REPORT

광운대학교
KwangWoon University

ASSIGNMENT3

담당교수: 신영주 교수님

학과: 컴퓨터정보공학과

학번: 2017202037

이름: 오민혁

Introduction

객체지향 프로그래밍 언어인 C++의 특성을 이용하여, CLASS를 이용하여 링크드 리스트를 만들어 CAFÉ의 메뉴판을 관리하는 시스템을 만드는 과제이다. 단, 중복된 메뉴가 있으면 안된다. 그리고 “Input.txt” 파일 안에 카테고리와 메뉴의 가격, 메뉴의 이름 정보가 들어있다. 이 정보들을 가지고 LOAD를 통해 메뉴판을 생성해야 한다. 단, 메뉴의 링크드 리스트에 삽입할 때는 가격의 오름차순으로 해야한다. 만약 가격이 같다면 메뉴 이름의 알파벳 오름차순으로 한다.

Algorithm

<**MENU NODE & CATEGORY NODE & LIST CLASS**>

CATEGORY NODE 안에 MENU NODE 가 속해 있는 형식으로 구현하였다. MENU NODE CLASS를 생성하고 PUBLIC MEMBER로 MENU의 이름, 가격 그리고 nextNode를 지정해주었다. 그 다음 CATEGORY NODE CLASS를 생성하고 PUBLIC MEMBER로 CATEGORY의 이름, 그리고 MENU NODE, nextNode를 지정해주었다. 그 다음 LIST 클래스를 생성하여 PRIVATE MEMBER로 CATEGORYNODE * HEAD를 지정해주었다. CATEGORY NODE가 MENU NODE보다 큰 범주임으로 따로 MENU NODE를 설정해 줄 필요가 없다고 생각하였다.

<**LOAD**>

우선 FILE STREAM을 이용해 “input.txt” 파일을 열었다. 그 다음 getline을 이용하여 텍스트 파일에 있는 정보들을 한 줄 씩 버퍼에 받아왔다. 그 다음 탭과 띄어쓰기를 기준으로 STRING TOKENIZING을 하여 카테고리, 메뉴, 가격 순서대로 각 변수에 저장하였다. 그 다음 구현해놓은 INSERT를 이용하여 링크드리스트를 생성하였다.

<**INSERT**>

함수의 인자로 카테고리, 메뉴, 가격을 설정하였다. 그 다음 카테고리가 만약 링크드리스트가 없으면 카테고리를 생성해주고, 있다면 해당 카테고리의 메뉴 노드에 덧붙여 주는 식으로 구현하였다. 이 노드를 추가해주는 방식은 가격이 크고 작음에 따라 삽입할 위치를 찾아서 노드를 삽입하는 Insertion Sort를 사용하였다. 순서는 가격의 오름차순이며, 만약 가격이 같을 시, 메뉴의 알파벳 오름차순이다. 만약 이 과정에서 인자로 받은 메뉴 이름이 이미 존재하는 메뉴이면 삽입을 하지 않는다.

<**DELETE**>

카테고리와 메뉴 링크드 리스트를 처음부터 끝까지 탐색하여, 인자로 받은 메뉴 이름과 일치하는 곳을 찾는다. 그리고 삭제할 노드의 부모 노드의 다음 노드를 찾은 노드의 다음 노드로 설정한다. 만약 일치하는 메뉴 이름이 존재하지 않는다면 함수를 종료시키는 예외처리를 하였다. 만약 메뉴 노드가 하나밖에 없는 카테고리에서 메뉴 노드를 삭제하면 카테고리 까지 삭제한다.

<**CATEGORY SEARCH**>

카테고리 이름을 인자로 받아서, 카테고리 링크드리스트의 헤드부터 끝까지 인자로 받은 이름과 같은 이름이 있는지 검사한다. 만약 같은 것이 있다면 해당 카테고리의 이름과 메뉴 목록을 모두 출력한다.

<**MENU SEARCH**>

메뉴 이름을 인자로 받아서, 카테고리 링크드리스트의 헤드부터 끝까지 그리고 해당 카테고리의 메뉴 링크드리스트의 처음부터 끝까지 검색하여 인자로 받은 이름과 같은 메뉴가 있는지 검사한다. 만약 같은 것이 있다면 해당 메뉴의 이름과 가격을 출력한다.

결과화면

```
1. LOAD
2. PRINT
3. INSERT
4. Category SEARCH
5. Menu SEARCH
6. DELETE
7. End of Program!
Select Number: 1

StrawberrySmoothie Invalid Input

ChocoSmoothie already exists!
LOAD Success!
1. LOAD
2. PRINT
3. INSERT
4. Category SEARCH
5. Menu SEARCH
6. DELETE
7. End of Program!
Select Number: 2

==========COFFEE==========

Espresso 4000
Americano 4200
CaramelMacchiato 4900

==========SMOOTHIE==========

ChocoSmoothie 4800
YogurtSmoothie 4900
MangoSmoothie 5000
GreenteaSmoothie 5400

==========TEA==========

EarlgrayBlackTea 3500
JasmineTea 4700
GrapefruitTea 4800
LimeTea 5600

==========ADE==========

GrapefruitAde 5500
LemonAde 5500
OrangeAde 5500
StrawberryAde 5700
```

```
Enter Category, Menu and Price:
dessert arownie 2500
Insert Success!
1. LOAD
2. PRINT
3. INSERT
4. Category SEARCH
5. Menu SEARCH
6. DELETE
7. End of Program!
Select Number: 3

Enter Category, Menu and Price:
dessert crownie 2500
Insert Success!

==========DESSERT==========

arownie 2500
Brownie 2500
crownie 2500
Macaroon 2500
Bagel 3000
HoneyBread 5000
```

```
Select Number: 6

Enter Menu which you want to delete :arownie
Delete Success!
1. LOAD
2. PRINT
3. INSERT
4. Category SEARCH
5. Menu SEARCH
6. DELETE
7. End of Program!
Select Number: 2

==========COFFEE==========

Espresso 4000
Americano 4200
CaramelMacchiato 4900

==========SMOOTHIE==========

ChocoSmoothie 4800
YogurtSmoothie 4900
MangoSmoothie 5000
GreenteaSmoothie 5400

==========TEA==========

EarlgrayBlackTea 3500
JasmineTea 4700
GrapefruitTea 4800
LimeTea 5600

==========ADE==========

GrapefruitAde 5500
LemonAde 5500
OrangeAde 5500
StrawberryAde 5700

==========DESSERT==========

Brownie 2500
crownie 2500
Macaroon 2500
Bagel 3000
HoneyBread 5000
```

```
1. LOAD
2. PRINT
3. INSERT
4. Category SEARCH
5. Menu SEARCH
6. DELETE
7. End of Program!
Select Number: 4

Enter Category which you want to find :crownie
crownie is category that do not exist

1. LOAD
2. PRINT
3. INSERT
4. Category SEARCH
5. Menu SEARCH
6. DELETE
7. End of Program!
Select Number: 4

Enter Category which you want to find :dessert
DESSERT
Brownie 2500
crownie 2500
Macaroon 2500
Bagel 3000
HoneyBread 5000

1. LOAD
2. PRINT
3. INSERT
4. Category SEARCH
5. Menu SEARCH
6. DELETE
7. End of Program!
Select Number: 5

Enter Menu which you want to find :crownie
crownie 2500
```

위의 결과화면들로 보아 1번부터 6번까지의 기능이 정상적으로 수행되고 있음을 알 수 있다.

고찰

싱글 링크드 리스트를 이용하여 카페의 메뉴판을 구현해보았다. 배열과 비교해 보았을 때, 링크드리스트는 각 요소에 대한 직접적인 접근이 어렵지만, 배열은 직접적인 접근이 가능하여 Insert나 Delete, Search가 더 간단하다. 하지만 이번 과제처럼 어떤 자료에 대해 부가적인 요소들이 존재할 때, 배열로 구현하는 것은 정말 복잡하다. 배열은 한 저장 공간에 한 종류의 데이터밖에 저장하지 못하지만, 링크드 리스트는 노드의 멤버 변수를 설정하여 여러 종류의 데이터를 담을 수 있다. 이러한 점을 이용하면, 단순 카페 메뉴판이 아니라 훨씬 복잡한 자료들의 저장 공간도 링크드리스트로 구현할 수 있을거라는 생각이 든다.

<2>

Introduction

Circular Doubly Linked List를 이용하여 강의명, 교수님 성함, 학년의 정보를 관리하는 강의 목록 관리 시스템을 구현하는 과제이다. 이 시스템의 기능으로는 맨 앞에 강의 정보 삽입, 마지막에 강의 정보 삽입, 원하는 위치에 강의 정보 삽입, 강의 삭제, 강의 정보 업데이트, 강의명으로 검색, 교수님 성함으로 검색, 학년으로 검색, 학년 순으로 강의 오름차순 정렬, 강의 목록 보여주기, 강의 목록 거꾸로 보여주기가 있다.

Algorithm

<**NODE AND LIST**>

NODE CLASS를 생성한다. 이 NODE CLASS의 PUBLIC MEMBER는 next, prev, lecture name, professor name, grade가 있다. Single이 아닌 double

linked list 이므로 next와 prev 둘 다 필요하다. 그리고 LIST CLASS를 생성하고 이 CLASS PRIVATE MEMBER로 head, tail, count를 설정해준다. Head는 말 그대로 linked list의 맨 앞을 가리키며 tail은 말 그대로 꼬리를 가리킨다.

<**INSERT**>

-at beginning

LIST는 CIRCULAR DOUBLY LINKED LIST 이므로 맨 앞에 노드를 삽입하기 위해서는 HEAD의 PREV에 삽입 할 노드를 지정해주고, 삽입할 노드의 NEXT에 HEAD를 지정해준다. 그 후 HEAD를 삽입 할 노드로 옮긴다.

-at last

at beginning에서 사용한 알고리즘과 비슷한데, 삽입할 노드가 tail에 위치해야 하기 때문에, tail의 next에 삽입할 노드를 지정해주고, 삽입할 노드의 prev에 tail을 지정해준다. 그 후 tail을 삽입할 노드로 지정해준다.

-at position

Private member의 count를 이용한다. 반복문을 이용하여 node=node->next를 count 만큼 반복한다. 그 다음 node의 next에 삽입할 노드를 지정해주고 삽입할 노드의 prev를 node로 지정해준다. 그리고 삽입할 노드의 next를 node의 next로 지정해준다. 그 다음 node의 next의 prev에 삽입할 노드를 지정해준다. 만약 이미 존재하는 강의명이라면 중복 예외 처리를 해준다.

<**DELETE**>

삭제할 강의명을 인자로 받는다. 그리고 LINKED LIST의 HEAD부터 TAIL까지 같은 강의명을 가진 NODE를 찾는다. 이 노드가 DELETE NODE가 된다. 그리고 DELETE NODE의 PREV의 NEXT를 DELETE NODE의 NEXT로 지정해주고, DELETE NODE의 NEXT의 PREV를

DELETE NODE의 PREV로 지정해준다. 그 후 노드를 삭제해준다.

<**UPDATE**>

수정 할 강의명을 인자로 받는다. 그리고 LINKED LIST의 HEAD부터 TAIL까지 같은 강의명을 가진 NODE를 찾는다. 그 후 이 노드의 멤버 변수들인 강의명, 교수님 성함, 학년을 입력 받고 저장한다.

<**SEARCH**>

-lecture

검색 할 강의명을 인자로 받는다. 그리고 LINKED LIST의 HEAD부터 TAIL까지 같은 강의명을 가진 NODE를 찾는다. 그리고 이 NODE의 강의명, 교수님 성함, 학년을 출력한다.

-professor

검색할 교수님 성함을 인자로 받는다. 그리고 LINKED LIST의 HEAD부터 TAIL까지 같은 강의명을 가진 NODE를 찾는다. 그리고 이 NODE의 강의명, 교수님 성함, 학년을 출력한다. Lecture와 다르게 교수님 성함이 같은 노드가 여러 개 있을 수 있으므로 이 과정을 교수님 성함이 같은 노드가 없을 때 까지 반복한다.

-grade

검색 할 학년을 인자로 받는다. 그리고 LINKED LIST의 HEAD부터 TAIL까지 같은 강의명을 가진 NODE를 찾는다. 그리고 이 NODE의 강의명, 교수님 성함, 학년을 출력한다. Lecture와 다르게 학년이 같은 노드가 여러 개 있을 수 있으므로 이 과정을 학년이 같은 노드가 없을 때 까지 반복한다.

<**SORT**>

Bubble Sort를 이용한다. LINKED LIST의 HEAD부터 TAIL까지 탐색하는 두 개의 NODE를 설정하고, 학년이 더 큰 노드를 계속해서 학년이 더 작은 노드와 변경하는 작업을 반복한다.

<**DISPLAY**>

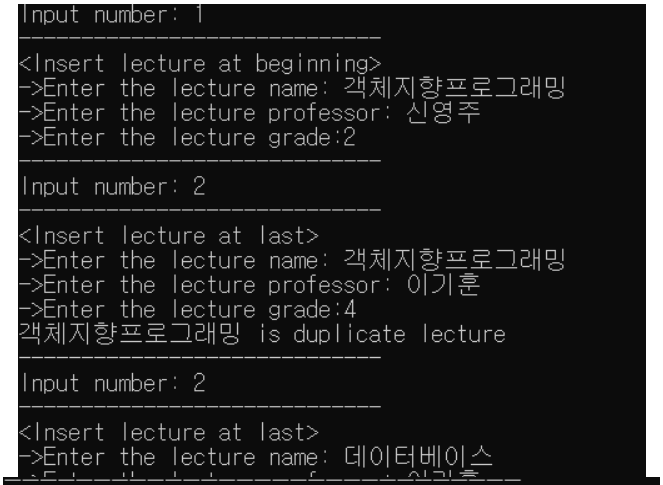
-Not Reverse

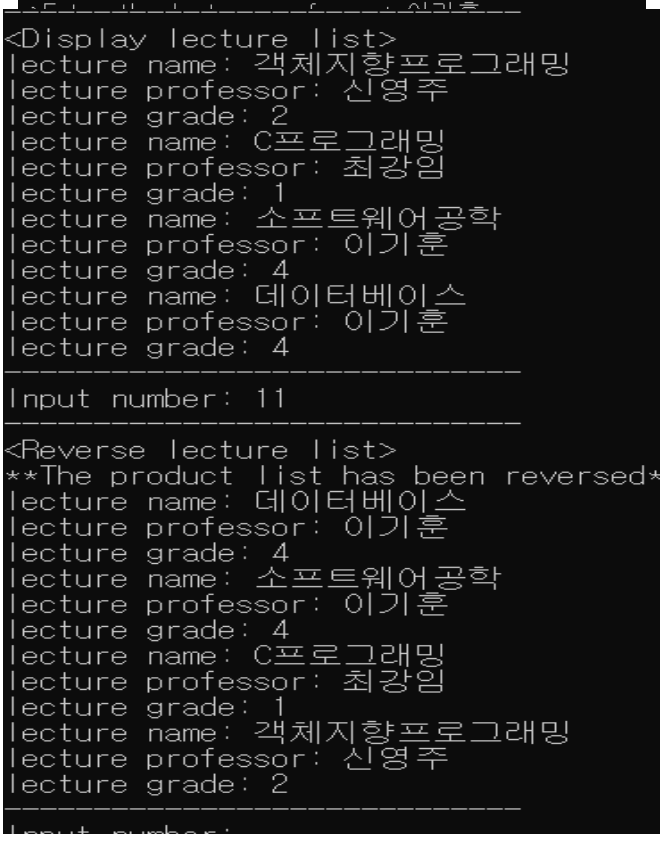
LINKED LIST의 HEAD부터 TAIL까지 탐색하며 반복문을 이용하여 교수님 성함, 강의명, 학년을 출력한다.

-Reverse

LINKED LIST의 TAIL부터 HEAD까지 탐색하여 반복문을 이용하여 교수님 성함, 강의명, 학년을 출력한다.

결과화면





위의 결과화면들은 1번부터 12번까지 모든 기능을 보여준다. 올바르게 결과 값을 출력하고 있음을 알 수 있다.

고찰

이번 과제를 하면서 많은 것들을 배웠다. 사실 Double linked list만을 구현하는 것도 쉽지 않았는데 거기에 Circular까지 더해지니 머릿속이 새하얘졌다. 링크드리스트의 중간에 Insert할 때나, Delete할 때는 노드의 prev가 존재하기 때문에 Single Linked List보다 구현이 쉬웠지만 만약 처음이나 끝에 Insert, Delete할 때는 Circular의 개념을 생각해주어야 한다는 점이 꽤나 어려웠다.

<3>

Introduction

이 과제는 사용자로부터 입력 받은 사이즈와, 숫자들을 가지고 Binary Search Tree를 구현하고, Queue(FIFO = First input First output)를 이용하여 Inorder, Preorder, Postorder로 Binary Search Tree에 있는 데이터들을 출력하는 프로그램을 구현하는 것이다. Queue는 Push(), Pop(), Front(), IsEmpty() 기능을 갖는다.

Algorithm

<**Queue**>

Queue Class는 private member로 BSTNode * Head와 size를 갖는다. 이는 이번 과제에서 쓰이는 Queue는 BSTNode를 저장 할 것이고, size는 Queue의 size이기 때문이다. 그리고 Queue Class는 size를 반환하는 Getsize(), Queue 가 비었는지 확인해주는 IsEmpty(), 그리고 Queue에 BSTNode를 삽입하는 Push(), 그리고 BST를 Queue에 preorder로 삽입하는 Pushpreorder(), inorder로 삽입하는 Pushinorder(), postorder로 삽입하는 Pushpostorder()를 갖는다. 그리고 Queue에 저장 돼있는

데이터들을 FIFO 방식에 맞게 꺼내줄 Pop()이 있다. 마지막으로 Front()는 맨 앞의 데이터를 반환한다.

<**BSTNode**>

BSTNode Class는 pLeft, pRight, pNext를 갖는다. pNext는 Queue에 있는 BSTNode를 연결하기 위해 사용하는 멤버이다. 이 Class는 멤버 변수들을 설정하거나 정보를 얻어올 수 있는 Get, Set 함수들을 Public 멤버로 가진다.

<**Tree**>

Tree Class는 BSTNode들을 가지고 있는 BST가 된다. 이는 BST를 만들어줄 Insert(), 그리고 해당 Node의 데이터가 중복되는지 확인해 줄 Search() 함수로 구성된다.

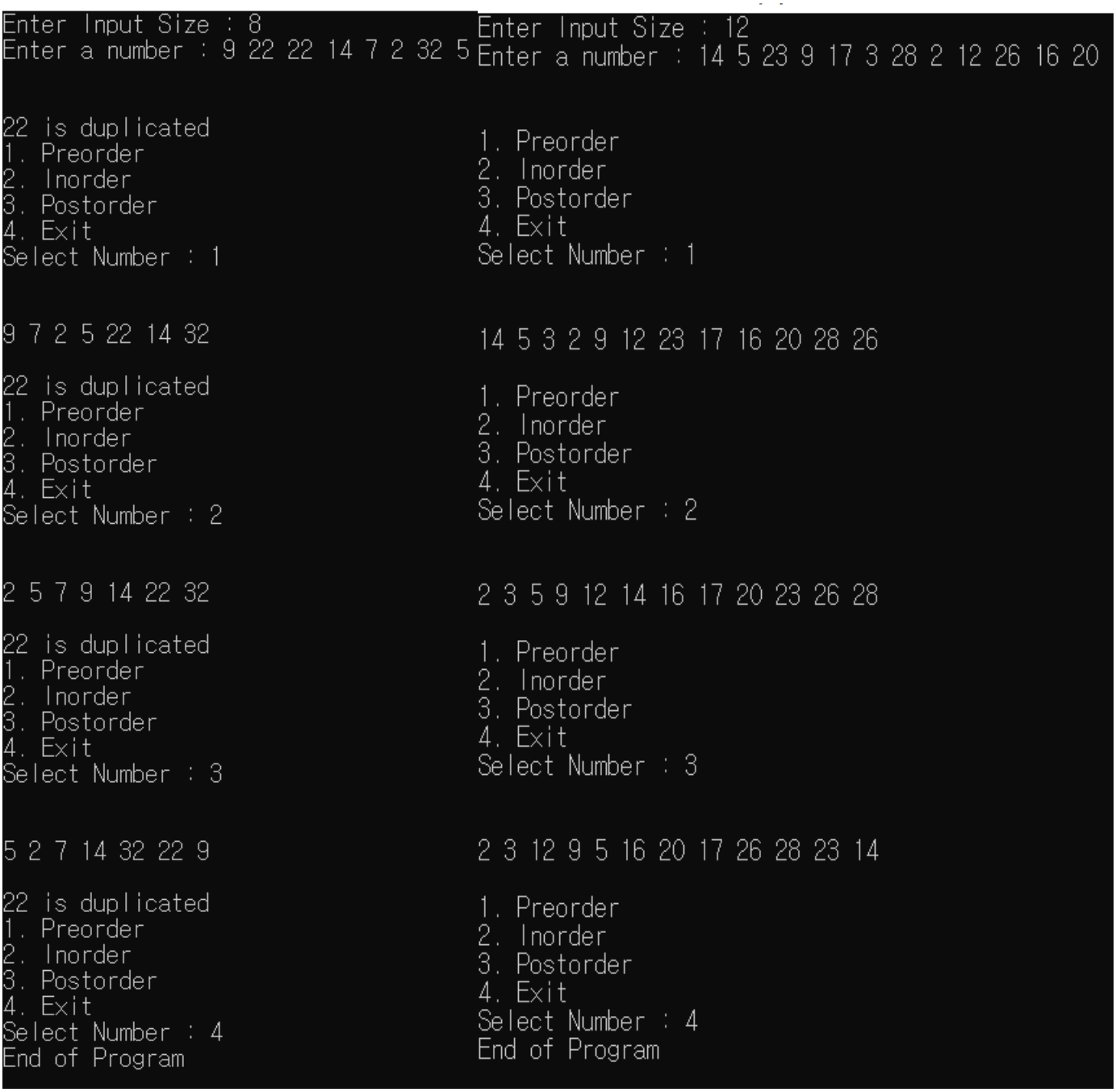
-Insert()

삽입할 노드를 Search()함수를 이용하여 중복검사를 시행 후, 중복되지 않는다면 루트 노드부터 비교를 하여 비교된 노드보다 더 크면 오른쪽 서브트리, 작으면 왼쪽 서브트리로 이동하여 알맞은 자리에 삽입한다.

-Search()

Data를 인자로 받아, root node data부터 비교한다. 만약 인자로 받은 data가 더 크다면 오른쪽 sub tree, 작으면 왼쪽 sub tree로 이동하며 탐색한다.

결과화면



위 결과화면은 Assignment3 제안서 pdf에 있는 테스트를 그대로 시행한 것이다. 결과 값이 올바르게 나오고 있음을 확인할 수 있다.

고찰

BST를 구현하는 것은 해본적이 있어 어렵지 않았지만, FIFO로 작동하는 Queue에 어떻게 preorder, inorder, postorder로 BSTNode를 저장할 지가 문제였다. 재귀 함수를 사용하여 해결할 수 있었는데 Queue 클래스에 있는 BSTNode* Head를 이용하여 재귀함수를 사용해 각 order에 맞게 삽입하는 함수를 만들었다. 그 후 Main 함수에서 Pop을 이용하여 출력하게 해서 프로그램을 완성할 수 있었다.

<4>

Introduction

이 과제는 Othello Game을 구현하는 것이다. Othello Game의 규칙은 다음과 같다.

1. 처음에 판 가운데에 사각형으로 엇갈리게 배치된 돌 4개를 놓고 시작한다.
2. 돌은 반드시 상대방 돌을 양쪽에서 포위하여 뒤집을 수 있는 곳에 놓아야 한다.
3. 돌을 뒤집을 곳이 없는 경우에는 차례가 자동적으로 상대방에게 넘어가게 된다.
4. 아래와 같은 조건에 의해 양쪽 모두 더 이상 놓을 수 없게 되면 게임이 끝나게 된다.
   4-1. 64개의 돌 모두가 판에 가득 찬 경우(가장 일반적)
   4-2. 어느 한 쪽이 돌을 모두 뒤집은 경우
   4-3. 한 차례에 양 쪽 모두 서로 차례를 넘겨야 하는 경우
5. 게임이 끝났을 때 돌이 많이 있는 플레이어가 승자가 된다. 만일 돌의 개수가 같을 경우는 무승부가 된다.

<5>

Introduction

이번 과제는 2048 Game을 구현하는 것이다. 시작 할 때 사용자가 선택한 사이즈의 판 위에 2개의 2 or 4가 있다. 여기서 방향을 입력하게 되는데, 입력하는 방향으로 숫자가 합쳐지게 된다. 이런 방식으로 게임을 진행하여 2048 블록을 만들면 게임에서 승리하게 된다.

고찰 (4 & 5)

4번과 5번은 시간 상의 이유로 구현하지 못하였다. 다른 강의 과제가 많이 겹쳐 이번 ASSIGNMENT3에 투자 할 시간이 많이 없었다. 4번과 5번이 어려워 보이긴 하지만 방학 때라도 꼭 구현해보고 싶다.