



Object-Oriented Programming Report

Assignment 2-1

Professor	Donggyu Sim
Department	Computer engineering
Student ID	2022202065
Name	박나림
Class (Design / Laboratory)	2 / C
Submission Date	2023. 4. 5

Program 1

□ 문제 설명

자료구조인 stack의 함수들을 직접 구현하는 문제이다. stack은 배열이 수직으로 되었다고 생각하는 것으로, 입력 값들이 맨 아래에서부터 쌓이다가 나갈 때에는 맨 위에서부터 나가는 형식으로 진행된다. 마지막으로 들어온 게 첫번째로 나간다는 뜻으로 LIFO라고 부르기도 한다. stack의 구성요소들 중 먼저 push에서는, 사용자가 입력하면 그 숫자가 stack의 맨 아래에서부터 저장되는 형태이다. print로는 stack 맨 아래부터 모든 요소를 출력하고, Top을 입력했을 때 현재 남아있는 stack의 맨 위 요소가 출력되게 한다. pop을 입력하면 마찬가지로 남아있는 stack의 요소 중 top을 출력하고 해당 요소만 지운다. Empty를 입력하면 스택이 비어 있을 때 1, 그렇지 않을 시 0을 출력하며, exit를 입력하면 종료시킨다.

-구현 방법

먼저 stack은 주어진 크기인 128로 int형 배열로 선언한다. 그리고 stack에서 계속 현재의 위치를 가리킬 t(top) 포인터가 필요하므로, 일단 int형 변수로 -1로 초기화 해준다. 그리고 명령어, 함수들의 이름을 입력 받을 char형 배열을 선언해주어서 exit를 입력 받기 전까지 while문으로 계속 반복하여 받는 형태로 만든다. 사용자가 명령어를 입력하면 문자열을 비교하는 strcmp 함수를 이용하여 if else문으로 각각 명령어와 같은 함수로 stack과 t의 주소값을 전달해준다. 이때 empty는 그냥 t 그대로만 전달해준다. 이 empty 함수로는 다른 함수들에서 확인하는 용(emp)과 사용자가 확인하는 용(empty)로 나누어서 써준다. emp은 bool형으로 스택의 요소 존재 여부에 따라 true, false 값을 반환하며 empty는 void형으로 존재 여부에 따라 바로 1과 0을 출력하는 식으로 한다. 마찬가지로 void형으로 만들 push 함수에서는 stack과 *t를 매개변수로 받아서 사용자로부터 숫자도 입력 받은 뒤, stack의 인덱스 값을 먼저 증가한 후에 숫자를 저장하는 식으로 한다. t가 -1로 초기화 되었기 때문에, 후위증가를 이용할 시에는 배열 값이 제대로 저장 안되고 나중에 쓰레기값으로 출력되기 때문이다. 그 다음 int형으로 pop 함수를 만들어서 마찬가지로 매개변수들을 받은 뒤, emp 함수를 이용한 조건문으로 스택의 요소 존재 여부에 따라 비어있지 않을 시 top을 출력 후 후위감소로 배열을 반환한다. 비어있으면 안내문 출력 후 NULL 값을 반환하도록 한다. 나머지 top과 print는 void형으로, 똑같은 매개변수를 받아서 top에서는 pop과 마찬가지로 조건을 따지다가 비어있지 않을 시 top을 그대로 출력하고, 비어있으면 안내문을 띄운다. print에서는 for문으로 0부터 *t까지 반복하여 스택의 맨 아래 요소부터 차례대로 출력하도록 한다.

□ 결과 화면

```
Microsoft Visual Studio 디버그 콘솔
push 3
push -4
push 5
push 7
push 8
print
3 -4 5 7 8
top
8
pop
8
pop
7
pop
5
print
3 -4
empty
0
pop
-4
pop
3
empty
1
exit
C:\Users\박나림\OneDrive\바탕 화면\과제\2
```

Push 를 입력하면 stack 에 삽입되는 것을 print 를 사용함으로써 알 수 있다. 그리고 top 과 pop 을 사용하여 차례대로 지워짐을 알 수 있고, empty 에서는 stack 에 남아있는 요소들이 존재하는지 확인할 수 있다. 마지막으로 exit 를 입력하면 프로그램이 종료되는 형태이다.

□ 고찰

이번 문제인 stack 이 자료구조의 내용이다 보니 처음에는 이론 먼저 공부를 하였다. 그래서 어떻게 구현할지에 대한 틀이 잡히기 시작해서 바로 함수들을 구현하였으나 배열들의 값이 제대로 저장이 되지 않는 문제가 발생하였다. 처음에 push 함수에서 배열 값을 증가시킬 때, 습관적으로 후위증가로 썼기 때문이었다. t 의 값을 -1 부터 잡았으니 먼저 증가시키지 않으면 존재하지 않는 -1 칸에 숫자가 저장되는 형태가 되어버려서 나중에 print 를 출력해도 쓰레기값이 출력되는 것이었다. 그래서 이 부분을 수정하였다. 또한 이렇게 다른 함수들도 다 void 형이다 보니 처음엔 pop 함수도 똑같이 했다가 stack 값이 지워지지 않는 현상이 발생하여 int 형으로 값을 반환해줘야 한다는 것도 깨달았다. 그리고 print 함수에서도

시행착오들을 겪었는데, 이것도 습관적으로 배열 값이 NULL 이 되기 전까지로 반복을 하였다가 아무것도 출력되지 않는 현상이 발생하였기 때문이었다. 그래서 그러한 조건이 아닌, 현재 가리키고 있는 *t 까지로 반복하였더니 정상적으로 작동됐다. 하지만 정확한 원리는 아직 잘 모르겠어서 이 부분은 조금 더 이론을 찾아봐야 될 것 같다.

Program 2

□ 문제 설명

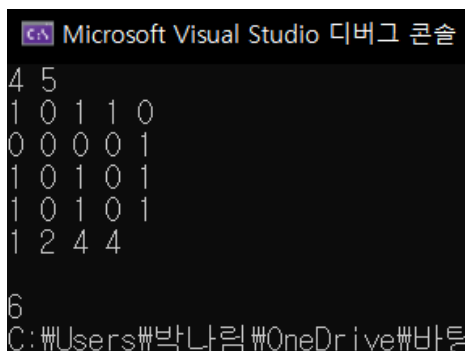
사용자로부터 미로의 구성요소 값들을 입력 받으면, 미로의 시작 지점부터 끝 지점까지의 최단 경로의 길이를 정수 값으로 알려주는 프로그램이다. 미로는 0 과 1 로 이루어지며, 1 은 벽으로써 갈 수 없는 곳이기 때문에 0 으로 된 길로만 움직일 수 있다. 미로의 사이즈는 행, 열 각각 1 부터 30 까지로만 받을 수 있다. 마지막으로 입력 받는 행 중에서 첫번째 두번째 값은 시작 지점, 세번째 네번째 값은 도착 지점을 나타낸다. 움직이는 방향은 동, 서, 남, 북 4 가지로만 구성되며, 미로를 통과하는 정답 경로는 1 가지로만 주어지는 문제이다.

-구현 방법

미로의 길을 찾으면서 그 과정을 저장하는 배열, 길을 잘못 찾을 시 그 배열 값에서 하나씩 지우면서 뒤로 가는 형식으로 진행해야 하므로 1 번 문제에서 구현한 스택을 활용한다. 여기서는 이동 횟수를 저장할 push, 횟수를 지울 pop, 스택이 비어 있는 지 검사할 emp(empty)만 사용하도록 한다. 먼저 사용자로부터 1 에서 30 까지의 범위로만 행, 열 값을 받는다. 그리고 2 차원 배열의 동적할당을 시켜주고, 미로의 구성요소 값들을 입력 받는다. 입력 받은 값들은 미로 길 찾기 함수로 전달하며, 동적할당을 해제하는 코드도 꼭 넣어주어서 main 함수는 끝낸다. 미로 길 찾기 함수에서는 2 차원 미로 배열, 시작 지점과 끝 지점의 행, 열 값들을 각각 매개변수로 전달받는다. 그리고 미로의 시작지점을 1 인 벽으로 바꿔주어서 시작지점보다 더 전으로 가지 못하게 만든다. 전체적인 while 문으로 출구를 찾을 때까지 반복시켜서, 4 가지 방향의 if else 문으로 조건을 검사한다. 만약 방향 값이 0 이 맞으면 그 방향으로 이동하고 현재 위치 값을 새로운 변수에 저장한다. 이동횟수도 증가시키고 push 함수에다 스택, 위치 변수, 이동 횟수 변수까지 전달한다. push 함수에서는 스택에다 경로 값(0 값)을 저장한다. 만약 4 가지

방향 모두 길이 아닐 경우에는 마지막 else 문에서 이동 횟수를 감소시키고, pop 함수로 스택과 현재 위치의 주소 값을 전달해준다. pop 함수에서는 emp 함수 여부에 따라 나누어서 비어 있지 않으면 현재위치(top)반환 후 인덱스 감소, 비어 있으면 NULL 값을 반환한다. 이 과정들을 반복하다가 끝 지점까지 도달하면 이동 횟수를 출력하고 프로그램을 종료한다.

□ 결과 화면



```
Microsoft Visual Studio 디버그 콘솔
4 5
1 0 1 1 0
0 0 0 0 1
1 0 1 0 1
1 0 1 0 1
1 2 4 4
6
C:\Users\박나림\OneDrive\바탕
```

행, 열 값과 미로 배열, 시작과 끝 위치를 입력하면 최단 루트로 가는 이동 횟수를 알려준다.

□ 고찰

저번 문제의 스택을 이용하면서 문제에 맞게 변환하는 게 어려웠던 것 같다. 특히 방향 전환을 할 때 조금 더 간략화 하기 위해서 구조체를 쓰는 방법도 해봤으나, 구조체 배열 값으로 조건문을 쓸 때 에러들이 발생하여 아직 미숙하다는 것을 깨닫고 if else 문으로 구현했다. 하지만 여기서도 처음에는 조건을 따질 때 바로 행이나 열 값을 증감하는 형식으로 하였더니 엑세스를 위반했다는 에러가 떴었다. 그래서 여러 시행 착오들을 겪다가, 증감하는 형식이 아니라 +1 이나 -1 로 배열 값을 바꾸지 않는 형태로 조건문을 따지고 그 안에서 조건이 맞을 시 증감하는 형태로 바꾸었더니 프로그램이 제대로 돌아갔다. 조건을 따질 때부터 증감 연산을 하면 그러한 배열 인덱스 값이 존재하지 않을 수도 있으니 에러가 났던 것 같다. 또한 이번 문제에서 2 차원 동적 배열을 할당하는 방법이 기존의 1 차원 배열의 동적 할당 방법과는 다르다는 점도 깨달았다. 1 차원처럼 한번에 쓰면 에러가

발생하고, 행의 수만큼 먼저 할당을 시키고 행만큼 반복하여 열자리의 메모리를 할당해주는 형식으로 해야 됐다. 할당을 해제할 때는 반대로 열 먼저 해제하고 마지막으로 기존처럼 해제하는 형식이었다. 이번 문제를 통해서 어떤 점이 안되고 어떤 점이 새롭게 되는 것인지 더 공부할 수 있었다.

Program 3

□ 문제 설명

oopstd 의 namespace 에 만든 1 차원 배열을 조작하는 함수들과 실제 기존 헤더파일(string, cstdlib)에 정의되어 있는 함수들을 비교하는 문제이다.

-구현 방법

oopstd.cpp 파일을 생성하여 namespace 안에다 memset, memcpy, strcmp, strncmp, strcpy, strncpy, strlen, atoi, atof 로 총 9 개의 문자열 관련 함수들을 구현한다. 그리고 main 함수 내에서 이들을 사용할 수 있도록 oopstd.h 로 헤더파일을 만들어서 이 namespace 의 함수들을 선언해준다. 그리고 양쪽의 cpp 파일에서 이 헤더파일을 불러와서 사용할 수 있게 만든다. memset 에선 메모리의 포인터, 채우고 싶은 값, 채우고 싶은 크기를 받아서 연속된 범위를 값으로 채울 수 있도록 구현한다. memcpy 에선 메모리의 포인터, 복사할 값, 복사할 사이즈를 받아서 값을 복붙하게 한다. strcmp 는 문자열 2 개의 포인터들을 받아서 비교하고 결과 값을 반환한다. strncmp 도 동일한 원리이지만 원하는 길이 변수를 추가로 받아서 그 길이만큼만 비교하고 반환한다. strcpy 에서는 저장할 공간인 문자열, 복사할 대상인 문자열 2 개의 포인터를 받아서 복사하고 저장하여 반환한다. 마찬가지로 strncpy 에서는 원하는 길이만큼만 복사하여 저장하고 반환하도록 한다. strlen 은 문자열의 길이를 세어서 반환하며, atoi 는 받은 문자열을 정수로, atof 는 실수로 반환하도록 구현한다.

□ 결과 화면

```
Microsoft Visual Studio 디버그 콘솔

/* memset */
Hello world
memset: ----- world

/* memcpy */
memcpy: abc, 21

/* strcmp */
str1: abcde
str2: abcd!
서로 다른 문자열입니다.

/* strncmp */
str1: abcde
str2: abcd!
비교 길이: 3
동일한 문자열입니다.

/* strcpy */
str: strcpy test
strcpy: strcpy test

/* strncpy */
str: strncpy test
복사 길이: 3
strncpy: str

/* atoi */

/* atoi */
str: 100
atoi/2: 50

/* atof */
str: 5
atof: 5.000000

C:\Users\박나림\OneDrive\바탕
```

네임스페이스 안에서 구현한 9 개의 문자열 관련 함수들을 차례대로 테스트해 본 결과이다.

아래처럼 구현한 함수의 헤더파일을 주석처리 하여 기존 함수들로 써도 동일한 결과 값이다.

```
oopstd.cpp  oopstd.h  Assignment 2-1_3.cpp  (전역 범위)
Assignment 2-1_3
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <iostream>
3  // #include "oopstd.h"
4  using namespace std;
5
```

```
Microsoft Visual Studio 디버그 콘솔

/* memset */
Hello world
memset: ----- world

/* memcpy */
memcpy: abc, 21

/* strcmp */
str1: abcde
str2: abcd!
서로 다른 문자열입니다.

/* strncmp */
str1: abcde
str2: abcd!
비교 길이: 3
동일한 문자열입니다.

/* strcpy */
str: strcpy test
strcpy: strcpy test

/* strncpy */
str: strncpy test
복사 길이: 3
strncpy: str

/* atoi */
str: 100
atoi/2: 50

/* atof */
str: 5
atof: 5.000000

C:\Users\박나림\One
```


□ 고찰

이번 문제에서는 두 개의 cpp 파일로 진행하다 보니 namespace 를 쓰고 가져오는 데에 조금 시행착오를 겪었다. 특히 헤더파일에 그대로 정의까지 할 경우에는 오류가 발생하므로 cpp 쪽에다가 정의를 하되 헤더파일에도 동일하게 namespace 를 쓰고, 그 안에서 함수의 원형들을 선언해줘야 main 함수에서까지 잘 전달되어 사용할 수가 있었다. 그래서 함수가 복잡해질 때에는 이렇게 파일을 나누는 편이 더욱 보기 편하다는 걸 깨닫게 되었다. 또한 문자열 함수들을 구현할 때 atoi 와 atof 에서 어려움이 있었던 것 같다. 문자의 형태를 숫자로 바꾸는 방법에 대해 생각해보다가, 해당 문자의 아스키 코드 값을 매칭시켜 찾은 뒤에 그 자릿수만큼 곱해서 더한다는 게 바로 안 떠올라서 그리면서 풀었더니 다른 함수들 보다 시간이 오래 걸렸다. 그래도 이번 기회에 이렇게 바꾸는 방식을 생각해 볼 수 있어서 도움이 된 것 같다.

Program 4

□ 문제 설명

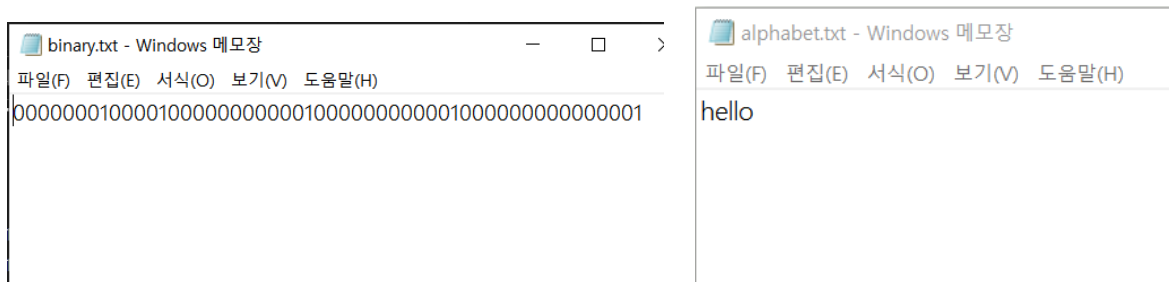
바이너리 데이터를 파일에서 읽어와서 클래스를 통해 알파벳으로 변환하고, 변환된 알파벳을 새로운 파일에 쓰는 프로그램이다.

-구현 방법

먼저 클래스에서 char 형 배열로 NULL 값 포함 27 칸을 생성하여 각 알파벳 소문자를 저장해준다. 이때 클래스에 그냥 쓰면 다른 함수들에서 접근할 수 없으므로 public 형태로 작성하여 준다. main 함수 내에서 클래스 객체를 생성하고, 파일에서 읽어 온 문자열과 알파벳으로 변환한 문자열을 저장할 char 형 배열을 각각 만들어 준다. 그 후 ifstream 을 통해 파일 객체를 생성하여 binary.txt 파일을 열고, ofstream 을 통해 파일 객체를 생성하여 alphabet.txt 파일을 열어준다. 이때 읽기용 객체는 이름 문제로 안 열릴 수 있기 때문에 예외처리 부분을 써준다. 그리고 while 문으로 파일 끝에 도달하기 전까지 파일을 한 줄씩 읽어온다. 다음 while 문에서는 무한 루프를 도는 형식으로, 저장한 문자열에서 처음부터

시작하여 1 을 만나기 전까지의 개수를 세어준다. 그리고 저장된 개수를 인덱스 값으로 갖는 클래스 객체의 배열을 통해 해당 알파벳 문자를 새로운 배열에 저장한다. 이 과정을 반복하다가 NULL 값을 만나면 반복문을 종료시키고 변환한 문자열을 쓰기용 파일에 출력한다. 마지막으로 읽기, 쓰기용 파일들을 다시 닫아주고 프로그램을 종료시킨다.

□ 결과 화면



binary.txt 파일에 데이터를 입력하고 나서 파일 읽기 모드를 통해 파일을 읽어 들이고 알파벳으로 변환한다. 그 후 파일 쓰기 모드로 alphabet.txt 파일을 생성하고 변환한 알파벳이 써진 모습이다.

□ 고찰

바이너리 문자를 알파벳으로 변환하는 과정에서 시행착오들을 겪었다. 처음에는 각 문자마다 배열을 생성하여 바이너리 데이터를 저장하는 식으로 했으나, 그럼 문자열을 비교하기도 까다롭고 컴파일 자체에서도 스택의 메모리를 많이 쓴다는 경고가 떴서 하나의 배열에 문자열을 저장하는 식으로 수정하였다. 그래서 인덱스 값을 이용하여 해당 문자열로 변환하는 것으로 했는데, 그 다음에는 반복문을 돌 때가 문제였다. 1 을 만나기 전까지의 0 의 개수를 세서 그만큼의 데이터를 갖는 알파벳으로 변환하는 원리인데, 처음 한번 과정을 거치고 두번째부터 다시 이어서 검사를 시작해야 되는 것을 제대로 변수 설정을 못해서 자꾸 무한 루프를 도는 상황이 발생하였다. 그래서 여러 수정을 하다가, 변수를 하나 더 만들어서 원래 증가 값을 저장하고 다시 그 값을 초기화로 갖는 for 문을 작성하는 걸로 바꿨더니 제대로 루프를 벗어날 수 있었다. 또한 이러한 과정에서 여러 변수들을 만들었기에, 나중에 보면 헷갈릴 수도 있을 것 같다. 앞으로는 보다 직관적으로 역할을 파악할 수 있도록 변수 이름을 지정할 것이다.