Infix to Postfix $A + D - Z * (C + E * U - (X * S + T))/B * Y \wedge Z$ as per algorithm

Infix/Input Char	Operator Stack	Output/Postfix String	Comments
A		A	
$+$	$+$	A	Should we solve +? Can't decide until we see next operato
D	$+$	AD	
$-$	$-$	$AD+$	Should we solve -? Can't decide until we see next operator. But now we may decide about the previous operator + on stack top. Since + and – have same precedence but associativity left to right, so, we pop + and append to postfix string.
Z	$-$	$AD+Z$	
$*$	$-*$	$AD+Z$	
$($	$-*($	$AD+Z$	
C	$-*($	$AD+ZC$	
$+$	$-*(+$	$AD+ZC$	
E	$-*(+$	$AD+ZCE$	
$*$	$-*(+*$	$AD+ZCE$	
U	$-*(+*$	$AD+ZCEU$	
$-$	$-*(-$	$AD+ZCEU*+$	Keep popping from the stack and append to output string until you find an operator of higher precedence
$($	$-*(-($	$AD+ZCEU*+$	
X	$-*(-($	$AD+ZCEU*+X$	
$*$	$-*(-(*$	$AD+ZCEU*+X$	
S	$-*(-(*$	$AD+ZCEU*+XS$	
$+$	$-*(-(+$	$AD+ZCEU*+XS*$	
T	$-*(-(+$	$AD+ZCEU*+XS*T$	
$)$	$-*(-$	$AD+ZCEU*+XS*T+$	
$)$	$-*$	$AD+ZCEU*+XS*T+-$	
$/$	$-/$	$AD+ZCEU*+XS*T+-*$	
B	$-/$	$AD+ZCEU*+XS*T+-*B$	
$*$	$-*$	$AD+ZCEU*+XS*T+-*B/$	
Y	$-*$	$AD+ZCEU*+XS*T+-*B/Y$	
$\wedge$	$-*\wedge$	$AD+ZCEU*+XS*T+-*B/Y$	
Z	$-*\wedge$	$AD+ZCEU*+XS*T+-*B/YZ$	
		$AD+ZCEU*+XS*T+-*B/YZ\wedge*-$	Input string finished so empty the stack and append the operators to output string

Objective:

- Implementing and analyzing an application of Queue. [*Process Scheduling [Round Robin Algorithm] and few competition problems.*

Task – 1:
A Double Ended Queue (**Deque (also called as DECK)**) is a queue in which you can insert and remove elements both at the *front* and at the *rear end* of the queue. You task is to create a class **DECK**, with the following interface:

- **void insertAtTail (T).**
- **void insertAtHead(T).**
- **T deleteAtTail( ).**
- **T deleteAtHead( ).**
- **isEmpty():**
- **isFull():**
- **int getCapacity()**
- **int getNoOfElements()**

Write routines to support the DECK that take constant amount of amortized time.

Task – 2:
In this task you are required to simulate ***Schedule***r (discussed in lecture as well). Scheduler is something, which is responsible for assigning the CPU time to the processes.

- The ***Scheduler*** should take in ***Processes*** and add them to a Queue.
- Once all the ***Processes*** are read, your program would execute each.
- But before executing any process, the program should print the name and other information about the process.
- Now it should start executing the processes in the Queue. For each process, execution just means that process stays at the head of the Queue until the time equal to its ***execution time*** has passed. The process is deleted from the list after that. At this moment your program should print that such and such process has finished execution. E.g. ***Messenger.exe, 6 completed execution***
- You must remember that it is a simulator. It means that you'll have to define your own timer, one which increments in unit steps.
- After the passage of every 10 time units, you program must stop the processing of current process and start the next process, the current process will be taken from the front of the queue to the end of the queue and the time remaining to complete process should be decremented.

Hints:
Design a class "Process" which will have following data members,

- int processId
- String processName
- int executionTime
- And a method to display its state.

Create a Queue and add the processes in it.
Process the "Process" at front of Queue and then remove it from front and add it at the end of queue if its execution time is still left and continue the same with next process in line.
Remove a "Process" from queue if it has left with '0' "execution time".
Repeat this process until the Queue is not empty.

* You have to create the following files and will have to code all the classes listed in these files.

```
//DRIVER.CPP
#include<iostream.h>
#include "schedular.h"

void main()
{
    Process arr[4] = { Process(1,"notepad",20), Process(13,"mp3player",5),
Process(4,"bcc",30), Process(11,"explorer",2) };
    Schedular s(arr, 4, 5);
    s.assignProcessor();
}

//PROCESS.h
class Process
{
public:
    int processId;
    string processName;
    int processExecTime;
    Process(int id=1, string pName="notepad", int burstTime=10 );
};
//SCHEDULAR.h
#include<iostream.h>
#include "CSTRING.h"
#include "queue.h"

class Schedular
{
    Process* processArray;
    int processArrayLength;
    int timeQuantum;

public:

    Schedular( Process* p, int length, int quantum );

    void assignProcessor();

};
//QUEUE.h
template<class T>
class Queue
{
//Queue interface as discussed in class
};
```

Task – 3: ***Computing Capital Gain***

When a person makes a profit on a purchase and eventual sale of a stock, he or she must pay taxes on the "capital gain", that is, the profit. (A capital loss can be used as a tax deduction.) However, sometimes a person purchases and sells shares of a stock over several transactions, and the capital gain depends on the interpretation of which shares were bought or sold. For example, suppose a person had the following transactions for stock of a certain company:

- bought 1000 shares at $3 each for $3000.
- bought 1500 shares at $2.5 each for $3750.
- sold 750 shares at $3.25 each for $2437.5.
- bought 1000 shares at $3.20 each for $3200.
- sold 2750 shares at $3.75 each for $10321.5

The total profit is (2437.50 + 10321.50) - (3000 + 3750 + 3200) = 5509. However, if a year ended after the third transaction (the first sale), then capital gains tax needed to be paid on the gain so far. But where did the 750 shares come from-- the original 1000 or the later 1500?

The standard way to compute capital gains is on a first-come-first-serve basis. Hence the 750 shares come out of the original 1000 shares, and their original price (their *basis*) is 2250, and so the capital gain was 187.5. If this person then sold another 750 shares the next year without buying any more, then 250 of those shares would be the remaining shares from the original purchase, and the other 500 would come from the 1500 bought in the second purchase. If sold at $3.50 per share, this would be a capital gain of (750 * 3.50) - (250 * 3 + 500 * 2.5) = 2625 - 2000 = 625.

Input and Output

Input will consist of a series of lines; first line contains the number of transactions. Each line will contain 'b' (bought) or 's' (sale) with next integer as number of shares sale/purchase and the next represents the rate of sale/purchase.

Output will consist of a value representing the capital gain.

Sample input

```
5
b 1000 3
b 1500 2.5
s 750 3.25
b 1000 3.2
s 750 3.5
```

Sample output

```
812.5
```

Queue ADT discussed in class

```
template<typename T>
class Queue
{
    int rear;
    int front;
    int capacity;
    int noOfElements;
    T * data;
    void reSize(int);
public:
    Queue();
    ~Queue();

    void enQueue(T);
    T deQueue();
    T getElementAtFront();
    bool isEmpty();
    bool isFull();
    int getNoOfElements();
    int getCapacity();
};
```

STL

Last but not the least, explore:

- https://www.cplusplus.com/reference/queue/queue/
- https://www.cplusplus.com/reference/deque/deque/