객체지향프로그래밍 보고서

3차과제

학번: 2023202022

이름: 이현지

#1.

총 10개의 노드로 이루어진 circular linked list를 설계한다. circular linked list는 한 방향으로 값만 옮길 수 있고, 노드는 자기자신의 값과 다음 노드로의 포인터로 총 두 개의 영역을 가진다. 사용자는 값을 특정 노드로 옮길 수 있어야 하며, 값 이동중, 10%의 가능성으로 어느 한 연결이 끊어질 수 있다. 이때에는 에러메세지를 띄워야 한다.

연결리스트를 만들기 위해서 Node클래스와 LinkedList클래스를 선언하였다. Node클래스에는 각 노드가 가질 value와 next포인터를 변수로 가지고, 각 변수를 이용할 수 있도록 getter와 setter함수를 만들어주었다. LinkedList클래스에는 연결리스트의 head포인터를 선언하고, 문제에서 요구하는 command를 작동시킬 initialize, transfer, print함수를 만들어주었다.

[initialize함수]

initialize를 command로 받을 때 빈칸을 기준으로 노드들에 넣을 값을 구분한다. find함수로 빈칸이 있는 위치를 확인하여, 문자열이 시작하는 위치부터 빈칸이 있기 전까지의 위치로 문자열을 길이를 파악할 수 있다. 그렇게 얻은 문자열의 길이를 이용해 입력받은 문자열을 원하는 길이로 잘라, 노드가 가져갈 value를 얻는다.

pHead가 빈칸이 아닐 경우, pNew에 얻은 value를 넣어 새로운 노드를 만들어, 현재까지 만들어진 연결리스트의 가장 끝의 노드의 다음에 위치하도록 한다. 또한 circulate linkedlist이므로 노드10개가 만들어지면, 연결리스트의 머리와 꼬리부분이 연결되도록 한다.

[transfer함수]

transfer에는 10개의 연결다리 중 1개가 끊어지도록 프로그램을 설계해야 한다. 랜덤성을 띠고 있기 때문에, srand와 rand함수를 사용하여 0~9까지의 숫자를 뽑도록 한다. 나온 숫자만큼 for문을 돌려서 나온 노드의 next포인터가 nullptr을 가리키도록 조정한다.

transfer도 숫자 2개를 입력받기 때문에 initialize에서 사용했던 문자열 구분 알고리즘을 사용하여 찾아야 하는 노드의 위치를 구하도록 한다. 그렇게 구해진 숫자는 linkedlist를 탐색하여 해당 위치에 있는 노드를 찾고, 그 value는 원하는 노드의 위치로 이동시켜준다. 이때 disconnected 부분이 있다면 error메시지를 띄운다.

[print함수]

print에서는 탐색을 위한 pHead를 curNode로 선언하여, 연결리스트 처음부터 끝까지 출력시킬 수 있도록 한다.

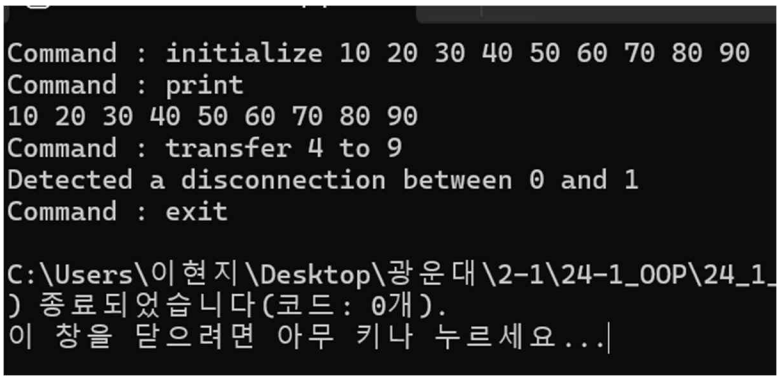
[main함수]

linkedlist클래스를 list로 선언하여, 각 command에 클래스의 올바른 멤버함수로 이동시켜 주도록 한다. 이때도 initialize함수와 동일하게, command단어만을 추출하여 strcmp함수로 문자열이 일치한지 확인하여 올바르게 작동할 수 있도록 한다.

[LinkedList소멸자]

linkedlist에서는 각 노드들을 동적할당하였기 때문에 모두 메모리해제를 해야 한다. print함수에서 사용했던 모든 노드를 훑어보는 알고리즘을 사용해 delete시킨다. delete하기 전에 다음 노드로 이동해야 하므로 tmp라는 임시 노드를 이용해 delete시킬 노드를 옮겨두고, 다음 노드로 이동한 후 메모리해제시킨다.

[출력화면]



[고찰]

연결리스트를 설계할 때, 가장 처음 클래스를 작동하면 node에 따로 동적할당을 하지 않으면 nullptr로 반환된다는 것을 명심하게 되었다. getter와 setter를 사용할 때, getNext함수에서 this가 nullptr일 경우에는 항상 오류를 반환한다. 따라서, while문을 이용하면서 getter를 쓸 때는 this가 nullptr이 되기 전, this->getNext() != nullptr로 직전에 미리 확인하여 this가 nullptr인채로 Node클래스로 넘어가지 않도록 해야 한다는 것을 알았다.

#2.

2차원 매트릭스 클래스를 설계하는 문제이다. 매트릭스는 4x4로 받는다고 설정한다. 연산자 오버로딩에 관한 문제로, 두 매트릭스의 동일한 위치의 성분에 대해 사칙연산을 진행할 수 있도록 재정의하면 된다.

Matrix클래스에는 data를 받도록 double 배열을 선언하고, 연산자에 대해 재정의하기 위

해 7개의 함수를 설계했다. +연산자에 대해 매개변수를 매트릭스로 받는지, scalar값을 받는지에 따라 연산이 달라진다. -와 *연산자도 동일하다. 우선, 매트릭스로 받을 때는 최종 결과값을 저장할 tmp Matrix클래스를 선언해주고, 이중 반복문을 사용하여 0,0부터 3,3까지의 성분 각각을 연산한다. return값으로는 결과를 저장한 tmp클래스로 한다. -와 *연산자는 +에 -와*로 변경되었다는 것만 다르다. scalar값을 매개변수로 받을 때는 /연산자도 추가된다. 이전과 동일하게 tmp선언하고, 이중반복문을 이용하여 원하는 클래스의 각 성분을 받아, 연산자에 따라 원하는 scalar로 연산할 수 있도록 한다. 반환값은 이전과 동일하게 tmp클래스로 한다.

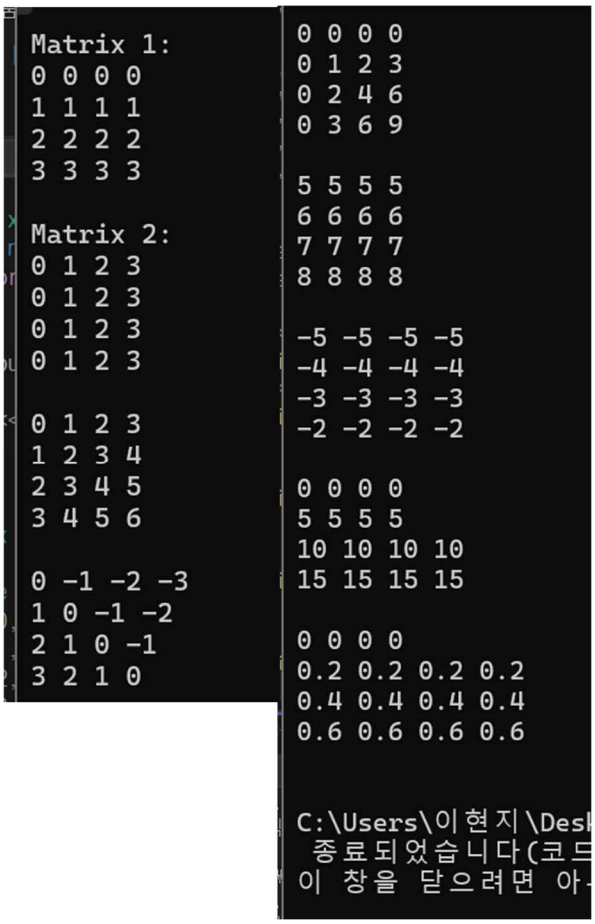
[setValue함수]

setValue함수는 외부에서 받은 Matrix성분 값을 각 클래스의 data배열에 저장하도록 한다. 이중반복문을 이용하여 data배열에 접근하여 arr값을 저장하면 된다.

[display함수]

클래스에 저장된 매트릭스의 data를 출력하도록 하는 함수이다. 이중반복문을 이용하여 data의 각 성분에 접근하여 출력시킨다.

[출력화면]



[고찰]

연산자 오버로딩에 대해서 알 수 있었다. 매트릭스를 매개변수로 받는 연산자 재정의에는 매개변수가 2개가 필요하다고 생각했었다. 그러나 강의자료에는 매개변수가 하나만 있어서 어떻게 처리하나 했더니, this포인터를 이용해 연산자 기준 왼쪽 매트릭스를 다룰 수 있다는 것을 보였다.

#3.

로그인 프로그램을 만드는 문제이다. 파일에 저장되는 회원들의 정보를 이용해서 로그인할 수 있도록 한다. 해당 프로그램을 닫고 재오픈했을 때도 정보가 사라지지 않아야 한다.

--프로그램 룰—

4개 옵션(login, register, withdraw, exit)

로그인은 아이디와 패스워드가 파일의 내용과 일치할 때 성공이다. 멤버십가입 시 ID는 겹치면 안되고, 패스워드는 10~20자로 알파벳, 숫자, 특수문자는 적어도 하나씩 포함되어야 한다. withdraw는 회원정보를 지우는 역할이다. 멤버십 정보는 구조체로 다룬다. 파일에 저장되는 회원의 패스워드는 caesar로 암호화한다. 알파벳의 경우 3칸 미뤄서 저장한다.(ex. a->d, z->c)

[Caesar함수]

a부터 w까지는 문자에 바로 +3을 해도 아스키코드에서 알파벳의 범주를 벗어나지 않는다. x부터 z는 +3을 했을 경우 알파벳의 범주를 벗어나므로 -23을 하여 규칙에 맞게 암호화해준다.

[Decipher함수]

부호화함수로 Caesar함수에서 해놓은 장치를 원래대로 돌려놓는 것이 목표이다.

사용자가 1을 입력했을 시, login알고리즘을 작동시킨다. 우선 회원정보가 저장되어있을 membership.txt파일을 열어 한줄씩 읽어오면서, 사용자가 입력한 id와 일치한 정보가 있는지 확인한다. 없을 경우, 로그인 실패를 띄우고 다시 command입력창을 띄우고, 일치할 경우, 로그인 성공을 띄운다. 이때 id는 상관없으나 password는 파일에 저장된 회원정보가 암호화되어있으므로 decipher함수를 이용해 부호화한 후 확인해주도록 한다.

사용자가 2를 입력했을 경우, register알고리즘을 작동시킨다. 사용자가 입력한 id가 membership파일에 있는지 확인하고, 있을 경우 register실패를 띄우고 없을 경우 password를 Caesar함수를 이용하여 암호화한 후 id와 함께 파일에 저장한다.

3을 입력했을 때는 withdraw알고리즘을 작동시킨다. 이것이 활성화될 때는 이미 회원 로그인이 되어있어야 하므로 bool 변수로 login이 되어있는지 확인 후 작동시킨다. 로그인이 되어있을 때 로그인되어있는 id와 password를 파일에서 삭제해야한다. 이때 tmp.txt라는 임시 파일을 만들어서, 삭제해야하는 id와 password만을 뺀 나머지 정보를 임시로 저장한다. 이후 membership파일에는 내용을 초기화하고 tmp에 있는 내용을 새로 작성하도록 한다.

[출력화면]

```
==================
Menu.
1. Login
2. Register
3. Withdrawal
4. Exit
 : 1
----------------------
==================
User id: abcdefg
password: abcdefg123!
Login failed. Invalid User id or password.
Menu.
1. Login
2. Register
3. Withdrawal
4. Exit
 : 2
----------------------
----------------------
==================
User id: abcdefg
password: abcdefg123!
Menu.
1. Login
2. Register
3. Withdrawal
4. Exit
 : 1
----------------------
----------------------
==================
User id: abcdefg
password: abcdefg123!
Login successful.
Login failed. Invalid User id or password.
Menu.
1. Login
2. Register
3. Withdrawal
4. Exit
 : 3
----------------------
```

```
----------------------
----------------------
==================
Logged in user : (abcdefg)
Menu.
1. Login
2. Register
3. Withdrawal
4. Exit
 : 1
----------------------
----------------------
==================
User id: abcdefg
password: abcdefg123!
Login failed. Invalid User id or password.
Menu.
1. Login
2. Register
3. Withdrawal
4. Exit
 : 4
----------------------
----------------------
==================

C:\Users\이현지\Desktop\광운대\2-1\24-1_OOP\
3\x64\Debug\3-3.exe(프로세스 42992개)이(가)
이 창을 닫으려면 아무 키나 누르세요...
```

[고찰]

파일에 입력되어있는 내용을 프로그램에서 효율적으로 삭제할 수 있는 방법을 찾지 못해서 tmp라는 임시 파일을 이용해 불완전한 정보 삭제를 진행하였다. 시간이 부족해 이 부분을 구현하지 못한게 아쉽다.

#4.

각 지역의 시간을 관리할 수 있는 클래스를 설계하는 것이 목표이다. 한국의 지역시간을 받아오고, 다른 지역은 한국과의 시차를 이용해 해당 지역시간을 설정하도록 한다. 또한 add [second]를 입력 받으면 second후의 시간을 set하도록 한다.

Time클래스에는 hour, minute, second를 다룰 수 있도록 변수를 선언하고, 각 지역의 시간을 설정할 수 있는 setTime, 원하는 second후의 시간을 설정하는 addTime, 각 지역의 시간을 출력하는 printTime 함수들을 선언했다.

각 지역에 따라 클래스를 사용했다. 각 지역의 생성자는 Time class로 초기화 시키고, Korea는 매개변수로 받은 현재시간을 setting하도록 한다. 다른 지역들도 현재시간으로 초기화시킨다.

[setTIime함수]

클래스 내의 변수에 매개변수의 값을 저장하도록 한다.

[addTime함수]

second는 60sec = 1m, 3600sec = 1h이므로 조건문으로 second의 값을 확인하는 기준은 3600과 60이다. 3600초가 넘을 경우, second를 3600으로 나눈 몫은 추가되어야 할 시간, 나머지는 60으로 다시 나누어서 나온 몫은 추가되어야 할 분, 나머지는 추가되어야 할 초로 계산하면 된다. 3600초는 넘지 않지만 60초가 넘을 경우에는 60을 나누어서 나온 몫과 나머지로 추가되어야 할 분과 초를 구한다. 60초도 넘지 않을 경우에는 초만 추가해주어 시간을 계산한다. 이때, 분과 초는 최대 59이므로, 각각 다음 구역으로 넘어가는지 확인도 필요하다. 시는 24가 넘어갈 경우 hour-24로 계산해주어야 한다.

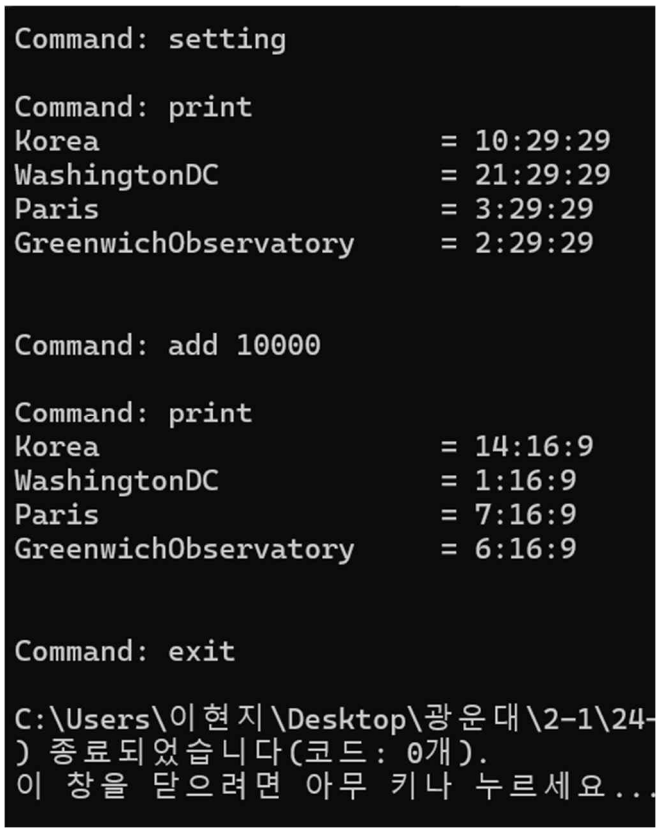
[각 지역의 클래스]

워싱턴보다 한국이 13시간 빠르고, 파리보다 7시간 빠르고 그리니치보다 8시간 빠르다. 이를 이용하여, setting 커맨드가 불렸을 때, 각 시차만큼 뺀 매개변수를 보내서 연산하도록 한다.

[메인함수]

ctime라이브러리를 이용해서 현재시각을 구하고 hour, min, sec를 각각 저장해준다. command를 입력 받기 전에 각 지역의 클래스를 부르고 초기화시킨다. setting이 입력되었을 때, setTime함수를 사용하여 시간을 설정한다. add를 작동시키기 위해 입력 받은 add값을 구하기 위해 원하는 만큼 잘라서, string으로 설정된 second를 stoi함수를 이용하여 int형태로 바꿔서 addTime함수를 실행시킨다.

[출력화면]



[고찰]

string 라이브러리에 존재하는 여러가지 매서드를 처음 써봤는데, 생각보다 과정이 번거로웠다. 이전까지는 string라이브러리만 있으면 모든 문제를 풀 수 있을 거라 생각했었으나, find함수가 포인터를 이용하여 탐색하는 함수이다보니, 커서를 직접 지정하고 문자열의 길이 또한 직접 구해야 한다는 것을 알게 되었다.

#5.

다항식을 연결리스트로 구현하여, 각종 연산을 진행하는 알고리즘을 설계하는 것이다. 각 term에는 계수와 차수가 들어가고, class overloading으로 연산자를 재정의한다. 또한 미분값과, x에 특정값을 대입했을 때의 결과값을 도출해내는 함수도 구현해야한다.

[Term 클래스]

단항식의 계수와 차수, next포인터를 변수로 하고, private으로 선언되어있으므로, getter와 setter를 이용해서 클래스 바깥에서 사용할 수 있도록 한다.

[Polynomial클래스]

연결리스트를 구현하는 것과 거의 동일하게 하면 된다. 연산자를 재정의하는 과정이 필요하다. 다항식 계산이므로 차수를 살피고, 동일한 차수가 있을 경우, 계수끼리 연산하여, 하나의 term으로 합치는 과정이 필요하다. 만약 동일한 차수가 없다면, 각각 다른 term에 저장해야 한다. 이를 설계하기 위해, 연산자 기준 왼쪽 term과 오른쪽 term으로 나누어서 각각의 차수를 직접 비교하면서 연산하도록 설계했다. 비교할 때, 왼쪽 term을 기준으로 돌렸기 때문에, 오른쪽 term에서 왼쪽 term과 차수가 일치하는 term이 없다면 누락되기 때문에, 한 번 더 검사가 필요하다.

[differentiation]

미분과정은 계수와 차수를 곱하고, 차수 = 차수 – 1을 한 다항식이 끝날 때까지 반복하면 된다.

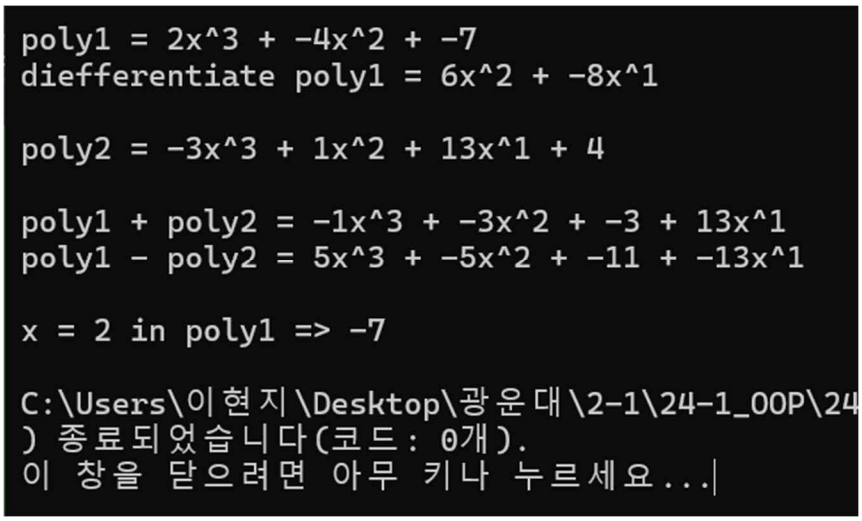
[addTerm 함수]

curTerm에 m_pHead를 받아 연결리스트를 돌아다닐 포인터로 설정하고, 노드의 가장 끝자락에 term을 추가하도록 한다.

[calculate 함수]

term에 있는 차수만큼 power연산을 통해 다항식에 x를 대입한 값을 계산하도록 한다.

[출력화면]



[고찰]

연결리스트를 설계하면서 가장 헷갈리는 부분은 nullptr을 어떻게 잘 피해가는 건지 였다. 항상 시작할 때는 노드가 비어있는지 확인하는 습관을 들여야하고, 탐색을 위한 노드를 설정했을 때는, 반복문 안에 노드를 다음 노드로 옮기는 습관도 필수라는 것을 깨닫게 되었다.

#6.

무궁화 꽃이 피었습니다를 구현해야 한다. tagger는 랜덤한 시간 주기로 Red Light와 Green Light를 반복하여 말한다. 플래이어는 Green Light일 때 오른쪽 방향키를 눌러 tagger에 다가가도록한다. 총 20번 다가가야하도록 만들어야한다.

tagger가 front state에 들어가면 Red Light를 외치고 2~10초동안 그 상태 유지 후 Back state로 바꾸도록 한다.

플레이어의 progress현황은 @로 알 수 있는데, tagger에 다가가는 것에 성공할 때마다 @가 차오르도록 설계해야한다.

만약 tagger가 front state일 때 player가 움직이면 lose이다.

[출력화면]

[고찰]

우선 미완성이다. 사용자가 오른쪽 방향키를 누르지 않았음에도 progress값은 진행되고, tagger의 상태 변화가 progress 옆쪽에서 나타난다. 여유를 가지고 작업했으면 성공했을 것 같아서 아쉬움이 남는다.

#7.

stack자료구조를 연결리스트를 이용하여 구현해야 한다. push, pop과 같은 매소드도 실행되도록 설계해야 한다. 이때 어떤 자료형이든 받을 수 있도록 templates을 사용하여라.

stack은 먼저 들어간 것이 가장 나중에 빠지는 구조를 가지고 있다. 따라서, m_Top은 노드가 추가될 때마다, 가장 끝 자락에 이어지는 구조가 되어야 한다.

노드 클래스는 다른 연결리스트와 동일하게 노드의 데이터와 next를 다루고, setter와 getter를 가지고 있다.

Stack 클래스는 m_Top을 변수로 가지고 있어, 연결리스트의 시작을 알 수 있도록 한다. stack을 사용할 수 있도록 push함수와 pop함수의 알고리즘을 작성해야한다.

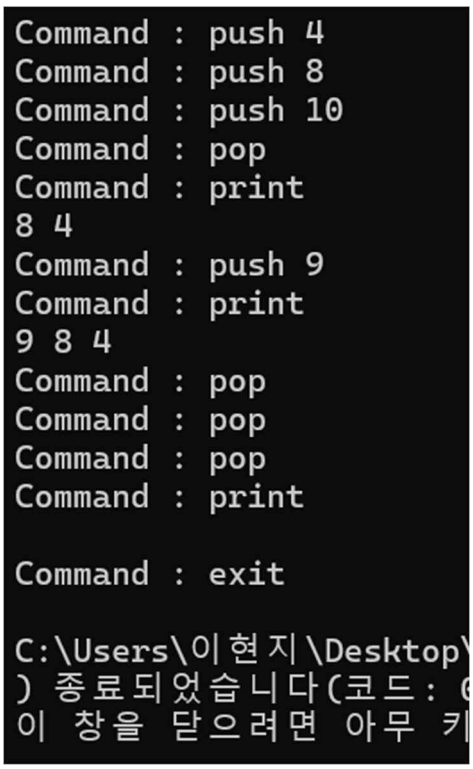
[push함수]

연결리스트에 노드를 추가하는 함수로, node는 가장 끝자락에 추가되고 m_Top이 항상 마지막에 위치한도록 조정한다,

[pop함수]

stack이 비어 있지 않다면, m_Top이 가리키고 있는 노드를 delete시키고 m_Top은 그 전 노드를 가리키도록 해야 한다.

[출력화면]



[고찰]

연결리스트를 이용한 자료구조가 다양하다는 것을 알 수 있었다. stack자료구조는 연결리스트에 비해, 자료를 다루는 것이 일회성인 느낌이 강해 간단한 자료를 정리하기에는 효

율적일 것 같다는 생각이 들었다.

#8.

queue자료구조를 연결리스트를 사용하여 만들어야 한다. 이전과 동일하게 자료형에 구애받지 않도록 template을 사용하도록 한다.

노드는 이전과 동일하다.

Queue클래스는 front와 back이 필요하다. 이전과는 다르게 head와 tail을 다루기 때문이다.

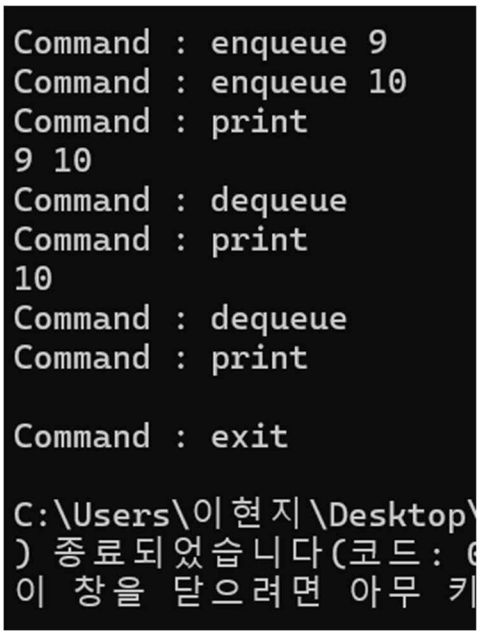
[enqueue함수]

m_Front가 비어있다면 m_Back에도 비어있을 테니, newNode로 채워준다. 비어있지 않을 경우에는, m_Back가 newNode의 값을 받도록 한다.

[dequeue함수]

이 자료구조는 m_Front부터 삭제하므로, stack과는 delete순서를 정반대로 설계하면 된다.

[출력화면]



[고찰]

또 다른 자료구조를 배울 수 있었다. 이전에 필요에 의해서 연결리스트의 private변수에 head와 tail을 선언해서 자료를 다룰까 생각했었는데, 해당 자료구조가 따로 있다는 것을

알게 되었다.

#9.

Binary Search Tree인 자료구조를 설계하는 문제이다. root node를 배열의 가운데 위치로 두고, root node를 기준으로 왼쪽 오른쪽 배열로 나눈다. 이 과정을 각 배열의 크기가 1이 될 때까지 재귀적으로 반복하면 된다. 이때, 각 요소가 배열에서 몇 번 나누어졌는지를 depth로 체크하여, 이를 print에 표시하도록 한다.

build함수는 mid를 설정하여 mid를 기준으로 배열을 나누어 역할을 한다.

[출력화면]