4. 클래스의 완성_2

5주차

192-194p

객체 배열
(ObjArr.cpp 1/3)

- 객체 배열로 사용할 클래스는 default constructor(인자 없는) 필수.
  - Constructor를 지정할 수 없으므로.
- 배열 메모리 해제될 때 각 원소 별로 destructor도 호출 됨.

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring>
using namespace std;
class Person {
        char* name;
        int age;
public:
        Person(char* myname, int myage) {
                int len = strlen(myname) + 1;
                name = new char[len];
                strcpy(name, myname);
                age = myage;
        }
        Person() {
                name = NULL;
                age = 0;
                cout << "called Person()" << endl;
        }
```

Person을 배열로 사용할 것이므로 필수!

생성자 호출 확인

192-194p

객체 배열
(ObjArr.cpp 2/3)

```
        void SetPersonInfo(char* myname, int myage)
        {
                name = myname;
                age = myage;
        }
        void ShowPersonInfo() const
        {
                cout << "이름: " << name << ", ";
                cout << "나이: " << age << endl;
        }
        ~Person()
        {
                delete []name;
                cout << "called destructor!" << endl;
        }
};
```

소멸자 호출 확인

192-194p

객체 배열
(ObjArr.cpp 3/3)

```
int main(void) {
	Person parr[3];
	char namestr[100];
	char* strptr;
	int age ,len;
	for (int i = 0; i < 3; i++) {
		cout << "이름: ";
		cin >> namestr;
		cout << "나이: ";
		cin >> age;
		len = strlen(namestr) + 1;
		strptr = new char[len];
		strcpy(strptr, namestr);
		parr[i].SetPersonInfo(strptr, age);
	}
	parr[0].ShowPersonInfo();
	parr[1].ShowPersonInfo();
	parr[2].ShowPersonInfo();
	return 0;
}
```

194-195p

객체 포인터 배열
(이전 소스에서 main 함수만 수정)(ObjPtrArr.cpp)

```
int main(void) {
	Person* parr[3];
	char namestr[100];
	char* strptr;
	int age, len;
	for (int i = 0; i < 3; i++) {
		cout << "이름: ";
		cin >> namestr;
		cout << "나이: ";
		cin >> age;
		parr[i] = new Person(namestr, age);
	}
	parr[0]->ShowPersonInfo();
	parr[1]->ShowPersonInfo();
	parr[2]->ShowPersonInfo();
	delete parr[0];
	delete parr[1];
	delete parr[2];
	return 0;
}
```

196-199p

This 포인터
(PointerThis.cpp 1/2)

- 클래스에 소속된 함수 안에서만 사용할 수 있는 포인터
- this가 사용된 객체 자신의 주소값을 정보로 담고 있는 포인터
- this의 타입 : 이를 사용한 함수가 포함된 클래스의 포인터
- 지역변수(매개변수 포함)와 같은 이름의 멤버 변수에 this 이용하여 접근 가능. (이런 상황을 만들지 말기)

```
#include <iostream>
#include <cstring>
using namespace std;
class SoSimple {
        int num;
public:
        SoSimple(int n) : num(n) {
                cout << "num=" << num << ", ";
                cout << "address=" << this << endl;
        }
        void ShowSimpleData() {
                cout << num << endl;
        }
        SoSimple* GetThisPointer() {
                return this;
        }
};
```

이 문장을 실행할 때의 객체 포인터를 반환.

196-199p

This 포인터
(PointerThis.cpp 2/2)

```
int main(void)
{
        SoSimple sim1(100);
        SoSimple* ptr1 = sim1.GetThisPointer();
        cout << ptr1 << ", ";
        ptr1->ShowSimpleData();

        SoSimple sim2(200);
        SoSimple* ptr2 = sim2.GetThisPointer();
        cout << ptr2 << ", ";
        ptr2->ShowSimpleData();
        return 0;
}
```

둘 다 this 를 반환한 값을 출력하지만 객체가 다르므로 서로 다른 주소값이 출력됨.

Self-reference의 반환
(SelfRef.cpp 1/2)

199-202p

- **Self-reference**
  - 객체 자신에 대한 참조자
  - this 포인터를 역참조하여 만들 수 있다.
  - 함수의 연속적인 호출에 유용

```
#include <iostream>
using namespace std;
class SelfRef {
        int num;
public:
        SelfRef(int n) : num(n) {
                cout << "객체생성" << endl;
        }
        SelfRef& Adder(int n) {
                num += n;
                return *this;
        }
        SelfRef& ShowTwoNumber() {
                cout << num << endl;
                return *this;
        }
};
```

this 포인터를 역참조하여 객체 자신을 반환.
반환타입을 참조로 하여 자신에 대한 참조를 반환하게 만듬.

Self-reference의 반환
(SelfRef.cpp 2/2)

```
int main(void)
{
	SelfRef obj(3);
	SelfRef& ref = obj.Adder(2);

	obj.ShowTwoNumber();
	ref.ShowTwoNumber();

	ref.Adder(1).ShowTwoNumber().Adder(2).ShowTwoNumber();
	return 0;
}
```

Adder, ShowTwoNumber 함수가 객체참조를 반환하기 때문에 가능한 문장.

201-206p

OOP 프로젝트 02단계

- **아래를 고려하여 구조체를 클래스로 바꾸기**
  - **캡슐화, 정보 은닉 고려**
  - **생성자, 소멸자 형태 고려**
- **객체 배열 대신 객체 포인터 배열 사용하기**

```
[결과]
-----Menu------
1. 계좌개설
2. 입    금
3. 출    금
4. 계좌정보 전체 출력
5. 프로그램 종료
선택: 1

[계좌개설]
계좌ID: 1234
이  름: choi
입금액: 1000
```

```
-----Menu------
1. 계좌개설
2. 입    금
3. 출    금
4. 계좌정보 전체 출력
5. 프로그램 종료
선택: 4

계좌ID: 1234
이  름: choi
잔  액: 1000
```

```
-----Menu------
1. 계좌개설
2. 입    금
3. 출    금
4. 계좌정보 전체 출력
5. 프로그램 종료
선택: 5
```

OOP 프로젝트 02단계
(BankingSystemVer02.cpp 1/8)

201-206p

강의자료실에서
"oop_01_chap01.cpp" 파일
다운로드 받아 수정하기.

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring>

using namespace std;
const int NAME_LEN = 20;
enum { MAKE = 1, DEPOSIT, WITHDRAW, INQUIRE, EXIT };

class Account {
        int accID;       // 계좌번호
        int balance;     // 잔     액
        char* cusName;   // 고객이름
public:
        Account(int ID, int money, char* name)
                : accID(ID), balance(money) {
                cusName = new char[strlen(name) + 1];
                strcpy(cusName, name);
        }
        int GetAccID() { return accID; }
```

struct를 class로 바꾸기.
- Private member data + Get/Set 함수
- Member functions

MakeAccount() 기능 가져오기.

OOP 프로젝트 02단계
(BankingSystemVer02.cpp 2/8)

```
        void Deposit(int money) {
                balance += money;
        }
        int Withdraw(int money) {    // 출금액 반환, 부족 시 0
                if (balance < money)
                        return 0;

                balance -= money;
                return money;
        }
        void ShowAccInfo() {
                cout << "계좌ID: " << accID << endl;
                cout << "이  름: " << cusName << endl;
                cout << "잔  액: " << balance << endl;
        }
        ~Account() {
                delete [] cusName;
        }
};
```

ShowAllAccInfo 기능 가져오기

201-206p

OOP 프로젝트 02단계
(BankingSystemVer02.cpp 3/8)

```
Account* accArr[100];   // Account 저장을 위한 배열
int accNum = 0;         // 저장된 Account 수

// 메뉴출력
void ShowMenu(void)
{
        cout << "-----Menu------" << endl;
        cout << "1. 계좌개설" << endl;
        cout << "2. 입     금" << endl;
        cout << "3. 출     금" << endl;
        cout << "4. 계좌정보 전체 출력" << endl;
        cout << "5. 프로그램 종료" << endl;
}
```

구조체의 배열에서
클래스 객체의 포인터 배열로.

201-206p

OOP 프로젝트 02단계
(BankingSystemVer02.cpp 4/8)

```
// 계좌개설을 위한 함수
void MakeAccount(void)
{
        int id;
        char name[NAME_LEN];
        int balance;

        cout << "[계좌개설]" << endl;
        cout << "계좌ID: ";          cin >> id;
        cout << "이  름: ";          cin >> name;
        cout << "입금액: ";          cin >> balance;
        cout << endl;

        accArr[accNum++] = new Account(id, balance, name);
}
```

배열 원소에 값 대입하는 대신 객체 할당 받아서 추가.

201-206p

OOP 프로젝트 02단계
(BankingSystemVer02.cpp 5/8)

```
// 입     금
void DepositMoney(void) {
        int money;
        int id;
        cout << "[입     금]" << endl;
        cout << "계좌ID: ";          cin >> id;
        cout << "입금액: ";          cin >> money;

        for (int i = 0; i < accNum; i++)
        {
                if (accArr[i]->GetAccID() == id)
                {
                        accArr[i]->Deposit(money);
                        cout << "입금완료" << endl << endl;
                        return;
                }
        }
        cout << "유효하지 않은 ID 입니다." << endl << endl;
}
```

Private 멤버에 대해 Get/Set 함수 사용

멤버 함수 사용.

OOP 프로젝트 02단계
(BankingSystemVer02.cpp 6/8)

201-206p

```
// 출     금
void WithdrawMoney(void) {
        int money;
        int id;
        cout << "[출     금]" << endl;
        cout << "계좌ID: ";          cin >> id;
        cout << "출금액: ";          cin >> money;

        for (int i = 0; i < accNum; i++) {
                if (accArr[i]->GetAccID() == id) {
                        if (accArr[i]->Withdraw(money) == 0) {
                                cout << "잔액부족" << endl << endl;
                                return;
                        }
                        cout << "출금완료" << endl << endl;
                        return;
                }
        }
        cout << "유효하지 않은 ID 입니다." << endl << endl;
}
```

Private 멤버에 대해 Get/Set 함수 사용

멤버 함수 사용.

201-206p

OOP 프로젝트 02단계
(BankingSystemVer02.cpp 7/8)

```
// 잔액조회
void ShowAllAccInfo(void)
{
	for (int i = 0; i < accNum; i++)
	{
		accArr[i]->ShowAccInfo();
		cout << endl;
	}
}
```

멤버 함수 사용.

OOP 프로젝트 02단계
(BankingSystemVer02.cpp 8/8)

```
int main(void) {
	int choice;
	while (1) {
		ShowMenu();
		cout << "선택: ";
		cin >> choice;
		cout << endl;

		switch (choice) {
		case MAKE:		MakeAccount(); break;
		case DEPOSIT:	DepositMoney(); break;
		case WITHDRAW:	WithdrawMoney(); break;
		case INQUIRE:	ShowAllAccInfo(); break;
		case EXIT:		return 0;
		default:		cout << "Illegal selection.." << endl;
		}
	}
	for (int i = 0; i < accNum; i++)
		delete accArr[i];
	return 0;
}
```

할당 받은 객체들을 메모리 해제.

5. 복사생성자

214-215p

C스타일의 초기화와 C++스타일의 초기화

C 스타일의 초기화

```
int num=20;
int &ref=num;
```

C++ 스타일의 초기화

```
int num(20);
int &ref(num);
```

객체에 대해서도 다음 두 문장은 동일한 문장으로 해석됨.

1번째 문장은 C++에서 자동으로 2번째 문장으로 변환됨.

2번째 문장일 때 “복사생성자”가 호출됨.

정의된 “복사생성자”가 없을 때는 자동으로 생성된 default “복사생성자”가 사용됨.

```
SoSimple sim2=sim1;
SoSimple sim2(sim1);
```

복사생성자 사용
(ClassInit.cpp 1/2)

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

class SoSimple {
private:
        int num1;
        int num2;
public:
        SoSimple(int n1, int n2)
                : num1(n1), num2(n2) {
                // empty
        }

        SoSimple(const SoSimple &copy)
                : num1(copy.num1), num2(copy.num2) {
                cout << "Called SoSimple(SoSimple &copy)" << endl;
        }

        void ShowSimpleData() {
                cout << num1 << endl;
                cout << num2 << endl;
        }
};
```

(1) 원본 내용을 실수로 바꾸지 않도록 const 사용
(2) 임시객체가 생성되면서 무한루프에 빠지지 않도록 참조자로 선언

복사생성자 호출 확인용 문장.

복사생성자 사용
(ClassInit.cpp 2/2)

```
int main(void)
{
	SoSimple sim1(15, 30);
	cout << "생성 및 초기화 직전" << endl;
	SoSimple sim2 = sim1;
	cout << "생성 및 초기화 직후" << endl;
	sim2.ShowSimpleData();
	return 0;
}
```

Default 복사생성자

- 자동으로 삽입되는 복사생성자
- 모든 멤버 변수를 복사해 줌
- 복사생성자를 명시적으로 정의하면 사용되지 않음.

```
class SoSimple
{
private:
     int num1;
     int num2;
public:
     SoSimple(int n1, int n2) : num1(n1), num2(n2)
     { }
     SoSimple(const SoSimple &copy) : num1(copy.num1), num2(copy.num2)
     { }
};
```

복사 생성자를 정의하지 않으면, 멤버 대 멤버의 복사를 진행하는 디폴트 복사 생성자가 삽입된다.

자동 형변환에 의한 복사생성자 실행 금지

◆ 예약어 explicit

- C스타일의 초기화 (SoSimple s3 = s2;)가 자동으로 C++ 스타일(SoSimple s3(s2);)로 변환되어 default 복사생성자 적용되는 것을 막음
  코드의 결과를 보다 예측 가능하게 만들어 줌.

  ❖ 앞선 예제의 복사생성자 앞에 explicit 붙여보자
  explicit SoSimple(const SoSimple &copy)

  ❖ SoSimple sim2 = sim1; 이 자동으로 C++스타일 초기화로 전환되지 않으므로 오류 발생됨

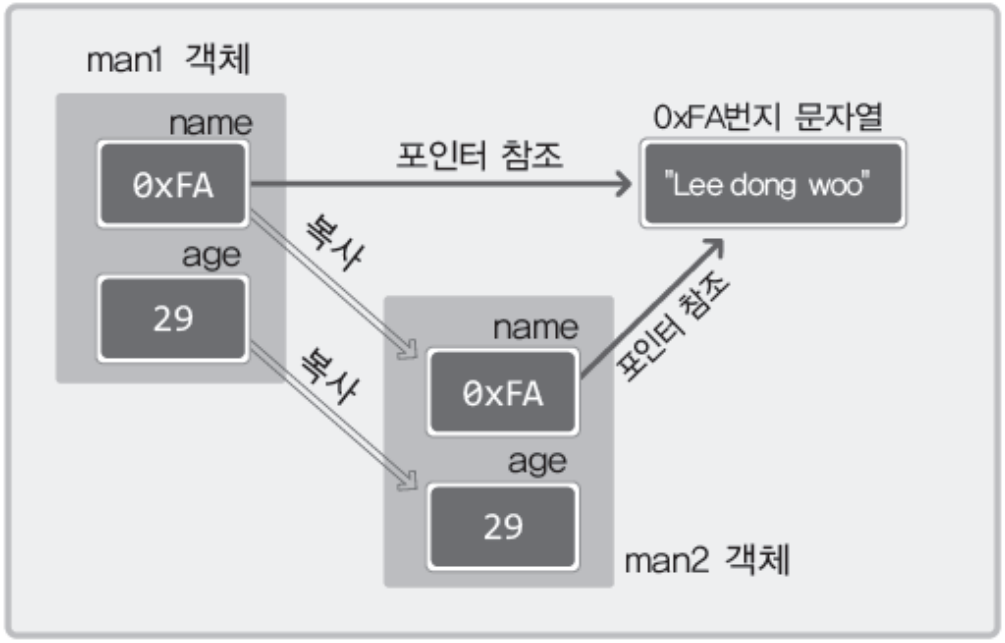
  ❖ SoSimple sim2(sim1); 으로 바꾸어 해결.

- 객체 생성시에 타입이 맞는 생성자가 있으면 오류를 내보내지 않고 자동으로 그 생성자를 이용하여 객체 생성해 주는 기능 금지

디폴트 복사 생성자의 문제점

- 멤버 중 할당된 메모리 주소를 가진 것이 있을 때 객체 복사하면…
- 디폴트 복사생성자에 의해 메모리 번지수만 복사됨.
- 원본과 사본 객체 중 하나가 소멸될 때 메모리를 delete 시키면 다른 객체가 갖고 있는 메모리 주소는 사용할 수 없게 됨.

- => deep copy 필요!



221-225p

디폴트 복사 생성자의 문제점
(ShallowCopyError.cpp 1/2)

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring>
using namespace std;
class Person {
        char* name;
        int age;
public:
        Person(char* myname, int myage) {
                int len = strlen(myname) + 1;
                name = new char[len];
                strcpy(name, myname);
                age = myage;
        }
        void ShowPersonInfo() const {
                cout << "이름: " << name << endl;
                cout << "나이: " << age << endl;
        }
        ~Person() {
                delete [] name;
                cout << "called destructor!" << endl;
        }
};
```

Default 복사생성자에 의해 name의 주소를 다른 객체가 갖고 있다면
(1) 여기서 delete한 후에는 다른 객체를 name을 사용할 수 없게 됨.
(2) 2번째 객체가 같은 위치에 대해 delete를 호출하면 오류 발생!

디폴트 복사 생성자의 문제점
(ShallowCopyError.cpp 2/2)

```
int main(void)
{
        Person man1((char*)"Lee dong woo", 29);
        Person man2 = man1;
        man1.ShowPersonInfo();
        man2.ShowPersonInfo();
        return 0;
}
```

자동으로 C++스타일 초기화로 바꿔어 타입에 맞는 생성자(여기서는 default 복사생성자)가 호출됨.

225-226p

Deep copy를 위한 생성자의 정의
(ShallowCopyError.cpp Person 클래스에 함수 추가)

```
Person(const Person& src) {
        name = new char[strlen(src.name)+1];
        strcpy(name, src.name);
        age = src.age;
}
```

Deep copy

240-241p

OOP 프로젝트 03단계
(BankingSystemVer03.cpp)

[BankingSystemVer02.cpp 에 Account 클래스의 복사 생성자만 추가하기.]

```
Account(const Account & ref)
        : accID(ref.accID), balance(ref.balance)
{
        cusName=new char[strlen(ref.cusName)+1];
        strcpy(cusName, ref.cusName);
}
```