Object-Oriented Programming Report

Assignment 2-1

Professor	Donggyu Sim
Department	Computer engineering
Student ID	2020202031
Name	Jaehyun Kim

Class (Design / Laboratory)	1 / B
Submission Date	2023. 04. 07

Program 1

❑ 문제 설명

스택의 FILO principle 을 구현하는 문제로, stack 을 배열로 선언합니다. Stack 을 int 형 배열로 선언하고, 현재 stack 의 top 의 위치를 알아야하기 때문에 int 형 변수 topIndex 를 선언했습니다. Main 에서 사용자로부터 exit 을 입력 받을 때까지 무한루프를 돕니다. 루프 내에서 조건문을 통해 입력 받은 명령어가 무엇인지에 따라 다른 함수를 실행시킵니다.

1) push 함수

현재 topIndex 위치에 정수를 추가하고, topIndex 를 1 증가시킵니다.

2) pop 함수

topIndex – 1 인덱스에 가장 최근 추가된 value 가 저장되어 있으므로 topIndex – 1 인덱스의 값을 출력하고 0 으로 초기화해줍니다. topIndex 를 1 감소시킵니다.

3) top 함수

topIndex – 1 인덱스의 값을 출력합니다.

4) print 함수

0 부터 topIndex – 1 까지의 값들을 모두 출력합니다.

5) empty 함수

topIndex 가 0 이면 stack 이 비어있다고 판단하고 1 을 출력, topIndex 가 0 이 아니면 0 을 출력합니다.

※ topIndex 가 0 이면 stack 이 비어 있기 때문에 pop, top, print 함수가 할 수 있는 역할이 없으므로 예외처리를 해주었습니다.

❑ 결과 화면

```
push 3
push -4
push 5
push 7
push 8
print
3 -4 5 7 8
top
8
```

stack 에 3, -4, 5, 7, 8 을 push 합니다.

print 를 실행시킨 결과, push 한 순서대로 value 가 잘 출력됐습니다.

top 을 실행시킨 결과, top 위치에 있는 8 이 잘 출력됐습니다.

```
pop
8
pop
7
pop
5
print
3 -4
empty
0
```

pop 을 3 번 실행하니 stack 에서 8, 7, 5 가 제거됩니다.

print 를 실행시킨 결과, 3, -4, 5, 7, 8 에서 8, 7, 5 가 제거된 3, -4 가 잘 출력됐습니다.

empty 를 실행시킨 결과, stack 에 3, -4 가 남아있으므로 0 이 출력됩니다.

```
pop
-4
pop
3
empty
1
exit
```

pop 을 두 번 실행시키니 stack 에 남아있던 -4, 3 마저 제거되고 더 이상 stack 에 남아있는 value 가 없으므로 empty 를 실행하면 1 이 출력됩니다.

exit 을 실행시키니 프로그램이 종료되는 모습입니다.

❑ 고찰

스택이 비어있거나 가득 차 있을 경우에 대한 예외를 생각했습니다. 스택이 비어있거나 가득 차 있는 상태가 기능에 영향을 미치는 함수는 무엇이 있는지

고민한 결과 스택에 value 를 추가 또는 제거하거나 value 를 출력해야하는 함수들이 영향을 받을 수 있겠다는 생각을 했고 그에 따라 value, pop, top, print 에 예외처리를 추가했습니다.

Program 2

❑ 문제 설명

작성할 맵의 크기를 미리 입력하고 그 크기에 맞게 0 과 1 을 입력합니다. 마지막으로 출발 지점과 도착 지점의 위치를 입력하면 0 이 입력된 경로를 따라 도착지점까지의 최단경로의 길이를 출력하는 문제입니다.

출발 지점으로부터 도착 지점까지의 경로는 단 하나만 존재하기에, 출발 지점으로부터 갈 수 있는 모든 경로를 탐색하여 도착 지점까지의 경로를 찾아내고자 백트래킹을 사용했습니다.

maze 함수에 맵, 현재 위치의 인덱스, 도착지점의 인덱스, 직전의 진행방향, 진행 횟수를 매개변수로 부여했습니다.

출발지점으로부터 뻗어나갈 수 있는 모든 방향으로 진행한 뒤 진행한 후 위치가 유효한지(벽이거나 맵을 벗어나지 않는지) 검사하고, 바로 직전에 진행한 방향을 제외한 세 방향으로 진행합니다. 그렇게 진행하다가 도착지점의 인덱스와 같아지면 경로의 길이를 출력합니다.

❑ 결과 화면

```
4 5
10110
00001
10101
10101
1 2 4 4
6
```

높이 4 너비 5 인 맵을 입력하고 출발 지점 (1, 2), 도착 지점(4, 4)를 입력했더니 경로의 길이 6 이 출력됨을 확인할 수 있습니다.

```
6 7
1011001
0001010
1011000
1011011
1000001
1111111
1 2 6 1
13
```

높이 6 너비 7 인 맵을 입력하고 출발 지점 (1, 2), 도착 지점(6, 1)을 입력했더니 경로의 길이 13 이 출력됨을 확인할 수 있습니다.

❑ 고찰

맵의 크기가 30x30 이내라는 조건이 있어서 30x31 배열을 선언하고 필요한 크기만큼만 사용하려고 했습니다. 하지만 그렇게 되면 유효성검사에 필요한 맵의 높이, 너비까지 매개변수로 넘겨줘야했고, 그렇게 된다면 매개변수의 수가 필요

이상으로 많아지게 됩니다. 따라서 맵의 크기에 맞게 동적할당하여 맵을 벗어나는 인덱스에 대해 보다 적은 매개변수로 판별할 수 있었습니다.

Program 3

❑ 문제 설명

<string.h> 헤더파일에 있는 다양한 함수들을 "oopstd.h" 헤더파일에 직접 구현하는 문제입니다.

1) memset

memset 은 메모리를 바이트 단위로 초기화해주는 함수이므로 void*형 ptr 인자를 unsigned char* 로 캐스팅한 값을 p 에 저장했습니다. p 부터 num 만큼의 바이트 각각을 value 로 초기화해줍니다.

2) memcpy

memcpy 는 메모리를 복사하는 함수로, 1 바이트씩 복사하도록 함수를 작성했습니다. 그 이유는 가장 작은 단위 바이트인 1 바이트씩 복사를 진행하므로써, 몇 바이트던지 데이터의 손실없이 복사를 진행하기 위해서입니다. destination 과 source 를 unsigned char*로 캐스팅해서 source 를 destination 에 1 바이트씩 복사해줬습니다.

3) strcmp

str1 과 str2 의 각 인덱스를 처음부터 비교합니다. 그러다가 두 문자가 달라지는 시점(두 문자열 중 한 문자열이 널 문자에 도달했을 때도 포함)에 크기를 비교해서 그 차이값을 반환합니다. str1 이 더 크면 차이값은 >0 일 것이고 str2 가 더 크면 차이값은 <0 일 것입니다. 하지만 str1 과 str2 의 같으면 0 을 반환합니다.

4) strncmp

strcmp 와 동일한 코드에서 while 문에 idx<num 이라는 조건만 추가해줬습니다. 구현방식은 완전히 동일하지만 num 개의 문자만 비교하는 것이기 때문입니다.

5) strcpy

strcpy 는 dest 배열이 source 배열보다 길다는 가정하에 쓰이는 함수이기 때문에 source 의 문자 하나하나를 dest 에 그대로 복사해줍니다.

6) strncpy

strncpy 역시 strcpy 와 동일한 코드에서 while 문에 idx<num 이라는 조건만 추가해줬습니다. 그러나 strncpy 는 num 이 source 의 길이보다 길면 복사하고 남은 dest 의 공간에 대해서는 null 문자를 채워줘야 하기에 반복문을 통해 구현해줬습니다.

7) strlen

strlen 은 문자열의 길이를 반환하는 함수로, 배열을 0 번째 인덱스부터 차례로 검사하며 null 문자가 나올때까지 인덱스를 늘려갑니다. null 이 나오는 인덱스가 곧 문자열의 길이이기 때문에 인덱스를 반환해줍니다. 주의해야할 점은 size_t 가 unsigned long long 형이기 때문에 idx 를 unsigned long long 으로 선언해줘야합니다.

8) atoi

while 문으로 white space(' ', '₩t', '₩n', '₩v', '₩f', '₩r')를 읽어서 건너뛴 후 바로 연달아 나오는 문자가 부호인지 검사한 후 연달아 나오는 숫자들을 int 형 변수 num 에 저장합니다. 형식에 맞지 않는 문자열이면 0 을 반환합니다.

9) atof

atoi 와 부호검사까지는 동일하게 합니다. 부호 다음 연달아 나오는 숫자를 정수부분을 저장할 int 형 변수 integer 에 저장하고, '.'이 나온다면 연달아 나오는 숫자를 소수부분을 저장할 int 형 변수 demical 에 저장합니다. 또한 정수부분이나

'.' 바로 다음에 'e'나 'E'가 온다면 연달아 나오는 숫자를 지수부분을 저장할 exponent 에 저장합니다.

double 형 변수 num 에 소수부분부터 저장하고 정수부분을 저장한 후 지수부분만큼 10 을 곱하거나 나눠줍니다. (exponent 는 10 의 지수이기 때문)

❑ 결과 화면

```
==========================memset==============================
before memset: almost every programmer should know memset!
after std::memset: ------ every programmer should know memset!
after oopstd::memset: ------ every programmer should know memset!
```

memset(str1, '-', 6); 을 실행시키니 다음과 같은 결과가 나왔습니다. char 형 배열의 str1 의 맨 앞 주소값으로부터 6 바이트까지의 각 바이트에 '-'를 저장한 결과 "almost"자리에 "------"가 잘 저장된 것을 확인할 수 있습니다.

```
==========================memcpy==============================
before memcpy: Hello
after std::memcpy: HelloHiHi
after oopstd::memcpy: HelloHiHi

before memcpy: Pierre de Fermat
after std::memcpy: Pierre de Fermat
after oopstd::memcpy: Pierre de Fermat
```

첫번째로 memcpy(str2 + 5, str3, strlen(str3) + 1); 을 실행시키니 다음과 같은 결과가 나왔습니다. char 형 배열 str2 의 맨 앞 주소값에 5 를 더하면 str2 의 널문자를 가리키게 되고, 널값의 위치에 str3 에 저장된 "HiHi"문자열이 복사된 것을 확인할 수 있습니다. memcpy 세번째 인자로 stren(str3) + 1 을 전달한 이유는 널문자까지 복사하기 위함입니다.

두번째로 memcpy(&copy1, &person, sizeof(person)); 을 실행시키니 다음과 같은 결과가 나왔습니다. 구조체를 대상으로 memcpy 를 진행해도 값의 복사가 잘 진행되는 것을 확인할 수 있었습니다.

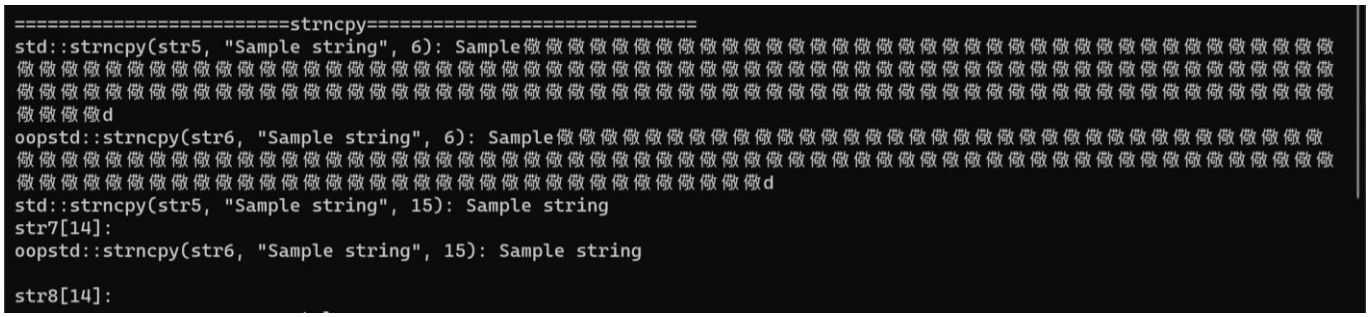
```
=========================strcmp==============================
std::strcmp("orange", "orange"): 0
oopstd::strcmp("orange", "orange"): 0
std::strcmp("orange", "apple"): 1
oopstd::strcmp("orange", "apple"): 14
std::strcmp("orange", "watermelon"): -1
oopstd::strcmp("orange", "watermelon"): -8
```

strcmp 의 경우 같은 문자열이 인자로 전달되면 0 이, 첫번째 인자와 두번째 인자가 달라지는 순간의 문자 크기에 따라 양수 또는 음수가 반환됩니다.

```
=========================strncmp==============================
std::strncmp("apple", "applemango", 5): 0
oopstd::strncmp("apple", "applemango", 5): 0
std::strncmp("apple", "applemango", 8): -1
oopstd::strncmp("apple", "applemango", 8): -109
std::strncmp("apple", "applemango", 3): 0
oopstd::strncmp("apple", "applemango", 3): 0
```

strncmp 의 경우 첫번째 인자와 두번째 인자를 세번째 인자의 크기만큼만 비교한다. 비교 방식은 strcmp 와 같습니다. "apple"과 "applemango"의 5 개 문자는 'a', 'p', 'p', 'l', 'e'로 같으므로 0 이 출력됩니다. "apple"의 8 개 문자는 'a', 'p', 'p', 'l', 'e', '₩0', '₩0', '₩0'이고 "applemango"의 8 개 문자는 'a', 'p', 'p', 'l', 'e', 'm', 'a', 'n'이므로 처음으로 달라지는 부분은 '₩0'과 'm'이 만나는 순간으로 'm'이 더 크기 때문에 음수가 반환됩니다. "apple"과 "applemango"의 3 개 문자는 'a', 'p', 'p'로 같으므로 0 이 출력됩니다.

```
=========================strcpy==============================
std::strcpy(str5, "Sample string"): Sample string
oopstd::strcpy(str6, "Sample string"): Sample string
```



char 형 배열 str5 에 널문자 포함 길이 14 인 문자열"Sample string"을 6 만큼만 복사하므로 Sample 이 복사되는데 null 문자는 복사되지 못하여 쓰레기값이 출력되는 것을 확인할 수 있습니다.

반면 15 만큼 복사하는 경우, 널문자까지 복사하고도 남는 한자리에도 널문자가 삽입되므로 15 번째 인덱스 str8[14]를 출력했을 때 아무것도 출력되지 않음을 확인할 수 있습니다.

```
==========strlen==========
str: HelloMyName
std::strlen(str): 11
oopstd::strlen(str): 11
```

"HelloMyName"은 널문자를 제외하고 길이가 11 인 문자열입니다. 따라서 strlen 을 실행시켰을 때 11 이 반환됩니다.

```
==========atoi==========
arr:     \t\n\v\f\r +-12345
std::atoi(arr): 0
oopstd::atoi(arr): 0

arr:     \t\n\v\f\r -12345abc
std::atoi(arr): -12345
oopstd::atoi(arr): -12345
```

첫번째 case 에서 whitespace 를 제외하면 +-가 나오는데 부호가 두개가 나와서 잘못된 형식이라 판단하고 0 을 반환합니다.

두번째 case 에서는 whitespace 를 제외하고 -부호 하나가 나온 후 숫자가 나오기 때문에 부호와 숫자를 고려해서 반환합니다. 마지막에 나오는 문자 a, b, c 는 방해를 주지 않는 문자들이기에 무시하고 진행이 됩니다.

```
=========================atof==============================
arr:     \t  \n  1234.5987e10
std::atof(arr): 1.2346e+13
oopstd::atof(arr): 1.2346e+13
```

whitespace 를 제외하고 처음 나오는 숫자를 정수부분, '.' 다음 나오는 숫자를 소수부분, 'e' 다음에 나오는 숫자를 지수부분으로 각각 저장한 후 이들을 조합하여 double 형으로 저장한 값을 반환합니다.

$$1234.5987\text{e}10 = 1234.5987 \times 10^{10} = 1.2345987 \times 10^{13} \approx 1.2346 \times 10^{13} = 1.2346e13$$

double 형은 값을 저장하는 방식 때문에 정확한 값을 저장할 수 없습니다. 그러므로 1.2345987 의 근삿값 1.2346 이 저장됩니다.

❑ 고찰

atof 에서 소수부분을 num 에 저장할 때 어떤 방식을 써야하는지에 대한 고민이 있었습니다. 소수부분의 앞자리부터 차례로 넣자니 int 형으로 저장돼있는 소수부분의 앞자리부터 다룰 수 있을 수가 없었습니다. 그래서 %10 으로 소수부분 뒷자리부터 num 에 저장한 후 num 을 10 으로 나눠 num 에 저장된 값들을 소수점 아래로 한칸씩 내리는 방식으로 소수부분을 저장했습니다.

Program 4

❑ 문제 설명

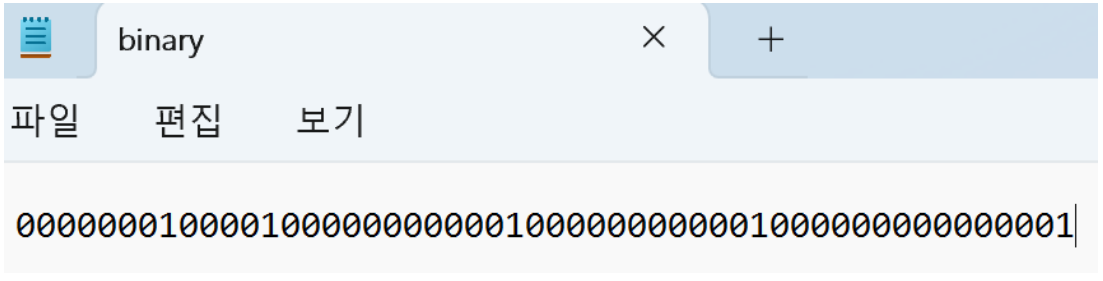
"binary.txt" 에 채워져 있는 0 과 1 을 주어진 표를 참고하여 적절한 알파벳으로 변환한 후 "alphabet.txt"에 출력하는 문제입니다.

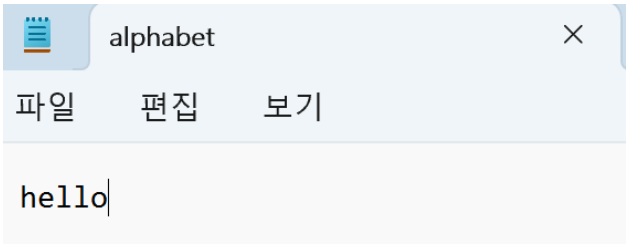
decode class 를 선언하고 멤버변수로 binary.txt 내의 0 과 1 을 저장할 배열 bin, 적절히 변환한 알파벳을 저장할 배열 alp 와 alp 의 인덱스를 나타낼 idx 변수, ifstream, ofstream 객체 fin, fout 을 선언했습니다.

메서드로는 텍스트파일을 열어줄 fileopen 메서드, 텍스트파일의 내용을 읽어와서 알파벳으로 변환 후 alp 에 저장해줄 fileread 메서드, 텍스트를 파일에 출력해줄 filewrite 메서드, 텍스트파일을 닫아줄 fileclose 메서드를 선언해줬습니다.

fileread 메서드에서 bin 배열을 작은 인덱스 순으로 차례로 검사하며 0 의 개수를 세고, 1 이 나오거나 0 의 개수가 25 개가 되면 0 의 개수에 맞는 알파벳으로 변환하여 alp 배열에 저장합니다.

❑ 결과 화면



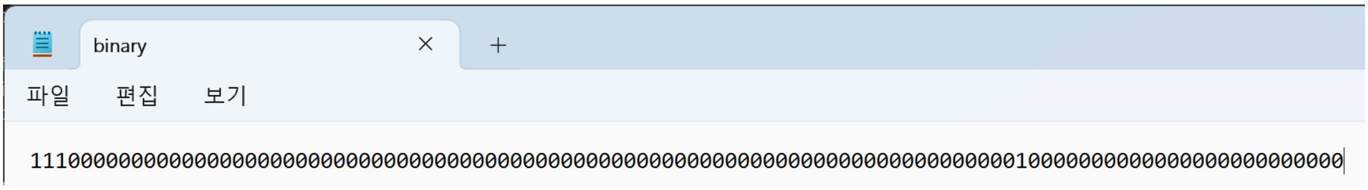


h: 00000001

e: 00001

l: 000000000001

o: 000000000000001



a: 1

z: 0000000000000000000000000

z 가 연속으로 나오거나 마무리가 z 일 때도 문제없이 잘 출력이 되는 모습입니다.

❑ 고찰

getline 메서드로 ‘1’을 기준으로 나눠서 입력 받으려고 시도했으나, ‘z’가 연속으로 나오는 경우를 예외 처리하는 부분에서 어려움을 겪었습니다. 그래서 “binary.txt”의 모든 텍스트를 한번에 입력 받고 배열을 검사하면서 1 을 만나거나 0 이 연속으로 25 개가 붙어있다면 그에 맞는 알바펫으로 변환하고 alp 배열에 저장하는 쪽으로 방향을 바꿨습니다.