



# 3차 과제

---

과목명	객체지향프로그래밍
실습분반	B(02)
담당교수	신영주 교수님
학과	컴퓨터정보공학부
학년	2학년
학번	2019202009
이름	서여지
제출일	20.06.10

## 1. 카페 메뉴 관리 프로그램

### 1) 문제 이해

카테고리에 따라 나누어진 메뉴를 관리할 수 있는 프로그램을 작성하는 문제이다. 이 문제에는 두 종류의 노드가 사용된다. 각 메뉴의 이름정보와 가격정보를 메뉴 노드에 저장하고, 메뉴들이 속하는 카테고리의 이름정보는 카테고리 노드에 저장한다. 한 카테고리의 메뉴 정보들은 연결리스트 형태로 저장되고, 여러 개의 카테고리 노드들도 마찬가지로 연결리스트 형태로 구현된다. 이 문제를 해결하기 위해서는 텍스트 파일에서 메뉴 정보를 읽어오기 위한 파일입출력에 대한 이해와, 연결리스트의 구조에 대한 이해가 필요하다.

### 2) 작성한 프로그램 설명

MenuNode에는 메뉴의 이름을 저장하는 문자열과 가격을 저장하는 정수형 변수, 연결리스트에서 다음 메뉴노드의 주소를 저장할 MenuNode형 포인터 변수가 public영역에 선언되었다.

CategoryNode에는 카테고리의 이름을 저장하는 문자열과 해당 카테고리에 속하는 메뉴의 개수를 저장할 정수형 변수, 카테고리의 연결리스트에서 다음 카테고리의 주소값을 저장할 CategoryNode형 포인터변수가 있고, 마지막으로 해당 카테고리의 속하는 첫 번째 메뉴의 주소값을 저장 할 MenuNode형 포인터 변수가 public영역에 저장되어있다. 이 MenuNode형 포인터 변수는 메뉴의 연결리스트의 헤드와 같은 의미이다.

List 클래스는 private영역에 리스트가 가지고 있는 카테고리의 개수를 저장하는 정수형 변수 하나와, 카테고리의 연결리스트에 속하는 첫 번째 카테고리의 주소값을 저장하는 CategoryNode형 포인터변수를 가진다. 위에서와 마찬가지로 이 포인터 변수는 카테고리의 연결리스트의 헤드이다.

List 클래스는 public영역에 다양한 메소드를 가진다. 생성자, 소멸자와 더불어 메뉴를 추가하거나 삭제하는 멤버함수, 리스트의 전체 혹은 일부를 출력하는 함수, 특정 노드를 검색해서 순서나 해당 노드의 주소값을 반환하는 멤버함수이다.

add함수는 리스트에 새 메뉴를 추가하는 함수이다. 이 함수는 오버로딩 되어있다. 인자를 받지 않는 함수와 인자로 문자열 하나를 입력 받는 함수는 함수의 내부에서 두 개의 문자열과 하나의 정수를 만들고, 그것을 모두 인자로 사용하는 add함수를 호출한다. 인자를 받지 않는 함수는 input.txt 파일에서 한 줄을 읽어들이어 '\t'문자를 기준으로 카테고리 이름과 메뉴 이름, 가격을 분리하고, 인자로 문자열 하나를 입력 받는 함수는 콘솔창에서 입력받은 한 줄의 문자열을 공백문자(스페이스바)를 기준으로 분리한다. 세 개의 인자를 받는 add함수는 카테고리명과 메뉴의 이름을 이용해 새 노드가 삽입될 위치에 따라 새 메뉴 노드와 필요한 경우 새 카테고리 노드를 생성하여 값을 저장한다. 노드가 삽입될 위치를 찾는 것에는 find함수가 이용된다. add함수는 중복된 메뉴가 있는 경우 해당 메뉴가 이미 존재한다는 문장을 출력하고 (-2)를 반환하여 상황에 따라 오류처리가 가능하게 한다.

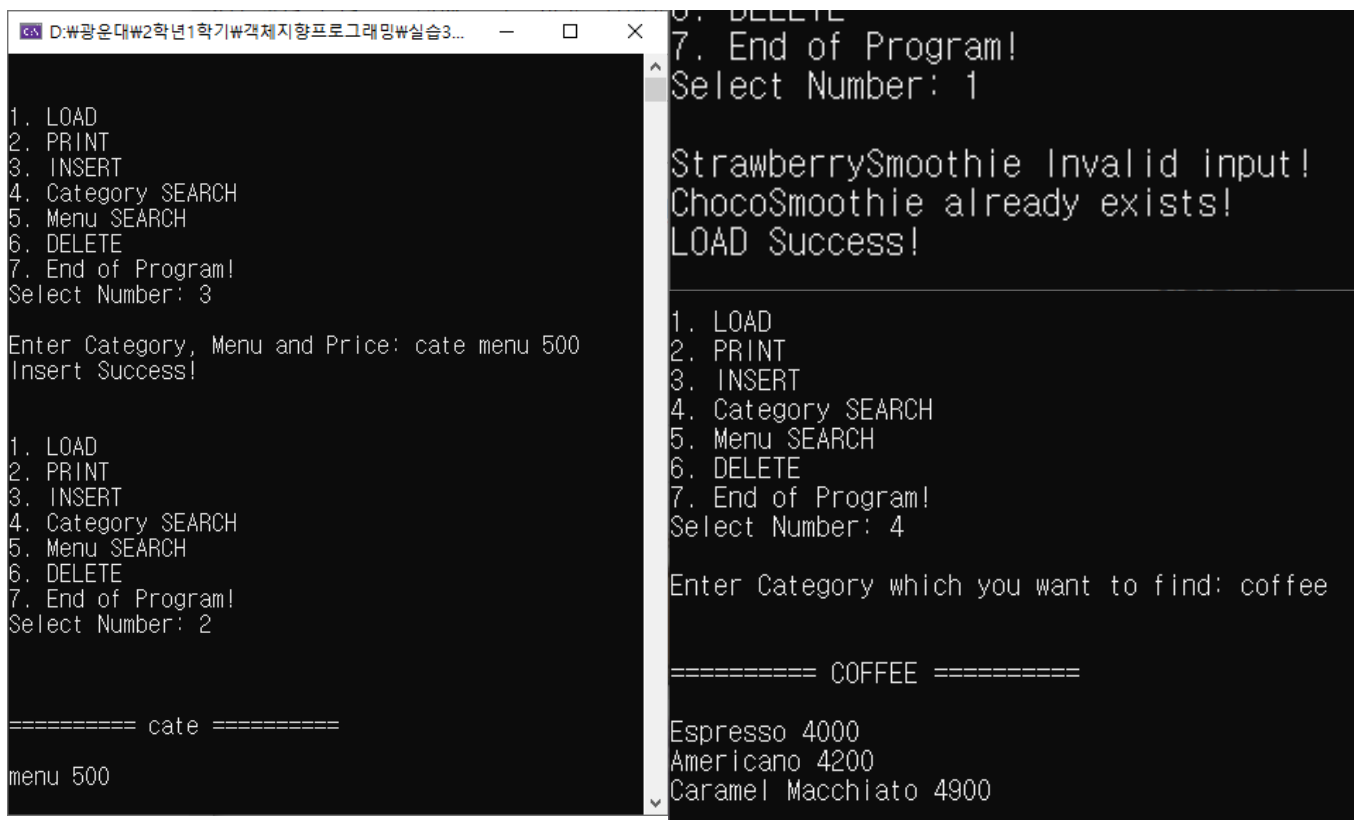
find 함수는 리스트에서 인자로 전달받은 문자열에 해당하는 카테고리나 메뉴를 찾아 그 순서에 대한 정보를 반환하는 함수이다. 이 함수 역시 카테고리 이름을 인자로 받는 경우와 카테고리의 순서와 메뉴의 이름을 인자로 받는 두 가지 경우로 오버로딩 되어있다. 두 함수 모두 해당하는 노드가 없는 경우 (-1)을 반환하고, 있다면 해당하는 노드의 연결리스트에서 순서를 반환한다. 이 함수는 메뉴의 생성과 삭제, 출력 등 프로그램에서 자주 사용된다.

remove함수는 리스트의 노드를 삭제하는 함수이다. 인자에 따라 카테고리 전체를 삭제하거나 메뉴 하나를 삭제한다. List 클래스의 소멸자에서는 카테고리 전체를 삭제하는 remove함수를 반복적으로 호출하여 메모리를 해제한다. 하나의 메뉴를 삭제하는 deleteMenu함수에서도 이 함수를 이용하여 메뉴 하나를 삭제한다.

printList함수는 인자가 없다면 리스트 전체를 출력하고, 카테고리의 순서가 인자로 입력되면 해당 순서의 카테고리에 속하는 모든 메뉴의 정보를 출력한다. 이와 유사한 printMenu함수는 문자열로 입력된 메뉴의 이름과 일치하는 정보를 가진 메뉴 노드를 찾아 내용을 출력한다.

main함수에서는 while반복문을 이용하여 메뉴를 반복적으로 출력하고, 사용자가 입력하는 정수를 switch-case문에 이용하여 각각의 기능을 하는 List 클래스의 메소드를 호출한다.

### 3) 결과화면



```
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: cate menu 500
Insert Success!

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

===== cate =====
menu 500

0. 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: 4

Enter Category which you want to find: coffee

===== COFFEE =====
Espresso 4000
Americano 4200
Caramel Macchiato 4900
```

#### 4) 고찰

일반적인 연결리스트는 몇 번 다루어본 경험이 있지만 이번 과제에서 사용된 연결리스트의 형태가 특이하여 작성에 어려움을 겪었다. 각각의 메뉴들을 연결하는 연결리스트는 간단하게 작성할 수 있었지만 각 메뉴 연결리스트의 헤드값을 가지고 있는 카테고리 노드와 그것을 연결하여 만드는 연결리스트는 처음에 그 구조를 이해하는 것에 시간이 들었다. 특히 새 메뉴를 추가하는 과정에서 기존에 카테고리가 없다면 새로 카테고리 노드를 생성하는 과정을 작성하며 시행착오를 여러 번 겪었다.

## 2. 강의목록 관리 프로그램

### 1) 문제 이해

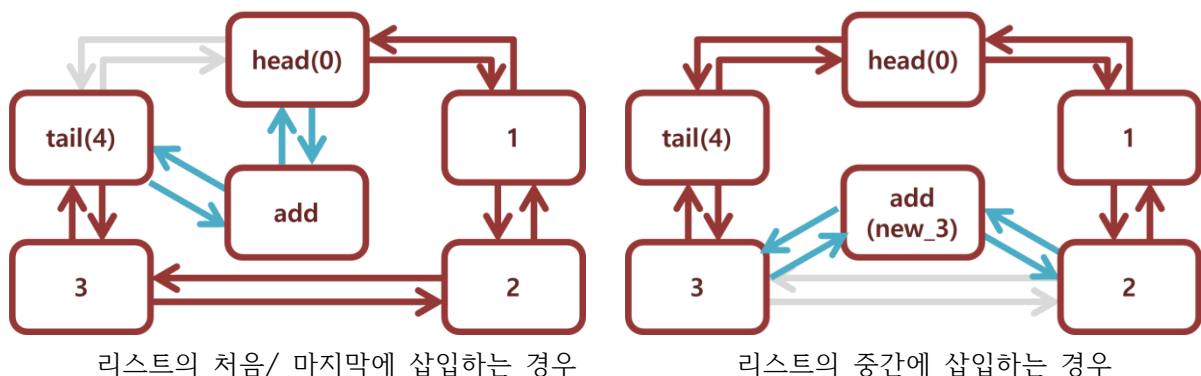
원형이중연결리스트를 이용하여 강의목록을 관리하는 프로그램을 작성하는 문제이다. 각 강의는 강의명, 교수명, 학년 정보를 가지고 있고, 이 프로그램은 강의삽입, 삭제, 내용수정, 검색, 정렬, 전체 출력, 순서 뒤집기 기능을 포함한다. 강의 삽입은 리스트의 가장 앞, 가장 뒤, 혹은 리스트의 특정 위치를 지정하여 삽입할 수 있다. 리스트 정렬은 학년의 오름차순이고, 학년이 같은 강의는 강의명에 따라 순서를 조정한다. 이 문제를 해결하기 위해 원형이중연결리스트에 대한 이해가 필요하다.

### 2) 작성한 프로그램 설명

Node 클래스는 각 강의의 정보를 저장할 변수가 public영역에 선언되어 있다. 강의명과 교수명은 문자열로 저장하고, 학년은 문자형으로 저장한다. 이중연결리스트에서 앞과 뒤에 위치하는 노드의 주소값을 저장하기 위한 Node형 포인터 변수를 두 개 포함하고 있다.

List클래스는 연결리스트에 저장된 노드의 개수를 세는 정수형 변수 하나와 연결리스트의 헤드인 Node형 포인터 변수 하나를 private영역에 가지고 있다. public영역에는 프로그램의 각 기능을 구현하기 위한 메소드가 선언되어 있다.

Insert함수는 새 노드를 삽입하려는 위치를 인자로 전달받아 작동한다. 새 노드를 할당하고 강의명과 교수명, 학년을 입력 받아 저장한 뒤, 해당 위치에 삽입한다.

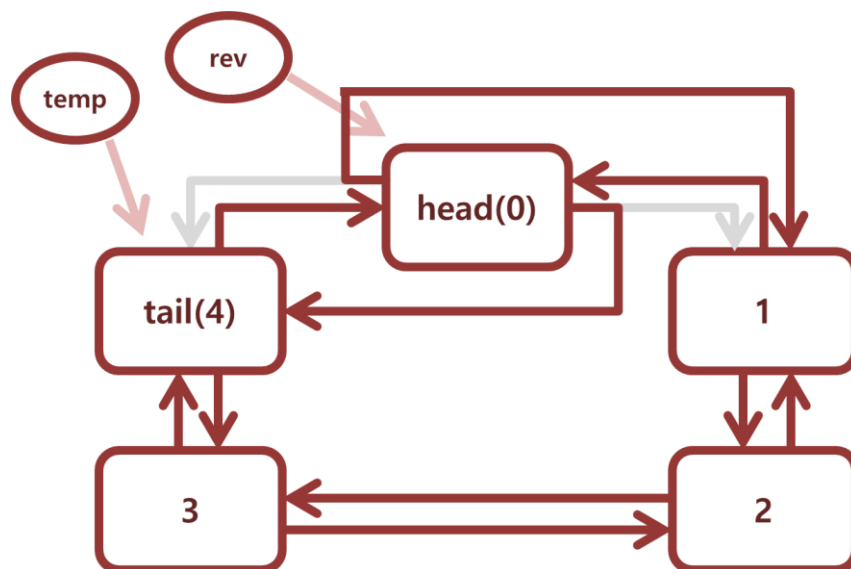


Delete 함수는 삭제하려는 과목명을 입력 받아 해당하는 노드를 삭제한다. 일치하는 노드가 없는 경우 다시 입력 받는다. Update 함수는 수정하려는 과목명을 입력 받아 해당 노드의 내용을 수정한다. 강의명을 입력 받아 일치하는 노드가 있는지 확인하고 동작한다는 점이 유사하지만, delete함수는 리스트에 없는 과목명이 입력된 경우 다시 입력 받지만, Update 함수는 오류메시지를 출력한다는 차이점이 있다.

검색에는 세 개의 함수가 있다. 강의명을 이용하는 SearchLecture함수와 각각 교수명과 학년으로 정보를 출력하는 SearchProfessor, SearchGrade 함수이다. SearchLecture함수는 노드의 순서를 반환하고 해당 노드가 존재하지 않으면 (-1)을 반환한다. 이 함수는 프로그램의 전반에서 어떤 노드가 존재하는지 확인하는 것에 쓰였다. 나머지 두 함수는 함수 내부에서 교수명 혹은 학년 정보를 입력받고 리스트의 노드를 하나씩 옮겨가며 일치하는 정보가 있는 노드의 내용을 출력한다. 세 함수는 매우 유사하게 동작한다.

리스트를 출력하는 Display함수는 인자를 받지 않는 경우와 노드의 순서를 인자로 받는 두 가지 경우로 오버로딩 되었다. 인자를 받지 않는 경우 함수는 리스트 전체의 노드를 하나씩 옮겨가며 내용을 모두 출력한다. 특정 위치의 노드를 인자로 받은 경우 해당 노드의 내용을 출력한다.

Reverse함수는 전체 리스트의 순서를 반대로 뒤집는다. 리스트의 헤드에서 시작해서 순서를 하나씩 바꾸어 나간다. 리스트가 비어있는 경우 (false)를 반환하고, 성공적으로 리스트를 뒤집은 경우 (true)를 반환한다.



Reverse 함수 동작

### 3) 결과화면

```
D:\광운대\2학년1학기\객체지향프로그래밍\실습3...
12.Exit
input number: 3
<insert lecture at position>
Enter the position of lecture insert: 2
->Enter the lecture name: C프로그래밍
->Enter the lecture professor: 최강임
->Enter the lecture grade: 1
-----
12.Exit
input number: 10
<Display lecture list>
lecture name: 객체지향프로그래밍
lecture professor: 신영주
lecture grade: 2
lecture name: C프로그래밍
lecture professor: 최강임
lecture grade: 1
lecture name: 소프트웨어공학
lecture professor: 이기훈
lecture grade: 4
lecture name: 데이터베이스
lecture professor: 이기훈
lecture grade: 4
-----
1. Insert lecture at beginning
12.Exit
input number: 6
<Search lecture>
->Enter the lecture name you want to search:c프로그래밍
* At position 2*
lecture name: C프로그래밍
lecture professor: 최강임
lecture grade: 1
-----
input number: 4
<Delete lecture>
Enter the lecture of lecture Delete: 소프트웨어공학
**소프트웨어공학 has been deleted from position3**
-----
input number: 5
<Update lecture>
->Enter the lecture name: 소프트웨어공학
No information to update
```

### 4) 고찰

리스트를 오름차순으로 정렬하는 함수를 작성하는 과정에서 어려움을 겪었다. 이 문제는 연결된 노드들이 오름차순인지 확인하는 부분과 오름차순이 아닌 것이 확인된 노드를 이동시키는 부분으로 나누어서 해결할 수 있었다. 학년이 같은 경우 이름 순으로 배치하게 되어있는데, 다시 과목명을 비교하는 과정이 앞에서 학년을 비교하는 과정과 거의 일치한 점이 특이했다. 1번 문제에서는 새로 삽입하는 노드의 가격을 고려하여 적절한 위치를 찾으려 했었지만, 이 문제에서는 이미 여러 개의 노드가 불규칙적으로 삽입된 상태에서 리스트 전체를 정렬하는 동작이라 더욱 복잡했던 것 같다.

## 3. BST

### 1) 문제이해

사용자로부터 노드의 개수를 입력 받고, 뒤에 입력되는 정수를 이진 탐색 트리 형태로 저장하는 문제이다. 이진 탐색 트리의 순회는 큐를 이용해서 구현한다. 이진 탐색 트리의 세 가지 순회방법인 Preorder, Inorder, Postorder를 실행할 수 있는 프로그램을 작성하는 문제이다. 이 문제를 해결하기 위해서는 이진 탐색 트리 구조와 순회에 대한 이해와 큐에 대한 활용이 필요하다.

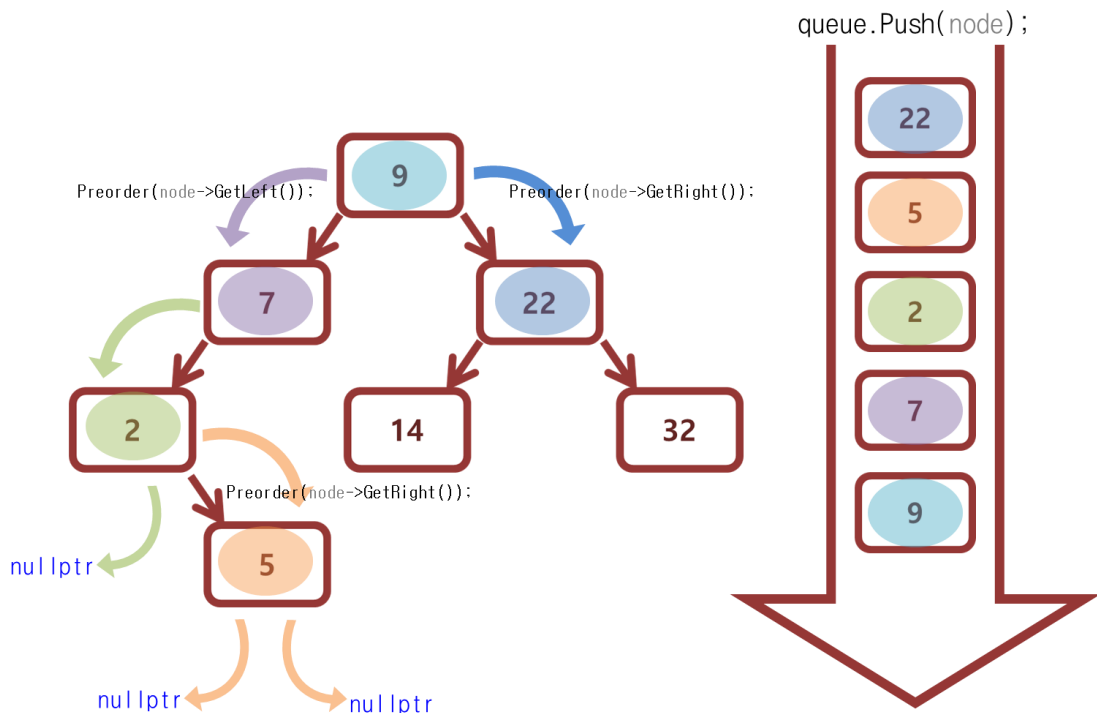
## 2) 작성한 프로그램 설명

BSTNode 클래스는 이진 탐색 트리를 구성하는 노드이다. BSTNode 클래스의 private 영역에는 연결된 노드의 주소값을 저장하는 BSTNode형 포인터 변수가 선언되어있다. 이들은 각각 노드의 부모 노드, 오른쪽, 왼쪽의 자식 노드, 큐에서 다음에 오는 노드를 의미하는 네 가지이다. 또한 각 노드는 하나의 정수형 변수에 데이터를 저장한다. BSTNode 클래스의 메소드는 private영역의 값을 반환하거나 저장하는 메소드 이다.

BST클래스는 트리에서 가장 위에 위치하는 노드의 주소값을 저장할 BSTNode형 포인터변수와 순회에 사용할 큐의 주소값을 저장할 Queue형 포인터 변수를 private영역에 가지고있고, public영역에는 이진 탐색 트리에 새 노드를 추가하는 함수와 세 가지 트리 순회를 실행하는 함수 등이 있다.

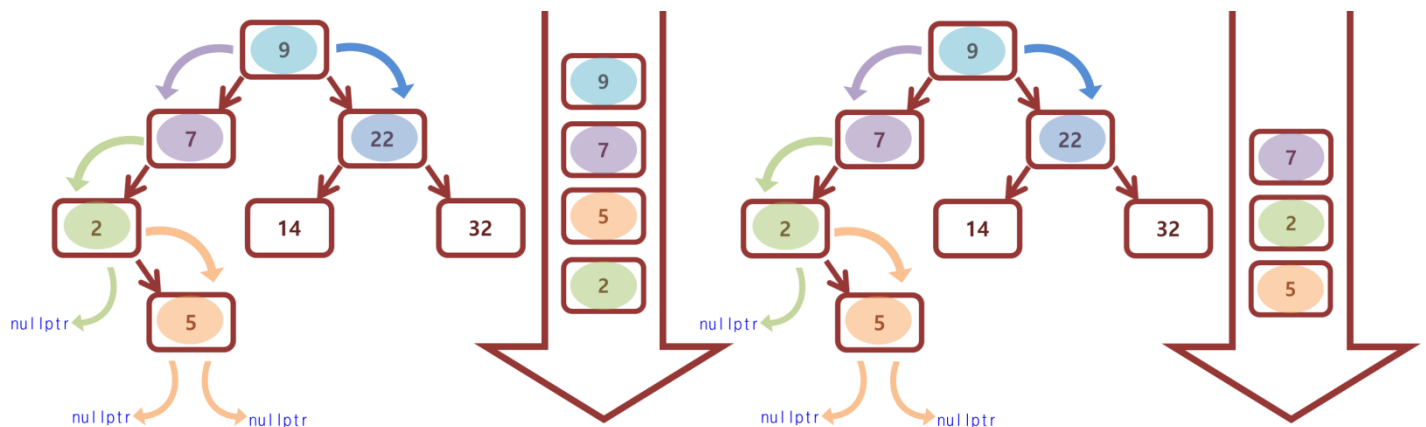
트리에 새 노드를 추가하는 함수는 인자로 새 노드가 가질 정보를 받는다. 새로 생성한 노드가 트리의 첫 번째 노드일 경우 트리의 root를 새 노드로 지정하고, 이미 노드가 있는 경우에는 새 노드가 삽입 될 위치를 찾아 적절한 위치에 삽입한다. 노드의 위치를 찾을 때 정보의 크기에 따라 오른쪽 혹은 왼쪽으로 나뉘는 특징을 이용하여 삼항연산자 ( ? : )를 이용하여 작성하였다. 이 문제에서 중복되는 정수는 허용되지 않기 때문에 기존 노드의 정보와 일치하는 정보를 갖게 된다면 메시지를 출력하고 오류처리한다.

세 개의 트리 순회 함수는 모두 재귀함수로 작성하였다. preorder, inorder, postorder에 맞는 순서로 큐에 노드를 삽입하거나, 현재 노드의 자식노드로 이동한다.



위 그림에서 나타난 preorder의 경우를 예를 들어 설명하면, Preorder함수에 인자로 BST의 가장 상위 노드인 '9'가 저장된 노드를 인자로 전달된다. Preorder함수는 큐에 인자로 전달된 노드를 삽입하는 동작을 가장 먼저 실행하고, 다시 현재 노드의 왼쪽 자식 노드의 주소값을 Preorder함수에 인자로 전달하여 함수를 호출한다. 이 예시에서 전달되는 노드는 '7'이 저장된 노드이다. 이 노드를 인자로 호출된 Preorder함수는 큐에 '7' 노드를 삽입하고, 다음 왼쪽 자식 노드를 인자로 하여 Preorder함수를 호출한다. 다음 실행에서, 인자로 전달된 '2'가 저장된 노드가 큐에 삽입되고, 다시 왼쪽 자식 노드를 인자로 같은 함수를 호출하지만, 이번에는 해당 노드의 왼쪽 자식 노드가 존재하지 않고, 그 주소값은 nullptr이기 때문에 바로 반환된다. 다시 '2'가 저장된 노드로 호출된 함수에서, 이번엔 오른쪽 자식 노드를 인자로 하여 함수를 호출한다. 따라서 '5' 노드가 큐에 삽입되고, 오른쪽 자식과 왼쪽 자식 노드에 대해 Preorder함수를 호출하지만 해당하는 노드가 없기 때문에 바로 반환된다. '5'가 저장된 노드에 의한 함수의 실행이 종료되었으므로, '2'의 노드에 의한 함수로 돌아가고, 이 과정이 반복되며 최초의 노드인 '9'가 저장된 노드까지 거슬러 올라간다. 이후 '9'의 노드의 오른쪽 자식에 대해서도 같은 과정이 반복되어 BST전체를 순회하게 된다.

Inorder와 Postorder함수 또한 비슷한 구조로 이루어져 있어서 해당 순회방식에 따라 함수를 반복해서 호출하고, 큐에 노드를 삽입한다.



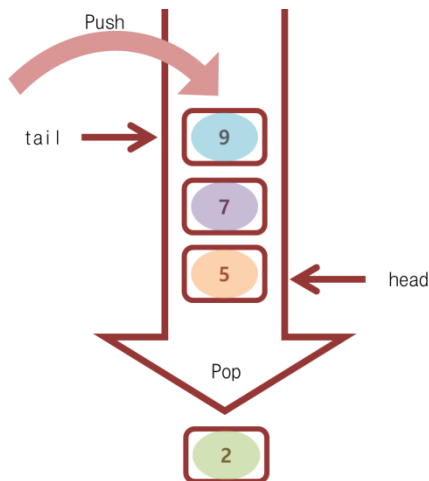
Inorder 순회 예시

Postorder 순회 예시

BST클래스의 소멸자는 postorder를 이용하여 작성한 Dtor함수를 호출한다. postorder를 이용하면 가장 하위의 노드부터 삭제할 수 있기 때문에 트리의 중간 노드가 삭제되어 메모리 누수가 일어나는 일 없이 모든 노드를 삭제할 수 있다.

마지막으로 Queue클래스는 큐의 head노드와 tail노드의 주소값을 저장하는 BSTNode형 포인터 변수 2개와 큐에 저장된 노드의 수를 세기 위한 정수형 변수 하나를 private영역에 갖고, 큐가 비었는지 확인하는 함수, 새 노드를 삽입하는 함수, 하나의 노드를 삭제하는 함수, 큐의 head 노드의 주소값을 반환하는 함수를 public영역에 가지고 있다.





큐에 새 노드를 추가하는 Push함수는 새로 저장한 노드를 큐의 마지막인 tail로 지정한다. 큐의 노드를 삭제하는 Pop함수는 큐의 가장 앞부분인 head 노드의 정보를 반환하고, 해당 노드는 삭제한다. 이 두 함수는 가장 먼저 삽입된 노드를 가장 먼저 삭제하는 큐의 특징(선입선출)을 보여준다.

### 3) 결과화면

```

Microsoft Visual Studio 디버그 콘솔
Enter Input Size: 7
Enter a number: 9 22 14 7 2 32 5

1. Preorder
2. Inorder
3. Postorder
4. Exit
Select Number: 1
Preorder:9 7 2 5 22 14 32

Select Number: 2
Inorder:2 5 7 9 14 22 32

Select Number: 3
Postorder:5 2 7 14 32 22 9

Select Number: 4
End of Program.

```

```

Microsoft Visual Studio 디버그 콘솔
Enter Input Size: 12
Enter a number: 14 5 23 9 17 3 28 2 12 26 16 20

Select Number: 1
Preorder:14 5 3 2 9 12 23 17 16 20 28 26

Select Number: 2
Inorder:2 3 5 9 12 14 16 17 20 23 26 28

Select Number: 3
Postorder:2 3 12 9 5 16 20 17 26 28 23 14

Select Number: 4
End of Program.

Microsoft Visual Studio 디버그 콘솔
Enter Input Size: 4
Enter a number: 6 6 2 7
Invalid input

```

### 4) 고찰

트리를 순회하는 함수를 작성하는 것에 재귀함수를 이용하는 것 대신 조건에 따라 nullptr를 확인하고, root노드에서 하위의 노드로 들어온 회수만큼 다시 부모 노드로 돌아가는 방법을 이용했었는데 조건을 작성하는 것이 너무 복잡해졌었다. BSTNode형 포인터변수를 한 칸씩 옮겨가며 while문 내부를 반복한다는 점에서 재귀함수를 떠올렸고, 매우 간단한 코드로 세 가지 순회를 모두 구현할 수 있었다. 코드를 작성한 뒤, 확인하는 과정에서 종이에 이진 탐색 트리를 직접 그려서 코드의 단계별 실행에 맞추어 하나씩 대조하는 방법이 순회 과정에 대한 이해를 도왔다.

#### 4. 오셀로

##### 1) 문제 이해

이 문제는 유명한 보드게임인 오셀로를 연결리스트를 이용해서 작성하는 문제이다. 오셀로는 8\*8 크기의 총 64칸으로 이루어진 판 위에서 두 명의 플레이어가 자신의 차례에 각자의 돌을 하나씩 올려 판 위의 돌들을 최대한 자신의 돌 색으로 바꾸는 것이 목적인 게임이다. 자신의 돌로 상대방의 돌을 양쪽에서 감싸면 자신의 돌로 둘러싸인 상대방의 돌을 모두 자신의 색으로 뒤집을 수 있고, 그 방향은 상, 하, 좌, 우 그리고 대각선을 모두 포함하는 8방향에 적용된다. 플레이어는 상대방의 돌을 적어도 하나 뒤집을 수 있는 위치에만 자신의 돌을 올릴 수 있다.

##### 2) 작성한 프로그램 설명

오셀로 게임을 만들기 위해서 두 개의 클래스를 이용하였다.

OthelloStone클래스는 게임에서 사용되는 돌을 의미하며 private영역에 돌의 색을 저장할 문자형 변수 하나만을 가진다. 또한 public영역에 돌의 색을 반환하거나 새로운 값을 저장하는 메소드를 가지고 있다. OthelloStone클래스는 게임이 진행되면서 플레이어가 특정 자리에 돌을 놓았을 때 동적 할당 되어 사용된다.

반면 게임의 배경이 되는 OthelloBoard 클래스는 OthelloStone클래스보다 구성이 복잡하다. 이 클래스는 게임 판 위의 64칸 중 한 칸을 의미한다. 칸 위에 돌이 있는 경우 그 주소값을 저장하기 위한 OthelloStone형 포인터 변수 하나와, 게임판의 다른 칸과의 관계를 나타내는 OthelloBoard형 포인터변수를 5개 가지고 있다. 이중 하나의 포인터 변수는 판의 (1,1) 지점의 주소값을 저장하기 때문에 모든 클래스에서 같은 값을 가지고 있고, 나머지 포인터 변수는 해당하는 칸의 상, 하, 좌, 우에 위치하는 칸의 주소값을 가진다. 만약 그 방향으로 더 이상 칸이 존재하지 않는다면 nullptr를 저장한다.

OthelloBoard클래스의 public영역에는 x,y 칸의 OthelloBoard의 주소를 반환하는 함수와 게임판을 생성, 출력,삭제하는 함수, 새 돌을 놓는 함수, 게임의 종료 조건 만족을 판별하는 함수 등이 선언되어있다. 새 돌을 내려놓기 위한 조건을 확인하는 함수를 방향 별로 8개, 내려놓은 돌에 대해 상대의 돌을 뒤집는 함수를 방향 별로 8개 선언하였다.

처음 OthelloBoard클래스가 생성하고, SetBoard함수를 이용해 64칸의 게임판을 연결한다. 이 함수에서 64개의 OthelloBoard 클래스가 문제에서 제시된 그림과 같은 형태로 연결된다. 게임판의 생성이 종료되면 SetStone함수를 4번 호출하여 말판 중앙의 4개의 돌을 내려놓는다.

보드를 출력하는 함수인 PrintBoard함수는 wchar\_t형과 wcout을 이용하여 말판과 돌을 나타내는 특수문자를 출력한다. 특정 칸에 돌이 놓인 경우에 문자가 두 가지이므로 삼항연산자를 이용하였다.

새 돌을 내려놓는 NewStone함수가 호출되면 가장 먼저 해당하는 칸에 사용자가 돌을 내려놓을 수 있는지를 확인한다. 이후 해당 칸으로 이동하여 OthelloStone클래스를 새로 동적할당하여 돌 색을 지정한다. 마지막으로 Turn 함수를 이용하여 새로 내려놓은 돌에 의해 양쪽이 둘러싸이게 된 상대방의 돌들을 모두 뒤집는다. 새 돌을 내려놓는 과정에서 사용되는 IsPossible함수와 Turn함수는 8 방향에 맞게 동작하는 8개의 함수를 각각 포함하고 있다.

IsPossible함수에서 돌을 놓을 수 있는지 판단할 때 가장 먼저 게임판 위의 좌표가 맞는지 확인하고, 다음으로 해당하는 칸에 이미 돌이 올려져있는지를 확인한다. 두 조건이 모두 확인된 후에 마지막으로 8가지 방향 중 어떤 방향이던 상대의 말을 뒤집을 수 있는 경우가 있는지 판단한다. 이 과정은 각각의 방향에 따라 다른 함수에서 동작한다. 방향에 따라 호출된 함수에서 새로 돌을 내려놓으려는 칸과 바로 이웃한 칸에 상대의 돌이 있는지 여부를 확인하고, 이 조건을 만족한다면 상대의 돌 끝에 자신의 돌이 존재하는지 확인한다.

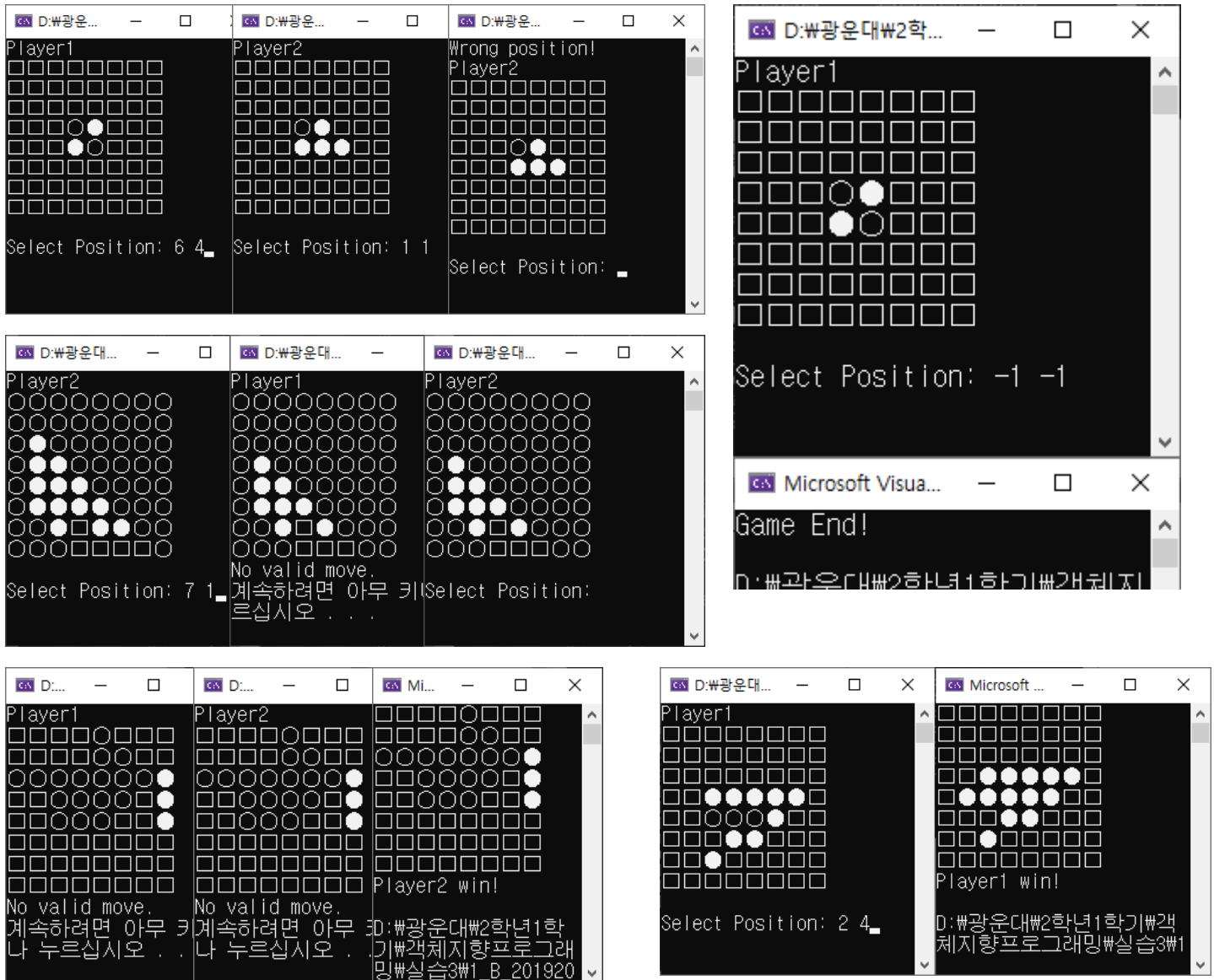
돌을 내려놓은 뒤 호출되는 Turn함수는 IsPossible함수에서처럼 방향별로 8개의 함수를 호출한다. Turn함수에서는 IsPossible함수를 이용하여 어떤 방향에 대해 상대의 돌을 뒤집을 수 있는 상황에서만 해당 방향에 대해 작동하는 함수를 호출한다. 각 방향의 Turn함수가 실행하는 내용은 간단하다. 새로 내려놓은 돌의 색과 일치하는 돌을 찾을 때 까지 색이 일치하지 않는 상대의 돌을 모두 자신의 색으로 바꾸는 것이다.

새 돌을 내려놓은 뒤에는 main함수의 흐름에 따라 종료조건을 확인하게 된다. 종료조건을 확인하는 멤버함수는 IsEnd함수이다. 이 함수에서는 모든 칸에 대해 올려져 있는 각 돌의 수를 센다. 두 종류의 돌에 대한 수의 합이 64인경우 게임판이 가득 찬 경우이므로 게임의 결과를 반환한다. 만약 두 돌 중 하나의 개수가 0인 경우 게임판 위의 돌이 한 가지 색으로 통일되어 게임이 종료된 경우이므로 마찬가지로 게임의 결과를 반환한다.

게임의 규칙에 따라 플레이어가 더 이상 말을 놓을 수 있는 칸이 없는 경우를 IsSkip함수를 통해 확인한다. 이 함수는 모든 칸에 대하여 IsPossible함수를 호출하여 플레이어가 칸을 놓을 수 있는 경우의 수가 있는지 확인한다. 게임이 종료된 후, 연결리스트를 해제 할 때는 DeleteBoard 함수를 이용한다. 이 함수를 실행하며 OthelloBoard클래스를 삭제하면, OthelloBoard클래스의 소멸자에서 해당 칸 위에 돌이 있는 경우 해당하는 OthelloStone클래스를 메모리 해제해준다.

main함수의 동작은 다음과 같다. OthelloBoard클래스를 생성하고, SetBoard 함수를 이용해 판을 만들고, 최초 4개의 돌을 올린다. 반복문의 내부에서 해당 순서의 플레이어가 돌을 올릴 수 있는 경우의 수가 있는지 확인한 후, 플레이어에게 돌을 올릴 칸을 입력받는다. 입력이 적절하지 않은 경우 다시 입력받고, 입력이 적절한 경우 새 돌을 올리고 종료조건을 만족하는지 확인한다. 종료조건을 만족한다면 결과를 출력하고 메모리를 해제한다. 종료조건을 만족하지 않은 경우 상대 플레이어에게 순서가 넘어가고, 위의 동작을 반복한다. 화면을 초기화 하는 함수를 이용되었다.

### 3) 결과화면 i 테스트케이스 출처



### 4) 고찰

main함수에서 생성된 하나의 OthelloBoard클래스가 다른 63개의 OthelloBoard클래스와 함께 연결리스트를 이루는 구조를 생성하는 것이 어려웠다. 처음에는 OthelloBoard의 생성자에서 다른 OthelloBoard를 동적할당하는 형태로 작성했는데, 새로 할당된 클래스에서 다시 생성자를 호출했기 때문에 오류가 일어났다. 오셀로 판을 구성하는 문자들은 모두 특수문자라서 wcout을 이용하여 처리하였다. wchar\_t형 변수를 처음 사용하는 것이었기 때문에 관련해서 유니코드에 대한 내용도 추가로 살펴보게 되었다. 새로운 돌을 추가하는 과정에서 해당 칸의 조건을 확인하는 방법을 떠올리는 것은 간단했지만, 8개의 방향에 대해 각각 실행되는 함수를 작성하여 코드의 길이가 과도하게 길어진 것 같다. 방향을 함수의 인자로 받아 처리하는 하나의 함수로 작성할 수도 있을 것 같았지만 구현에 실패해서 아쉬움이 남는다.

## 5. 2048

### 1) 문제이해

4\*4 크기의 공간에서 무작위 위치에 생기는 수를 상, 하, 좌, 우 네 방향 중 한 쪽의 벽으로 밀어서 블록의 수들을 합치고, 최종적으로 2048을 만들면 승리하는 게임인 2048 게임을 만드는 과제이다. 이 문제에서 무작위 위치에 생기는 수는 2로 고정되어있다. 수가 적힌 블록은 마치 특정 방향으로 중력을 작용시키듯 밀어낼 수 있다. 이때 밀어낸 방향으로 이웃하는 두 개의 블록의 수가 일치한다면, 두 블록의 합의 값을 갖는 하나의 블록으로 합쳐진다. 16개의 칸이 모두 수로 채워지고, 그 수들이 어떤 방향으로든 더 이상 합쳐질 수 없을 때 플레이어의 패배로 게임이 종료된다. 이 문제에서 제시된 좌표는 (x,y)이고, x값은 세로축, y값은 가로축을 의미한다.

### 2) 작성한 프로그램 설명

2048게임을 만들기 위해 2개의 클래스를 정의하였다.

MyBlock클래스는 게임의 배경이 되는 16개의 블록 중 한 칸을 의미한다. 클래스의 private영역에 해당 칸이 가지고 있는 수와 x,y좌표를 저장할 정수형 변수 세 개와 4\*4의 공간에서 해당 칸과 이웃한 칸의 주소값을 저장할 MyBlock형 포인터 변수를 4개 가지고 있다. 각 변수의 내용을 수정하거나 반환하는 함수들이 public영역에 선언되어 있다.

MyBoard클래스는 MyBlock클래스를 이용하여 만든 4\*4크기의 게임판이다. private영역에는 게임판의 (0,0)위치에 해당하는 칸의 주소값을 저장할 포인터 변수가 있다. public영역에는 생성자, 소멸자와 함께 게임판을 출력하는 함수, 특정 좌표에 해당하는 칸의 주소값을 반환하는 함수, 무작위 위치에 2를 생성하는 함수, 상, 하, 좌, 우로 밀어내기 동작을 하는 네 개의 함수와 게임의 종료조건을 확인하는 함수가 선언되어 있다.

클래스의 생성자에서는 총 16개의 MyBlock 클래스를 동적할당하고, 좌표값을 부여한 뒤 다른 칸과 연결한다. 소멸자에서는 모든 칸에 대해 할당했던 메모리를 해제한다. 보드 출력 함수인 print함수에서는 형태에 맞추어 보드의 모든 칸의 정보를 출력하고, 무작위로 2를 생성하는 random2함수는 rand함수를 이용해 생성한 (x,y)위치에 2를 삽입한다. 이때 해당 좌표에 이미 수가 저장되어있다면 다시 무작위 좌표를 생성한다. 16개의 칸 중 빈 칸이 존재하는지 여부를 IsFull함수에서 모든 칸에 대한 값을 조회하여 확인한다.

게임의 종료조건을 확인하는 IsEnd함수는 16칸의 블록에 접근하여 2048이 저장된 칸을 찾고, 동시에 빈칸의 개수를 센다. 2048이 있는 경우 게임은 승리로 종료되고, 빈칸이 하나라도 있는 경우 계속 진행된다. 빈칸은 없지만 어떤 방향으로 밀었을 때 블록의 수가 더해져 게임을 계속할 수 있는 경우를 찾기 위해 모든 블록의 모든 방향에 대해 같은 수가 연달아 존재하는 경우를 찾는다. 같은 수가 연달아 존재하는 경우 게임은 계속 진행되고, 아닌 경우 플레이어의 패배로 게임이 종료된다.

밀어내기 동작을 수행하는 네 개의 함수는 방향을 제외하면 동작이 같다. Up, Down 함수는 각 열에 대해 동작을 반복하고, Left, Right 함수는 각 행에 대해 동작을 반복한다. 해당 동작에서 블록은 밀린 방향에 0이 있는 경우, 즉 칸이 비어있는 경우 빈칸위치로 이동한다. 벽에 도착하거나 숫자가 저장된 블록을 만날 때까지 이동을 반복한다. 숫자가 저장된 블록을 만났을 때 두 블록의 수가 같은 경우 블록에 저장된 값을 두 수의 합(저장된 수의 2배)으로 바꾸고, 수가 바뀐 블록의 주소값을 저장한다. 이것은 한 번의 실행에서 값이 더해진 블록에 대해서 추가로 값을 더하는 일이 없게 하기 위함이다. 밀어내기 함수들은 블록의 이동이나 값의 변경이 일어나면 true를 반환하고, 아무런 변화가 없는 경우 false를 반환한다. 16개의 칸이 가득 찬 방향에서 변화가 일어나지 않는 방향에 대한 명령을 실행해도 새로운 2가 생성되지 않도록 예외처리를 하기 위함이다.

main함수의 흐름은 다음과 같다. MyBoard를 생성하여 무작위 2를 하나 생성하고, 반복문 내부에서 보드와 출력문을 출력한 뒤 사용자에게 메뉴를 입력받는다. 사용자의 입력에 따라 밀어내기 함수를 실행하고, 게임의 종료조건이 만족되었는지 판별한다. 종료조건이 만족된 경우 결과를 출력하고 프로그램을 종료한다. 아닌 경우 다시 플레이어에게 메뉴를 입력받아 위의 과정을 반복한다.

### 3) 결과화면

The screenshots illustrate the 2048 game interface. Each screen shows a 4x4 grid board with numbers, a menu for movement (1.Up, 2.Down, 3.Left, 4.Right), and a score display. The sequence of screens shows the game's progression, including winning and losing states.

**Screen 1:** Initial board state with a score of 2. Menu: 1.Up 2.Down 3.Left 4.Right. Input: 2.

**Screen 2:** Board state after moving 2 to the right. Menu: 1.Up 2.Down 3.Left 4.Right. Input: 3.

**Screen 3:** Board state after moving 2 to the left. Menu: 1.Up 2.Down 3.Left 4.Right. Input: 2.

**Screen 4:** Board state after moving 2 to the right. Menu: 1.Up 2.Down 3.Left 4.Right. Input: 3.

**Screen 5:** Board state after moving 2 to the left. Menu: 1.Up 2.Down 3.Left 4.Right. Input: 2.

**Screen 6:** Board state after moving 2 to the right. Menu: 1.Up 2.Down 3.Left 4.Right. Input: 3.

**Screen 7:** Board state after moving 2 to the left. Menu: 1.Up 2.Down 3.Left 4.Right. Input: 2.

**Screen 8:** Board state after moving 2 to the right. Menu: 1.Up 2.Down 3.Left 4.Right. Input: 3.

**Screen 9:** Board state after moving 2 to the left. Menu: 1.Up 2.Down 3.Left 4.Right. Input: 2.

**Screen 10:** Board state after moving 2 to the right. Menu: 1.Up 2.Down 3.Left 4.Right. Input: 3.

**Screen 11:** Board state after moving 2 to the left. Menu: 1.Up 2.Down 3.Left 4.Right. Input: 2.

**Screen 12:** Board state after moving 2 to the right. Menu: 1.Up 2.Down 3.Left 4.Right. Input: 3.

#### 4) 고찰

4번의 오셀로 문제와 다르게 블록을 여러 개 동적할당하여 게임판을 만드는 클래스를 이용하여 문제를 해결하였다. 4번 문제는 (x,y)에서 x좌표가 가로, y좌표가 세로를 의미한 반면 이번 문제에서는 x좌표가 세로, y좌표가 가로를 의미하고 있어 혼동이 있었다. 밀어내기 함수에서 밀어낸 후 같은 수를 가진 블록과 이웃하게 되면 두 블록의 수를 더하도록 코드를 작성했었는데 한 번의 실행에서 여러 개의 블록이 합쳐지는 오류가 발생하였었다. 이는 한 번 수정된 블록의 수가 연속해서 변경되지 않도록 처리하여 오류발생을 막을 수 있었다. 프로그램을 작성하는 것에 큰 어려움은 없었으나, 완성된 프로그램을 테스트 하는 과정에서 게임에 실패하여 여러 번 도전했다. 결국 한 번의 성공을 이뤘으나, 결과 화면을 촬영하지 못해 위의 결과화면은 임의로 1024를 두 개 생성하여 더한 것으로 대체하였다. 직접 게임을 통해 2048을 완성했을 때도 승리메시지가 출력되는 등의 동작은 동일하게 작동하였다.

---

#### i 오셀로 테스트케이스 출처

- 1) 전체 통일 (흑돌) <https://www.youtube.com/watch?v=xvX6hmDRvvA>
- 2) 차례 독점 (백돌) <https://www.youtube.com/watch?v=2wAbQM98s78>
- 3) 양쪽 모두 패스 (백돌) <https://www.youtube.com/watch?v=B2RKnHTrbTs>
- 4) 64:0 차례독점 (흑돌) <https://www.youtube.com/watch?v=jtPs9Afv5ds>