 도구

상속과 관련된 문제: 급여관리 프로그램

- 직원 근무 형태가 정규직(Permanent) 하나인 급여관리 시스템.

- 클래스
  - 데이터 클래스 : PermanentWorker
  - 컨트롤 클래스 : EmployeeHandler

272-278p

상속과 관련된 문제: 급여관리 프로그램 (EmployeeManager1.cpp 1/3)

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring>
using namespace std;

class PermanentWorker {
private:
	char name[100];
	int salary;
public:
	PermanentWorker(char* name, int money)
	  : salary(money) {
		strcpy(this->name, name);
	}
	int GetPay() const {
		return salary;
	}
	void ShowSalaryInfo() const {
		cout << "name: " << name << endl;
		cout << "salary: " << GetPay() << endl << endl;
	}
};
```

272-278p

상속과 관련된 문제: 급여관리 프로그램 (EmployeeManager1.cpp 2/3)

```
class EmployeeHandler {
private:
        PermanentWorker* empList[50];
        int empNum;
public:
        EmployeeHandler() : empNum(0) { }
        void AddEmployee(PermanentWorker* emp) {
                empList[empNum++] = emp;
        }
        void ShowAllSalaryInfo() const {
                for (int i = 0; i < empNum; i++)
                        empList[i]->ShowSalaryInfo();
        }
        void ShowTotalSalary() const {
                int sum = 0;
                for (int i = 0; i < empNum; i++)
                        sum += empList[i]->GetPay();
                cout << "salary sum: " << sum << endl;
        }
        ~EmployeeHandler() {
                for (int i = 0; i < empNum; i++)
                        delete empList[i];
        }
};
```

구조체의 배열

상속과 관련된 문제: 급여관리 프로그램 (EmployeeManager1.cpp 3/3)

272-278p

```
int main(void)
{
	// 직원관리를 목적으로 설계된 컨트롤 클래스의 객체생성
	EmployeeHandler handler;

	// 직원 등록
	handler.AddEmployee(new PermanentWorker((char*)"KIM", 1000));
	handler.AddEmployee(new PermanentWorker((char*)"LEE", 1500));
	handler.AddEmployee(new PermanentWorker((char*)"JUN", 2000));

	// 이번 달에 지불해야 할 급여의 정보
	handler.ShowAllSalaryInfo();

	// 이번 달에 지불해야 할 급여의 총합
	handler.ShowTotalSalary();
	return 0;
}
```

272-278p

상속과 관련된 문제: 급여관리 프로그램
(문제 제시)

- 직원 고용 형태 다양화
  - 정규직(기존)
    - 연봉제. 매달 고정 급여.
  - 영업직(Sales)
    - 기본 급여 + 인센티브
  - 임시직(Temporary)
    - 시간당 급여 x 일한 시간

- 데이터 클래스의 추가와 더불어 컨트롤 클래스인 EmployeeHandlder 에 대한 변경으로 이어짐.
  - 좋은 코드는 요구사항의 변경 및 기능의 추가에 따른 변경이 최소화되어야 함.
  - 이를 위한 해결책으로 상속이 사용됨.

상속의 문법적인 이해

- A클래스(유도클래스.derived class)가 B클래스(기초클래스. Base class)를 상속받으면 A클래스는 B클래스의 모든 멤버를 물려 받는다.
  - A클래스에서 선언한 멤버들과 B클래스의 멤버들을 다 갖게 됨.
- 유도클래스 생성자에서 기초클래스 생성자를 호출하여 기초클래스 멤버 초기화. (initializer 이용)
- private 멤버는 유도 클래스에서도 접근이 불가능
  - 기초클래스의 public 함수를 통해서 기초 클래스의 private 멤버에 접근.

278-284p

상속의 문법적인 이해
(UnivStudentInheri.cpp 1/2)

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring>
using namespace std;

class Person {
private:
	int age;		// 나이
	char name[50];	// 이름
public:
	Person(int myage, char* myname) : age(myage) {
		strcpy(name, myname);
	}
	void WhatYourName() const {
		cout << "My name is " << name << endl;
	}
	void HowOldAreYou() const {
		cout << "I'm " << age << " years old" << endl;
	}
};
```

상속의 문법적인 이해
(UnivStudentInheri.cpp 2/2)

278-284p

```
class UnivStudent : public Person {
private:
        char major[50];     // 전공과목
public:
        UnivStudent( char* myname, int myage, char* mymajor )
                : Person(myage, myname) {
                strcpy(major, mymajor);
        }
        void WhoAreYou() const {
                WhatYourName();
                HowOldAreYou();
                cout << "My major is " << major << endl << endl;
        }
};
int main(void) {
        UnivStudent ustd1((char *)"Lee", 22, (char*)"Computer eng.");
        ustd1.WhoAreYou();

        UnivStudent ustd2((char*)"Yoon", 21, (char*)"Electronic eng.");
        ustd2.WhoAreYou();
        return 0;
};
```

상속

기초클래스용 데이터까지 다 인자로 받음.

기초클래스의 생성자를 호출하는 member initializer

상속받은 함수 호출

유도 클래스의 객체 생성과정 (DerivedCreOrder.cpp 1/3)

- 유도 클래스의 객체 생성 과정에서 기초 클래스의 생성자는 100% 호출됨.
  - 유도클래스 생성자에서 기초클래스 생성자를 명시적으로 호출하지 않으면 기초클래스의 default constructor가 호출됨.
  - 호출순서
    - 기초클래스의 constructor 먼저
    - 유도클래스의 constructor 나중

```
#include <iostream>
using namespace std;
class SoBase {
        int baseNum;
public:
        SoBase() : baseNum(20) {
                cout << "SoBase()" << endl;
        }
        SoBase(int n) : baseNum(n) {
                cout << "SoBase(int n)" << endl;
        }
        void ShowBaseData() {
                cout << baseNum << endl;
        }
};
```

유도 클래스의 객체 생성과정
(DerivedCreOrder.cpp 2/3)

285-291p

```
class SoDerived : public SoBase {
private:
        int derivNum;
public:
        SoDerived() : derivNum(30)
        {
                cout << "SoDerived()" << endl;
        }
        SoDerived(int n) : derivNum(n)
        {
                cout << "SoDerived(int n)" << endl;
        }
        SoDerived(int n1, int n2) : SoBase(n1), derivNum(n2)
        {
                cout << "SoDerived(int n1, int n2)" << endl;
        }
        void ShowDerivData()
        {
                ShowBaseData();
                cout << derivNum << endl;
        }
};
```

상속

기초클래스의 constructor를 명시적으로 호출 안 했으므로 기초클래스의 default constructor가 호출됨.

유도 클래스의 객체 생성과정
(DerivedCreOrder.cpp 3/3)

285-291p

```
int main(void)
{
        cout << "case1..... " << endl;
        SoDerived dr1;
        dr1.ShowDerivData();
        cout << "------------------" << endl;
        cout << "case2..... " << endl;
        SoDerived dr2(12);
        dr2.ShowDerivData();
        cout << "------------------" << endl;
        cout << "case3..... " << endl;
        SoDerived dr3(23, 24);
        dr3.ShowDerivData();
        return 0;
};
```

Base, derive : 둘 다 default constructor 사용.

Base, derive : 둘 다 default 아닌 constructor 사용.

Base: default constructor 사용.
derive : default 아닌 constructor 사용.

[실행결과]

```
case1.....
SoBase()
SoDerived()
20
30
------------------
case2.....
SoBase()
SoDerived(int n)
20
12
------------------
case3.....
SoBase(int n)
SoDerived(int n1, int n2)
23
24
```

유도 클래스의 객체 소멸과정 (DerivedDestOrder.cpp 1/2)

- 유도 클래스의 객체 소멸 과정에서 기초 클래스의 소멸자는 100% 호출됨.
  - 호출순서
    - 유도클래스의 destructor 먼저
    - 기초클래스의 destructor 나중
- 스택에 생성된 객체(지역변수로서 객체 차례로 선언)의 소멸 순서
  - 생성 순서와 반대

```
#include <iostream>
using namespace std;
class SoBase {
        int baseNum;
public:
        SoBase(int n) : baseNum(n) {
                cout << "SoBase : " <<  baseNum <<  endl;
        }
        ~SoBase() {
                cout << "~SoBase : " << baseNum << endl;
        }
};
```

유도 클래스의 객체 소멸과정
(DerivedDestOrder.cpp 2/2)

```
class SoDerived : public SoBase {
private:
        int derivNum;
public:
        SoDerived(int n) : SoBase(n), derivNum(n)
        {
                cout << "SoDerived : " << derivNum << endl;
        }
        ~SoDerived()
        {
                cout << "~SoDerived : " << derivNum << endl;
        }
};
int main(void)
{
        SoDerived drv1(15);
        SoDerived drv2(20);
        return 0;
};
```

[실행결과]

```
SoBase : 15
SoDerived : 15
SoBase : 20
SoDerived : 20
~SoDerived : 20
~SoBase : 20
~SoDerived : 15
~SoBase : 15
```

유도클래스의 생성자와 소멸자 정의 모델 (DestModel.cpp 1/2)

```
#define _CRT_SECURE_NO_WARNINGS

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

class Person {
private:
        char* name;
public:
        Person(char* myname) {
                name = new char[strlen(myname) + 1];
                strcpy(name, myname);
        }
        ~Person() {
                delete[]name;
        }
        void WhatYourName() const {
                cout << "My name is " << name << endl;
        }
};
```

자신이 할당한 메모리를 해제.

유도클래스의 생성자와 소멸자 정의 모델
(DestModel.cpp 2/2)

293-294p

```
class UnivStudent : public Person {
private:
        char* major;
public:
        UnivStudent(char* myname, char* mymajor)
                :Person(myname) {
                major = new char[strlen(mymajor) + 1];
                strcpy(major, mymajor);
        }
        ~UnivStudent() {
                delete[]major;
        }
        void WhoAreYou() const {
                WhatYourName();
                cout << "My major is " << major << endl << endl;
        }
};
int main(void) {
        UnivStudent st1((char*)"Kim", (char*)"Mathmatics");
        st1.WhoAreYou();
        UnivStudent st2((char*)"Hong", (char*)"Physics");
        st2.WhoAreYou();
        return 0;
};
```

자신이 할당한 메모리만 해제.
상속받은 변수의 해제는 ~Person에서 이루어지므로

295, 296p 연습문제를 교재에 풀어보자.

297-303p

protected로 선언된 멤버의 접근 허용범위

- 허용 범위
  - private < protected < public
- protected
  - 상속받은 클래스에서만 접근 가능
  - But 상속관계에서도 정보은닉은 지켜지는 것이 좋으므로 가능한 사용 안함.
- 3가지 형태의 상속
  Derived 클래스를 다시 상속 받은 클래스 (DeDerived클래스)에서의 Base 클래스(DeDerived클래스의 2대 상위 클래스) 멤버 접근에 대한 제한 기준.
  - class Derived : public Base
  - class Derived : protected Base
  - class Derived : private Base
- 상속은 대부분 public 사용!

protected 상속과 private 상속

```
class Base
{
private:
    int num1;
protected:
    int num2;
public:
    int num3;
};
```

```
class Derived : private Base
{
    // empty!
};
```

private
상속의 결과

```
class Derived : private Base
{
접근불가:
    int num1;
private:
    int num2;
private:
    int num3;
};
```

때문에 이 이상의 상속은 무의미할 수 있다.

```
class Derived : protected Base
{
    // empty!
};
```

protected
상속의 결과

```
class Derived : protected Base
{
접근불가:
    int num1;
protected:
    int num2;
protected:
    int num3;
};
```