2020년도 2학기 컴퓨터공학설계및실험Ⅰ

3주차 기초 C++ 프로그래밍 CPP-1 : Range Array 결과 보고서

20161663 허재성

1. 실습 목적

C++는 객체지향 프로그래밍 개념을 도입하여 클래스(Class)를 확장하고 수정하는 수단으로 클래스 상속(Class Inheritance)을 제공한다. 이번 실험에서는 효율적인 프로그래밍을 위하여 상속 방법을 익혀보도록 한다.

2. 실습 구현 내용

주요 개념

객체 지향 프로그래밍(OOP) : 객체 지향 이론의 기본 개념은 실제 세계는 사물(객체)로 이루어져 있고 발생하는 모든 사건들은 사물 간의 상호작용으로 간주하는 것이다. 실제 사물의 속성과 기능을 분석한 다음 데이터(멤버 변수)와 기능(함수)로 정의함으로써 실제 세계를 컴퓨터 속에 옮겨 놓은 것과 같은 가상 세계를 구현할 수 있다. 1960년대 중반 이후로 객체 지향 이론을 프로그래밍 언어에 적용한 시뮬라(Simula)라는 최초의 객체 지향 언어가 탄생하였다.[1] 객체 지향 이론인 데이터의 추상화, 캡슐화와 정보 은닉, 상속성, 다형성, 코드의 재활용 등을 접목시켜 프로그램을 설계하는 방법이 OOP이다.[2]

OOP를 쓰는 이유 : 객체 지향 프로그램의 가장 큰 장점은 코드의 재사용성이 높아 기존의 코드를 이용하여 새로운 코드를 쉽게 작성할 수 있다. 또한 코드간의 관계를 이용하여 적은 노력으로 쉽게 코드를 변경할 수 있어서 유지 보수가 용이하다는 점이다. 이러한 장점은 프로그램의 개발과 유지보수에 드는 시간과 비용을 획기적으로 개선할 수 있다. 그리고 제어자와 메서드를 이용하여 데이터를 보호하고 올바른 값을 유지할 수 있도록 하며 코드의 중복을 제거하여 코드의 불일치로 오작동하는 일을 방지할 수 있어 프로그램의 신뢰도를 높일 수 있다.[3]

객체(Object) : 객체의 사전적 정의는 실제로 존재하는 것으로 주변의 책상, 의자, 자동차와 같은 유형적인 사물은 물론 개념이나 논리 같은 무형적인 것들도 객체로 간주할 수 있다. 프로그래밍에서의 객체는 클래스에 정의된 내용대로 메모리에 생성된 것을 의미한다.[4] 객체는 객체가 가지고 있는 속성 또는 특성을 나타내는 상태(State)와 객체의 기능 및 반응을 나타내는 행동(Behavior)로 이루어져 있다. 객체의 상태는 멤버 변수(Member Varivale)을 이용해 구현하며, 객체의 행동은 멤버 함수(Member Function) 또는 메소드(Method)로 구현된다.[5]

클래스(Class) : 클래스는 객체를 정의해 놓은 것이다. 객체가 사물에 대응된다면 클래스는 사물을 만들기 위한 틀(Template) 또는 설계도에 대응된다. 자동차나 TV 같은 사물을 설계도를 이용해 만들어내듯이, 클래스는 객체를 생성하는 데 사용된다. 클래스는 객체를 생성하기 위해 사용되지만 클래스 그 자체가 객체는 아니므로 원하는 기능의 객체를 사용하기 위해서는 클래스로부터 객체를 생성하는 과정이 먼저 이루어져야 한다. 따라서 프로그래밍의 순서는 클래스를 작성한 후에 클래스로부터 객체를 생성한다.[6]

인스턴스(Instance) : 클래스로부터 객체를 만드는 과정을 클래스의 인스턴스화(Instantiate)라고 하며, 어떤 클래스로부터 만들어진 객체를 해당 클래스의 인스턴스라고 한다. 인스턴스와 객체는 비슷한 의미지만 인스턴스는 특정 클래스로부터 만들어진 것임을 강조하는 의미가 더 강하다.[7]

상속(Inheritance)의 개념과 사용 이유

상속(Inheritance) : 상속이란 기존의 클래스를 재사용하여 새로운 클래스를 만드는 것이다.[8] 클래스가 인스턴스를 생성할 때 기존에 존재한 다른 클래스의 데이터(맴버 변수)/기능(멤버 함수)를 가져와서 자신이 가진 데이터와 기능과 합쳐 하나의 인스턴스를 생성하는 것이다.[9] 데이터와 기능을 기존 클래스에서 가져올 때 기존 클래스를 기반 클래스(Base Class)라고 하고 가져온 새로운 클래스를 파생 클래스(Derived Class)라 하며 기반 클래스를 파생 클래스가 상속한다고 한다. 기반 클래스와 파생 클래스는 기반 클래스의 데이터와 기능을 공통으로 가지고 있고 파생 클래스는 파생 클래스만의 기능과 데이터를 따로 가지고 있다. 파생 클래스와 기반 클래스 사이에는 IS-A (~는 ~이다, 여기서는 파생 클래스는 기반 클래스이다.) 관계가 성립한다.

상속의 사용 이유 : 상속을 통해서 클래스를 작성하면 새로운 클래스를 작성할 때 기존 클래스와 공통된 부분을 코드로 작성할 필요 없이 기존 클래스를 기반 클래스로, 새로운 클래스를 파생 클래스로 하여 파생 클래스가 기반 클래스를 상속하도록 하면 보다 적은 양의 코드로 새로운 클래스를 작성할 수 있고 코드를 공통적으로 관리할 수 있기 때문에 코드의 추가 및 변경이 용이하다. 상속의 이러한 특징은 코드의 재사용성을 높이고 코드의 중복을 제거하여 프로그램의 생산성과 유지보수를 개선한다.[10]

Array 클래스

멤버 변수

멤버 변수는 모두 접근 제어자가 protected로 선언되어 있어서 Array 클래스와 Array 클래스를 상속하는 RangeArray 클래스에서 직접 접근이 가능하다.

(1) int\* data

int형 변수를 저장할 1차원 배열의 역할을 할 포인터 변수이다. Array 객체가 생성되어 배열의 길이가 정해지면 해당 길이만큼 동적할당된다.

(2) int len

배열의 길이를 나타내는 변수이다. Array 클래스의 생성자에 의해 객체가 생성될 때 초기화된다.

멤버 함수

멤버 함수는 접근 제어자가 public으로 선언되어 있어 클래스 외부에서도 접근이 가능하다.

(1) Array(int size)

Array 클래스의 생성자로 Array 객체를 만들기 위해 반드시 호출된다. 인자로 size를 전달받아 size가 0보다 큰 유효한 정수이면 멤버 변수 len을 size로 초기화하고 size 크기만큼 1차원 배열을 동적 할당받아 멤버 변수 data가 가리키게 한다. 인자로 0 이하의 값이 전달되면 에러 메시지를 출력하고 종료한다.

input : 배열의 크기가 될 정수 size

return : 없음

(2) ~Array()

Array 클래스의 소멸자로 Array 객체가 소멸될 때 호출된다. 생성자에서 동적할당한 메모리를 해제하는 역할을 한다.

input : 없음

return : 없음

(3) int length() const

Array 클래스의 멤버 변수인, 배열의 길이를 나타내는 변수 len의 값을 반환한다. 멤버 변수의 접근 제어자가 protected로 Array 클래스와 파생 클래스 밖에서는 접근할 수 없으므로 length() 함수를 통해 간접적으로 멤버 변수 len의 값을 얻어온다. 함수 내에서 배열의 길이 len이 변해서는안되므로 const 선언을 해준다. 이처럼 멤버 변수에 저장된 값을 수정하지 않는 함수는 const 선언을 해주는 것이 좋다.[11]

input : 없음

return : 배열의 길이 len

(4) void print()

Array 클래스의 동적 배열 data에 저장된 값을 인덱스 0부터 len-1까지 출력 형식에 맞게 차례대로 출력한다.

input : 없음

return : 없음

연산자 다중정의

연산자 다중정의는 접근 제어자가 public으로 선언되어 있어 클래스 외부에서도 접근이 가능하다.

(1) int& operator [](int i)

연산자 []를 다중정의한 것으로 동적할당한 배열 data의 i번째 인덱스의 원소 data[i]에 접근할 수 있도록 한다. 전달 인자 i가 0 이상, len – 1 이하로 유효할 경우, data[i]의 값에 대입할 수 있도록(left-value) data[i]의 참조자를 반환한다. i가 유효하지 않을 경우 에러 메시지를 출력하고 다중 정의 내 static 변수인 tmp의 참조자를 반환한다.

input : 접근할 배열의 인덱스 i

return : i가 유효할 경우 data[i]의 참조자(left-value), 유효하지 않을 경우 다중 정의 내 static 변수인 tmp의 참조자 반환

(2) int operator [](int i) const

연산자 []를 다중정의한 것으로 동적할당한 배열 data의 i번째 인덱스의 원소 data[i]를 얻어 올 수 있도록 한다. 전달 인자 i가 0 이상, len – 1 이하로 유효할 경우, data[i]의 값(right-value)을 반환한다. i가 유효하지 않을 경우 에러 메시지를 출력하고 0을 반환한다. const 선언을 통해 int& operator [](int i)와 구별된다. const 선언도 함수 다중정의의 조건이 된다.[12]

input : 접근할 배열의 인덱스 i

return : i가 유효할 경우 data[i]의 값(rightt-value), 유효하지 않을 경우 0 반환

RangeArray 클래스

기반 클래스 Array를 상속하는 파생 클래스로 Array 클래스의 모든 멤버 변수, 멤버 함수를 사용할 수 있다. Array 클래스와 달리 배열의 시작 인덱스로 0이 아닌 값을 허용하며 음수 인덱스도 허용한다. 이를 위해 RangeArray 클래스에 시작 인덱스와 끝 인덱스를 의미하는 멤버 변수를 추가하고, Array 클래스의 생성자를 호출하고 시작 인덱스와 끝 인덱스를 초기화하는 생성자, 시작 인덱스와 끝 인덱스 값을 반환하는 멤버 함수도 각각 추가한다. 또한 [] 연산자 다중 정의에서는 Array 클래스에서 다중 정의한 [] 연산자를 호출한다.

멤버 변수

멤버 변수는 모두 접근 제어자가 protected로 선언되어 있어서 RangeArray 클래스와 RangeArray 클래스를 상속하는 클래스에서 직접 접근이 가능하다.

(1) int low

배열의 첫 번째 인덱스를 의미하는 변수이다. 기존 배열과 다르게 0이 아닌 값도 가능하며 음수 인덱스도 허용한다.

(2) int high

배열의 끝 인덱스를 의미하는 변수이다. 음수 인덱스도 허용한다.

멤버 함수

멤버 함수는 접근 제어자가 public으로 선언되어 있어 클래스 외부에서도 접근이 가능하다.

(1) RangeArray(int l, int h)

RangeArray 클래스의 생성자로 RangeArray 객체를 만들기 위해 반드시 호출된다. 인자로 l과 h를 전달받아 h – l + 1을 새로운 size로 하여 Array 클래스의 생성자에 전달하여 Array 클래스의 생성자를 호출한다. 또한 RangeArray 클래스의 멤버 변수 low와 high를 각각 l과 h로 초기화한다.

h – l + 1을 전달받은 Array 클래스의 생성자는 h – l + 1을 size로 간주하여 유효할 경우 위에서 설명한대로 1차원 배열을 동적할당하여 data가 가리키게 한다. RangeArray 클래스의 생성자는 RangeArray의 고유한 멤버 변수를 초기화하고 Array 클래스의 생성자를 호출하는 일만 한다.

input : 배열의 시작 인덱스 l, 끝 인덱스 h

return : 없음

(2) ~RangeArray()

RangeArray 클래스의 소멸자로 RangeArray 객체가 소멸될 때 호출된다. 동적할당한 메모리는 ~Array 소멸자에서 해제하므로 ~RangeArray() 소멸자가 따로 할 일은 없다..

input : 없음

return : 없음

(3) int baseValue() const

RangeArray 클래스의 멤버 변수인, 배열의 첫 번째 인덱스를 나타내는 변수 low의 값을 반환한다. 멤버 변수의 접근 제어자가 protected로 RangeArray 클래스와 파생 클래스 밖에서는 접근할 수 없으므로 baseValue() 함수를 통해 간접적으로 멤버 변수 low의 값을 얻어온다. 함수 내에서 배열의 길이 low가 변해서는안되므로 const 선언을 해준다.

input : 없음

return : 배열의 첫번째 인덱스 low

(4) int endValue() const

RangeArray 클래스의 멤버 변수인, 배열의 끝 인덱스를 나타내는 변수 high의 값을 반환한다. 멤버 변수의 접근 제어자가 protected로 RangeArray 클래스와 파생 클래스 밖에서는 접근할 수 없으므로 endValue() 함수를 통해 간접적으로 멤버 변수 high의 값을 얻어온다. 함수 내에서 배열의 길이 high가 변해서는안되므로 const 선언을 해준다.

input : 없음

return : 배열의 끝 인덱스 high

연산자 다중정의

연산자 다중정의는 접근 제어자가 public으로 선언되어 있어 클래스 외부에서도 접근이 가능하다.

(1) int& operator [](int i)

연산자 []를 다중정의한 것으로 첫 번째 배열의 인덱스가 low인 배열의 i번째 인덱스의 원소는 첫 번째 배열의 인덱스가 0인 인덱스의 i-low 번째 인덱스의 원소 data[i-low]에 대응한다. 따라서 Array 클래스에 다중 정의된 int& operator [](int i)에 인자로 i-low를 전달하여 호출한다.

input : 접근할 배열의 인덱스 i

return : Array 클래스의 다중 정의 int& operator [](int i)에 인자로 i-low를 전달하여 호출해서 얻은 data[i-low]의 참조자

(2) int operator [](int i) const

연산자 []를 다중정의한 것으로 첫 번째 배열의 인덱스가 low인 배열의 i번째 인덱스의 원소는 첫 번째 배열의 인덱스가 0인 인덱스의 i-low 번째 인덱스의 원소 data[i-low]에 대응한다. 따라서 Array 클래스에 다중 정의된 int operator [](int i) const에 인자로 i-low를 전달하여 호출한다.(const 선언으로 구분된다.)

input : 접근할 배열의 인덱스 i

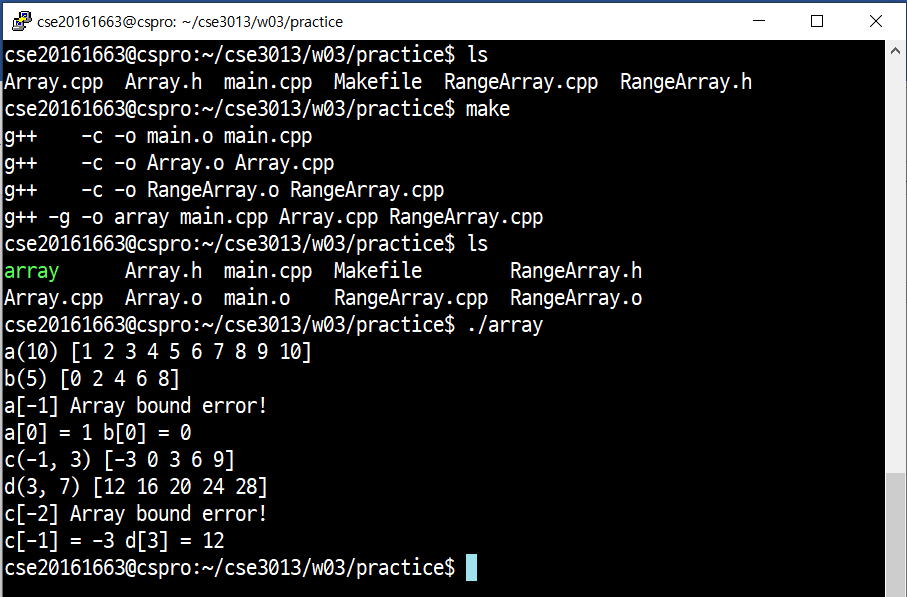
return : Array 클래스의 다중 정의 int operator [](int i) const에 인자로 i-low를 전달하여 호출해서 얻은 data[i-low]의 값

3. 실습 환경

CSPRO 서버 : CSPRO.sogang.ac.kr

4. 실습 결과 및 분석

교재에 나와있는 예시 main 함수의 실행 결과이다.



배열 c는 index가 -1부터 3까지, d는 인덱스가 3부터 7까지인 RangeArray 인스턴스이다. 배열의 시작 인덱스가 0이 아닌 것도 허용하고 음수 인덱스도 허용하는 것을 알 수 있다. 유효한 범위 내의 인덱스의 원소에 접근하여 값을 대입할 수도 있고(left-value), 값을 가져올 수도 있다(right-value).

RangeArray 클래스를 구현하기 위해서는 Array 클래스의 구현이 선행되어야 한다. Array 클래스 구현의 핵심인 연산자 다중정의 코드이다.

Array.cpp 파일의 일부분

|  |
| --- |
| #include “Array.h”  /\* 중간 생략 \*/  int& Array::operator [](int i) {  static int tmp;  if(0 <= i && i <= len – 1)  return data[i];  else {  cout<<”Array bound error!”<<endl;  return tmp;  }  }  int Array::operator [](int i) const {  if(0 <= i && i <= len – 1)  return data[i];  else {  cout<<”Array bound error!”<<endl;  return 0;  }  } |

Array 클래스의 객체, RangeArray 클래스의 객체 모두 실제 배열 역할을 하는 것은 Array 클래스의 멤버 변수인 int\*형 변수인 data 변수이다. Array 객체 또는 RangeArray 객체 모두 배열의 크기가 정해지면 그 크기만큼 인덱스가 0부터 시작하는 기초적인 자료구조의 배열을 동적으로 할당받아 data가 가리키게 한다. Array 클래스는 기존의 배열과 역할과 사용 방법이 완전히 같다. RangeArray 클래스의 경우 배열의 시작 인덱스 l와 끝 인덱스 h를 생성자에 인자로 전달하여 생성한다. l과 h로 배열의 실제 크기 h – l + 1을 계산하여 그 크기만큼 동적 배열을 생성한다.

생성한 동적 배열 data는 0부터 시작하는 인덱스이다. RangeArray 객체 배열 c의 시작 인덱스는 low이고 끝 인덱스는 high이다. 따라서 data[0]은 c[low]에 대응하고 data[len – 1]=data[high-low]은 c[high]에 대응한다. 그러므로 임의의 인덱스 i에 대하여 c[i]는 data[i-low]에 대응한다.

따라서 RangeArray 배열의 새로운 인덱스로 접근하기 위해 RangeArray 클래스의 연산자 다중 정의에서는 Array 클래스에서 다중 정의된 연산자를 호출하여 보정된 인덱스(i -> i-low)를 전달한다.

RangeArray.cpp 파일의 일부분

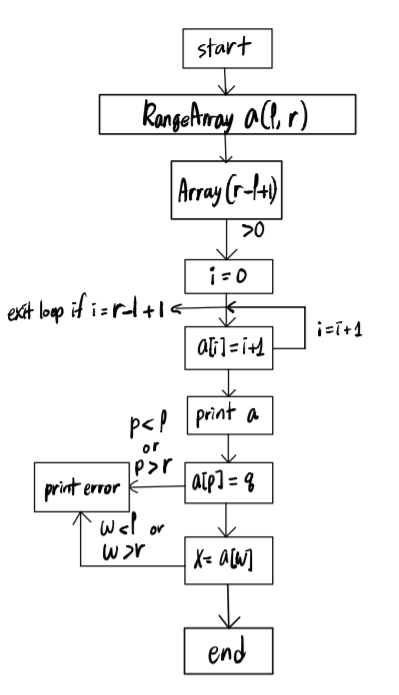
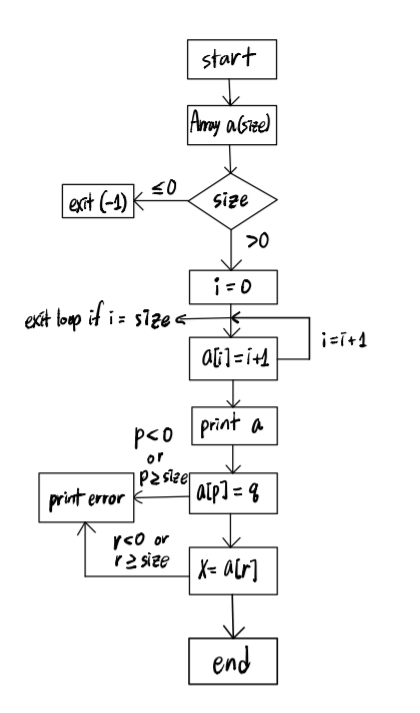
|  |
| --- |
| #include “RangeArray.h”  /\* 중간 생략 \*/  int& RangeArray::operator [](int i) {  return Array::operator [](i-low);  }  int RangeArray::operator [](int i) const {  return Array::operator [](i-low);  } |

RangeArray 클래스의 const 선언된 연산자 다중정의에서는 멤버 변수의 값을 변경할 수 없으므로 Array 클래스의 const 선언된 연산자 다중정의를 불러오게 된다.

사용한 자료구조 및 알고리즘

기초 자료구조인 1차원 배열을 동적 할당하여 int\*형 변수 data가 가리키게 했다. 인덱스가 low부터 시작하여 high로 끝나는 배열 RangeArray의 길이를 구하고 그 길이를 가지는 인덱스 0부터 시작하는 배열 data가 있을 때 RangeArray의 임의의 인덱스 i에 대응되는 data의 인덱스는 i-low이다.

프로그램의 구조도



Array 객체 생성 및 대입 연산자 실험 구조도(왼쪽)

RangeArray 객체 생성 및 대입 연산자 실험 구조도(오른쪽)

5. 과제

문자열 역할을 하는 클래스 Str의 멤버 함수를 구현한다.

주어진 멤버 변수

접근 제어자를 private로 하여 클래스 외부에서는 접근할 수 없도록 한다.

(1) char\* str

문자열의 내용을 저장할 동적할당한 배열을 가리킬 char형 포인터 변수이다.

(2) int len

문자열의 길이를 나타내는 변수이다.

구현할 멤버 함수

멤버 함수는 접근 제어자가 public으로 선언되어 있어 클래스 외부에서도 접근이 가능하다.

(1) Str(int leng)

Str 클래스의 객체를 만들기 위해 호출되는 생성자 중 하나이다. 인자로 정수형 변수 하나가 주어졌을 때 호출된다. Str 클래스의 멤버 변수 len을 전달받은 인자 leng으로 초기화하고 널 문자를 고려하여 len + 1 길이만큼의 문자 배열을 할당받아 str이 가리키게 한다.

input : 문자열의 길이 leng

return : 없음

(2) Str(const char\* neyong)

Str 클래스의 객체를 만들기 위해 호출되는 생성자 중 하나이다. 인자로 문자열을 가리키는 변수가 주어졌을 때 호출된다. <string.h> 헤더파일의 함수 strlen으로 neyong의 길이를 계산하여 해당 길이만큼 len을 초기화하고 널 문자를 고려하여 len + 1 길이만큼의 문자 배열을 할당받아 str이 가리키게 한다. <string.h> 헤더파일의 함수 strcpy를 이용하여 neyong의 내용을 str에 복사한다.

input : 저장할 문자열의 내용에 해당하는 문자열 neyong

return : 없음

(3) ~Str()

Str 클래스의 객체를 소멸할 때 호출되는 소멸자이다. Str 객체를 생성할 때 동적 할당되었던 메모리를 해제한다.

input : 없음

return : 없음

(4) int length() const

Str 클래스의 멤버 변수인, 문자열의 길이를 나타내는 변수 len의 값을 반환한다. 멤버 변수의 접근 제어자가 private로 Str 클래스 밖에서는 접근할 수 없으므로 length() 함수를 통해 간접적으로 멤버 변수 len의 값을 얻어온다. 함수 내에서 문자열의 길이 len 변해서는안되므로 const 선언을 해준다.

input : 없음

return : 문자열의 길이 len

(5) char\* contents() const

Str 클래스의 멤버 변수인 문자열 str을 반환하는 함수이다. 멤버 변수의 접근 제어자가 private로 Str 클래스 밖에서는 접근할 수 없으므로 contents() 함수를 통해 간접적으로 문자열을 얻어온다. 함수 내에서 문자열이 변해서는안되므로 const 선언을 해준다.

input : 없음

return : 문자열 str

(6) int compare(class Str& a)

해당 객체(compare 함수를 호출한 객체)의 문자열 str과 인자로 전달받은 Str 클래스의 객체 a에 저장된 문자열을 사전 편찬순으로 비교한다. 함수 내에서 <string.h> 헤더파일의 strcmp 함수를 호출하여 비교한다. a 객체 자체는 문자열이 아니므로 a.contents()를 호출하여 str과 비교한다.

input : 문자열 클래스 Str의 객체의 참조자 a

return : str과 문자열 a.contents()를 사전 편찬 순으로 비교한 결과. str이 앞서면 음수, 같으면 0, str이 뒤에 나오면 양수 반환

Str.cpp의 일부분

|  |
| --- |
| #include “Str.h”  /\* 중간 생략 \*/  int Str::compare(class Str& a) {  return strcmp(str, a.contents());  } |

(7) int compare(const char\* a)

해당 객체(compare 함수를 호출한 객체)의 문자열 str과 인자로 전달받은 문자열 a를 사전 편찬순으로 비교한다. 함수 내에서 <string.h> 헤더파일의 strcmp 함수를 호출하여 비교한다.

input : 문자열 a

return : str과 a를 사전 편찬 순으로 비교한 결과. str이 앞서면 음수, 같으면 0, str이 뒤에 나오면 양수 반환

연산자 다중정의

연산자 다중정의는 접근 제어자가 public으로 선언되어 있어 클래스 외부에서도 접근이 가능하다.

(1) void operator=(const char\* a)

연산자 =를 다중정의한 것으로 인자로 전달받은 문자열 a를 Str 객체의 문자열 str에 붙여 넣어 대입한다. <string.h> 헤더파일의 strlen 함수를 이용하여 a의 길이를 측정한다. 만약 str의 길이 len이 a의 길이보다 짧을 경우 할당된 동적 배열을 해제하고 (a의 길이 + 1) 만큼 배열을 다시 할당한다. 길이가 작거나 같을 경우는 메모리 해제 및 재할당 과정을 생략 가능하다. 그 다음 len의 길이를 a의 길이로 갱신하고 <string.h> 헤더파일의 strcpy 함수를 이용하여 a의 내용을 str에 복사한다.

input : 대입할 문자열 a

return : 없음

(2) void operator =(class Str& a)

연산자 =를 다중정의한 것으로 전달받은 Str 객체 a에 저장된 문자열을 호출한 Str 객체의 문자열 str에 붙여 넣어 대입한다. a에 저장된 문자열의 길이는 a.length()로 알아낼 수 있다. 위와 같은 방법으로 str과 객체 a에 저장된 문자열의 길이를 비교 후 재할당 여부를 결정한다. 그리고 len을 갱신한 다음 <string.h> 헤더파일의 strcpy 함수를 이용하여 a.contents()의 내용을 str에 복사한다.

Input : 대입할 문자열을 저장한 Str 객체의 참조자 a

return : 없음

Str.cpp의 일부분

|  |
| --- |
| #include “Str.h”  /\* 중간 생략 \*/  void Str::operator=(const char\* a) {  if(strlen(a) > len) {  delete[] str;  str = new char[strlen(a) + 1];  }  len = strlen(a);  strcpy(str, a);  }  void Str::operator=(class Str& a) {  if(a.length() > len) {  delete[] str;  str = new char[a.length() + 1];  }  len = a.length();  strcpy(str, a.contents());  } |

6. 결론

실습과 과제를 통해 OOP가 코드의 재사용성을 높이고 유지보수를 용이하게 하는 것을 확인하였다. 특히 상속을 통해 공통된 멤버 변수와 멤버 함수를 기반 클래스에 정의하고 기반 클래스를 상속한 파생 클래스를 정의하면서 공통된 부분과 추가된 부분을 따로 관리할 수 있게 되었다. C의 경우 데이터가 서로 비슷하지만 조금 다른 두 구조체가 있을 경우 중복된 부분이 많아도 각각의 구조체에 따로 선언해야 해서 코드의 길이가 길어지고 공통 부분의 코드를 수정해야 할 경우 두 구조체의 공통 부분을 둘 다 각각 수정해야 해서 유지, 보수하는데 시간, 비용이 많이 소모되었는데 상속을 통해 공통 부분은 기반 클래스만 관리하면 되므로 유지, 보수의 난이도가 낮아지게 되었다.

또한 연산자 다중정의를 통해 논리적으로는 가능하지만 지원하지 않았던 연산(ex 구조체 변수 간의 대입, 비교 연산 등)을 클래스 내에 연산자를 다중 정의하면서 객체 간의 연산으로 확장할 수 있다. 연산자 없이 멤버 함수로 구현하는 것은 코드의 길이도 길고 가독성도 좋지 않은데 연산자 다중정의를 통해 한 눈에 논리적으로 이해할 수 있을 정도로 가독성을 개선할 수 있다.

7. 참고 문헌

[1][3][4][6][7][8][10] 남궁 성, “Java의 정석 3rd Edition”, *도우출판*, 2016

[2][5][9] 서강대학교 컴퓨터 공학과 교수진, “컴퓨터공학 설계 및 실험 I”, *서강대학교 공학부 컴퓨터공학과*, 2012

[11][12] 윤성우, “윤성우의 열혈 C++ 프로그래밍[개정판]”, *오렌지미디어,*  2010