OOP - Midterm 1 - 2025.04.11

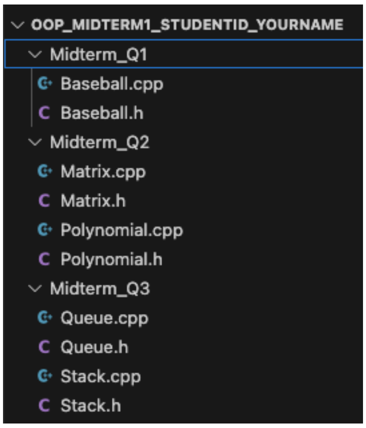
Exam Time: 18:30 ~ 21:20

There are three questions. You must finish all the questions.

Note: This specification contains 20 pages. Please make sure to read all of them carefully.

Rules

1. Put your student ID card on the table.
2. If you want to go to the toilet, raise your hand and tell TA.
3. The main.cpp file for each question:
   It is only a basic test for your code. We will also test your code with other test data.
4. You can start to upload your answer files after **20:45** and before the deadline.
5. TA will not accept any reason for uploading files too late except e3 is crashed.
6. Submit your files in the following structure:



7. You must use the template to do this midterm.
8. You are not allowed to include other libraries except for the ones specified in the question.
9. You must follow the output format.
10. If your source code cannot be built, your score is 0.
11. If you cheat, your score is 0.

Q1. Baseball ! (20%)

A baseball field has four bases in total: first base, second base, and third base, home base.

Rules

- Do not change the names of the functions and the class.
- Do not change the print function to avoid generating incorrect results.
- You must use private member variables to implement this class and can't add other variables in this class. (i.e. Access to member variables from external sources is restricted.)

Each baseball game consists of 9 innings, with each inning allowing 3 **outs**. When there are 3 outs, the inning ends. Once all 9 innings are completed, the game is over. Each batter will have a specific outcome during their plate appearance (PA), and both the batter and runners will proceed based on the result. Runners should pass first base, second base, third base and home base sequentially, when a runner successfully returns to home base, the team will get 1 point.

Each plate appearance (PA) can result in one of the following outcomes:

- On-base outcomes:
  - **Single (1B)** : The batter reaches first base, runners on base advance one base.
  - **Walk (BB)** : The batter reaches first base, runners on base advance one base.
  - **Double (2B)** : The batter reaches second base at once, runners on base advance two bases.
  - **Triple (3B)** : The batter reaches third base at once, runners on base advance three bases. For example, if there is a runner on first base, they advance to home plate and score 1 point.
  - **Home run (HR)** : Both the batter and all runners on base return to home base.

- Out outcomes:
  - **Fly out (FO)**
  - **Ground out (GO)**

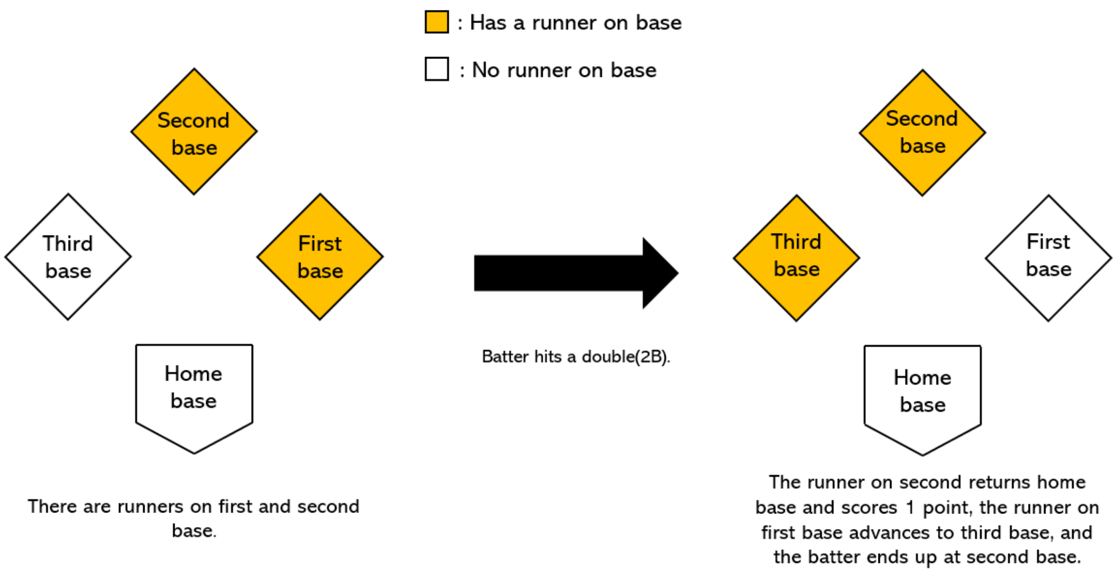
- **Strike out (SO)**

Traditional chinese version :

- 上壘結果（On-base outcomes）：
  - **一壘安打** ： 打者上到一壘，壘上的跑者各推進一個壘包。
  - **保送** ： 打者上到一壘，壘上的跑者各推進一個壘包。
  - **二壘安打** ： 打者上到二壘，壘上的跑者各推進兩個壘包。
  - **三壘安打** ： 打者上到三壘，壘上的跑者各推進三個壘包。舉例：若跑者在一壘，則跑者回到本壘得一分。
  - **全壘打** ： 打者與壘上的所有跑者皆回到本壘得分。

- 出局結果（Out outcomes）：
  - **飛球出局**
  - **滾地球出局**
  - **三振出局**

Example:



Operations	Description
A. (5%) void load_data(string)	Load data from input file.
B. (5%) int get_PA_result(string)	Converts a plate appearance (PA_result) into an integer representing base advancement: "1B" or "BB" returns 1 "2B" returns 2 "3B" returns 3 "HR" returns 4 "FO"," SO"," GO" returns 0 Note: Please output the result
C. (10%) Play()	Simulate a simplified baseball game according to input data and calculate the final point. • For each plate appearance (PA), convert the result into a number indicating how many bases to advance • If the PA result is 0, increase the out count • If the PA result is greater than 0, place the batter on base and move all existing runners accordingly • If a runner passes home base, a point is added When 3 outs are reached, reset all bases and the out count to simulate the next inning

(Question 1 provides a check-answer system that displays any errors in the command line.)

Error Version

```
Input Index: 1
[Test 1] : Line 5 mismatch:
   Your output    : 0 0 0 4 4 0 0 4 4 2 0 0 0 0 1 0 4 0 0 0 1 4 4 4 0 0 1 0 4 0 0 0 0 0 2 0 1 0 0 0 0 0
   Expected output: 0 0 0 1 1 0 0 1 1 2 0 0 0 0 4 0 1 0 0 0 4 1 1 1 0 0 3 0 1 0 0 0 0 0 2 0 3 0 0 0 0 0
Line 6 mismatch:
   Your output    : Point: 16
   Expected output: Point: 8
False
Input Index: 2
[Test 2] : Line 5 mismatch:
   Your output    : 0 0 1 0 4 0 0 0 0 4 4 4 4 0 2 0 0 0 0 4 0 0 0 0 0 0 0 4 0 4 2 0 0 0 0 0 0 4 4 0
   Expected output: 0 0 4 0 1 0 0 0 0 1 1 1 1 0 2 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 2 0 0 0 0 0 0 1 1 0
Line 6 mismatch:
   Your output    : Point: 10
   Expected output: Point: 5
False
```

Correct Version

```
Input Index: 1
[Test 1] : All lines match!
True
Input Index: 2
[Test 2] : All lines match!
True
```

Input

FO FO SO

1B 1B FO GO 1B 1B 2B FO

FO SO SO

HR SO 1B FO FO

FO HR 1B 1B 1B 1B GO GO

3B GO 1B GO GO

SO SO GO

2B SO 3B GO GO

GO FO SO

Output

```
==========================================================================================================================
PA:
FO FO SO 1B 1B FO GO 1B 1B 2B FO FO SO SO HR SO 1B FO FO FO HR 1B 1B 1B 1B GO GO 3B GO 1B GO GO SO SO GO 2B SO 3B GO GO GO FO SO
Base advancement:
0 0 0 1 1 0 0 1 1 2 0 0 0 0 4 0 1 0 0 0 4 1 1 1 1 0 0 3 0 1 0 0 0 0 0 2 0 3 0 0 0 0 0
Point: 8
==========================================================================================================================
```

Q2. Polynomial & Matrix (60%)

This question is divided into two parts.

The first part requires you to implement several operator functions for polynomial arithmetic. For more details, please refer to the polynomial operations described in the Part 1 instruction.
In the second part, you are required to implement operator functions for matrices to facilitate the evaluation of matrix polynomials. For more details, please refer to the matrix operations described in the Part 2 instruction.

The Part 1 folder includes Sample_Input.txt, Sample_Output.txt, and configure.txt for testing polynomial functionality only. You are required to test polynomial operations exclusively in this part.
The Part 2 folder includes Sample_Input.txt, Sample_Output.txt, and configure.txt for both polynomial and matrix operations. In this part, you will test the complete system, covering both functionalities.

Rules

- Do not change the names of the functions and the class.
- Please complete the functions in public.
- You must use private member variables to implement this class and can't add other variables in this class. (i.e. Access to member variables from external sources is restricted.)
- Do not use any built-in data structures such as std::queue or std::stack.

(Question 2 provides a check-answer system that displays any errors in the command line.)

Error Version: If your output is not correct, then you will see some error messages.
For Example:

```
Input Index: 1
Line 3 mismatch:
   Your output    : (2x^4 + 2x^3 + x^2) + (4x^4 + 4x^3 + 4x^2 + 3) = −2x^4 − 2x^3 − 3x^2 − 3
   Expected output: (2x^4 + 2x^3 + x^2) + (4x^4 + 4x^3 + 4x^2 + 3) = 6x^4 + 6x^3 + 5x^2 + 3
Line 5 mismatch:
   Your output    : (−2x^4 − 2x^3 − 3x^2 − 3) − (5x^4 + 4x^3 + 4x^2 + 3x + 3) = 3x^4 + 2x^3 + x^2 + 3x
   Expected output: (6x^4 + 6x^3 + 5x^2 + 3) − (5x^4 + 4x^3 + 4x^2 + 3x + 3) = x^4 + 2x^3 + x^2 − 3x
Line 7 mismatch:
   Your output    : (3x^4 + 2x^3 + x^2 + 3x) ∗ (x^4 + x^3 + x^2 + x + 1) = 3x^8 + 5x^7 + 6x^6 + 9x^5 + 9x^4 + 6x^3 + 4x^2 + 3x
   Expected output: (x^4 + 2x^3 + x^2 − 3x) ∗ (x^4 + x^3 + x^2 + x + 1) = x^8 + 3x^7 + 4x^6 + x^5 + x^4 − 2x^2 − 3x
False
```

Correct Version

```
Input Index: 1
All lines match!
True
Input Index: 2
All lines match!
True
```

Example:
From Part 1 of Q2, you may obtain a polynomial such as $P(x) = a + bx + cx^2 + dx^3 + ex^4$.
In Part 2, you will evaluate this polynomial by substituting a matrix into it.

Let $A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$ .You need to compute the matrix polynomial as following:

$P(A) = aI + bA + cA^2 + dA^3 + eA^4$, where I is 2x2 identity matrix.

Part 1 – Polynomial

The supported operations and their descriptions are listed below:

Operations	Description
A. (6%) operator+(poly)	Add another polynomial instance to this instance. $(2x^4 + 2x^3 + x^2) + (4x^4 + 4x^3 + 4x^2 + 3) = (6x^4 + 6x^3 + 5x^2 + 3)$
B. (6%) operator-(poly)	Subtract this instance by another polynomial instance. $(4x^4 + 4x^3 + 4x^2 + 3) - (2x^4 + 2x^3 + x^2) = (2x^4 + 2x^3 + 3x^2 + 3)$
C. (6%) operator*(poly)	Multiply this instance by another polynomial instance. $(x^4 + 2x^3 + x^2 - 3x) * (x^4 + x^3 + x^2 + x + 1)$ $= (x^8 + 3x^7 + 4x^6 + x^5 + x^4 - 2x^2 - 3x)$
D. (6%) operator<<(out, poly)	Output the polynomial to the output stream out. The format should follow standard mathematical notation. For example, the polynomial $(x^8 + 3x^7 + 4x^6 + x^5 + x^4 - 2x^2 - 3x)$ should be printed as: x^8 + 3x^7 + 4x^6 + x^5 + x^4 - 2x^2 - 3x

The following operations are provided for implementation. Their descriptions are listed in the table below:

Operations	Description
A. (0%) Polynomial()	Default constructor. Initializes a polynomial with all coefficients set to 0. The vector has size $SIZE = 11$, representing powers from x^0 to x^{SIZE-1}.
B. (0%) Polynomial(vector&)	Initialize the polynomial using the input coefficient vector. Automatically trims trailing zeros to maintain the correct degree of the polynomial.

Part 2 – Matrix

All matrix operations are guaranteed to be valid. All matrices are square (i.e., the number of rows equals the number of columns), and there is no need to handle invalid input or dimension mismatch.

The supported operations and their descriptions are listed below:

Operations	Description
A. (6%) setIdentity()	Set the current matrix as an identity matrix (only works for square matrices). All diagonal elements will be set to 1, others to 0.
B. (6%) evaluate()	Evaluate the matrix polynomial $P(A) = aI + bA + cA^2 + dA^3 + eA^4 + \cdots$ using the polynomial calculated in Part 1 and the current matrix A.
C. (6%) operator^(matrix)	Performs matrix exponentiation. Returns a new matrix raised to the specified power.
D. (6%) operator+=(matrix)	Add another matrix to this one in-place (element-wise).
E. (6%) operator*=(matrix)	Multiply this matrix by another matrix in-place.
F. (6%) operator*(int)	Return a new matrix where each element is multiplied by a given scalar integer.

The following operations are provided for implementation. Their descriptions are listed in the table below:

Operations	Description
A. (0%) clear()	Release the memory used by the matrix. Sets m_data to nullptr.
B. (0%) ~Matrix()	Destructor. Automatically calls clear() to release allocated memory.
C. (0%) init()	Allocate memory for the matrix based on row_size and col_size.
D. (0%) print()	Print the matrix in 2D format using cout. For debugging or display.
E. (0%) printPoly()	Print the internally stored polynomial using std::cout.
F. (0%) operator>>(in, matrix)	Read matrix data from file input stream. Initializes matrix and stores values row by row.
G. (0%) operator<<(out, matrix)	Write matrix data to file output stream in 2D format.

Input Format

Polynomial

A polynomial is represented by five integers, corresponding to the coefficients of the following form: $a + bx + cx^2 + dx^3 + ex^4$

Example:

0 0 1 2 2 Represents: $x^2 + 2x^3 + 2x^4$

3 3 4 4 5 Represents: $3 + 3x + 4x^2 + 4x^3 + 5x^4$

Matrix

A matrix is represented by two integers n and m, indicating the number of rows and columns respectively in an n × m matrix. n and m are 2 or 3.

The next n lines each contain m integers, representing the rows of the matrix.

Example:

If n = 2 and m = 2, and the next n lines are:

1 2

2 1

Then the matrix can be represented as $\begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$.

If n = 3 and m = 3, and the next n lines are:

1 2 3

2 1 2

3 2 1

Then the matrix can be represented as $\begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 2 \\ 3 & 2 & 1 \end{bmatrix}$.

(Input code is already provided. You only need to focus on implementing the logic.)

1. The first line begins with the character 'p', which indicates the first polynomial input. In this specification, we will temporarily refer to this polynomial as poly1.
2. The second line begins with the character '+', followed by another polynomial. You are required to add this polynomial to poly1.
3. The third line begins with '-', followed by a polynomial to subtract from the result of the previous operation.
4. The fourth line begins with '*', followed by a polynomial to multiply with the result of the previous operation.

The first character of each line indicates the operation to perform. While 'p' must appear first to define the initial polynomial, the order of the subsequent operations (+, -, *) may vary.
After completing all polynomial operations, you are required to use the resulting polynomial to evaluate a matrix polynomial**.**

5. The fifth line begins with 'm', followed by two integers n and m, indicating an $n \times m$ matrix. The next n lines contain the matrix elements row by row. You are required to evaluate the current polynomial as a matrix polynomial by substituting this matrix into it.

Input For Polynomial

p 0 0 1 2 2
+ 3 0 4 4 4
- 3 3 4 4 5
* 1 1 1 1 1

Output For Polynomial

```
Initial Polynomial = 2x^4 + 2x^3 + x^2

(2x^4 + 2x^3 + x^2) + (4x^4 + 4x^3 + 4x^2 + 3) = 6x^4 + 6x^3 + 5x^2 + 3

(6x^4 + 6x^3 + 5x^2 + 3) − (5x^4 + 4x^3 + 4x^2 + 3x + 3) = x^4 + 2x^3 + x^2 − 3x

(x^4 + 2x^3 + x^2 − 3x) ∗ (x^4 + x^3 + x^2 + x + 1) = x^8 + 3x^7 + 4x^6 + x^5 + x^4 − 2x^2 − 3x
```

Input For Polynomial and Matrix

p 0 0 1 2 2

+ 3 0 4 4 4

- 3 3 4 4 5

* 1 1 1 1 1

m 2 2

1 2 2 1

Output For Polynomial and Matrix

You can check more detailed in **Sample_output_1.txt** or **Sample_output_2.txt.**

```
Initial Polynomial = 2x^4 + 2x^3 + x^2

(2x^4 + 2x^3 + x^2) + (4x^4 + 4x^3 + 4x^2 + 3) = 6x^4 + 6x^3 + 5x^2 + 3

(6x^4 + 6x^3 + 5x^2 + 3) - (5x^4 + 4x^3 + 4x^2 + 3x + 3) = x^4 + 2x^3 + x^2 - 3x

(x^4 + 2x^3 + x^2 - 3x) * (x^4 + x^3 + x^2 + x + 1) = x^8 + 3x^7 + 4x^6 + x^5 + x^4 - 2x^2 - 3x

[[1, 2],[2, 1]]
[[8169, 8166],[8166, 8169]]
```

Q3. Stack & Queue (20%)

This question is divided into two parts.
In the first part, you are required to implement a stack.
In the second part, you must use the stack implementation from part 1 to simulate a queue using two stacks.

In Part 1, you are required to test only the stack functionalities.
In Part 2, you will test a queue that is simulated using two stacks.

The Part 1 folder includes Sample_Input.txt and Sample_Output.txt for testing stack functionality only. You are required to test stack operations exclusively in this part.
The Part 2 folder includes Sample_Input.txt and Sample_Output.txt for testing queue operations.

Rules

- You must implement this class using pointers. If you do not, you will receive zero points.
- Do not change the names of the class or its functions.
- Complete the implementation of the functions in the public section.
- You must implement the stack by yourself.
- Do not use any built-in data structures such as std::queue, std::vector, or std::stack.

(Question 3 provides a check-answer system that displays any errors in the command line.)

Error Version

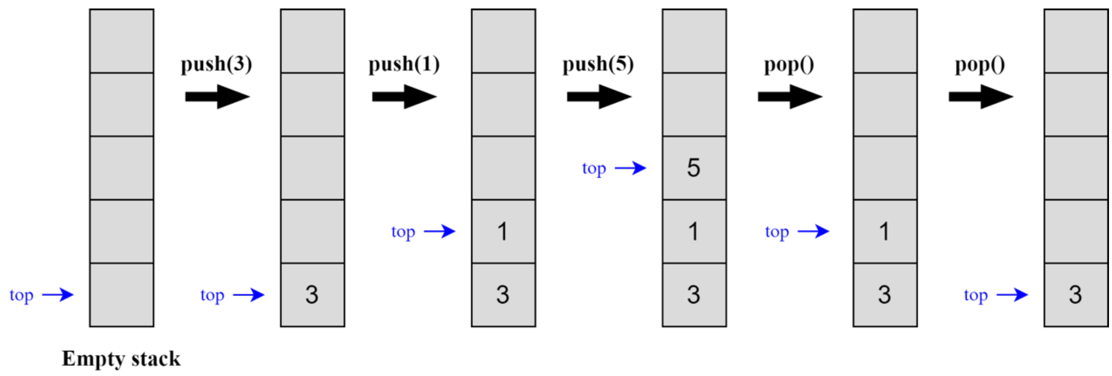
```
Input Index: 1
Line 3 mismatch:
   Your output    : The stack is Full.
   Expected output: 110 100 90 80 70 60 50 40 30 20 10
Line 4: Extra line in your output: 100 90 80 70 60 50 40 30 20 10
False
```

Correct Version

```
Input Index: 1
All lines match!
True
Input Index: 2
All lines match!
True
Input Index: 3
All lines match!
True
```

Part 1 – Stack

The following diagram illustrates the instruction flow of stack operations. The stack follows the LIFO (Last-In, First-Out) principle, where push operations add values to the top of the stack.



Assume all the input data are positive integers.
The supported operations and their descriptions are listed below:

Operations	Description
A. (2%) Stack()	Constructor. Initialize the stack with an empty array and sets the top index to -1.
B. (2%) push(x)	Push a new element x onto the stack if the stack is not full.
C. (2%) pop(&x)	Pop the top element from the stack and stores it in x if the stack is note empty. Return -1 if the stack is empty.
D. (2%) isFull(Type type)	Return true if the stack is full; otherwise, returns false. Only print "The stack is Full." if the type is ONLY_STACK and the stack is Full; otherwise, do not print anything.

The following operations are provided for implementation. Their descriptions are listed in the table below:

A. (0%) isEmpty()	Return true if the stack is empty; otherwise, returns false. Only print "The stack is Empty." if the type is ONLY_STACK and the stack is Empty; otherwise, do not print anything.
B. (0%) getTopValue()	Retrieve the top value of the stack and stores it in x. Returns -1 if the stack is empty.
C.(0%) size()	Return the number of elements currently in the stack.

Input For Stack

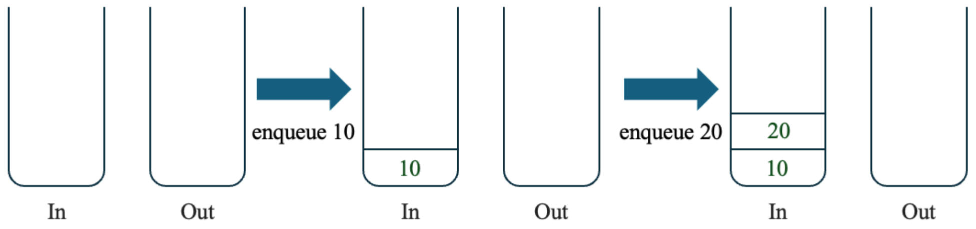
push 10
pop
pop
push 20
push 30
pop
push 40
push 50
push 10
push 60
push 70
push 80
push 90
push 100
push 110
push 120
push 130
push 140
pop
pop
pop
print

Output For Stack

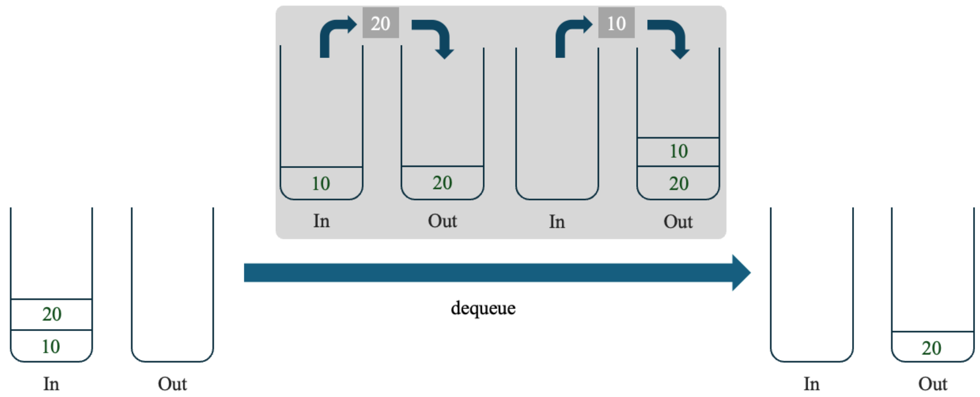
10
The stack is Empty.
30
The stack is Full.
The stack is Full.
120
110
100
90 80 70 60 10 50 40 20

Part 2 – Queue

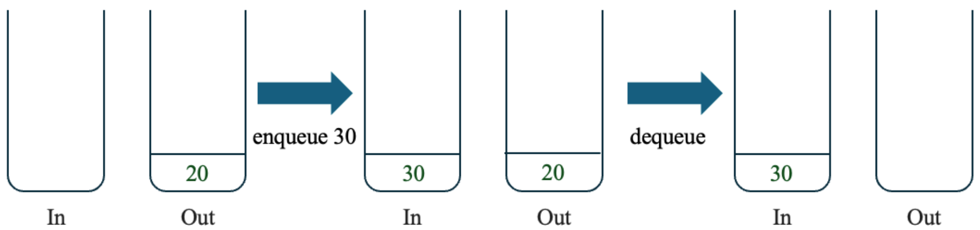
The following diagram illustrates the instruction flow of queue operations. The queue follows the FIFO (First-In, First-Out) principle.



The **In and Out stacks** are initially empty. First, an **enqueue 10** operation adds 10 to **the top of the In stack,** resulting in the middle state shown in the diagram. Next, an **enqueue 20** operation adds 20 **on top of the In stack**, resulting in the final state.



The left **In and Out stacks** are performing a **dequeue** operation. Since the **Out stack is empty**, all elements from the **In stack** are moved to the **Out stack** (as shown by the gray background flow). Then, the **top of the Out stack** is dequeued.



The **left In and Out stacks** are performing an **enqueue** 30 operation, which pushes 30 onto **the top of the In stack**, resulting in the middle state shown in the diagram. Next, a **dequeue** operation is performed. Since the **Out stack is not empty**, you can directly remove the top element from **the Out stack**.

The supported operations and their descriptions are listed below:

Operations	Description
A. (2%) enqueue(x)	Insert element x at the end of the queue. Internally, this pushes x to the input stack.
B. (2%) dequeue(&x)	Remove the element from the front of the queue and stores it in x. Internally, this pops from the output stack; if empty, elements are transferred from the input stack first. Return -1 if the queue is empty.
C. (2%) front(&x)	Retrieve the element at the back of the queue and stores it in x, without removing it. Return -1 if the queue is empty.
D. (2%) back(&x)	Retrieve the element at the back of the queue and stores it in x, without removing it. Return -1 if the queue is empty.
E.(2%) size()	Return the current number of elements in the queue.
F.(2%) print()	Print the elements in the queue in the correct front-to-back order.

The following operations are provided for implementation. Their descriptions are listed in the table below:

A. (0%) isEmpty()	Return true if the queue is empty; otherwise, returns false. Print "The queue is Empty." if the queue is empty.
B. (0%) isFull()	Return true if the queue has reached its maximum capacity (sum of input and output stacks equals capacity); otherwise, returns false. Print "The queue is Full." if the queue is full.

Input

enqueue 10
dequeue
dequeue
enqueue 20
enqueue 30
dequeue
enqueue 40
enqueue 50
enqueue 10
enqueue 60
enqueue 70
enqueue 80
enqueue 90
enqueue 100
enqueue 110
enqueue 120
enqueue 130
enqueue 140
dequeue
dequeue
dequeue
dequeue
print
front
back
dequeue
dequeue
dequeue
dequeue
dequeue
dequeue
dequeue
dequeue
print

Output

```
10
The queue is Empty.
20
The queue is Full.
The queue is Full.
30
40
50
10
60 70 80 90 100 110 120
60
120
60
70
80
90
100
110
120
The queue is Empty.
```