Object-Oriented Programming Report

Assignment 2-3

Professor	Donggyu Sim
Department	Computer engineering
Student ID	2022202061
Name	Seoeun Yang
Class (Design / Laboratory)	1 / B (미수강시 0로 표기)
Submission Date	2023. 4. 28

Program 1

❑ 문제 설명

직원들의 정보를 저장하는 프로그램을 만든다. Class 로 구성되며 멤버 변수에는 이름, 나이, 국적, 직업을 저장한다. 최대 10 까지 저장할 수 있으며 직원 추가, 저장된 직원 출력, 특정 직원 출력, 직원 정보 변경을 할 수 있어야 한다. 직원을 추가할 때 생성자 함수를 호출해 저장한다. 특정 직원 출력의 경우, 입력한 이름의 직원 정보를 출력한다. 만약 이름이 동일한 직원이 없다면 아무것도 출력하지 않는다. 정보 변경의 경우에도 검색할 직원과 변경할 내용을 같이 입력하는데, 동일한 이름의 직원 정보를 찾아 변경해주면 된다. 멤버 변수는 모두 private 으로, 멤버 함수는 public 으로 선언해 main 함수에서도 접근할 수 있도록 했다.

❑ 결과 화면

문제지 예시

❑ 고찰

처음 구현할 땐 생성자에서 strcpy() 함수를 통해 직원 정보들을 저장했는데 멤버 변수가 포인터로 선언되어 있어 주소값이 넘어가기 때문에 다음 직원 정보를 저장할 때 미리 저장한 정보들까지 함께 변했다. 이를 방지하기 위해 생성자에서 char 형인 멤버 변수를 main()함수에서 입력 받은 이름, 국적, 직업 글자의 길이만큼 동적으로 할당해주어 멤버 변수에 입력 정보를 복사하도록 구현했다. change()함수의 경우, 기존 정보들은 할당해주고 새로운 정보를 동적 할당해 복사했다. 소멸자는 나이를 제외한 모든 멤버 변수를 할당 해제하는 것으로 코딩했다.

Program 2

❑ 문제 설명

학생 정보를 저장, 출력하는 프로그램이다. 중첩 클래스를 사용하는데 하나는 student 클래스로 이름, 나이, 반 이름을 멤버 변수로 가진다. 다른 클래스인 school 클래스는 student 객체 100 개를 저장할 수 있는 배열과 학생 수를 멤버 변수로 가진다. 이 프로그램은 학생 추가, 이름 정렬, 모두 출력, 특정 반 출력 등을 할 수 있다. 이름 정렬의 경우 알파벳 순으로 정렬해 학생 정보를 출력한다. 특정 반 출력은 입력한 반의 학생들만 출력한다.

❑ 결과 화면

```
new_student Jim 8 Hope
new_student Alice 9 Love
new_student Claude 8 Hope
new_student Megan 11 Wish
print_all
=====print_all=====
Name: Jim
Age: 8
Class: Hope
----------
Name: Alice
Age: 9
Class: Love
----------
Name: Claude
Age: 8
Class: Hope
----------
Name: Megan
Age: 11
Class: Wish
----------
sort_by_name
print_all
=====print_all=====
Name: Alice
Age: 9
Class: Love
----------
Name: Claude
Age: 8
Class: Hope
----------
Name: Jim
Age: 8
Class: Hope
----------
Name: Megan
Age: 11
Class: Wish
----------
print_class Hope
=====print_class=====
Name: Claude
Age: 8
Class: Hope
----------
Name: Jim
Age: 8
Class: Hope
----------
Number of classmates: 2
exit
```

문제지 예시

❑ 고찰

sort_by_name 을 구현할 때 각 학생 이름의 아스키코드 값을 비교하면 정렬이 될 줄 알았다. 하지만 이렇게 구현할 경우 이름을 구성하는 알파벳의 아스키코드 합으로 비교하기 때문에 문제에서 원하는 알파벳 순으로 정렬할 수 없었다. 대문자와 소문자의 구분, 이름의 길이 등 수많은 예외가 발생하기 때문에 strcmp() 함수를 사용해 정렬했다. 이 함수의 경우 문자열에서 동일한 자리를 비교하기 때문에 알파벳 정렬이 가능하다. 예를 들어 Alice 와 Claude 를 비교할 때 각 첫번째 자리가 A 와 C 이기 때문에 Alice 가 배열의 앞에 위치되어 Claude 보다 먼저 출력된다. 이러한 방식으로 정렬할 경우 Ann 과 Anne 도 알맞게 비교할 수 있어 효율적이다.

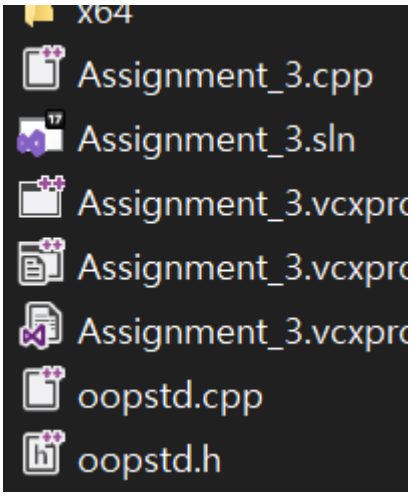
Program 3

❑ 문제 설명

string 에 사용할 수 있는 함수를 구현하는 프로그램이다. 직접 구현해야 할 함수는 생성자, 소멸자 포함 17 개이다. string class 를 선언해 구현하는데 생성자의 개수만 3 개이다. 자동생성자, 문자열 생성자, 복사 생성자를 통해 문자열 데이터를 가리키는 주소와 문자열 길이를 저장한다. 소멸자는 s 가 동적으로 할당되기 때문에 할당 해제를 해준다. assign()함수는 string 에 새로운 값을 저장한다. 따라서 기존에 존재하던 값은 사라진다. at()함수는 특정 위치의 문자를 탐색한다. 매개변수로 위치를 넘기면 해당 인덱스의 문자를 반환한다. c_str()함수는 string 형 문자열을 char 형으로 변환해 반환한다. clear()함수는 string 을 지운다. 즉 NULL 의 형태로 바꿔준다. push_back()함수는 기존에 있던 문자열의 맨 뒤에 문자를 추가로 삽입하는 함수이다. compare()함수는 문자열을 비교하는 함수이다. 동일하면 0, 비교하는 string 이 작으면 -1, 크면 1 을 반환한다. replace()함수는 특정 위치에 존재하는 문자열을 교체한다. 매개변수로 바꿀 위치, 길이, 교체할 문자열을 넘기면 시작 위치부터 길이만큼의 문자를 삭제하고 그 자리에 교체할 문자열을 삽입한다. substr()함수는 특정 위치부터 길이만큼 문자열을 잘라 반환한다. find()함수는 특정 문자를 검색해 존재하면 해당 인덱스 값, 존재하지 않으면 -1 을 반환한다. stoi(), stof()함수는 string 에 저장되어 있는 숫자를 정수형, 실수형 변수에 저장하고 뒤에

문자가 나온다면 문자가 나오기 시작하는 인덱스 값을 반환하는 함수이다. 반대로 to_string 은 각각 정수, 실수를 string 으로 변환하는 함수이다.

❑ 결과 화면



class string 을 사용자 정의 헤더 파일에 선언하고, oopstd.cpp 파일에 함수들을 정의해준 후 Assignment_3.cpp 의 main()함수에서 직접 구현한 함수들을 사용한다.

```
oopstd::string a;//assign
a.assign("Hi");
cout << a.c_str() << endl;
cout << endl;

oopstd::string example("Hello World");//at
cout << example.at(2) << endl;
cout << endl;

const char* b;//c_str
b = example.c_str();
cout << b << endl;
cout << endl;

oopstd::string example2("Hello World");//clear
example2.clear();
cout << example2.c_str() << endl;
cout << endl;
```

```
oopstd::string example3("Hello World");//push_back
example3.push_back('b');
cout << example3.c_str() << endl;
cout << endl;

oopstd::string ex1("HELLO");//compare
cout << ex1.compare("Hello") << endl;
cout << ex1.compare("HELLO") << endl;
cout << ex1.compare("HA") << endl;
cout << endl;

oopstd::string example4("Hello World");//replace
cout << example4.replace(0, 5, "Hi").c_str() << endl;
cout << endl;
```

```
Hello Worldb

-1
0
1

Hi World
```

```
oopstd::string example5="Hello World";//substr
oopstd::string e = example5.substr(5, 10);
cout << e.c_str() << endl;
cout << endl;

oopstd::string example6 = "Hello World";//find
cout << example6.find("World") << endl;
cout << endl;

oopstd::string number = "10bus";//stoi
size_t sz;
int num = oopstd::stoi(number, &sz,10);
cout << num << endl;
cout << sz << endl;
cout << endl;
```

```
 World

6

10
2
```

```
oopstd::string float_number = "1.5abc";//stof
size_t sz_1;
float float_num = oopstd::stof(float_number, &sz_1);
cout << float_num << endl;
cout << sz_1 << endl;
cout << endl;

oopstd::string y = oopstd::to_string(123);//to_string(int)
cout << y.c_str() << endl;
cout << endl;

oopstd::string z = oopstd::to_string((float) - 24.3);//to_string(float)
cout << z.c_str() << endl;
cout << endl;
```

```
1.5
3

123

-24.299999237060546875
```

❑ 고찰

push_back()함수를 구현할 때 char* 변수를 새로운 길이로 동적할당 해줘야 했다. 이 때 할당 크기가 관건이었는데 처음 구현할 땐 기존 길이보다 1 크게 할당하여 뒤에 문자를 삽입했다. 하지만 이 경우 함수가 종료될 때 소멸자를 호출하는데 할당 해제할 때 디버깅 오류가 났다. 기존 크기보다 2 크게 할당할 땐 정상적으로 프로그램이 종료됐다. 아직도 정확한 이유는 모르지만 문자열을 할당할 땐 항상 크기를 신경써야 한다는 것을 깨달았다. 또한 실수형을 string 형으로 변환하는 to_string(float val)함수를 구현하고 main()함수에서 불러올 때 매개변수로 실수를 넣으면 자동으로 double 형 변수로 취급되어 오류가 났다. float 형으로 강제 형변환을 시켜준 후 프로그램을 실행하니 숫자가 깔끔하게 떨어지지 않음을 결과 화면으로 알 수 있다. float 로 강제적으로 변환해주면서 숫자의 오차가 생기는 것으로 파악된다. 숫자의 자릿수를 매개변수로 넘겨줄 수도 없고, 함수 내에서 다시 double 형으로 변환해도 오차가 사라지지 않아 어쩔 수 없이 남겨두게 되었다.

Program 4

❑ 문제 설명

입력된 행렬을 통해 계산을 해주는 프로그램이다. 행렬 2 개의 합, 차, 곱 계산과 행렬에 상수와 사칙연산을 하는 함수, determinant 계산과 전치행렬, 수반행렬, 역행렬을 구하는 프로그램이다. Class 를 통해 구현하는데 멤버 변수로 행과 열의 크기, 2 차원 배열 포인터를 가진다. 모든 함수들이 Class 밖에 선언되어 있기 때문에 멤버 함수에 변수들을 설정하고 반환하는 함수를 작성해 접근해야 한다. Class 밖에 선언된 함수들은 모두 결과를 저장할 행렬 변수가 매개 변수로 들어가고 반환된다. 따라서 함수들의 반환형은 class Matrix 이다.

❑ 결과 화면

```
mat :
2       2       3
1       1       2
1       2       3
determinant of mat : -1

elementAdd 3
5       5       6
4       4       5
4       5       6

elementSub 2
0       0       1
-1      -1      0
-1      0       1

elementMul 4
8       8       12
4       4       8
4       8       12

elementDiv 3
0.666667        0.666667        1
0.333333        0.333333        0.666667
0.333333        0.666667        1
```

mat 행렬의 determinant 를 계산해주고, mat 에서 3 을 더해준 결과값, mat 에서 2 를 빼준 결과값, mat 에서 4 를 곱해준 결과값, mat 에서 3 을 나눠준 결과값이다. 행렬과 상수의 사칙연산 결과이다.

```
mat5 :
1       2       3
4       5       6
7       8       9

add func with mat, mat5
3       4       6
5       6       8
8       10      12

sub func with mat, mat5
1       0       0
-3      -4      -4
-6      -6      -6

mul func with mat, mat5
31      38      45
19      23      27
30      36      42

transpose of mat5
1       4       7
2       5       8
3       6       9
```

```
mat :
2       2       3
1       1       2
1       2       3
```

mat5 행렬을 선언해 mat 행렬과의 합, 차, 곱 연산을 한 행렬을 출력한다. transpose 는 mat5 의 전치행렬이다.

```
mat9 :
1       4       1
2       3       1
0       2       1
adjoint matrix of mat9 :
1       -2      1
-2      1       1
4       -2      -5
inverse matrix of mat9 :
-0.333333       0.666667        -0.333333
0.666667        -0.333333       -0.333333
-1.33333        0.666667        1.66667
```

mat9 행렬을 선언해 수반행렬과 역행렬을 계산하는 프로그램이다. 이들을 구할 때 determinant() 함수와 transpose() 함수를 적극적으로 호출한다.

❑ 고찰

determinant()함수를 구현할 때 행렬의 크기가 제한되지 않기 때문에 소행렬을 구해 함수를 재귀적으로 호출했다. 3*3 행렬의 경우 2*2 크기의 소행렬을 선언해 소행렬의 determinant 를 먼저 계산해주는 함수를 작성했다. adjoint()함수의 경우, determinat 를 구하는 방식과 흡사함을 이용, 비슷하게 코드를 작성했다. 하지만 이 경우 모든 인덱스의 소행렬을 계산해줘야 되기 때문에 결과적으로 4 중 for 문을 도는 함수가 되었다. 인덱스 행과 열을 제외한 인덱스 값들로 소행렬을 만들어 determinant()함수를 호출해 새로운 행렬에 채워준다. 반환하기 전에 transpose() 함수를 통해 행렬을 전치시켜주면 수반행렬 연산이 끝난다. inverse()함수는 determinant()함수와 adjoint()함수를 통해 행렬을 구하고 각 인덱스 값을 det 값으로 나눠준 행렬을 반환한다.

사칙연산 연산은 어렵지 않았는데 행렬 크기의 제한이 없는 상황에서 모든 상황에서 사용 가능한 detminant, adjoint 함수를 만드는 것이 어려웠다. determinant 계산의 경우,

$$D = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} \begin{matrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{matrix} - \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} \begin{matrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{matrix}$$

이처럼 구할 수도 있지만 3*3 크기 이외의 행렬에도 적용 가능해야 했기에 연산의 정의에 입각해 코드를 작성했다.