2020.09.28 - 5주차

객체지향설계

생성자와 소멸자

Content

1. 지난주 과제 풀이
2. 생성자
3. 소멸자
4. 과제
5. QnA

과제 01

- "실습 1"의 코드를 C 코드로 변경해 보자
- 100자 이내로 입력된다고 가정하고 별도의 예외 처리는 하지 않아도 됨

homework01.cpp

```
int main() {
    std::string s;
    std::cout << "문자를 입력하세요(100자 이내)." << std::endl;
    std::cin >> s;
    std::cout << "입력된 문자는 " << s << "입니다." << std::endl;

    system("pause"); // keep terminal open
    return 0;
}
```

과제 01 풀이_1

homework01.c

```c
#include <stdio.h>

int main() {

    char s[100];
    printf("문자를 입력하세요(100자 이내).₩n");
    scanf("%s", s);
    printf("입력된 문자는 %s 입니다.₩n", s);

    system("pause"); // keep terminal open
    return 0;
}
```

과제 01 풀이_2

homework01.c

```c
#include <include.h>

int main() {

    char *s;
    s = (char*)malloc(100)l

    printf("문자를 입력하세요(100자 이내).₩n");
    scanf("%s", s);
    printf("입력된 문자는 %s 입니다.₩n", s);

    free(s);
    system("pause");
    return 0;
}
```

과제 02 풀이

- 아래 코드의 swap 함수는 인풋 파라미터를 서로 바꾸는 역할을 한다.
- 정상적으로 동작하도록 프로그램을 수정하세요. (힌트: 레퍼런스 활용)

homework02.cpp

```
#include<iostream>
void swap(int first, int second){
    int temp = first;
    first = second;
    second = temp;
}
int main(){
    int a = 2, b = 3;
    swap(a, b);
    std::cout << a << " " << b << std::endl;
    return 0;
}
```

생성자

- 생성자의 개념
  - ✓ 객체를 생성함과 동시에 멤버변수 초기화를 가능하게 함
  - ✓ 생성자는 클래스 이름과 동일한 이름으로 구현됨
- 생성자의 특징
  - ✓ 반환값이 없음
  - ✓ 매개변수 다양화를 통해 여러 번 정의될 수 있음
- 기본생성자
  - ✓ 별도의 생성자를 구현하지 않으면 자동생성 됨
  - ✓ 매개변수를 자지지 않으면 매개변수들은 0, NULL 등으로 초기화됨

생성자

- 기본생성자
  - ✓ 디폴트 생성자의 암시적 호출 : Animal animal;
  - ✓ 디폴트 생성자의 명시적 호출 : Animal animal = Animal();

```
class Animal {
public:

private:
    int numberOfLeg;
};

int main() {
    Animal a;
    return 0;
}
```

생성자

- 하지만 아래와 같이 numberOfLeg를 인자로 받는 생성자를 추가할 경우 컴파일 오류가 발생하였다.
- 이유를 생각해보고 오류를 고쳐 컴파일 해보자.

```
class Animal {
public:
    Animal(int numberOfLeg) {
        this->numberOfLeg = numberOfLeg;
    }
private:
    int numberOfLeg;
};

int main() {
    Animal a;
    return 0;
}
```

생성자

- 얇은복사와 깊은복사
  - ✓ 변수에 다른 변수의 값을 대입하기 위해서는 대입 연산자(=)를 사용하면 됨
  - ✓ 이와 같은 방법으로 객체에 또 다른 객체의 값을 대입하는 것이 가능

```
Animal dog("jack", 13);
Animal cat = dog;
```

  - ✓ 하지만 위의 경우 값이 아닌, 값을 가리키는 포인터를 복사하는 얇은 복사(shallow copy)가 수행됨
  - ✓ 객체의 멤버가 메모리공간의 힙 영역을 참조할 경우 문제 발생

생성자

➢ 복사생성자

- ✓ C++에서 복사 생성자란 자신과 같은 클래스 타입의 다른 객체에 대한 참조(reference)를 인수로 전달받아, 그 참조를 가지고 자신을 초기화하는 방법

- ✓ 복사 생성자를 이용한 대입은 깊은 복사(deep copy)를 통한 값의 복사이기 때문에 새롭게 생성되는 객체가 원본 객체와 같지만 완전한 독립성을 가짐

- ✓ 다음과 같은 상황에 주로 사용됨
  - 객체가 함수에 인수로 전달될 때
  - 함수가 객체를 반환값으로 반환할 때
  - 새로운 객체를 같은 클래스 타입의 기존 객체와 똑같이 초기화할 때

- ✓ Animal(Animal & a);

생성자

➢ 복사생성자

```
#include <iostream>
#include <memory>
class Animal {
	public:
	char* name;
	int age;

	Animal(int age_, const char* name_) {
		age = age_;
		name = new char[strlen(name_) + 1];
		strcpy(name, name_);
	}
	Animal(Animal& a) { //복사 생성자
		age = a.age;
		name = a.name;
	}
	void changeName(const char* newName) {
		strcpy(name, newName);
	}
	void printAnimal() {
		std::cout << "Name: " << name << " Age: "
		<< age << std::endl;
	}
};
```

```
void main() {
	Animal A(10, "Jenny"); //10살 Jenny 생성
	Animal B = A; // 10살 Jenny인 A를 B에게 복사
	A.age = 22;//A의 나이를 22살로 바꿈
	A.changeName("Brown"); //A의 이름을 Brown으로 바꿈

	A.printAnimal();
	B.printAnimal();
	getchar();
}
```

소멸자

- 소멸자 필요 이유
  - ✓ C++에서 생성자는 객체 멤버와 함께 객체를 사용하기 위한 외부 환경도 초기화하며 객체의 수명이 끝나면 생성자의 반대 역할을 수행할 멤버 함수도 필요
  - ✓ 소멸자는 객체의 수명이 끝나면 컴파일러에 의해 자동으로 호출되며, 사용이 끝난 객체를 정리

- 소멸자 사용법
  - ✓ C++에서 클래스 소멸자의 이름은 해당 클래스의 이름과 같으며, 이름 앞에 물결 표시(tilde, ~)를 붙여 생성자와 구분

소멸자

- 소멸자 특징
  - ✓ 소멸자는 인수와 반환값이 없으며 단 하나만 가질수 있음
  - ✓ 소멸자는 const, volatile 또는 static으로 선언될 수는 없지만, const, volatile 또는 static으로 선언된 객체의 소멸을 위해서 호출 가능

- 소멸자 호출
  - ✓ 호출 시기는 컴파일러가 처리
    - 데이터 영역 : 해당 프로그램이 종료될 때
    - 스택 영역 : 해당 객체가 정의된 블록을 벗어날 때
    - 힙 영역 : delete를 사용하여 해당 객체를 반환할 때
    - 임시 객체 : 임시 객체의 사용을 마쳤을 때

소멸자

➢ 소멸자 사용예시

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

int i;

class A{
public:
    ~A(){
        i = 10;
    }
};
int foo(){
    i = 3;
    A ob;
    return i;
}
int main(){
    cout << foo() << endl;
    return 0;
}
```

결과값 : 3

소멸자

➢ 소멸자 사용예시 – 래퍼런스 사용

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

int i;

class A{
public:
    ~A(){
        i = 10;
    }
};
int &foo(){
    i = 3;
    A ob;
    return i;
}
int main(){
    cout << foo() << endl;
    return 0;
}
```

결과값 : 10

생성자

➢ 이동생성자

- ✓ https://docs.google.com/document/d/1R_GIK8oYn30M82AKi-3DgJorRmzPt5H2S-ZMR23gmOM/edit

- ✓ 백터
  - 벡터란 크기를 변경할 수 있는 연속적인 컨테이너
  - 컨테이너랑 같은 타입의 데이터를 모은 객체
  - 벡터는 인접하는 메모리 위치에 원소를 저장하고, 연산자 []를 통해 아무 원소에나 직접적인 접근이 가능

- ✓ 백터 선언 방법
  - vector〈자료형〉 컨테이너이름;

- ✓ 벡터 끝에 원소를 추가하는 법
  - 컨테이너이름.push_back(추가할원소)

생성자

- 이동생성자
  - ✓ 복사 생정자만 존재할 경우 push_back시 vector의 제일 뒤에 값이 추가됨
  - ✓ vector의 크기가 부족할 경우 크기가 큰 vector를 생성해 값을 이동시키고 기존의 vector는 메모리에서 제거
  - ✓ vector에 새로운 오브젝트가 추가될 때 오브젝트의 복사 생성자가 호출 (미 정의시 컴파일러에서 자동생성)

  → example01.ccp 파일 참조

생성자

- 이동생성자
  - ✓ example01.ccp 처럼 구현할 때의 문제점은, 새로운 vector 생성에 따라 기존 vector에 저장된 오브젝트들이 새로운 vector로 복사 될 때마다 copy constructor가 호출되고 이는 성능 저하를 일으킨다는 점
  - ✓ 이러한 문제를 해결하기 위해, 클래스에 shallow copy를 수행하는 이동 생성자 (Move constructor)를 정의해 준다. 그러면 새로운 vector로 이동 시, move constructor가 호출되며, 불필요한 메모리 할당을 줄일 수 있음
  - ✓ 이때 주의할 점은
    1. move constructor에 noexcept 키워드가 지정되어 있어야 한다. 이는 move constructor 수행 도중에 예외가 없다는 것을 컴파일러에 알려주는 것
    2. Shallow copy가 일어나는 변수에 nullptr을 넣어줌.
    3. Destructor에서 메모리 할당에 관련된 변수가 nullptr인지 체크.

→example02.ccp 파일 참조

과제 01

- 12페이지의 결과 값 출력 시, A와 B의 이름이 동시에 바뀐 것을 볼 수 있다.
- A의 이름만 Brown으로 바뀌도록 복사 생성자 (Animal(Animal &a){})를 수정하라.

homework_05_01.cpp

```
class Animal {
.....
        Animal(Animal& a) { //복사 생성자
                age = a.age;
                name = a.name;
        }
.....
}
```

과제 02

- 15페이지 코드의 실행 결과값이 10이 되도록 foo 함수를 수정해보세요.
- 단, foo 함수 내부의 nested block (중첩 블럭) {} 을 이용할 것.

homework_05_02.cpp

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

int i;

class A{
public:
    ~A(){
        i = 10;
    }
};
int foo(){
    i = 3;
    A ob;
    return i;
}
int main(){
    cout << foo() << endl;
    return 0;
}
```

과제 03

- https://docs.google.com/document/d/1R_GIK8oYn30M82AKi-3DgJorRmzPt5H2S-ZMR23gmOM/edit
- example01.cpp, example02.cpp를 실행
- 이동생성자에 대해 이해한 것을 보고서에 정리

과제 제출

- 제출
  - ✓ 10월 04일 24시 까지 사이버캠퍼스에 보고서 및 깃헙 링크 업로드
  - ✓ 지각 제출시 만점의 50%
  - ✓ Copy 적발시 0점

- 보고서 작성방법
  - ✓ 표지 : 제출일, 학번, 이름, 주차(week05)
  - ✓ 과제 수행 과정, 실행결과 캡쳐 후 결과분석
  - ✓ 과제를 수행하며 어려웠던 점 또는 새로 알게 된 부분을 고찰한 내용
  - ✓ 과제의 해답 코드를 깃허브에 올리고 링크를 보고서에 기입
  - ✓ 파일형식 : PDF

QnA

사이버캠퍼스 질문게시판 또는
ljy8972@naver.comfh 이나
ymc12377@naver.com 로 메일