Assignment #3, Object Oriented Programming, Spring Semester, 2024

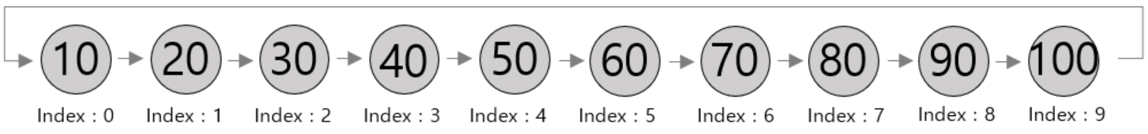
Due date: 2024.05.17

1. Implement a circular linked list that passes data through each node. The list consists of 10 nodes, with the last node linked to the first node. Each node in the list stores its own value and can move value to other nodes. The program follows the below rules:
   A. Circular linked lists can only send value in one direction.
   B. Nodes have two fields Its own value, and a pointer to the next node.
   C. The user can set the value of the node to any value (between 0 to 100).
   D. Users can transfer the value of a specific node to another node in the list.
   E. There is a certain probability that the connection will be lost while sending value.
      - If the transfer process detects a disconnect, notify the user with an error message.
      - There is a 10% probability of being disconnected.

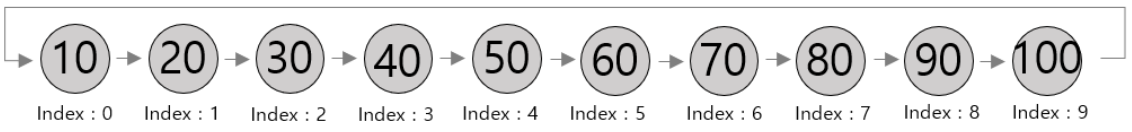
Command	Format	Description
initialize	initialize [values]	Initializes all values of a circular linked list.
transfer	transfer [A] to [B]	Transfer the value of the node at index A to the node at index B. If the transfer is not possible, print the location of the disconnection and exit the program.
print	print	Print the values of all nodes in the list, starting at the head.
exit	exit	Exit the program.

(Program Command)

```
Command : initialize 10 20 30 40 50 60 70 80 90 100
```



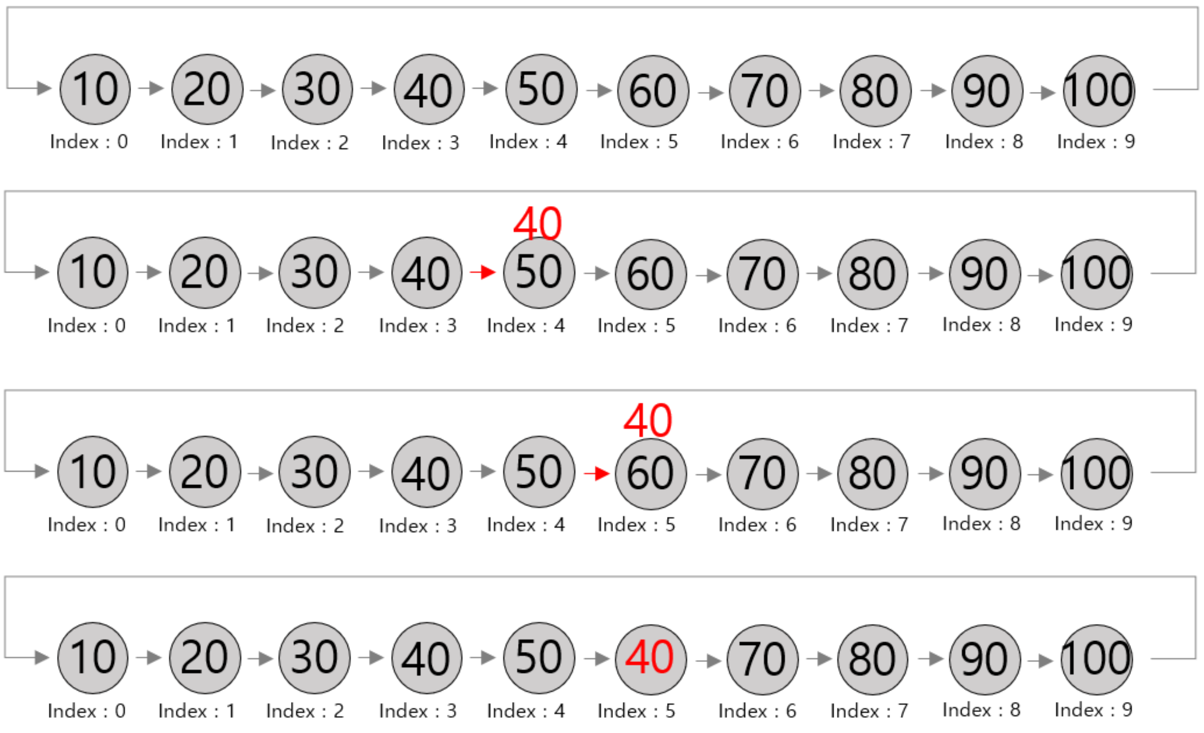
(Initialize Example)



```
Command : print
Output    : 10 20 30 40 50 60 70 80 90 100
```

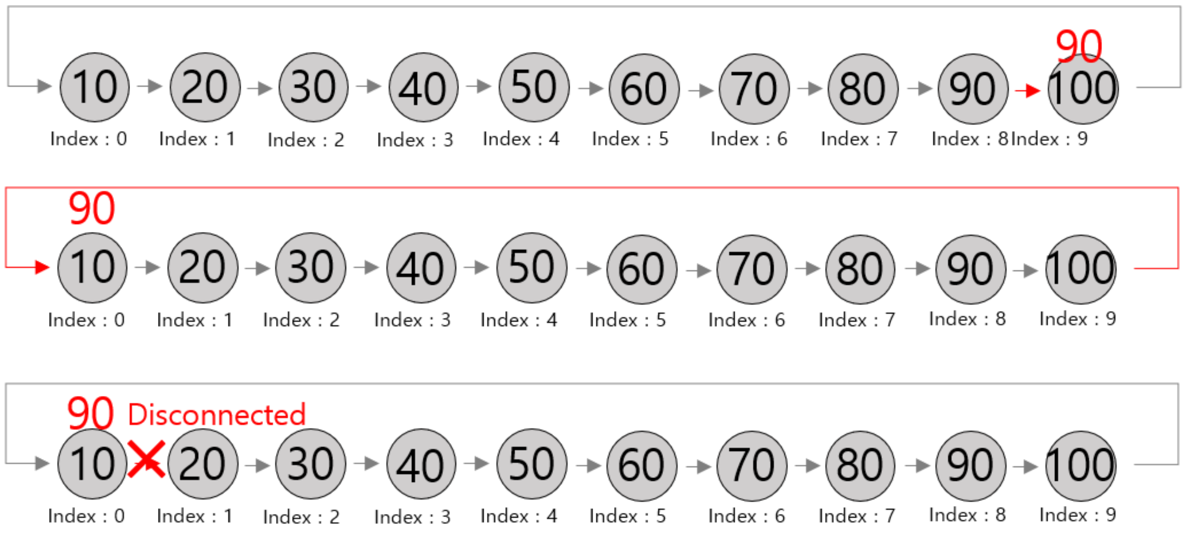
(Print Example)

```
Command : transfer 3 to 5
```



(Transfer Example 1 (case that won't disconnect))

```
Command : transfer 8 to 2
Output    : Detected a disconnection between 0 and 1
```



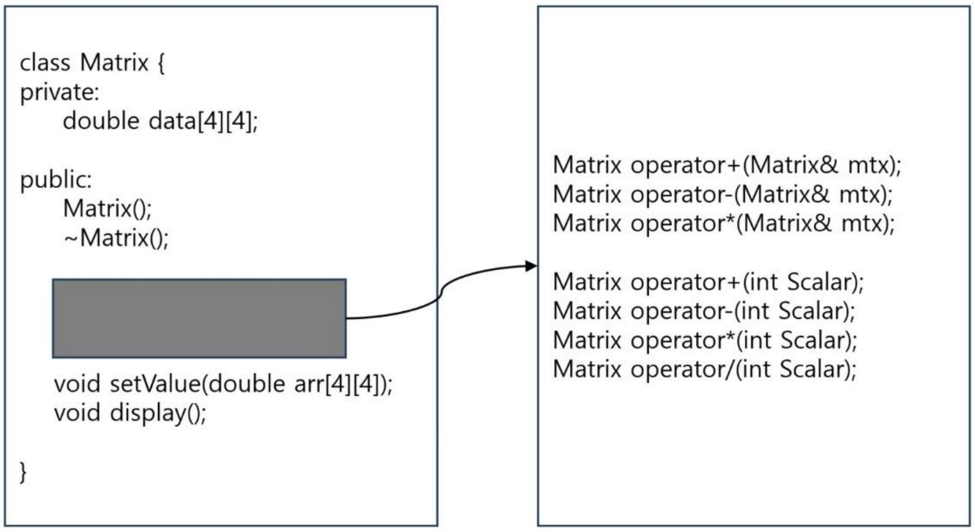
(Transfer Example 2 (case of disconnect))

Assignment #3, Object Oriented Programming, Spring Semester, 2024

Due date: 2024.05.17

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.

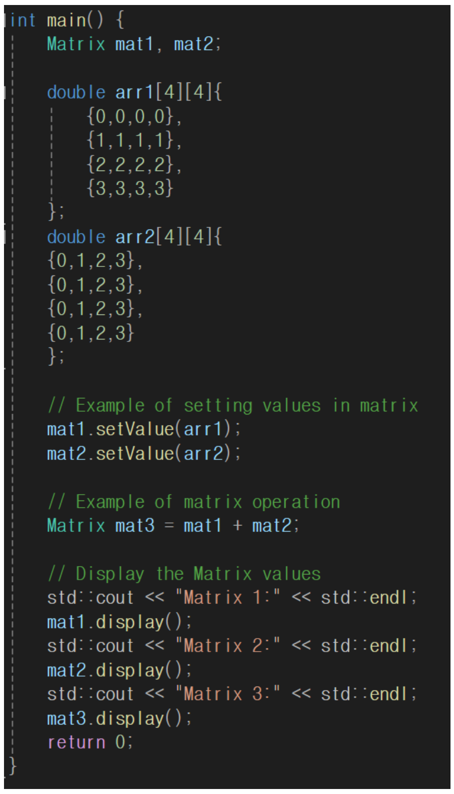
    - Binary Operations
        - "Matrix – Matrix": + (addition), - (subtraction), * (multiplication)
        - "Matrix – Scalar": + (addition), - (subtraction), * (multiplication), / (division)



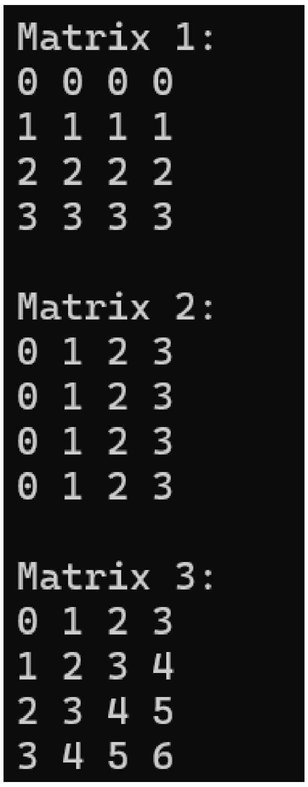
```
class Matrix {
private:
    double data[4][4];

public:
    Matrix();
    ~Matrix();

    void setValue(double arr[4][4]);
    void display();

}
```

```
Matrix operator+(Matrix& mtx);
Matrix operator-(Matrix& mtx);
Matrix operator*(Matrix& mtx);

Matrix operator+(int Scalar);
Matrix operator-(int Scalar);
Matrix operator*(int Scalar);
Matrix operator/(int Scalar);
```

(Matrix Class Sturcture)



```
int main() {
    Matrix mat1, mat2;

    double arr1[4][4]{
        {0,0,0,0},
        {1,1,1,1},
        {2,2,2,2},
        {3,3,3,3}
    };
    double arr2[4][4]{
    {0,1,2,3},
    {0,1,2,3},
    {0,1,2,3},
    {0,1,2,3}
    };

    // Example of setting values in matrix
    mat1.setValue(arr1);
    mat2.setValue(arr2);

    // Example of matrix operation
    Matrix mat3 = mat1 + mat2;

    // Display the Matrix values
    std::cout << "Matrix 1:" << std::endl;
    mat1.display();
    std::cout << "Matrix 2:" << std::endl;
    mat2.display();
    std::cout << "Matrix 3:" << std::endl;
    mat3.display();
    return 0;
}
```



```
Matrix 1:
0 0 0 0
1 1 1 1
2 2 2 2
3 3 3 3

Matrix 2:
0 1 2 3
0 1 2 3
0 1 2 3
0 1 2 3

Matrix 3:
0 1 2 3
1 2 3 4
2 3 4 5
3 4 5 6
```

(Example Program Code using Matrix Class)

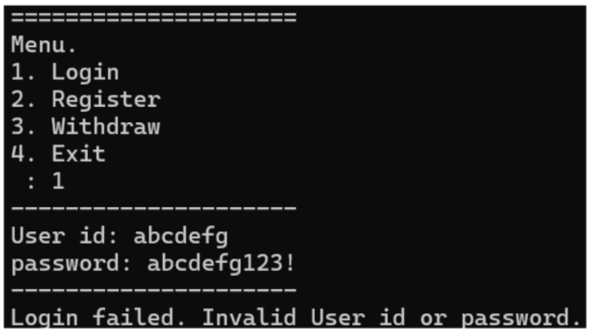
Assignment #3, Object Oriented Programming, Spring Semester, 2024

Due date: 2024.05.17

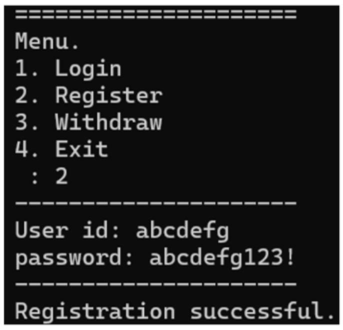
3. Implement a login program using a membership file. Implement a login program using a membership file. The program should maintain member information even after it is closed and reopened. The program follows the below rules:
   A. There should be four menu options (Login, Register, Withdraw, and Exit).
   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];
}
```

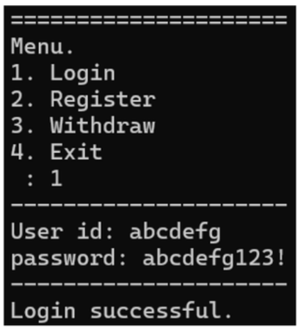
(Member struct)



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

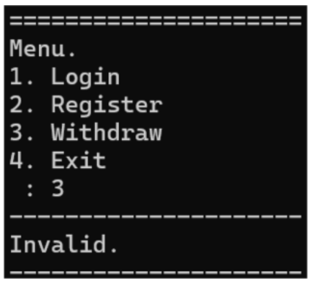


```
=====================
Menu.
1. Login
2. Register
3. Withdraw
4. Exit
 : 2
---------------------
User id: abcdefg
password: abcdefg123!
---------------------
Registration successful.
```

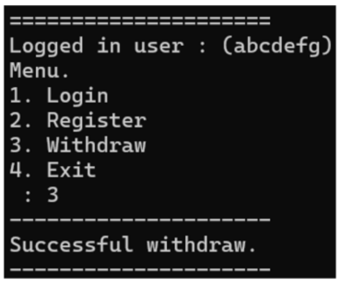


```
=====================
Menu.
1. Login
2. Register
3. Withdraw
4. Exit
 : 1
---------------------
User id: abcdefg
password: abcdefg123!
---------------------
Login successful.
```

(Login and Registration Example)



```
=====================
Menu.
1. Login
2. Register
3. Withdraw
4. Exit
 : 3
---------------------
Invalid.
---------------------
```



```
=====================
Logged in user : (abcdefg)
Menu.
1. Login
2. Register
3. Withdraw
4. Exit
 : 3
---------------------
Successful withdraw.
---------------------
```

(Withdrawal Example)

Dept. of Computer Engineering, Kwangwoon University

4. Implement a class to manage the time of each region. The time for each region is determined based on the standard time zone relative to the Greenwich Observatory. Show how the times of other regions are set based on the current time in Korea. This program has commands for setting, add, print, and exit. Use the 'ctime' library to read the current time.

Command	Format	Description
setting	setting	The time is set using a function that calculates the current time.
add	add [second]	Add the entered time to the set time.
print	print	Print the time for each country.
exit	exit	Exit the program.

(Program Command)

```
class Time {
private:
    int hour;
    int minute;
    int second;

public:
    Time(int hour, int min, int sec);
    ~Time();

    void setTime(int hour, int min, int sec);
    void addTime(int sec);
    void printTime();

}
```

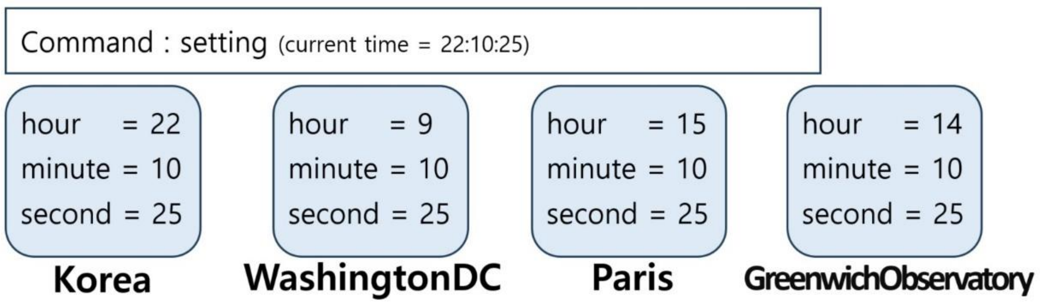
```
class Korea : public Time {

public:
    Korea(int hour, int min, int sec);
    ~Korea();
}
```

```
class WashingtonDC : public Time {

public:
   WashingtonDC(int hour, int min, int sec);
    ~WashingtonDC();
}
```

```
class Paris : public Time {

public:
    Paris(int hour, int min, int sec);
    ~Paris();
}
```

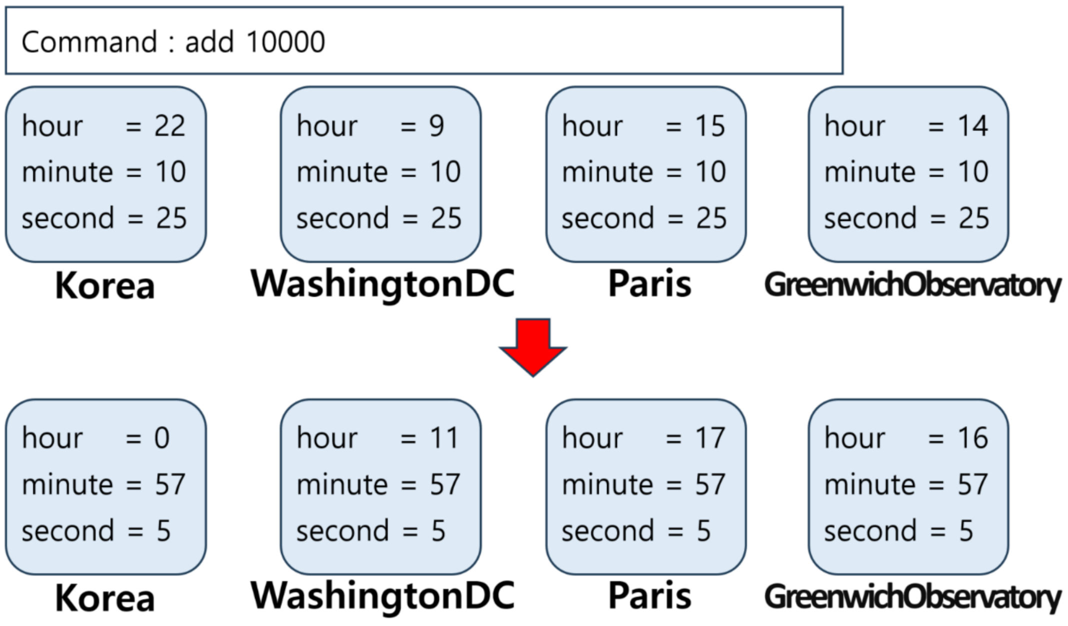
```
class GreenwichObservatory : public Time {

public:
    GreenwichObservatory(int hour, int min, int sec);
    ~GreenwichObservatory();
}
```

(Time, Each Country Class Structure)



```
Command : print

Output :
Korea                = 22:10:25
WashingtonDC         = 09:10:25
Paris                = 15:10:25
GreenwichObservatory = 14:10:25
```

(Setting and Print Example)



(Add Example)

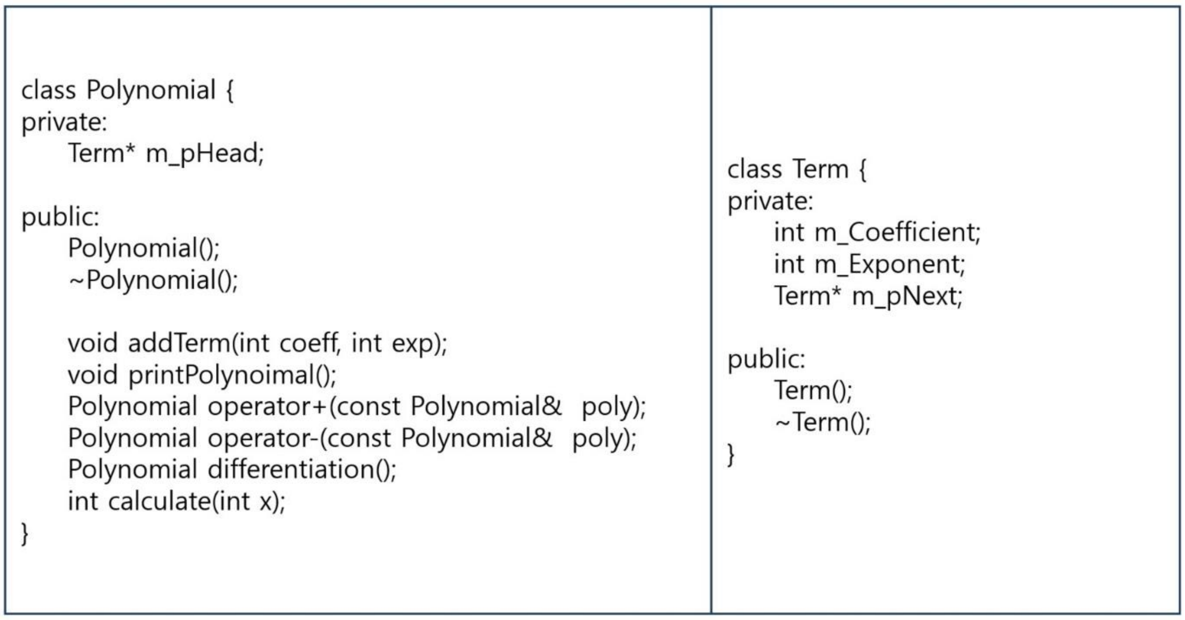
Assignment #3, Object Oriented Programming, Spring Semester, 2024

Due date: 2024.05.17

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:

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



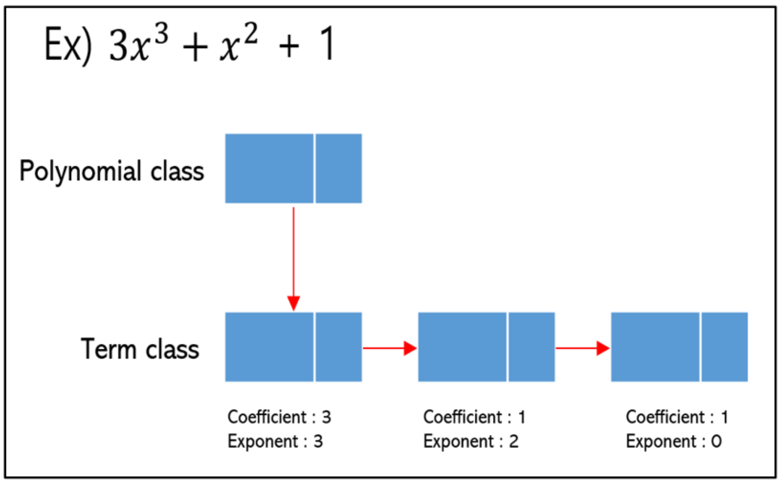
```
class Polynomial {
private:
    Term* m_pHead;

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);
}
```

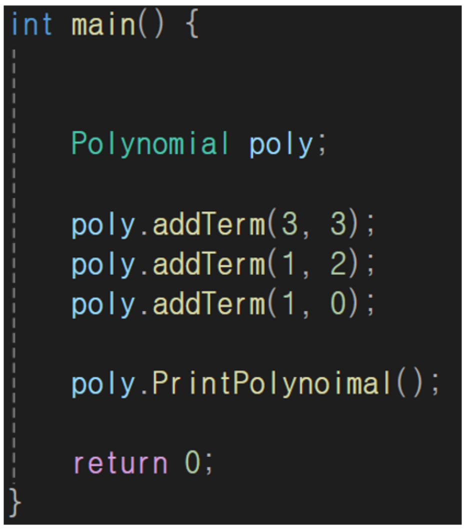
```
class Term {
private:
    int m_Coefficient;
    int m_Exponent;
    Term* m_pNext;

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

(Polynomial, Term Class Structure)

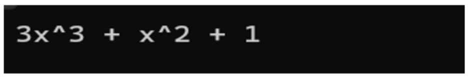
Assignment #3, Object Oriented Programming, Spring Semester, 2024

Due date: 2024.05.17



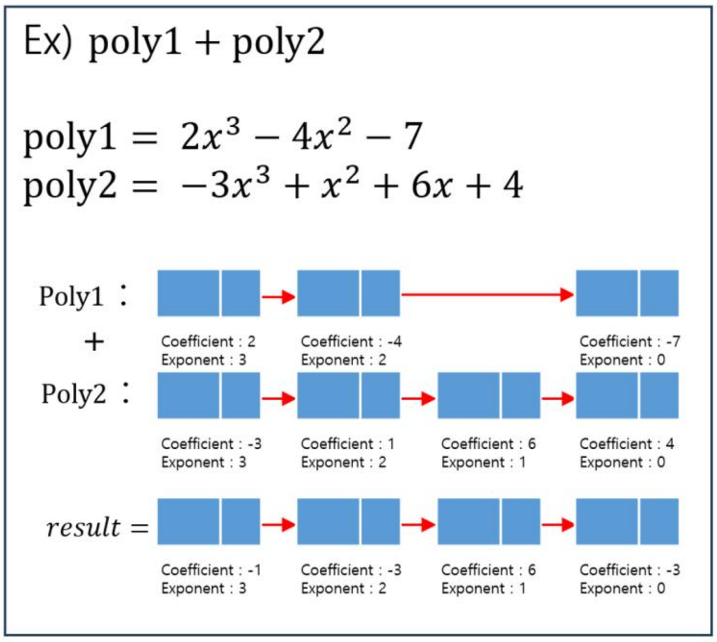


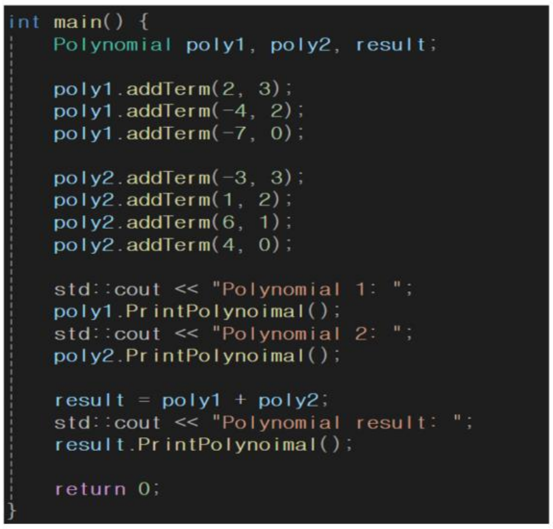
```
int main() {

    Polynomial poly;

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

    poly.PrintPolynoimal();

    return 0;
}
```



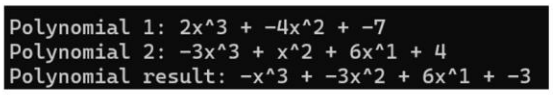
```
3x^3 + x^2 + 1
```

(Example of Polynomial)





```
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)

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.

Assignment #3, Object Oriented Programming, Spring Semester, 2024

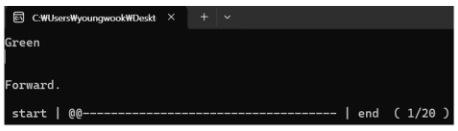
Due date: 2024.05.17

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

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

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

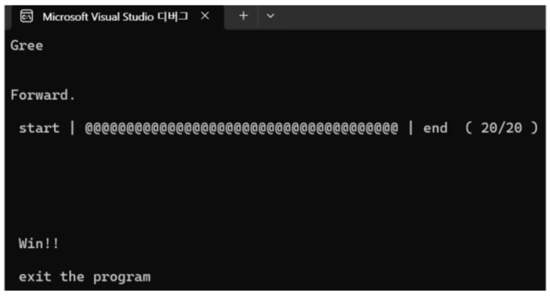
```
class Player
{
private:
    TaggerState * state;
    int progress;

public:
    Player();
    ~Player();

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

(TaggerState, Player Class Structure)

Success case

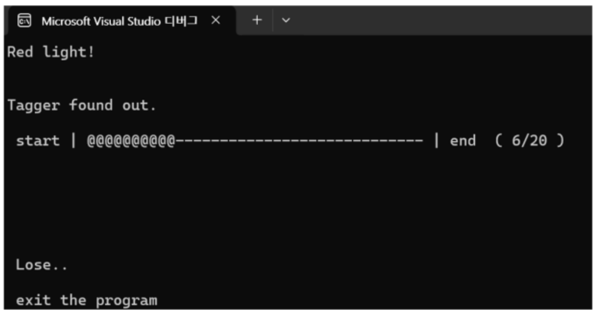


```
Green

Forward.
 start | @@-------------------------------------- | end  ( 1/20 )
```



```
Gree

Forward.

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




 Win!!

 exit the program
```

Unsuccess case



```
Red light!

Tagger found out.

 start | @@@@@@@@@@@@---------------------------- | end  ( 6/20 )




 Lose..

 exit the program
```

(Screen Composition)

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<T>* m_Next;
    T m_Data;

public:
    Node();
    ~Node();

}
```

(Stack Class Structure)

Assignment #3, Object Oriented Programming, Spring Semester, 2024

Due date: 2024.05.17

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<T>* m_Next;
    T m_Data;

public:
    Node();
    ~Node();

}
```

(Queue Class Structure)

Assignment #3, Object Oriented Programming, Spring Semester, 2024

Due date: 2024.05.17

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.

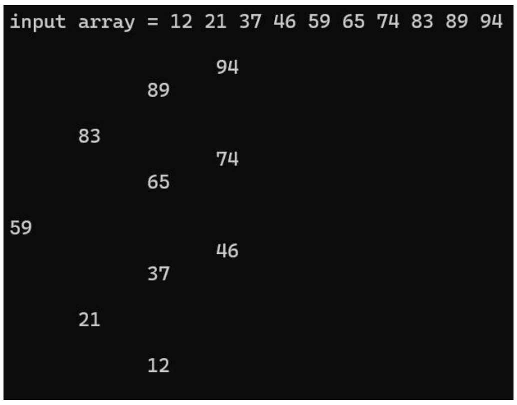
Print Rules:

- 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.

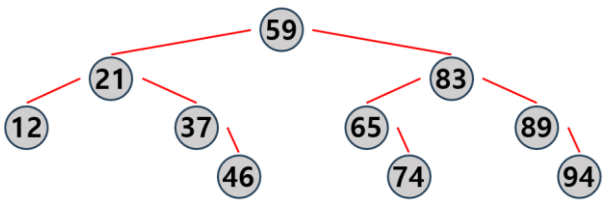
```
template <typename T>
class BST
{
private:
      Node<T>* m_root;

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<T>* m_left;
      Node<T>* 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)

Dept. of Computer Engineering, Kwangwoon University