INHA UNIVERSITY

ICE2004 자료구조론

9

제 목

자료구조론 과제 1 보고서

보고서 작성 서약서

1. 나는 타학생의 보고서를 베끼거나 여러 보고서의 내용을 짜집기하지 않겠습니다.
2. 나는 보고서의 주요 내용을 인터넷사이트 등을 통해 얻지 않겠습니다.
3. 나는 보고서의 내용을 조작하지 않겠습니다.
4. 나는 보고서 작성에 참고한 문헌의 출처를 밝히겠습니다.
5. 나는 나의 보고서를 제출 전에 타학생에게 보여주지 않겠습니다.

나는 보고서 작성시 윤리에 어긋난 행동을 하지 않고 정보통신공학인으로서 나의 명예를 지킬 것을 맹세합니다.

2021년 10월 10일

학부

학년

성명 성시열

학번

[개요]

- class의 선언되어 있는 헤더파일과 멤버 함수를 정의하는 소스파일, 그리고 main함수가 포함된 실행파일로 구성되어있다.

  헤더파일 : Media.h, VTR.h, DVD.h, LegalDVD.h

  소스파일 : Media.cpp, VTR.cpp, DVD.cpp, LegalDVD.cpp

  실행파일 : testMedia.cpp

- 상속관계와 class를 구성하는 멤버는 문제에서 제시된 바와 같다.

[구현상 특징]

- static변수를 활용하여 모든 객체에서 활용할 수 있는 정수를 선언하여 어떤 클래스의 객체이든 객체가 생성될 때마다 1씩 증가하는 count 변수를 구현할 수 있었다.

또한, 상속구조이기 때문에 자식 객체는 부모 객체의 public, proteceted 멤버를 사용할 수 있다. 같은 이름의 함수가 있다면 부모객체에 virtual을 붙여준다. 어떠한 클래스에서 그 함수를 실행시킬 때 부모클래스의 함수가 아닌 자식클래스의 함수가 실행된다. 즉, 같은 이름이라도 다양한 기능의 함수를 구현할 수 있고, 이는 객체지향프로그래밍의 큰 특징 중 하나인 다형성이다. 그 외에도 상속에서의 특징을 보면 자식클래스를 활용하여 객체를 생성할 때, 부모클래스의 생성자도 함께 생성되고, 순서도 부모클래스를 활용한 객체의 생성자가 먼저 실행된다.

연산자 오버로딩 중 <<연산자를 사용하여 포인터 객체가 가지고 있는 함수의 출력값을 출력하였다. 또한 +연산자를 활용하여 Date type이 서로 다른 값을 연산하는 operator+를 선언하였고, 포인터 객체 값과 포인터 객체의 주소를 활용해 각 객체의 멤버변수끼리 더하는 연산을 구현하였고, 이를 새로운 포인터객체에 값을 할당하였다. 마지막으로 =(대입) 연산자를 활용하여 새로운 포인터객체에 할당한 값을 대입하고, 할당시킨 객체는 소멸시켜 비어있는 공간을 delete하지 않도록 구현하였다. new를 통해 heap 공간에 생성한 메모리는 stack과 달리 자동으로 청소를 해주지 않기 때문에 그 값을 소멸시키지 않으면 memory leak이 발생하게 된다. 이를 방지하기 위해 new를 통해 생성했던 포인터 객체들을 모두 delete해주었다.

[코드 분석]

1. 헤더파일에서의 공통점.

1)

#ifndef ~~~

#define ~~~

…

#endif

: 헤더파일이 중복되어 선언하는 것을 방지하기 위해 모든 헤더파일에 작성.

2)

#include<iostream>

#include<string>

using namespace std;

: iostream과 string 라이브러리에 포함된 이름공간의 모든 함수를 사용할 수 있다.

3) 접근지정자

private : 클래스 외부에서 접근할 수 불가. 주로 변수

public : 클래스의 내부, 외부에서 모두 접근 가능. 주로 함수

4) 생성자, 소멸자

생성자, 소멸자를 작성하지 않더라도 기본생성자가 생성되지만, 작성을 해줌으로써 생성자와 소멸자가 실행될 때 추가적인 실행이 가능하다. 생성자는 class의 이름을 그대로 사용하고 반환형이 없다. 소멸자는 생성자처럼 class의 이름을 그대로 사용하고 앞에 ~을 붙여준다. 반환형이 없으며, parameter도 없다.

5) set함수, get함수

Class의 private 멤버의 값을 writing 하거나 reading 하기 위해 사용되는 함수.

set함수는 반환값이 없고, parameter로 상황에 따라 int형 변수 or string 변수를 가진다.

private의 값을 지정해주고, 조건을 통해 값을 필터링해주는 역할을 한다.

get함수는 parameter가 없으며 private의 값을 반환하는 역할을 한다.

6) 상속관계 - class (자식-class) : (접근지정자) (부모-class)

```
class DVD: public Media
class VTR: public Media
class LegalDVD: public DVD
```

상속관계를 표현할 때 다음과 같이 코딩한다. 접근 지정자를 어떻게 설정하냐에 상속받아 사용할 수 있는 범위가 달라진다. Public으로 설정하면 객체지향프로그래밍의 특징인 다형성을 활용할 수 있다.

2. 소스코드 파일에서의 공통점

1) (클래스 이름) :: (함수이름)

함수를 정의할 때 어느 클래스의 함수인지 명시해준다.

2) 각종 라이브러리(iostream, string ...)

3. 파일별 특징적인 부분

1) Media.h

Line	코드, 목적 및 상세설명
13-14	`protected:` `    string title;`
	각 객체의 제목을 나타낼 때 사용하는 문자열변수 title. 이 변수를 protected에 포함시켜 자식 관계(상속)에서도 title을 사용할 수 있음. 생성자와 소멸자가 실행될 때 출력문에 어떤 객체의 생성자인지, 어떤 객체의 소멸자인지 구분을 명확히 하기 위해 제목까지 함께 출력.
17	`static int count;`
	여러 객체가 하나의 변수를 공유하고자 할 때 static변수를 사용. public변수이기 때문에 객체 생성 이전에도 접근이 가능하다. 과제의 조건에 따라 main함수 이전에 static int count의 값을 0으로 초기화 해야하기 때문에 public에 포함시켰다.
20	`virtual ~Media();`
	virtual은 (상속관계에서) 자식 class에 같은 함수가 있을 때와 소멸자가 있을 때, 부모 class 선언 시 헤더파일의 함수 prototype 앞에 붙여준다. virtual이 붙은 함수를 보면 자식 클래스의 같은 이름의 함수가 있는 것을 예상할 수 있고, 부모와 자식에 같은 이름의 함수가 있을 때 자식의 함수로 실행된다. 소멸자에 virtual을 붙이는 이유는 상속관계에서 하나의 객체가 부모 class 자식 class에 의해 두 번 소멸될 수 있고, 이는 오류를 발생시키기 때문이다.
25	`virtual void play();`
	play()함수가 모든 자식class에 있기 때문에 20번 라인처럼 virtual을 사용.
	`friend ostream& operator<<(ostream&, Media&);`

26	friend를 사용하여 외부함수에 접근할 수 있다. 드라이버 파일에 선언된 << 연산자를 media class로 접근할 수 있다. 복사를 방지하기 위해 ostream 뒤에 &를 붙여주었다. << 연산자를 우리가 ostream 클래스 내부에 정의할 수 없기 때문에 전역함수로 작성하고 friend를 통해 접근한다.
28	`Media* operator+(Media*);`
	+ 연산자 오버로딩으로 parameter와 return의 type으로 포인터객체 Media*를 사용. main함수에서는 ( ) + ( )의 형태로도 사용이 가능하다.
29	`Media* operator=(Media*);`
	= (대입) 연산자 오버로딩으로 parameter와 return의 type으로 포인터객체 Media*를 사용. main함수에서는 ( ) = ( )의 형태로도 사용이 가능하다.

2) DVD.h, VTR.h, LegalDVD.h

DVD.h, VTR.h, LegalDVD.h에서 string이라는 기능을 사용한다. 하지만, #include<string>을 하지 않은 이유는 string을 포함한 Media.h를 include 했기 때문이다. LegalDVD는 DVD.h를 포함하는데 그 DVD.h가 Media.h를 include했기 때문에 꼭 #include<string>하지 않아도 된다.

3) Media.cpp

Line	코드, 목적 및 상세설명
9	`count++;`
	헤더 파일에서 static int형으로 선언한 count, Media 클래스를 사용한 객체가 하나씩 생성될 때마다 count++을 해준다. 자식-class의 객체가 생성이 되어도 부모-class의 생성자도 생성이 되므로 어느 객체든지 하나가 생성(new)될 때마다 하나씩 생성된다..
10, 14	`cout << "Media object constructor for [ " << title << " ]₩n";` `cout << "Media object destructor for [ " << title << " ]₩n";`
	접근지정자 protected에 title을 포함시킨 것에서 알 수 있듯이 생성자와 소멸자가 실행될 때 출력문에 어떤 객체의 생성자인지, 어떤 객체의 소멸자인지 구분을 명확히 하기 위해 title까지 함께 출력.

33-40	`Media* Media::operator+(Media* val) {`
	`  int newLength;`
	`  newLength = length + val->length;`
	
	`  Media* newmp = new Media("new", newLength);`
	
	`  return newmp;`
	`}`

	+ 연산자 오버로딩 코드. 문제 식 : mp[3] = *mp[1] + mp[2] 문제에서 mp[2]는 객체의 주소값이기 때문에 주소를 저장할 수 있는 포인터객체 Media*형태의 val parameter로 사용.

<table>
<tr><td></td><td>return값 또한 mp[3]가 객체의 주소값이기 때문에 포인터객체 Media*를 사용했다.
+ 기호의 왼쪽이 this가 되고 오른쪽이 파라미터가 된다.
newlength에 mp[1]과 mp[2]에 생성된 객체의 length값을 더한다. length 앞에는 (this->)가 생략되어있다. 포인터 객체를 return해주어야 하기 때문에
Media* newmp = new Media("new", newlength);를 통해 Media 클래스를 활용한 포인터객체 newmp를 생성하여 heap 공간에 새로운 객체 하나를 생성한다.
return type이 Media*이기 때문에 return type에 맞춰 newmp를 반환한다.</td></tr>
<tr><td rowspan="2">42-
50</td><td><pre>
Media* Media::operator=(Media* m) {
        if (this == m) {
                return this;
        }

        this->length = m->length;
        delete m;
        return this;
}
</pre></td></tr>
<tr><td>=(대입) 연산자 오버로딩 코드.
문제 식에서 우변은 operator +를 통해 포인터 객체인 Media* newmp가 return 되었다. 좌변은 포인터객체인 mp[3] 이다.
return 값은 포인터 객체인 Media 형태이고, parameter은 반환받은 Media* newmp에서 m이 된다. this는 좌변의 포인터객체인 mp[3]이다.
만약 문제 식에서 좌변과 우변이 같을 경우에는 좌변의 포인터 객체를 바로 넘겨준다. 조건에 해당이 안된다면 newmp의 length값을 mp[3]의 length에 대입한다. 그 이후에 newmp값은 mp[3]에게 값을 전달한다는 임무를 다 했고, new로 동적 할당된 공간은 delete를 통해 반드시 없애준다. 이는 heap 공간의 특성 때문이다. 이후 this를 반환하여 newmp값이 대입된 mp[3]를 반환한다.</td></tr>
</table>

※ 모든 소스코드 파일에 play() 함수가 있다. 이 때 play함수는 클래스마다 출력되는 문장이 다르다. 상속의 특성으로 인해 부모보단 자식 클래스에 정의된대로 실행되므로 같은 이름의 함수임에도 각자 다른 함수가 실행된다. 이를 통해 객체지향프로그래밍의 다형성을 확인할 수 있다.

4) DVD.cpp, VTR.cpp, Legal.cpp

```
DVD::DVD(string tit, int len, string lic) : Media(tit, len){
VTR::VTR(string tit, int len, string formatVideo) : Media(tit, len) {
LegalDVD::LegalDVD(string tit, int len, string lic, string cop):DVD(tit,len,lic) {
```

위 3줄은 각 클래스의 생성자를 정의하는 첫 줄이다. 여기서 초기화기(Member Initializer)을 통해 부모 클래스에서 생성되었던 변수 중 그대로 가져와 자식 클래스의 대입해줄 값을 찾아 작성한다.

5) testMedia.cpp

Line	코드, 목적 및 상세설명
10	`int Media::count = 0;`
	Media 클래스에서 생성된 count라는 int형 변수는 static 변수이기 때문에 전역에서 초기화해줘야 한다. 문제에서 제시된 조건에 맞게 0으로 값을 초기화한다.
12	`ostream& operator<<(ostream& os, Media* m) {`
	<< 연산자 오버로딩. ostream 클래스 내부에 정의할 수 없기 때문에 전역함수로 지정한다. 그리고 Media 클래스 내부에서 friend를 통해 외부함수인 operator <<를 Media 클래스 내부에서도 사용할 수 있게 한다. 두번째 parameter에 const를 사용하면 함수 호출을 할 수 없다.

13-15	`m->play();`
	`return os;`
	`}`

	mp[0]의 play함수를 실행하기 위해 << 연산자 오버로딩을 사용한다. mp[0]의 멤버함수인 play를 실행하고 ostream 내부의 os를 return한다.
20	`Media* mp[4];`
	Media 클래스를 사용한 포인터객체를 생성하는데, 크기가 4인 배열로 생성함. 각 배열의 값을 동적으로 할당해준다. new를 통해 heap 공간에 클래스를 활용하여 만들고, 포인터 객체를 통해 값을 연결한다. new를 통해 heap 공간에 데이터를 만들 시, 반드시 delete를 활용해 값을 소멸시켜야 한다.

29-31	`for (int i = 0; i < Media::count; i++) {`
	`mp[i]->play();`
	`}`

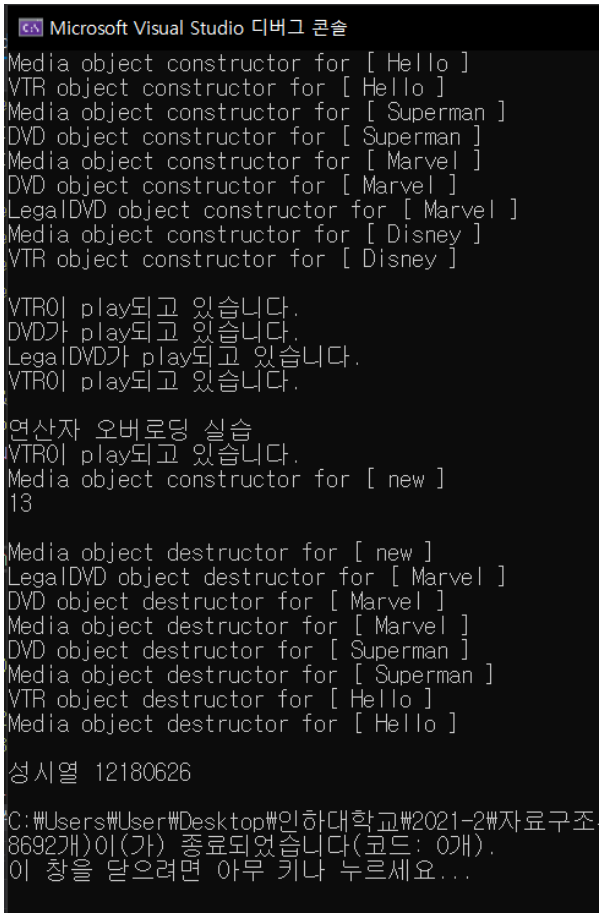
	선언된 모든 클래스에 play함수가 존재한다. 각자의 클래스에 play함수가 존재하기 때문에 같은 play함수더라도 어떤 클래스를 통해 생성된 객체에서 사용되냐에 따라 다르게 실행된다. 또한, 이 클래스들은 상속 관계인데, 부모 클래스에 play함수가 있더라도 자식 클래스의 play함수를 실행한다. count는 0으로 초기화한 뒤 new할 때마다 1씩 증가하도록 코드를 구성했으므로 이 for문에서 count는 4이다. for문의 조건부분에 i < count를 사용하면, 각자 포인터 객체(mp[0~3])의 play함수가 생성된 클래스에 따라 각기 다르게 실행된다. 이는 다형성을 보여줄 수 있는 코드이다.
35	`cout << mp[0]`
	포인터 객체의 주소인 mp[0] 를 cout << 하지만 결과는 mp[0]이 출력된다. 이를 위해 << 연산자 오버로딩을 사용해야 한다. operator<< 을 위와 같이 선언하고 정의하면 몇번째 배열의 mp가 cout << 뒤에 위치해도 클래스에 맞는 play함수가 출력된다

37-38	`mp[3] = *mp[1] + mp[2];` `cout << mp[3]->getLength() << endl;`
	mp[1]이라는 주소의 값 + mp[2]의 주소(이것은 operator의 parameter을 통해 value의 값으로 바뀜) 이 두개의 연산으로 인해 생기는 값 mp[3]가 포인터 객체. 대입 연산자와 +연산자 오버로딩을 통해 mp[3]는 이전의 mp[3]와 새로 생성된 포인터 객체가 mp[3]에 대입된다. 이 객체는 변수 2개(title, length)를 private의 데이터 멤버로 하는 Media 객체이다. mp[3]의 private에 있는 length값을 불러오기 위해서는 Media 클래스의 getLength 함수를 활용하여 새로운 mp[3]의 length를 출력한다

42-44	`for (int i = 3; i >= 0; i--)`
	`delete mp[i];`
	`}`

	위에서 포인터 객체 4개의 값을 지정학 위해 new를 활용하여 heap 공간에 데이터를 동적으로 할당하였다. new를 사용하여 생성한 데이터는 delete를 사용하여 소멸시켜야한다. LIFO에 따라 소멸은 생성의 역순으로 하여, mp[3], mp[2], mp[1],mp[0]을 순서대로 소멸시킨다. 소멸이 될때마자 각 객체의 소멸자가 생성되고, 그 객체의 부모객체에서도 소멸자가 생성된다.

[실행결과분석]



객체가 생성될 때 생성자가 실행되는데, 상속관계일 때 부모 객체의 생성자도 생성된다. (단, 부모객체의 생성자가 먼저 생성) 그렇게 new를 통해서 생성한 객체 4개에 대한 생성자가 출력.

이후 다형성을 보기 위해 각각의 포인터 객체의 play함수를 실행한 결과가 출력된다. 상속관계에서 부모 객체에 play함수가 있더라도 자식 객체의 play함수가 실행되어 다음과 같은 문장 4개를 출력한다.

이후 main함수의 cout을 통해 "연산자 오버로딩 실습"을 출력한다.

<< 연산자 오버로딩 사용을 통해 포인터 객체 mp[0]의 멤버함수인 play함수를 실행한다. "VTR이 play되고 있습니다." 가 출력되고, 이후 ostream 내부의 os를 return한다.

이후 + 연산자 오버로딩과 =(대입) 연산자 오버로딩을 활용하여

mp[3] = *mp[1] + mp[2]를 계산한다. + 연산자 오버로딩을 실행할 때 new를 통해 새로운 객체(newmp)가 생성되는데 이때 생성자도 실행된다.

이후 = (대입) 연산자를 통해 mp[3]에 연산을 거친 포인터 객체가 생성된다.

이후 mp[3]->getLength를 통해 mp[3]의 private에 있는 length 값인 13을 출력한다.

이후 for문을 통해 mp[3], mp[2], mp[1], mp[0]을 차례로 delete해준다. 객체가 소멸되면서 소멸자가 출력된다.

이후 학번 이름을 출력한다.

[결론 및 느낀점, 어려웠던 점]

이번 자료구조론 첫 번째 과제를 통해 코드를 구성해보면서 포인터가 활용하기 까다로운 형태라는 것을 새삼 느끼게 되었다. 그 중에서도 이 과제는 포인터 객체의 배열이었기 때문에 객체지향프로그래밍이 능숙하신 분들에게는 쉬웠을 수 있는 과제이지만 나에게는 너무나 어려웠다. 강의노트와 교수님 필기를 최대한 곱씹어보며 어떠한 기능의 코드는 왜 사용하는 것인지? 알아갈 수 있는 과제였다. 어려움이 있었던만큼 마음처럼 되지 않았던 코드들이 있었다. mp[3] = *mp[1] + mp[2]의 식과 그 조건을 봤을 때 mp[3]의 length 값만 바꿀 수 있지 않을까 라는 생각을 했다. 하지만 대입 연산자 중 return type을 좌변의 mp[3]와 맞추기 위해 포인터 객체를 사용했고, 이를 위해 Media*를 type으로 가지는 포인터객체를 생성하고 값을 동적으로 할당하게 되었다. 결국 그 생성한 포인터객체를 넘겨주고 mp[3]에 대입을 하여 기존의 mp[3]는 새롭게 생성된 포인터객체에 대입되었다. 또한, 생성자와 소멸자의 실행 당시 부모클래스의 생성자와 소멸자가 실행되는 것이 맞지만, 출력상 깔끔하지 않아, 가장 자식에게 있는 생성자와 소멸자만 실행되게 하고싶어 방법을 구상했지만, 상속이 한 단계만 이루어지는 것이 아니라 두 단계가 이루어져 해결하지 못했다. 하지만 상속받은 자식클래스의 객체를 생성할 때, 부모클래스의 객체의 생성자도 실행된다는 것을 출력하는 것은 더욱 자세한 정보 전달을 위해 필요하다고 생각한다.