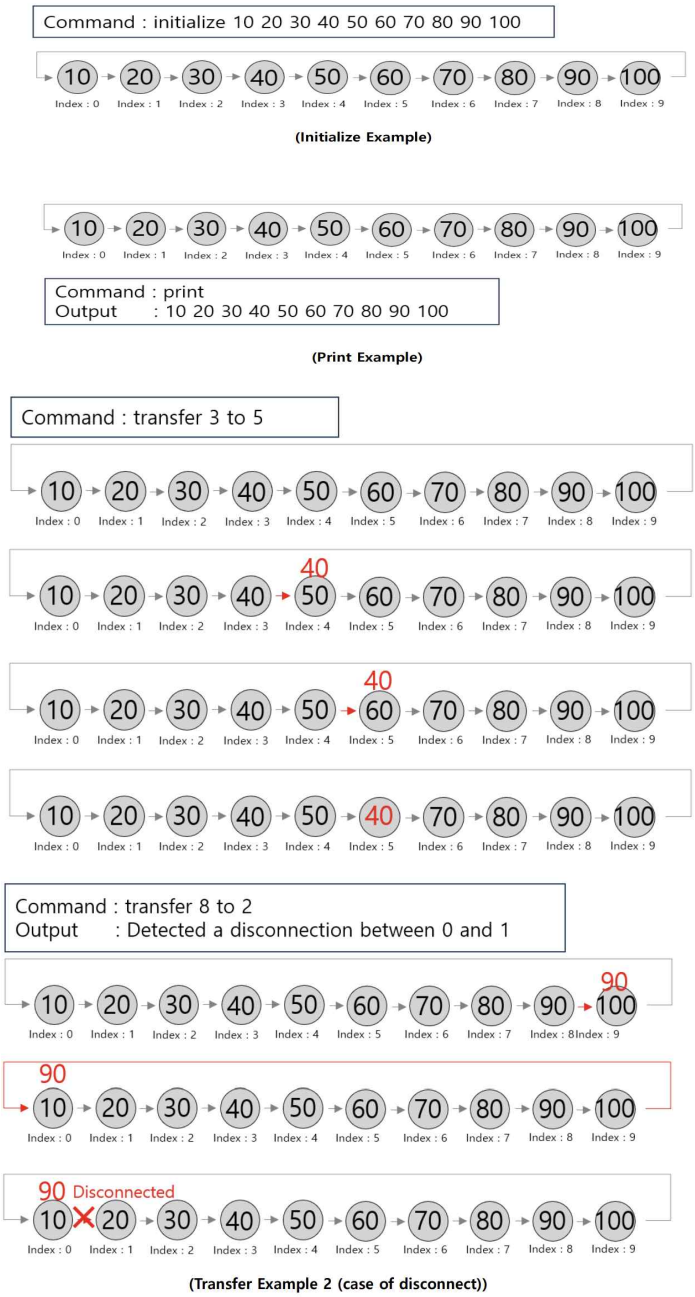
객체지향프로그래밍 3차 과제 보고서

2021202085 전한아솔

<Assignment #3-1>



10개의 원소를 입력하여 리스트를 구성하고 tail노드와 head노드를 이어서 원형리스트를 만든다. 이 원형리스트에서 transfer 커맨드를 이용하여 한 인덱스에서 다른 인덱스로 값을 전달할 수 있다. 한 노드에서 다른 노드로 옮겨갈 때마다 10%의 확률로 연결이 끊길 수 있다.

<알고리즘>
이전 과제에서도 사용했던 리스트와 똑같이 head, tail로 멤버를 구성하여 tail의 다음 노드로 새로운 노드를 이어서 리스트를 구성했다. 10개의 원소로 구성된 원형리스트이므로 리스트의 원소가 10개라면 tail의 다음 노드를 head로 이어서 원형리스트를 구현했다.
이 프로그램의 핵심 함수인 transfer에서는 인자로 인덱스 값 2개를 전달 받는다. transfer a to b의 입력이라면 a인덱스의 값을 b인덱스의 값으로 저장하게 된다. head부터 노드를 순회하여 a 인덱스를 가진 노드에 도달한다. 이후 해당 노드의 데이터를 저장하고, a 인덱스부터 b 인덱스까지 while문을 통하여 순회한다. 이때 random number generation을 이용하여 0~9의 랜덤 값을 발생시키고, 랜덤 값이 0이라면 해당 노드와 다음 노드 사이의 연결이 끊어졌다는 메시지를 출력하고 해당 노드의 다음 노드를 nullptr로 지정하여 연결을 끊는다. 이렇게 되면 0이 나올 확률은 1/10이므로 10%이다. 0이 아니라면 다음 노드로 이동한다. 원형리스트이므로 9 인덱스의 다음은 0 인덱스로 이동한다. while문의 조건을 b 인덱스가 아닐 때까지 반복하는 것으로 설정하여 b 인덱스를 가진 노드에 도달하면 반복문을 탈출하고 b 인덱스를 가진 노드의 데이터를 아까 저장한 a 인덱스 노드의 데이터로 초기화한다.
출력은 do while문을 이용하여 head부터 순회한다. 종료 조건은 노드가 nullptr이거나 head일 때 종료된다. do while문을 이용하므로 처음 head는 건너뛰고 실행되면 한 바퀴를 돌아서 다시 head로 돌아오면 조건에 만족하지 않으므로 출력이 중단된다. 또한 transfer을 실행하다가 연결이 끊기면 다음 노드가 nullptr이 되므로 nullptr을 만나도 출력이 중단되도록 구현했다.
<결과>

```
initialize 10 20 30 40 50 60 70 80 90 100
transfer 2 to 7
print
10 20 30 40 50 60 70 30 90 100
transfer 4 to 0
print
50 20 30 40 50 60 70 30 90 100
transfer 8 to 6
Detected a disconnected between 8 and 9
exit
Exit the program
```

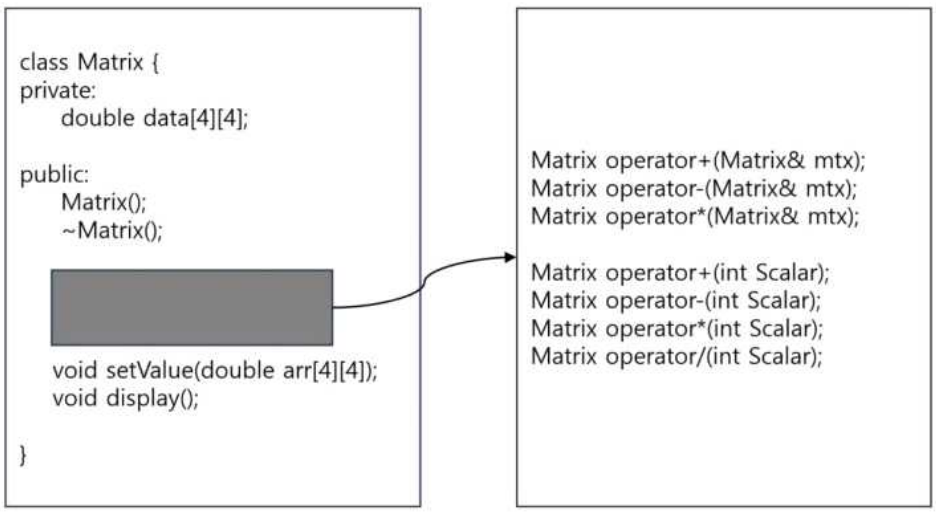
<고찰>
큐를 원형으로 구현한 원형 큐는 보았으나, 리스트도 원형리스트가 있다는 것을 이 문제를 통하여 처음 알게 되었다. 또한 처음에 10%의 확률로 연결이 끊겼다는 메시지를 출력하라는 것을 보고 어떻게 확률을 구현해야 할지 막막했으나 수업 시간에 자주 언급하신 random number generation을 통해서 0~9의 랜덤 값을 발생시키고 이 중에 한 숫자를 지정하면 1/10, 즉 10%임이 생각났고, 이 방법을 통하여 문제에 접근했고 처음 접해본 유형의 문제라서 고민이 많았으나 과제를 다 마친 후에는 많은 것을 배웠다는 생각이 들었다.
또한 연결을 끊는 방법의 설명이 나와 있지 않아서 연결이 끊긴다면 해당 노드의 다음 노드를 nullptr로 지정하여 연결을 끊었다. 이렇게 되면 연결이 끊긴 후에는 다시 transfer이나

print 커맨드를 사용할 경우 에러가 발생할 수 있다.

<Assignment #3-2>

2. Implement a Matrix Class consisting of 2D array (data) and the number of rows (rows) and columns (cols) and then implement matrix operations using operation overloading. Let assume that the matrix is square and matrix size is 4x4.

    - Binary Operations
        - "Matrix – Matrix": + (addition), - (subtraction), * (multiplication)
        - "Matrix – Scalar": + (addition), - (subtraction), * (multiplication), / (division)



(Matrix Class Sturcture)

4×4의 크기로 지정된 행렬 클래스를 구성하는데, 행렬끼리의 합, 뺄셈, 곱을 할 수 있도록 연산자 오버로딩을 구현하고 행렬과 Scalar의 합, 뺄셈, 곱, 나눗셈을 할 수 있도록 연산자 오버로딩을 구현하는 프로그램이다.

<알고리즘>

연산자 오버로딩이란 기존에 정의되지 않은 기능을 연산자를 통해서 새롭게 정의하기 위한 것으로 반환형은 해당 클래스의 이름과 같다. 연산자 오버로딩 함수 내에서 새롭게 객체를 생성하거나 자신을 가리키는 객체를 바꾸어서 반환하도록 구현할 수 있다. 나는 함수 내에서 새로운 객체를 만들어 저장하여 해당 객체를 반환하도록 오버로딩을 구현했다.

먼저, 행렬끼리의 합, 곱, 뺄셈을 진행하기 위해서 인자로 Matrix 객체를 참조형으로 전달받는다. 행렬끼리의 합, 뺄셈은 위치가 같은 (4x4로 고정이므로 위치가 같음) 원소끼리의 합, 뺄셈으로 구성된다. 따라서 2중 반복문을 통하여 같은 위치의 원소를 더하거나 빼주면 된다.

행렬의 곱은 조금 다른데, 두 행렬을 곱하여 새롭게 만든 행렬의 I행 j열의 원소는 첫 행렬의 I행과 두 번째 행렬의 j열의 내적으로 구할 수 있다. 즉, $A_{ik} \times B_{kj}$(k=1부터 4까지)의 합으로 행렬을 구성할 수 있다. 위 프로그램에서는 행렬을 2차원 배열로 구성하였으므로 (하나의 행렬의 I행, 0~3열의 원소) × (다른 행렬의 0~3행, j열의 원소) 의 결괏값을 모두 더해주면 새로운 행렬의 원소들을 구성할 수 있다.
행렬과 Scalar의 연산은 인자로 받은 int형 데이터를 행렬의 모든 원소에 더하거나 빼거나 곱하거나 나누어주면 된다.
<결과>

```
Matrix 1:
1 2 3 4
5 7 8 4
9 5 7 1
5 8 7 4

Matrix 2:
5 1 2 3
4 1 2 3
3 1 2 3
2 1 2 3

Mat1 * Mat2:
30 10 20 30
85 24 48 72
88 22 44 66
86 24 48 72

Mat1 + 3:
4 5 6 7
8 10 11 7
12 8 10 4
8 11 10 7
```

<고찰>
오버로딩에 대한 개념이 흔들려서 여러 블로그와 책을 찾아보며 공부한 결과, 오버로딩이란 함수의 함수의 이름은 동일하나 매개변수의 개수, type이 다른 여러 함수를 정의할 수 있는 기능이라는 것을 알게되었다. 그 중에서도 특히 연산자 오버로딩은 기본적으로 정의되어있는 연산자를 내가 설계한 클래스에서도 맞추어 사용하기 위해서 연산자의 기능을 더 확장할 수 있는 기능이라는 것을 배웠다. 또한 오버로딩과 비슷한 오버라이딩이 있는데, 오버라이딩은 상속 관계에서 기초 클래스의 함수를 유도 클래스에서 재정의하는 기능인 것도 배웠다.
오버로딩을 찾아보던 중 오버라이딩이 함께 나와서 헷갈렸는데 확실히 개념을 잡아두니 헷갈리지 않게 되었다.

<Assignment #3-3>

로그인, 계정 가입, 탈퇴의 기능을 지원하는 로그인 프로그램이다. 기존의 멤버쉽 파일을 읽어와 기존 정보를 저장해놓을 수 있고, 종료 후에도 파일에 정보를 저장하여 프로그램이 종료해도 정보는 남아있도록 구현해야한다.
먼저 기존에 있는 id와 password가 일치해야 로그인이 가능한데, 이때 비밀번호의 알파벳은 id의 길이만큼 오른쪽으로 밀린다. 예를 들어 아이디의 길이가 3이라면 A는 D, Z는 C로 변환된다. 이런 방식으로 비밀번호는 암호화되어 저장된다.
회원가입을 위해서는 첫째, 중복되는 id가 없어야 한다. 둘째, 비밀번호의 길이가 10 이상, 20

3. Implement a login program using a membership file. Implement a login program using a membership file. The program should maintain member information even after it is closed and reopened. The program follows the below rules:
   A. There should be four menu options (Login, Register, Withdraw, and Exit).
   B. Login should only be successful when the ID and password entered match those in the membership file.
   C. For membership registration, there should be no overlapping IDs in the existing membership information, and the password must be set to meet the set rules to register.
      1. The password must contain at least one alphabets (only lowercase), at least one numbers, at least one special characters (!@#$%^&*()). And the password must contain at least 10 characters, at most 20 characters.
   D. Membership withdrawal should only be able to delete the record of the logged-in members from the membership information file.
   E. The membership information consists of structures, as shown in the example below.
   F. Among passwords, the alphabet is transformed into a Caesar password and stored, and each user password is encrypted by pushing an id length.
      1. Caesar's password is to encrypt each letter by pushing it a fixed number of times.
      2. For example, when you encrypt, if you push each letter three spaces to the right, 'A' changes to 'D' and 'Z' changes to 'C'.

```
struct Member {
    char id[20];
    char password[20];
}
```

(Member struct)

이하여야 한다. 셋째, 비밀번호에 알파벳 소문자, 숫자, 특수문자가 모두 1개 이상 포함되어야 한다.

로그인에 성공하면 로그인 상태로 들어가는데, 이 상태에서만 탈퇴할 수 있다.

<알고리즘>

먼저, 파일에서 id와 password를 입력받아서 정보를 저장하기 위해서 ifstream 객체를 이용하여 id와 password를 읽는다. 파일에는 id 빈칸 password의 형태로 저장시켰다. 이렇게 읽어온 id와 password를 객체에 저장하기 위해서 membership 클래스의 멤버로 Member 포인터 inform, 가입한 사람의 수를 나타내는 int형 변수 cnt_num을 선언했고 inform[cnt_num]에 해당 id와 password를 저장하고 cnt_num을 증가시키는 방식으로 파일의 정보를 저장했다.

<set_enclnform 함수>

입력받은 비밀번호를 id만큼 오른쪽으로 밀어 암호화시켜서 저장하는 함수로, 인자로 id와 password를 전달받는다. for문을 통해서 password의 문자를 하나하나 순회하는데, 소문자나 대문자 알파벳일 경우, id의 길이 -26(알파벳의 개수)의 값을 더해준다. 이때 아스키코드 상으로 a나 A보다 작다면 다시 26을 더해준다. 만약 비밀번호의 문자가 a이고 id의 길이가 3이라면 97+(3-26)=74가 된다. 이는 a의 아스키코드인 97보다 작으므로 26이 더해지고 100이 된다. 이는 d이므로 오른쪽으로 3만큼 밀린 것과 같은 양상을 보인다. 만약 z일 때, id의 길이가 4라면 122+(4-26)=100이고, 이는 a의 아스키코드값보다 크다. 따라서 그냥 d가 되고 이는 z를 오른쪽으로 4만큼 민 것과 같은 효과이다. 이런 방식을 통하여 알파벳을 모두 바꿔 password를 암호화하고 저장한다.

<Check_id, Check_pass 함수>

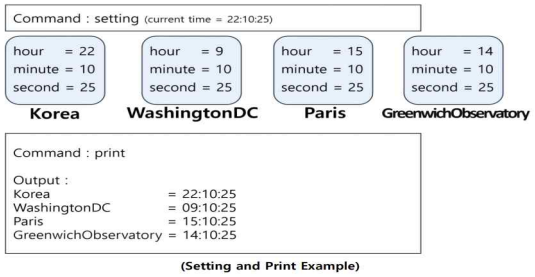
먼저, Check_id 함수는 반환형이 bool type으로 0부터 cnt_num-1까지 반복하여 해당 id와 같은 id가 있다면 true를 반환, 없다면 false를 반환하는 함수이다.

Check_pass 함수 또한 반환형이 bool type이다. 인자로 id와 password를 전달받아서 set_encInform 함수에서 사용했던 방식을 똑같이 사용하여 password를 암호화하고 암호화된 비밀번호와 객체 배열 원소들의 비밀번호를 비교하여 같은 비밀번호가 있다면 true, 없다면 false를 반환하도록 구현했다.

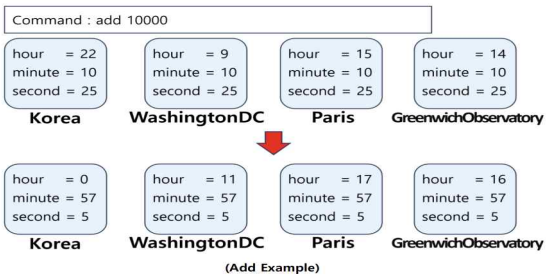
<Withdraw 함수>
인자로 전달받은 id와 반복문을 통해서 객체를 순회하며 각 객체의 id와 비교를 진행한다. id가 일치하는 객체가 있다면 해당 객체의 id와 password를 빈 문자열로 바꾸어서 탈퇴시킨다.
<saveInfo 함수>
프로그램이 종료될 때, 저장된 객체의 정보를 파일에 저장해야 하므로 ofstream 객체를 선언하여 입력을 받아오는 파일과 같은 파일을 연다. 반복문을 통해서 객체의 id, 빈칸, password 순서로 파일에 쓴 후 파일을 닫아서 파일에 정보를 저장한다.
<pass_condition 함수>
반환형이 bool type으로 알파벳 소문자의 개수, 숫자의 개수, 특수문자의 개수를 저장하는 int형 변수를 선언했다. 먼저 인자로 전달받은 password의 길이가 10 미만, 20 초과라면 true를 반환한다. for문을 통해서 password의 각 문자를 순회하는데, if ('a' <= pass[i]&& pass[i] <= 'z')를 통해서 소문자를 체크, else if ('1' <= pass[i]&& pass[i] <= '9')를 통해서 숫자를 체크하는데 else문에는 대문자와 특수문자만 남게 된다. 따라서 else문에 if ('A' <= pass[i]&& pass[i] <= 'Z')를 사용하면 대문자가 걸리게 되고 결국 else문에 특수문자만 남게된다. 따라서 알파벳 소문자, 숫자, 특수문자 중 하나라도 0이라면 true를 반환하고 모든 조건을 충족하면 flase를 반환하도록 구현했다.

Login command에서는 id 중복 체크, 해당 id가 있다면 password 체크, password까지 일치한다면 로그인 상태로 넘어간다. 이 로그인 상태에서는 탈퇴만 가능한데 탈퇴를 하고나면 해당 id와 password가 빈 문자열로 바뀌고 해당 정보를 저장한 후 프로그램이 종료된다.
Register command에서는 id 중복을 체크하여 해당 id가 이미 있다면 중복되었다는 메시지를 출력, 해당 id가 없다면 암호 조건을 판단한다. 암호 조건까지 만족한다면 비밀번호를 암호화하여 id와 같이 저장한다.
<고찰>
1차 과제에서 수행했던 암호화가 잘 기억나지 않아서 다시 블로그를 보며 공부했고 여러번 디버깅을 통해서 암호화를 구현했다. 그리고 비밀번호에서 특수문자를 구분하는 것을 해본 적이 없어서 어려웠는데 비밀번호는 알파벳 소문자, 대문자, 숫자, 특수문자로 구분되므로 소문자와 숫자는 if와 else if문과 아스키 코드를 이용하여 거를 수 있고, 대문자 또한 아스키 코드를 이용하여 거를 수 있으므로 else를 이용하여 나머지들은 모두 특수문자로 취급하는 방법을 배웠다.

<Assignment #3-4>



(Setting and Print Example)



(Add Example)

한국, 워싱턴 DC, 파리, 그리니치 천문대의 클래스를 구현하여 각 나라의 시간을 출력, 시간 추가 등의 기능을 할 수 있도록 구현하는 프로그램이다. 이 때 각 클래스는 Time 클래스는 public으로 상속받아서 구현한다.

<알고리즘>

각 클래스의 기초 클래스가 될 Time클래스는 멤버로 시, 분, 초를 int형 변수로 가진다. 멤버 함수 중 하나인 setTime함수는 인자로 시, 분, 초를 전달받아서 해당 객체 멤버 변수에 저장한다. addTime 함수는 인자로 초(sec)를 전달받아서 시, 분, 초로 바꾸어 더한다. while문을 이용하여 sec가 3600 미만이 될 때까지 sec에서 3600을 빼고 시를 증가시킨다 (1시간은 3600초). 또 while문을 이용하여 sec가 60 미만이 될 때까지 sec에서 60을 빼고 분을 증가시킨다 (1분은 60초). 이렇게 전달받은 초를 시, 분, 초로 나누어 저장하고 더해준다. 하지만 이렇게만 하면 원래 저장되어 있던 시, 분, 초와 더했을 때 범위를 초과할 수 있으므로 더해진 초가 60 미만이 될 때까지 60을 빼고 분을 증가시킨다. 똑같이 더해진 분이 60 미만이 될 때까지 60을 빼고 시를 증가시킨다. 마지막으로 날짜는 신경 쓰지 않으므로 해당 시를 24로 나눔으로써 범위를 제한시켜서 addTime 함수를 구현했다.
Time 클래스의 멤버 함수는 가상함수가 아니므로 Time 클래스를 상속받는 모든 클래스 내에서 Time 클래스와 같은 함수를 재정의하도록 구현했다. 특히 addTime 함수는 모든 클래스 내에서의 동작이 같으므로 따로 추가한 기능은 없다. 하지만 Korea 클래스를 제외한 클래스들은 모두 시차가 존재하므로 각 클래스의 setTime 함수 내에서 시차만큼을 빼주었다. 이때 해당 시간이 0 미만이라면 24를 더해서 구현했다.

<결과>

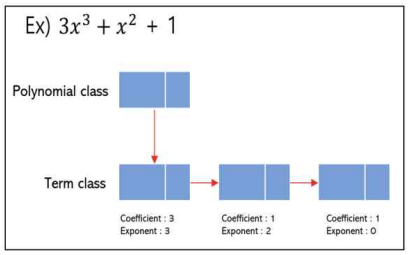
<고찰>

ctime 라이브러리를 통해서 srand의 시드 값으로 현재 시간을 넣어서 실행마다 다른 랜덤 값을 발생시키는 것만 해보았는데, time_t 데이터형을 통해서 현재 시간을 구하고 struct tm 구조체를 사용하여 구조체의 멤버인 tm_hour, tm_min 등을 사용하여 현재의 시, 분, 초를 구할 수 있다는 사실을 처음 알게 되었다. 또한 ctime 라이브러리에 이런 구조체나 멤버가

```
Command : setting
Command : print
Korea                = 0:43:33
washingtonDC         = 11:43:33
Paris                = 17:43:33
GreenwichObservatory = 16:43:33
Command : add 3600
Command : print
Korea                = 1:43:33
washingtonDC         = 12:43:33
Paris                = 18:43:33
GreenwichObservatory = 17:43:33
Command : add 3599
Command : print
Korea                = 2:43:32
washingtonDC         = 13:43:32
Paris                = 19:43:32
GreenwichObservatory = 18:43:32
Command : exit
exit the program
```

포함되어 있다는 사실도 알게 되었다.

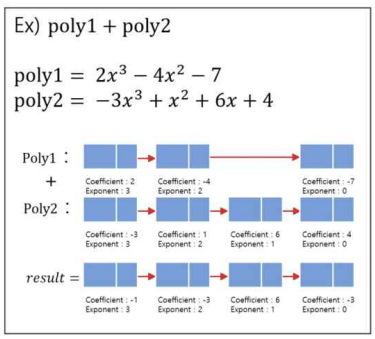
<Assignment #3-5>




```cpp
int main() {


    Polynomial poly;

    poly.addTerm(3, 3);
    poly.addTerm(1, 2);
    poly.addTerm(1, 0);

    poly.PrintPolynoimal();

    return 0;
}
```

```
3x^3 + x^2 + 1
```

(Example of Polynomial)




```cpp
int main() {
    Polynomial poly1, poly2, result;

    poly1.addTerm(2, 3);
    poly1.addTerm(-4, 2);
    poly1.addTerm(-7, 0);

    poly2.addTerm(-3, 3);
    poly2.addTerm(1, 2);
    poly2.addTerm(6, 1);
    poly2.addTerm(4, 0);

    std::cout << "Polynomial 1: ";
    poly1.PrintPolynoimal();
    std::cout << "Polynomial 2: ";
    poly2.PrintPolynoimal();

    result = poly1 + poly2;
    std::cout << "Polynomial result: ";
    result.PrintPolynoimal();

    return 0;
}
```

```
Polynomial 1: 2x^3 + -4x^2 + -7
Polynomial 2: -3x^3 + x^2 + 6x^1 + 4
Polynomial result: -x^3 + -3x^2 + 6x^1 + -3
```

(Addition of Polynomial Example)

5. Implement a class to represent a polynomial. The terms of the polynomial are represented in the form of a linked list. Each term stores a coefficient and a degree, both of which are integers. The class supports overloading of the + and - operators to create a new polynomial, and a 'calculate' member function that returns the calculated result for a given x value , and it also provides a 'differentiation' function to return a new polynomial.

   The polynomial is represented as follow:

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_2 x^2 + a_1 x + a_0$$

```
class Polynomial {
private:
    Term* m_pHead;

public:
    Polynomial();
    ~Polynomial();

    void addTerm(int coeff, int exp);
    void printPolynoimal();
    Polynomial operator+(const Polynomial&  poly);
    Polynomial operator-(const Polynomial&  poly);
    Polynomial differentiation();
    int calculate(int x);
}
```

```
class Term {
private:
    int m_Coefficient;
    int m_Exponent;
    Term* m_pNext;

public:
    Term();
    ~Term();
}
```

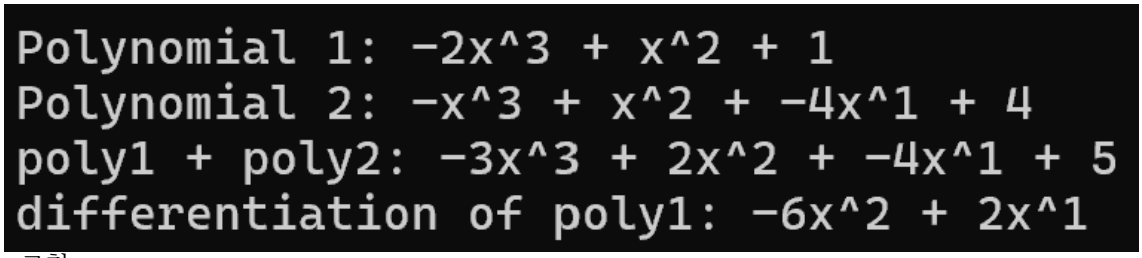
(Polynomial, Term Class Structure)

연결리스트를 이용하여 방정식의 각 미지수를 한 노드에 저장하도록 구현한다. 한 노드는 계수, 차수로 구성된다. 이런 미지수들을 연결리스트로 구성하여 하나의 방정식을 완성하도록 한다. +, - 연산자 오버로딩을 통해서 방정식끼리의 연산이 가능하도록 구현한다. 또한 한 방정식의 미분도 가능하도록 해야하며 인자로 정수를 전달하여 해당 방정식의 결괏값을 반환하는 함수도 필요하다.

<알고리즘>
하나의 미지수를 구성하는 Term 클래스의 멤버인 계수(coe), 차수(exp)는 private 멤버이므로 Term 클래스 내에 public 함수에서 직접 접근하여 멤버 변수를 새롭게 저장, 반환할 수 있도록 구현했다.
미지수를 추가하는 addTerm함수는 이전에 구현했던 연결리스트와 똑같이 마지막 노드의 뒤 노드로 새로운 노드를 이어 붙이는 방식으로 구현했다.
입력으로 같은 차수가 여러 개 들어온다면 m_head 노드부터 순회하여 입력받은 차수와 같은 차수를 가지는 노드에 도달할 때까지 다음 노드로 이동한다. 같은 차수를 가지는 노드를 찾았다면 해당 노드의 계수+입력받은 계수로 새로운 계수를 만들어 해당 노드의 계수로 다시 저장한다. 하나의 노드는 같은 차수를 가지므로 같은 차수가 들어온다면 더 이상 새로운 노드를 만들지 않는다. 따라서 함수를 종료하도록 구현했다.
<printPolynomial 함수>
방정식의 미지수를 출력하는 함수이다. 먼저 계수를 출력하는데, 계수가 1일 때는 차수가 0이라면 1을 출력하고, 차수가 0이 아니라면 1은 생략한다. -1도 마찬가지로 차수가 0이라면 -1을 그대로 출력하고 차수가 0이 아니라면 -만 붙여서 출력한다. 또한 0은 아예 출력하지 않도록 했고 나머지 숫자는 그대로 출력한다. 차수의 출력은 계수가 0이 아닐 때만 차수를 출력하도록 했다. 미지수의 사이에 + 기호를 붙여서 출력해야 하는데, 한 노드에서 coe가 0이라면 방정식에서는 출력하지 않지만, 해당 노드는 존재하므로 head부터 노드를 순회하여 다음 노드가 nullptr이 아닐 때까지 반복하며 해당 노드의 coe가 0이 아닐 때만 +를

출력하여 해당 방정식을 출력한다.
<+연산자 오버로딩>
두 방정식을 더하려면 두 미지수의 차수가 맞아야 덧셈을 진행할 수 있다. 따라서 각 방정식(리스트)의 head부터 끝까지 순회하며 차수를 비교한다. 한 방정식의 미지수가 차수가 높다면 해당 미지수를 새로운 리스트에 그대로 저장하고 다음 노드로 이동하여 차수를 내린다. 이를 두 미지수의 차수가 같을 때까지 반복한다. 차수가 같아지면 두 계수를 더하고 같은 차수를 방정식에 저장한다. 이렇게 하면 같은 차수의 계수들은 더해져서 새로운 방정식을 만들고 차수가 높은 미지수는 그대로 저장된다.
<-연산자 오버로딩>
A-B를 수행한다고 할 때, A를 1 방정식, B를 2 방정식이라고 서술하겠다.
-연산자도 +연산자와 마찬가지로 같은 차수의 미지수만 뺄셈이 가능하므로 각 리스트의 head부터 끝까지 순회하며 차수를 비교한다. 여기서 +연산자와 다른 점은 1 방정식의 차수가 2 방정식의 차수보다 낮으면 2 방정식의 계수를 저장해야 하는데 - 연산이므로 2 방정식의 미지수 계수의 부호를 바꾸어서 저장해야 한다. 이외에는 +연산자와 같은 로직으로 구현했다,
<differentiation 함수>
미지수를 상수, 1차, 2차 이상으로 구분한다. 상수는 미분하면 0이므로 아무것도 실행하지 않는다. 차수가 1이라면 계수를 그대로 유지하며 차수만 하나 줄어든다(차수가 0이 된다). 차수가 2 이상이라면 차수와 계수를 곱한 값을 새로운 계수로 저장하고 차수를 하나 줄여 새로운 노드(미지수)를 구성한다.
<calculate 함수>
정수를 인자로 전달받아 해당 정수의 결괏값을 반환하는 함수이다. 인자로 전달받은 정수에 미지수의 차수만큼 제곱을 하기위해서 pow_함수를 정의했다.
num^exp=num*num^(exp-1)이므로 exp가 0이라면 1을 반환, 0이 아니라면 exp-1을 인자를 전달하여 재귀함수의 형태로 호출한다. pow_함수를 이용하여 미지수에 인자로 전달받은 정수를 대입한 값을 구하고 이 값에 계수를 곱하여 반환하면 된다.

<결과>



<고찰>
+ 연산자 오버로딩을 통해서 두 객체를 더하고 미리 선언한 객체에 대입할 때, 소멸자가 호출된다. 하지만 생성자에서 동적 할당을 진행하지 않았으므로 소멸자에서는 아무 기능도 하지 않는다. 따라서 대입 연산자 오버로딩이 필요하지 않다.
또한 대입 연산자와 복사 생성자의 호출 시점에 차이가 있는데, 두 객체가 이미 생성&초기화 된 상태에서 대입을 진행하려면 대입 연산자가 호출되고, 객체가 새로 생성되는 시점에서 대입을 진행하려면 복사 생성자가 호출된다는 것을 알게 되었다.

<Assignment #3-6>

6. Implement the game "무궁화 꽃이 피었습니다 (Red Light, Green Light)". It's a game where you try to get to tagger without getting caught. Players can only move when the tagger is looking back, and if they move when the tagger is looking front, they lose the game. There are classes that represent the **player** and tagger states. The **Back** class represents the tagger's looking back, and the **Front** class represents the tagger's looking front.

States rules:
- A tagger repeats the behavior of looking back and front.
- Each behavior is held for as long as it outputs the phrases "Red Light!" and "Green Light!
- After entering the 'Front' state, the phrase 'Red Light!' appears immediately and lasts for 2-10 seconds. Then it will switch to the 'Back' state.
- Once in the 'Back' state, the 'Green Light!' phrase will be printed in 1 second increments. A random number of character from 2 to 6 will be displayed per second. When it reaches '!', it will switch to the 'Front' state.
- The initial state of the tagger is 'Back'

Player rules:
- Take 'Right arrow keyboard' input from a user and perform a single action (Moving forward).
- The tagger's state determines whether to move forward or end the game.

Screen composition:
- The tagger stays in a state for a random time whenever the state changes.
- The progress bar depends on how far forward you are, and replaces '@@' for each step forward.
- Shows your current progress and remaining distance, as shown in the example.

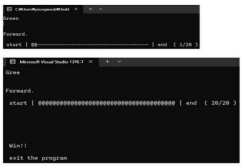
Termination condition:
- Win: If the player has moved forward 20 times.
- Lose: When action is detected while in the Front state.

```
class TaggerState {
public :
    virtual void toward() = 0;
}

class Back : public TaggerState {
public :
    void toward() override;
}

class Front : public TaggerState {
public :
    void toward() override;
}
```

```
class Player
{
private:
    TaggerState * state;
    int progress;

public:
    Player();
    ~Player();

    void SetState();
    void toward() {
        state->toward();
    }
    int GetProgress();
}
```

(TaggerState, Player Class Structure)

Success case



Unsuccess case

(Screen Composition)

"무궁화 꽃이 피었습니다" 게임을 구현하는 프로그램이다. 오른쪽 방향키를 눌러서 앞으로 이동이 가능하며, 술래가 뒤를 바라보고 있는 상태에서만 이동이 가능하다. 한번의 뒤를 바라보고 있는 상태에서 여러 번 이동이 가능하며, 20번을 이동하면 게임에서 승리한다. 술래가 앞을 바라보고 있는 상태에서 방향키를 누르면 게임에서 지게된다.
TaggerState 클래스를 public 상속받아서 Back, Front 클래스를 구현한다. TaggerState 클래스의 toward함수는 pure virtual로 선언되어 있으므로 Back, Front 클래스에서 함수를 똑같은 형태로 선언하되 다르게 정의하여 각 용도에 맞게 사용하면 된다.
<알고리즘>
Back State(클래스)는 술래가 뒤를 바라보고 있는 상태로, 방향키 오른쪽을 눌러서 앞으로 이동이 가능하다. string형 변수를 Green Light!로 초기화하여 1초에 2~6의 랜덤 값 만큼 문자를 출력한다. SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE) 함수를 이용하여 cmd창의 출력 위치를 바꿀 수 있는데 문자열이 얼마나 출력되었는지를 가리키는 int형 변수(cursor)를 선언하여 랜덤 값 만큼 for문을 통해서 출력한다. 이후 해당 cursor에 랜덤 값을

더하고 SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE) 함수의 인자로 cursor을 전달하면 해당 문자열이 출력된 위치부터 다시 출력을 시작하게 할 수 있다. 이렇게 1초마다 2~6 크기의 문자를 출력하고 마지막 !에 다다르면 Back state가 끝남을 의미하는 bool형 변수 flag_BtoF를 true로 초기화한다.
Front State(클래스)는 술래가 앞을 바라보고 있는 상태로, 이동이 불가능하고 만약 이동한다면 게임에서 지게된다. Front 클래스에서는 Red Light! 라는 문구가 2~10초의 랜덤 시간동안 유지된다. Sleep함수의 인자의 단위는 ms이므로 1000* 랜덤 값을 인자로 전달하여 2~10초 동안 Red Light! 문구가 유지되도록 구현했다.
Player 클래스는 멤버로 TaggerState형 객체, int pogress가 있으나 현재의 상태를 알고 있어야 할 필요가 있다고 판단하여 bool State_back이라는 변수를 추가했다. 이 멤버 변수는 back 상태라면 true, front 상태라면 false를 가진다.
또한 게임이 시작하면 back 상태에서 시작하므로 Player 클래스의 생성자에서는 state를 back으로 초기화했다.
<SetState 함수>
인자로 TaggerState형 객체를 전달받으므로 back이나 front 형도 전달될 수 있다. 이렇게 전달받은 상태를 새로 설정하는데 delete를 통해서 이전의 state를 삭제하고 새로운 상태를 할당한다.
<toward 함수>
멤버인 state는 TaggerState형 포인터이므로 back과 front를 모두 가리킬 수 있다. 따라서 state->towrd를 통해서 해당 상태에 맞는 오버라이딩된 toward함수가 호출된다.
또한 멤버 변수인 progress와 State_back을 반환하는 함수도 정의했다. 또한 인자로 bool형 변수를 전달받아서 상태를 저장하는 SetState_back 함수도 정의해놓았다.
<main 함수>
6번 문제에 이해를 돕는 코드에 나왔던 방법을 그대로 사용하여 1초마다 실행이 되도록 구현했다. 현재의 cursor만큼 @@를 출력하고 20에서 현재 cursor을 뺀 만큼ㅡ을 출력하여 현재의 진행 상태를 나타내주었다. GetState 함수를 이용하여 현재의 상태를 bool 자료형으로 반환받아서 true라면 back, false라면 front상태로 판단한다. 만약 back 상태에서 _kbhit 함수를 이용하여 입력이 들어왔음을 판단했다면 _getch 함수를 이용하여 int형으로 input을 저장한다. 이때, 방향키의 입력은 아스키코드로 224이므로 해당 input이 244라면 다시 _getch 함수를 이용하여 값을 재할당한다. 방향키 오른쪽은 아스키코드로 77이므로 한번 더 if문을 사용하여 77이라면(오른쪽 방향키가 입력되었다면) Forward. 라는 메시지를 출력하고 cursor을 하나 증가시킨다. 또한 Sleep(500)을 이용하여 0.5초를 대기 후 SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE) 함수를 이용하여 같은 행의 처음으로 커서를 이동시켜 공백을 출력하여 Forword. 메세지를 지운다. 여기서 _kbhit으로 입력을 판단할 때, while에 조건으로 집어넣어서 한 번의 Back 상태에서 여러 번 오른쪽 방향키를 누르면 여러 번 앞으로 이동하도록 구현했다. 이렇게 이동을 반복하여 cursor이 20이 된다면 게임에서 승리했으므로 프로그램을 종료한다.
Front 상태에서 _kbhit 함수를 통해서 입력을 확인하면 Back 상태에서와 똑같이 오른쪽 방향키임을 확인하고 오른쪽 방향키가 입력되었다면 게임에서 패배하므로 바로 프로그램을 종료하며 게임에서 졌음을 표기하는 문구를 출력하도록 구현했다.
<결과>

```
Tagger found out.
start | @@@@@@---------------------------------- | end  ( 3/20 )


Lose..

exit the program




start | @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ | end  ( 20/20 )


Win!!

exit the program
```

<고찰>
<conio.h>, <windows.h> 헤더파일을 처음 보고 써보아서 처음엔 굉장히 어려웠다. 또한 반복문을 1초마다 반복되도록 하는 방법도 이번 과제의 예제를 보면서 처음 알게 되어서 어떻게 적용할지 굉장히 막막했다. 처음에는 cmd창에서 back, front state와 앞으로 이동하는 것이 따로따로 실행되는 것인줄 알았다. 하지만 앞으로 이동하는 것을 어떻게 cmd 창에서 나타냈는지 예제로 올려주신 코드를 이해할때까지 한줄씩 디버깅 해보았고 그 결과, while문을 통해서 계속 출력하면서 SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE) 함수를 이용하여 출력의 위치를 바꿔가며 출력하면 따로따로 돌아가는 것처럼 보일 수 있다는 것을 깨닫고 문제에 적용했다. 내가 지금까지 해왔던 코딩이랑 결이 다른 문제였다는 생각이 들었다.

<Assignment #3-7>

연결리스트를 사용하여 스택을 구현하는 프로그램이다. 스택이 기본적으로 가지는 push, pop, top, isEmpty, print 등의 함수를 멤버로 가지도록 클래스를 구현한다. 또한 템플릿을 사용하여 스택의 데이터가 여러 가지 자료형으로 정의될 수 있도록 구현한다.

<알고리즘>
이전 과제에서의 Node는 구조체를 이용하여 구현했는데, 이번 문제에서 Node는 클래스에 정의 되어있다. m_Next와 Data가 private 멤버이므로 이전 과제에서 사용했듯, node->data처럼 접근하여 멤버를 수정하거나 값을 얻어올 수 없다. 따라서 Node 클래스

7. Implement a stack class using linked list. The stack should include the following functionalities, and it should be able to store various types of data using templates

Command	Format	Description
push	push [element]	Adds an element to the top of the stack.
pop	Pop	Removes and returns the top element of the stack.
top	top	Returns the top element of the stack.
isEmpty	isEmpty	Checks if the stack is empty, returning true if empty and false otherwise.
print	print	Print the values of all nodes starting at the top of the stack.
exit	exit	Exit the program.

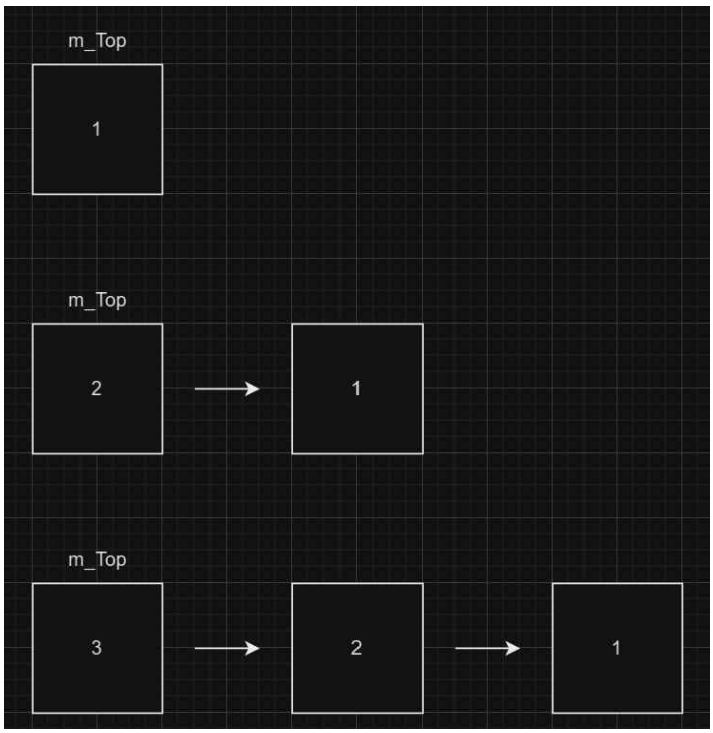
(Program Command)

```
template <typename T>
class Stack
{
private:
    Node<T>* m_Top;

public:
    Stack();
    ~Stack();

    void push(T data);
    T pop();
    bool isEmpty();
    T top();
    void print();
}
```

```
template <typename T>
class Node
{
private:
    Node* m_Next;
    T m_Data;

public:
    Node();
    ~Node();

}
```

(Stack Class Structure)

내에서 m_next와 data를 수정하고 반환할 수 있는 함수들을 정의했다.



내가 push할 노드인 temp의 데이터를 인자로 전달받은 T형 변수를 temp의 데이터로 저장한다.

스택은 FILO의 구조를 지니기 때문에, 처음 스택에 값을 push 할 때, m_Top이 nullptr 이라면 스택이 비어있으므로 m_Top를 temp로 저장한다. 스택이 비어있지 않다면 temp의 다음 노드를 m_Top노드로 할당하고 m_Top노드를 temp로 지정한다. FILO의 구조에 맞게 먼저 들어간 input은 나중으로 밀리는 형태이다.
pop함수를 구현하기 위해서 m_Top의 노드의 데이터를 저장하고 임시 노드인 temp에 m_Top 노드를 저장한다. 이후 m_Top=m_top->Getnext()를 통해서 m_TOP노드를 다음 노드로 옮긴 후 temp의 메모리를 해제하면 스택의 가장 위 요소가 반환, 삭제된다.
<결과>

```
push 3.2
push 3.5
push 4.5
print
4.5
3.5
3.2
pop
removed value is 4.5
print
3.5
3.2
pop
removed value is 3.5
print
3.2
pop
removed value is 3.2
print
Stack is empty
push 1.1
top
Value of top is 1.1
isEmpty
Stack is not empty
exit
Exit the program
```

<고찰>
Node를 항상 구조체로 정의했는데 클래스로 정의하여 스택을 구현하려고 하니까 private 멤버에 접근할 수 없어서 어떻게 할지 고민되었다. 수업 시간에 들었던 private에 대한 내용을 생각해보니 같은 클래스 내에서만 접근이 가능하므로 Node클래스 내에서 멤버변수에 접근할 수 있도록 함수를 선언하여 스택을 구현했다.
스택을 연결리스트로 구현해본게 처음이어서 헷갈렸는데, 수업 시간에 강의해주신 그대로 push할 때는 m_Top의 앞에 노드를 이어붙이고 m_Top 노드를 왼쪽으로 이동시키는 방법으로 구현했고, pop도 마찬가지로 m_Top 노드를 임시로 저장해놓고 m_Top 노드를 오른쪽으로 이동시켜 임시로 저장한 노드의 메모리를 해제하는 방법으로 구현하니 잘 동작했다. 또한 템플릿을 사용한지 오래되어서 문법이 헷갈렸는데 이번 과제를 통해서 앞으로 템플릿을 어렵지 않게 사용할 수 있겠다는 생각이 들었다.

<Assignment #3-8>

8. Implement a queue class using linked list. The queue should include the following functionalities, and it should be able to store various types of data using templates.

Command	Format	Description
enqueue	enqueue [element]	Adds an element to the back of the queue.
dequeue	dequeue	Removes and returns the front element of the queue.
front	front	Returns the front element of the queue.
isEmpty	isEmpty	Checks if the queue is empty, returning true if empty and false otherwise.
print	print	Print the values of all nodes starting at the front node of the queue.
exit	exit	Exit the program.

(Program Command)

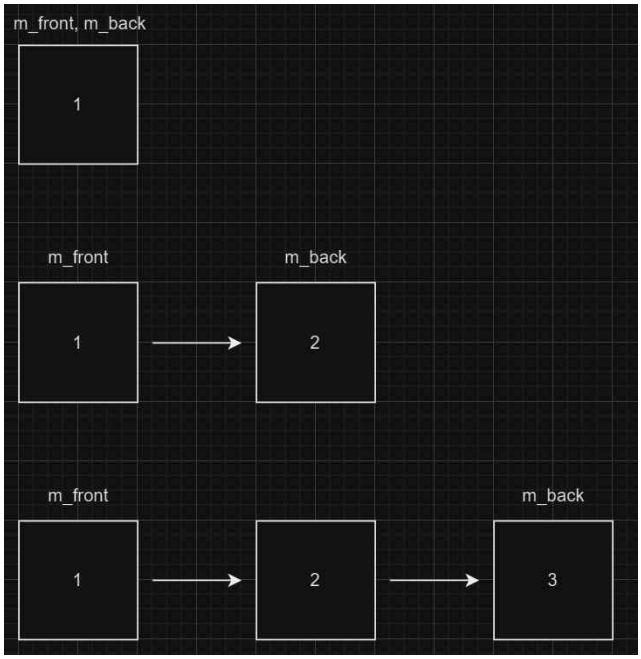
```
template <typename T>
class Queue
{
private:
    Node<T>* m_Front;
    Node<T>* m_Back;

public:
    Queue();
    ~Queue();

    void enqueue(T data);
    T dequeue();
    bool isEmpty();
    T Front();
    void print();
}
```

```
template <typename T>
class Node
{
private:
    Node* m_Next;
    T m_Data;

public:
    Node();
    ~Node();

}
```

(Queue Class Structure)

연결리스트를 이용하여 큐를 구현한다. 7번과 마찬가지로 템플릿을 사용하여 여러 가지 자료형으로 큐의 데이터가 저장될 수 있도록 구현하고 큐의 기본 기능인 inqueue, dequeue, front, isEmpty, print 함수도 클래스의 멤버 함수로 구현한다.

<알고리즘>
스택과 비슷하나 큐는 FIFO의 구조를 지닌다. 따라서 새로 저장할 노드를 m_back 노드의 뒤 노드로 연결하고 m_back 노드를 그 다음 노드로 옮기는 방식으로 enqueue 함수를 구현한다.
dequeue는 7번 문제의 pop과 비슷한 방식으로 m_front 노드를 임시로 다른 노드에 저장해놓고 m_front를 다음 노드로 옮긴다. 이후 임시로 저장한 노드의 메모리를 해제하여 dequeue의 기능을 구현했다.



<결과>

```
enqueue 1
enqueue 2
enqueue 3
print
1
2
3
dequeue
Value of front is 1
print
2
3
front
Value of front is 2
print
2
3
isEmpty
Queue is not empty
exit
Exit the program
```

<고찰>

7번 문제에서 클래스로 구현된 Node와 템플릿에 관한 어려움을 해결하고 8번 문제를 풀었기 때문에 쉽게 구현에 성공했다.

<Assignment #3-9>

9. Implement a Binary Search Tree (BST) class using a sorted array, the sorted array takes input from the user and is a positive integer (between 10 to 99). you can use the following method:

   Step 1. Set the middle element as the root node.
   - Middle element is ((start + end) / 2) index node.

   Step 2. The left sub-array becomes the left child of the root node, and the right sub-array becomes the right child of the root node.

   Step 3. Repeat the above process recursively.

Print Rules:

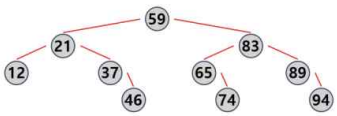
- Nodes with different depths are separated by three spaces.
- Each line contains one node. If either the left or right node is missing, a blank node should be printed, represented by two blank spaces.
- The left and right nodes of a node should be located at the same distance.

```
template <typename T>
class BST
{
private:
    Node* m_root;

public:
    BST();
    ~BST();

    void build(T arr[ ], int start, int end );
    void insert(T n, int depth);
    void printTree();
}
```

```
template <typename T>
class Node
{
private:
    Node* m_left;
    Node* m_right;
    T m_Data;
    int depth;

public:
    Node();
    ~Node();
}
```

(BST Class Structure)

```
input array = 12 21 37 46 59 65 74 83 89 94
            94
         89
      83
            74
         65
59
            46
         37
      21
            12
```



(PrintTree Example)

이진 탐색을 할 수 있는 트리인 이진탐색트리(BST)를 클래스 템플릿을 이용하여 구현하는 문제이다. 이진탐색트리는 이진 탐색을 수행하므로 원소들은 정렬된 상태로 입력이 들어온다. build 함수에 인자로 들어오는 start와 end의 중간 값인 mid를 root node로 설정하고 해당 루트 노드보다 작은 값은 왼쪽 자식노드, 루트 노드보다 큰 값은 오른쪽 자식노드로 지정한다. 이 과정을 재귀적으로 반복하여 BST를 구현한다. 루트 노드의 깊이를 0으로 하고 하나씩 내려갈수록 깊이가 1 증가한다.

출력 양식: 배열의 깊이는 1당 3개의 공백으로 출력, 한 줄에는 하나의 노드만 출력한다. 하나의 노드에 자식 노드가 하나만 존재한다면 비어있는 노드는 공백 2칸으로 대체하여 출력하도록 한다.

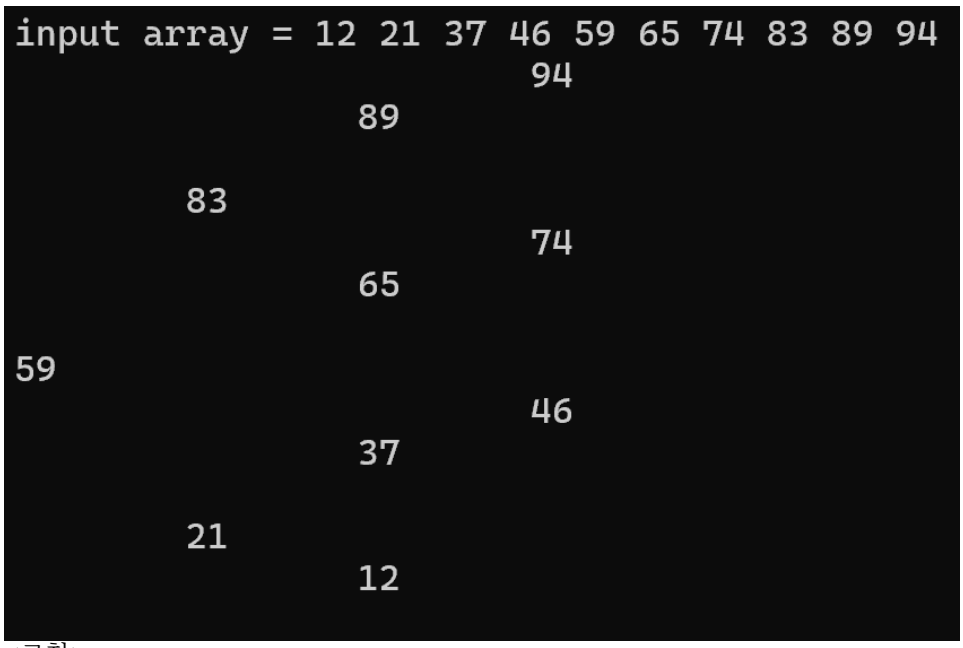
<알고리즘>

build 함수의 인자로 배열, start, end의 인덱스가 들어오는 것을 보고 이전 과제에서 구현했던 이진 탐색과 비슷한 방법을 생각했다. start가 end보다 작거나 같을 때만 실행하며 맨 처음 실행하면 생성자를 통해서 m_root가 nullptr로 초기화 되어있다. 따라서 m_root가 nullptr이라면(초기의 상태에서 m_root 노드가 없다면) m_root함수를 내가 저장할 노드(node)로 초기화하고 왼쪽, 오른쪽 자식 노드를 nullptr로 초기화한다.

m_root이 nullptr이 아니라면 루트 노드가 존재하므로 루트 노드부터 해당 노드를 저장할

위치를 탐색할 ptr 노드를 선언하고 루트 노드로 선언한다. node의 데이터와 ptr의 데이터를 비교하여 node의 데이터가 더 작다면 ptr을 왼쪽 자식노드로 이동, 더 크다면 ptr은 오른쪽 자식 노드로 이동하는 과정을 반복한다. 이때 한번의 실행마다 깊이가 1씩 증가한다. ptr이 nullptr이 아닐 때까지 반복하므로 반복을 마친 후, ptr은 내가 저장할 노드의 부모 노드가 된다. 이를 temp라는 새로운 노드로 저장하여 다시 temp와 node의 데이터 값을 비교한다. node의 데이터가 더 작다면 temp의 왼쪽 자식노드로 저장하고, 더 크다면 temp의 오른쪽 자식노드로 저장한다. 이렇게 저장하는 코드를 구현한 후, 재귀를 이용해야 하므로 build(arr, start, mid-1)을 전달하여 왼쪽 서브 트리를 재귀적으로 구성하고 build(arr, mid+1, end)을 전달하여 오른쪽 서브 트리를 재귀적으로 구성하도록 구현한다.

트리는 가장 오른쪽 맨 아래 노드부터 출력하므로 재귀를 이용하여 출력한다. 함수의 구조는, 인자로 전달받는 node의 오른쪽 자식노드를 먼저 전달하고 함수를 호출하여 오른쪽 서브 트리부터 출력하도록 한다. for문을 이용하여 해당 노드의 깊이 만큼 3칸의 공백을 출력하고 해당 노드의 데이터를 출력한다. 이때 해당 노드의 왼쪽 자식 노드가 nullptr이라면 (해당 노드의 깊이+1)*3 만큼 공백을 출력하고 비어있는 노드를 공백 2칸으로 대체하여 출력한다. 오른쪽 노드가 nullptr일 때도 마찬가지로 (해당 노드의 깊이+1)*3 만큼 공백 출력, 비어있는 노드는 공백 2칸으로 대체 출력한다.
<결과>



<고찰>
트리 구조를 처음 구현해 보았는데, 특히 재귀를 사용하여 트리를 구성하는 과정이 굉장히 어려웠다. 하지만 이진탐색트리라는 점에서 이전에 구현했던 이진 탐색에서 사용했던 방식으로 (start, mid-1), (mid+1, end)로 나누어 왼쪽, 오른쪽 서브 트리로 나누어서 구현하는 방식을 사용했다. 또한 트리를 출력하는 과정에서도 재귀를 사용해야 하는데 오른쪽 노드부터 해당 노드가 nullptr이 아닐 때, 오른쪽 자식 노드를 전달하여 함수를 호출, 해당 노드 출력, 왼쪽 자식 노드를 전달하여 함수를 호출하는 방식으로 함수를 구현하면 가장 아래, 오른쪽 노드가 먼저 출력되고, 그 노드의 부모 노드로 올라와 그 부모 노드를 출력,

왼쪽 자식 노드로 내려가 출력, 부모 노드의 부모 노드로 올라가서 출력...을 반복하여 출력하도록 구현했다. 직접 트리를 그려보고 순회 과정을 일일이 살펴보며 구현했기에 더 기억에 잘 남는 문제다.