



Object-Oriented Programming Report

Assignment 3-1

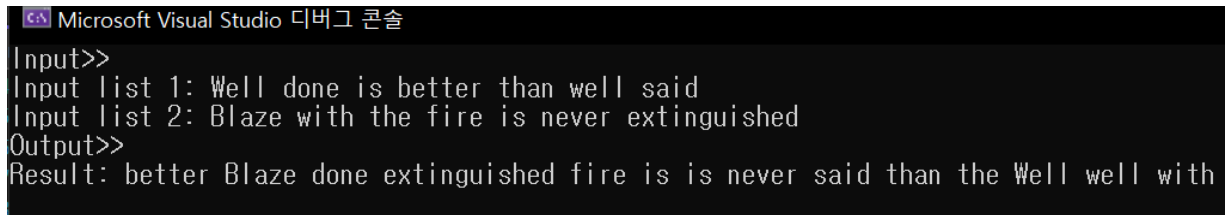
Professor	Donggyu Sim
Department	Computer engineering
Student ID	2022202061
Name	Seoeun Yang
Class (Design / Laboratory)	1 / B (미수강시 0로 표기)
Submission Date	2023. 5. 4

Program 1

□ 문제 설명

두 문장을 입력 받아 알파벳 순으로 정렬한 후 정렬된 두 문장을 다시 알파벳 순으로 정렬하여 한 문장으로 합치는 문제이다. Node 클래스를 선언하여 단어 별로 노드에 저장하고 포인터로 각 노드들을 연결한다. 노드들을 연결할 때 알파벳을 비교하여 중간 삽입을 통해 정렬한다. p1, p2 를 완성하고 나면, p3 노드에 두 문장의 첫 노드부터 비교해 작은 노드를 먼저 저장하며 정렬한다. 모든 과정이 끝나면 p3 에 지금까지 저장한 모든 노드가 있기 때문에 p3 를 통해 노드를 메모리 해제를 해주고 프로그램을 종료한다.

□ 결과 화면



```
Microsoft Visual Studio 디버그 콘솔
Input>>
Input list 1: Well done is better than well said
Input list 2: Blaze with the fire is never extinguished
Output>>
Result: better Blaze done extinguished fire is is never said than the Well well with
```

문제지 예시

□ 고찰

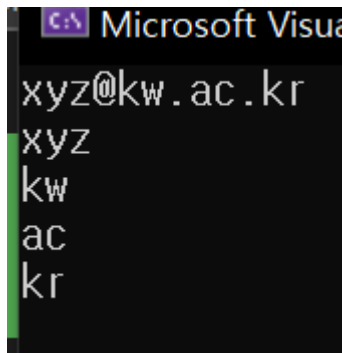
중간 삽입은 한 번도 구현해본 적이 없어서 노드 연결 순서와 포인터 연결에 어려움을 겪었다. 새로운 노드의 포인터를 먼저 연결하고 그 다음에 이전 노드가 새로운 노드를 가리키게 해야 정상적으로 중간 삽입이 이루어짐을 배울 수 있었다. 또한 merge list 를 하고 나면 p1, p2 가 NULL 이 되어야 하는 줄 알았는데 함수 내에서는 NULL 을 가리키지만 main 함수로 반환되면서 p3 에서 자신이 가리켰던 노드를 다시 가리킨다. 노드 자체가 merge 된 것이기 때문에 p3 의 노드들만 메모리를 해제해주면 다 해제가 되어 메모리 누수가 나지 않는다.

Program 2

□ 문제 설명

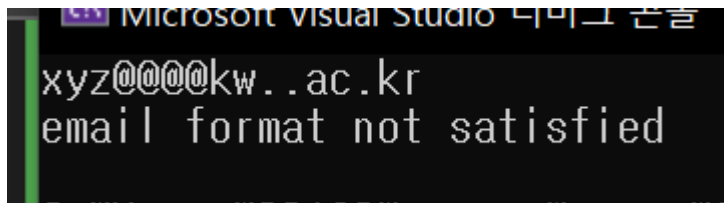
standard library 에 존재하는 strtok() 함수를 구현하는 문제이다. 구분자는 @와 .으로
구분자가 나오면 문자열을 잘라서 그 이전 문자열을 반환한다. 구분자가 연속으로
나올 땐 무시하고 다음 구분자를 찾아 자른다.

□ 결과 화면



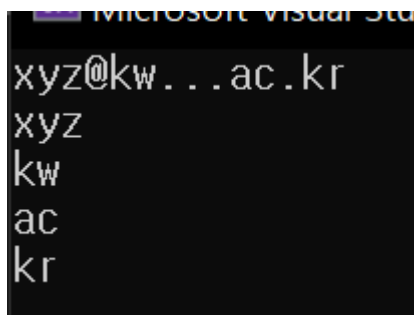
```
xyz@kw.ac.kr
xyz
kw
ac
kr
```

문제지 예시 (구분자 연속 없음)



```
xyz@@@kw..ac.kr
email format not satisfied
```

@ 2 개 이상 (이메일 규약 만족 X)



```
xyz@kw...ac.kr
xyz
kw
ac
kr
```

구분자 연속으로 존재할 때

□ 고찰

static 변수를 사용해본 적이 없어서 static 변수에 대해 이해하는 것이 시작이었다. static 변수는 그 값을 기억하고 있기 때문에 strtok() 함수가 여러 번 호출되어도 처음에 세팅해준 값 그대로 사용 가능하다. 구분자가 여러 개이면 index 와 temp 포인터를 다 한 칸 오른쪽으로 이동시켜 구분자를 무시하고 문자열을 찾도록 구현했다. 이 문제에서는 입력 형식이 이메일 형식으로 한정되어 있기 때문에 @가 2 개 이상이면 바로 프로그램을 종료하도록 코딩했다.

Program 3

□ 문제 설명

링크드 리스트로 queue 를 구현하는 문제이다. queue 는 FIFO (First In First Out) 형식으로 처음 삽입된 노드가 제일 먼저 삭제된다. 노드 삽입을 enqueue, 노드 삭제를 dequeue 라고 한다. rear 은 가장 최근에 삽입된 노드를 가리키고, front 는 제일 먼저 삭제될 노드를 가리킨다.

enqueue 는 첫 노드를 삽입할 때와 아닐 때를 구분한다. 첫 노드를 삽입할 때는 front 와 rear 을 삽입하는 노드를 가리키게 하고, 이어서 삽입하는 경우엔 rear 앞에 노드를 삽입하고 rear 가 삽입된 노드를 가리키게 한다.

dequeue 는 front 가 가리키는 노드를 따로 저장해주고, front 가 이전 노드를 가리키게 한다. 마지막 노드를 삭제할 땐 front 와 rear 을 NULL 로 초기화시켜서 쓰레기값을 방지한다. IsFull(), IsEmpty() 함수를 사용해 예외를 처리해주면 프로그램이 완성된다. 프로그램의 완성도를 높이기 위해 Node 클래스에서 포인터를 2 개 선언해 다음 노드뿐만 아니라 이전 노드도 연결하도록 했다.

□ 결과 화면

```
Please enter maximum size of queue: 3
Please Enter Command(enqueue, dequeue, check_empty, check_full, print, exit): enqueue 2
enqueue success
Please Enter Command(enqueue, dequeue, check_empty, check_full, print, exit): enqueue 4
enqueue success
Please Enter Command(enqueue, dequeue, check_empty, check_full, print, exit): enqueue 8
enqueue success
Please Enter Command(enqueue, dequeue, check_empty, check_full, print, exit): enqueue 3
queue is full
enqueue unsuccessful
Please Enter Command(enqueue, dequeue, check_empty, check_full, print, exit): print
8    4    2
Data count: 3
Please Enter Command(enqueue, dequeue, check_empty, check_full, print, exit): check_full
queue is full
Please Enter Command(enqueue, dequeue, check_empty, check_full, print, exit): check_empty
queue is not empty
Please Enter Command(enqueue, dequeue, check_empty, check_full, print, exit): dequeue
2
Please Enter Command(enqueue, dequeue, check_empty, check_full, print, exit): dequeue
4
Please Enter Command(enqueue, dequeue, check_empty, check_full, print, exit): dequeue
8
Please Enter Command(enqueue, dequeue, check_empty, check_full, print, exit): dequeue
queue underflow
Please Enter Command(enqueue, dequeue, check_empty, check_full, print, exit): print

Data count: 0
Please Enter Command(enqueue, dequeue, check_empty, check_full, print, exit): check_empty
queue is empty
Please Enter Command(enqueue, dequeue, check_empty, check_full, print, exit): check_full
queue is not full
Please Enter Command(enqueue, dequeue, check_empty, check_full, print, exit): exit

C:\Users\W82108\source\repos\OOP_2023_2-1\3-1\Assignment_3\64\Debug\Assignment_3.exe(프로젝트)
```

□ 고찰

dequeue() 함수가 노드를 반환하고 main()함수에서 메모리 해제를 해준다. 마지막 노드를 삭제할 때 front 와 rear 을 NULL 로 초기화해줘야 한다. 마지막 노드를 삭제할 때 size 가 0 으로 바뀌고 dequeue()함수가 종료되기 때문에 main()함수에서 size 가 0 일 경우 front 와 rear 을 리셋을 하도록 구현했다. enqueue 와 dequeue 모두 포인터를 같이 움직여줘야 해서 구현할 때 많이 헷갈렸지만 front 는 삭제, rear 은 삽입으로 생각하고 구현하니 덜 헷갈렸다.

Program 4

□ 문제 설명

링크드 리스트로 stack 을 구현하는 문제이다. stack 은 LIFO (Last In First Out)으로 처음 삽입된 노드가 가장 마지막에 삭제된다. 즉 최근에 삽입된 노드가 제일 먼저 삭제된다. 노드 삽입은 push, 삭제는 pop 이라고 한다. top 은 최근에 삽입된 노드를 가리키고 bottom 은 가장 먼저 삽입된 노드를 가리킨다.

push 는 처음 삽입할 때와 이어 삽입할 때로 나뉘는데 처음 삽입할 땐 top 과 bottom 을 삽입하는 노드를 가리키게 하고, 이어 삽입할 땐 setNext() 함수를 통해 노드를 연결하고 top 만 움직인다.

pop 은 top 을 이전 노드로 움직이고 top 이었던 노드를 삭제한다. 마지막 노드를 삭제할 땐 top 과 bottom 을 NULL 로 리셋해줘야 한다.

□ 결과 화면

```
Microsoft Visual Studio 17.1.2
Please enter maximum size of stack: 3
push 3
push success
push 1
push success
push 8
push success
push 4
stack is full
push unsuccessful
print
3 1 8
check_empty
stack is not empty
check_full
stack is full
pop
8
pop
1
pop
3
pop
check_empty
stack is empty
print
exit
```

□ 고찰

queue 를 구현하고 stack 을 구현하다보니 어렵지는 않았던 것 같다. 다만 수업자료에서는 Node 클래스에 다음 노드를 연결하는 포인터만 있는데 나는 각 노드를 양방향으로 연결했다. pop 을 할 때 이전 노드로 top 을 옮겨야 하기 때문인데 굳이 양방향으로 연결하지 않아도 pop 을 구현할 수 있다는 것을 깨달았다. bottom 부터 getNext()가 NULL 일 때까지 반복문을 돌아 top 을 찾아 삭제하면 되기 때문이다. 정말 코딩은 무수히 달라질 수 있음을 다시 한 번 깨닫는 문제였던 것 같다.