



# Object-Oriented Programming Report

## Assignment 2-1

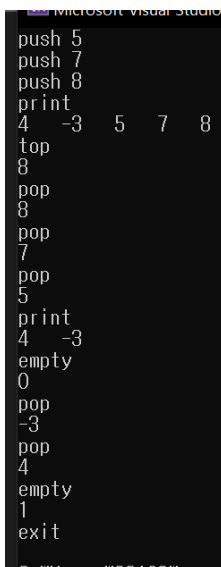
Professor	Donggyu Sim
Department	Computer engineering
Student ID	2022202061
Name	Seoeun Yang
Class (Design / Laboratory)	1 / B (미수강시 0로 표기)
Submission Date	2023. 4. 07

## Program 1

### □ 문제 설명

stack 을 구현하는 문제이다. 사용자가 command 를 입력하면 그에 맞는 연산을 수행하거나 cmd 창에 출력한다. 스택은 선형 데이터 구조로 입력하는 순서대로 저장한다. 하지만 최근에 저장했던 자료를 순서대로 지운다. 따라서 LIFO(last in first out)으로 설명 가능하다. 스택의 경우 1 차원 배열로도 충분히 구현 가능하다. 하지만 배열로 구현할 경우 결국 데이터 저장 개수에 한계가 생기기 때문에 연결 리스트로 구현했다. 사용자가 입력하는 command 는 총 6 개로 각 push, pop, top, print, empty, exit 가 있다. push 는 새로운 데이터를 저장하는 것이고, pop 은 저장된 데이터를 한 개씩 삭제한다. push 는 별도로 데이터를 입력을 받아야 한다. 노드에 저장하고 각 노드를 포인터로 연결한다. pop 을 할 때는 노드를 삭제하고 삭제한 노드를 가리키는 포인터를 NULL 로 초기화해준다. top 은 스택에서 가장 위에 있는 데이터 즉 가장 최근에 저장된 데이터를 출력한다. print 는 현재 저장된 데이터를 모두 출력하는데 bottom 부터 출력한다. bottom 은 top 과 다르게 가장 먼저 저장된 데이터이다. 즉 bottom 은 제일 마지막에 삭제된다. empty 는 스택에 데이터의 존재유무를 파악하고 exit 은 프로그램을 종료한다.

### □ 결과 화면



```
push 5
push 7
push 8
print
4 -3 5 7 8
top
8
pop
8
pop
7
pop
5
print
4 -3
empty
0
pop
-3
pop
4
empty
1
exit
```

문제지에 나와있던 예시

```

push 4
push 3
pop
3
pop
4
pop
stack underflow
exit

```

C:\Users\#82108\so stack 에 저장되어 있는 값이 없는데 pop 을 했을 경우

```

empty
1
push 4
push 98
print
4 98
pop
98
empty
0
pop
4
empty
1
print
exit

```

C:\Users\#82 stack 에 데이터를 저장하기 전, 중, 모든 pop 을 수행한 후의 empty 값

## □ 고찰

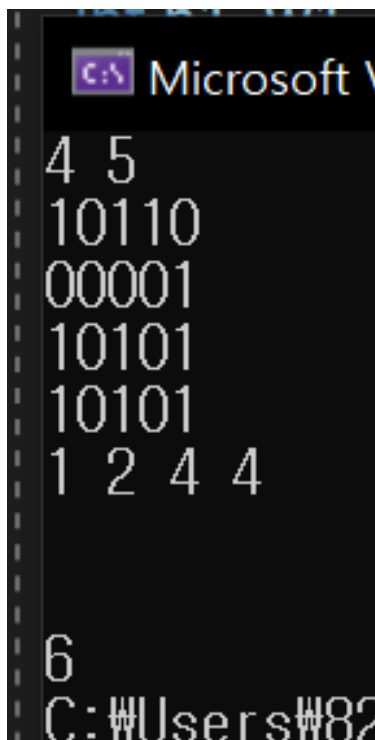
조교님께서 추가적으로 stack 의 크기를 알려주지 않으셨을 때 1 번을 구현했었다. 배열로도 충분히 구현 가능하지만 결국 배열이 꽉 차면 다음 push 를 수행할 수 없기 때문에 연결 리스트로 구현했다. 처음 구현해보는 거라 시행착오가 많았는데 노드를 연결한다는 개념 자체를 이해하는 과정이 가장 오래 걸렸다. push 의 경우 처음 노드를 삽입할 때와 아닐 때를 구분하고 pop 의 경우 일반 노드를 삭제할 경우와 제일 처음 삽입한 노드를 삭제할 경우를 구분 지어 구현해야 한다. stack 의 최대 크기가 정해져 있는 상황이라면 배열로 구현하는 것이 훨씬 쉬울 것 같다.

## Program 2

### □ 문제 설명

미로에서 최단 거리를 구하는 문제이다. 미로의 크기를 사용자로부터 입력 받아 동적 할당해주고 미로 또한 사용자가 입력한다. 공백 없이 미로를 입력하기 위해 char 형으로 입력 받고 다시 int 형으로 미로를 저장한다. 시작 좌표와 도착 좌표를 입력 받으면 두 좌표 간의 최단 거리를 찾아 출력한다. 좌표를 입력 받으면 find 함수를 호출해 동서북남 순으로 주변 좌표를 탐색한다. 만약 주변에 갈 수 있는 길이 없다면 다시 돌아가며 cnt 를 감소시켜 최단거리를 측정한다. 도착 좌표에 도착하는 그 즉시 cnt 를 출력하고 함수를 즉시 종료한다. 경로를 탐색하면서 그 자리의 값을 1 로 바꿔 경로를 탐색할 때 중복을 제외하고 탐색할 때마다 미로 범위 안에 있는지 확인해야 한다.

### □ 결과 화면



```
C:\> Microsoft V
4 5
10110
00001
10101
10101
1 2 4 4

6
C:\Users\82
```

문제지에 나와있던 예시

```
C:\> Microsoft
4 4
0100
0101
0101
0001
1 1 4 1

10
C:\Users\user>
```

다른 미로

```
C:\> Microsoft
4 5
10111
10001
11101
00001
2 1 1 4

9
C:\Users\user>
```

다른 미로

## □ 고찰

맨 처음에 구현을 int 형 2 차원 배열로 해서 char 형으로 입력 받아 int 로 변환시켰는데 굳이 그럴 필요는 없는 것 같다. char 형으로 거리를 탐색해도 충분히 값을 얻을 수 있을 것으로 판단된다.

재귀함수를 통해 프로그램을 구현하고 탐색하는 순서가 동서남북으로 정해져 있기 때문에 도착 좌표로 가는 길이 한 개 이상이라면 알맞은 값이 출력되지 않을 수도 있다. 문제를 보자마자 재귀함수를 통해 구현하고자 했는데 중복을 제외하고 탐색하는 과정에서 돌아가는 방법을 생각해내는 게 너무 어려웠다. 탐색하다가 길을 잘 못 들어서 cnt 를 감소시켜야 하는데 얼마나 어떻게 감소시켜야 할지 고민이 많았다. 만약 현재 위치한 칸에서 동서남북 모두 탐색했을 때 갈 수 있는 길이 없다면 cnt 를 감소시키고 전 칸으로 이동해 다른 방향을 탐색한다.

처음엔 함수를 하나만 만들어서 그 안에 동서남북을 다 넣었다가 동서남북 함수를 각 1 개씩 만들었다가 4 개를 한 번에 호출할 수 있는 또 다른 함수를 만들어 총 5 개의 사용자 정의 함수가 만들어졌다. 함수 호출이 너무 많아서 미로의 크기가 매우 커진다면 효율적이진 않을 것 같다.

## Program 3

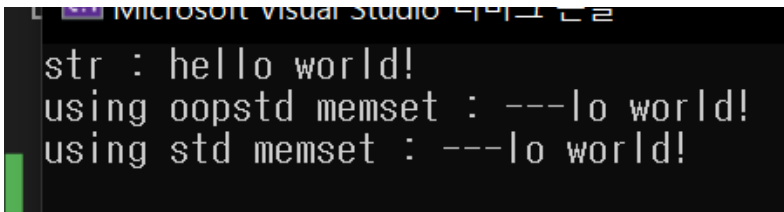
### □ 문제 설명

아홉 가지 함수를 직접 구현해 std 에 저장되어 있는 함수와 똑같이 구현되는지 확인하는 프로그램이다. 구현해야 할 함수는 memset, memcpy, strcmp, strncmp, strcpy, strncpy, strlen, atoi, atof 함수이다.

1. memset – 문자열의 앞부터 주어진 숫자의 길이만큼 사용자가 원하는 문자로 교체한다.
2. memcpy – 주어진 숫자의 길이만큼 사용자가 원하는 문자열을 복사한다.
3. strcmp – 두 문자열을 비교해 동일하면 0, 동일하지 않으면 1 또는 -1 을 반환한다.
4. strncmp – 두 문자열을 입력 받은 숫자의 길이만큼 비교해 동일하면 0, 동일하지 않으면 1 또는 -1 을 반환한다.
5. strcpy – 문자열을 복사해 복사된 문자열을 반환한다.
6. strncpy – 주어진 숫자의 길이만큼 문자열을 복사해 반환한다.
7. strlen – 문자열의 길이를 반환한다.
8. atoi – 문자열에 저장되어 있는 수를 정수형으로 변환해 반환한다.
9. atof – 문자열에 저장되어 있는 수를 실수형으로 변환해 반환한다.

### □ 결과 화면

<모든 실행 결과에 대한 코드는 Assignment\_3.cpp 파일에 주석 처리되어 있음>



```
str : hello world!  
using oopstd memset : ---lo world!  
using std memset : ---lo world!
```

memset 함수 (3 칸 '-'로 변환)

```
Microsoft Visual Studio 디버그 콘솔
str : hello world!
using oopstd memcpy : hello
using std memcpy : hello wo
```

memcpy 함수 (oopstd memcpy 로 5 자리 복사, std memcpy 로 8 자리 복사)

```
Microsoft Visual Studio 디버그 콘솔
str : hello world!
str2 : hello world!
str3 : Hello world!
str & str2 : 0
str & str3 : 1
```

strcmp 함수 (동일하면 0, 동일하지 않으면 1 or -1 출력)

```
Microsoft Visual Studio 디버그 콘솔
str : hello world!
str2 : hello world!
str3 : Hello world!
5자리 비교
str & str2 : 0
str & str3 : 1

Microsoft Visual Studio 디버그 콘솔
Looking R2 :
str : R2D2
str2 : R2D6
str3 : R5D8
Found :
R2D6
```

strncmp 함수 (str 과 나머지 문자열 5 자리 비교해 동일 여부 출력, str 과 나머지 문자열 2 자리 비교해 동일하면 출력)

```
Microsoft Visual Studio 디버그 콘솔
str : hello world!
using oopstd strcpy : hello world!
using std strcpy : hello world!

C:\Users\82108\source\repos\객체지향\
연습\1\1\main.cpp(9:11) : error C4013: 'strcpy' 함수 (str 복사해
```

strcpy 함수 (str 복사해 출력)

```

Microsoft Visual Studio 디버그 콘솔
str : hello world!
oopstd 3자리 복사, std 7자리 복사
using oopstd strcpy : hel
using std strcpy : hello w
C:\Users\82108\source\repos\객체지향\strcpy 함수

```

```

Microsoft Visual Studio 디버그 콘솔
str : hello world!
str2 : I wanna sleep....
using oopstd strlen : 12
using std strlen : 17

```

strlen 함수 (문자열 길이 출력, 각 str, str2 길이 출력)

```

Microsoft Visual Studio 디버그 콘솔
3426      -45.754
char to int using oopstd atoi : 3426
char to float using oopstd atof : -45.754
C:\Users\82108\source\repos\객체지향\atoi, atof 함수

```

atoi, atof 함수 (첫번째 줄은 문자열에 저장된 수, 2,3 번째 줄은 정수형, 실수형 변수에 저장된 수 출력)

```

Microsoft Visual Studio 디버그 콘솔
3.26      -1
char to int using oopstd atoi : 3
char to float using oopstd atof : -1
char to float using oopstd atof : -1.000000
C:\Users\82108\source\repos\객체지향\printf로 출력

```

printf 로 출력하면 소수점 아래 6 자리까지 출력할 수 있다. cout 의 경우 실수형이어도 소수점 아래 자리가 없으면 소수점을 자르고 출력한다.



## □ 고찰

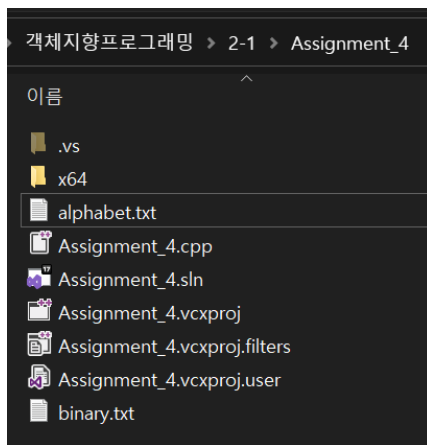
함수 자체를 구현하는 건 별로 어렵지 않았던 것 같다. 작년에 접해본 문제도 있었고 지금까지 배웠던 내용으로 충분히 구현할 수 있었던 문제였던 것 같다. atoi, atof 함수를 구현할 때 음수 또한 신경 써주는 것이 포인트였던 것 같다. 또한 문자열이기 때문에 정수형, 실수형으로 변환할 때 각 자리 수에서 '0'만큼 빼주고 알맞은 10의 거듭제곱을 곱해줘야 한다. 또한 실수를 atoi에 입력했을 때, 정수가 atof에 입력됐을 때 등 여러가지 예외를 처리해주는게 쉽지 않았다. 오히려 헤더 파일로 나누는 게 더 낫설어서 어려웠던 것 같다. 헤더파일 뿐만 아니라 따로 cpp 파일에 함수를 정의해줘야 해서 프로그램을 구현하기 전에 세팅이 어려웠다.

## Program 4

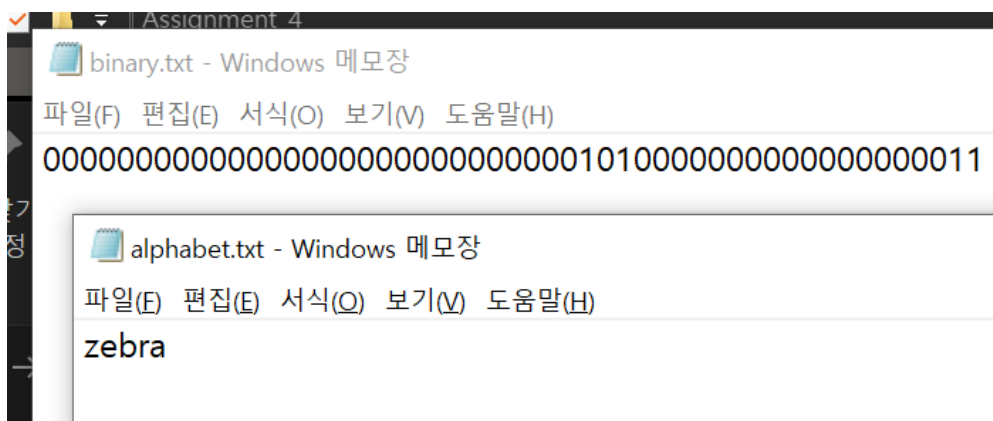
### □ 문제 설명

binary.txt에 있는 0과 1로만 이루어진 정보를 가져와 alphabet 정보로 변환해 alphabet.txt 파일에 저장하는 프로그램이다. a는 1, b는 01, c는 001로 점점 0의 개수가 많아진다. z는 1 없이 0으로만 이루어져 있다. a의 0의 개수는 0개, b는 1개... z는 25개임을 이용해 0의 개수를 탐색해 알파벳으로 변환해준다. 1을 발견하면 그 전까지 썼던 0의 개수를 파악해 알맞은 알파벳을 문자열에 저장하고 예외적으로 0이 25개면 z를 문자열에 저장하고 초기화해준다.

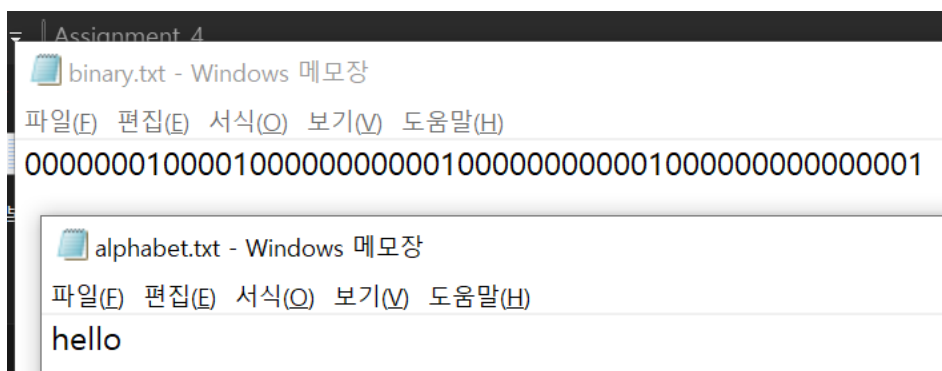
## □ 결과 화면



파일에 저장되어 있는 텍스트 파일



zebra 변환



hello 변환

## □ 고찰

맨 처음엔 알파벳 리스트를 모두 2차원 배열에 저장해 프로그램을 구현하려고 했는데 구현하다 보니 리스트가 필요하지 않음을 깨달았다. 0의 개수로 컨트롤할 수 있음을 깨닫고 0의 개수에 a의 아스키 코드값을 더해 저장하면 원하는 알파벳이 저장된다는 점을 활용해 프로그램을 구현했다. 알파벳에서 바이너리로 변환하는 프로그램이었다면 리스트가 필요했을 것 같다.

z를 예외 처리하는 과정에서 cnt를 1로 초기화해줘야 되는데 그 이유는 나머지 문자들은 1로 끊어지지만 z는 아니기 때문이다. 따라서 cnt를 1로 초기화해 다음 수를 세어야 오류가 나지 않는다.