EE 441 HOMEWORK #2

Stack and Priority Queue with Linked Lists

Due: December 4, 2019, 23:59
For questions: anil.gokce@metu.edu.tr

Homework Statement:

You are expected to implement a `Stack` and a `PriorityQueue` Class which store the data in linked lists. Each member function that you should implement is explained by comments. You are supposed to use the `Node` class in the lectures. Note that these classes are to be implemented in different .h files. For each class, a template file is shared with you. You should add them to your project and write all methods in those files.

The class Declarations are as follows:

```
template <class T>
class Stack
{
    private:
        // a linked list to store the stack items
        Node<T>* stackTop;// top item of the stack, head pointer of
the linked list

    public:
        // constructor
        Stack(void);

        // stack access methods
        void Push(const T& item);
        T Pop(void); // deletes the top item and returns it
        T Peek(void); // returns the top item without deleting it

        // stack test and clear methods
        int StackEmpty(void) const; // return 1 if the stack is
empty
        void ClearStack(void); //dynamically allocated memory for
the nodes must be returned
        void ShowStack (void) const; // prints the content of the
     stack from top to bottom on the standard output (with cout)

};
```

```
template <class T>
class PQueue
{
    private:
      // a linked list to hold the queue items
Node<T>* queueFront;//pointer to the head of the list, head pointer
of the linked list
int count;//number of elements in the queue

    public:
      // constructor
         PQueue(void);

         // queue access methods
         void  PQInsert(const  T&  item);//inserts  the  item  in  the
correct position according to its priority.
         T PQDelete(void);//Deletes the highest priority item at the
queue front and returns it

         // queue access returns the highest priority element that is
in the queue in O(1) time
         T PQFront(void);

         // queue test methods
         int PQLength(void) const; // returns the length of the queue
         int PQEmpty(void) const; // returns 1 if the queue is empty
         void  PQClear(void);//dynamically  allocated  memory  for  the
nodes must be returned
void  ShowPQ  (void)  const;  //  prints  the  content  of  the  priority
queue from front to rear on the standard output (with cout)
};
```

You should (first implement, and then) use the following `Element` data type in order to test the `PriorityQueue` Class.

```
template <class T>
class Element
{
    private:
      T Data; // the data of the element
      int Priority;//non-negative value, similar to the processes in
HW1 a smaller value shows a higher priority
      public:
      // constructor
           Element(const T data, const int priority);//returns with
error if priority is initialized a negative integer
         T ShowData(void)
{return Data;)
int ShowPriority (void) const
{return Priority};
//overload the comparison operator < such that < returns true for
//element1<element2 if element2.Priority<= element1.Priority
bool operator< (const Element<T>& rhs) const;

};
```

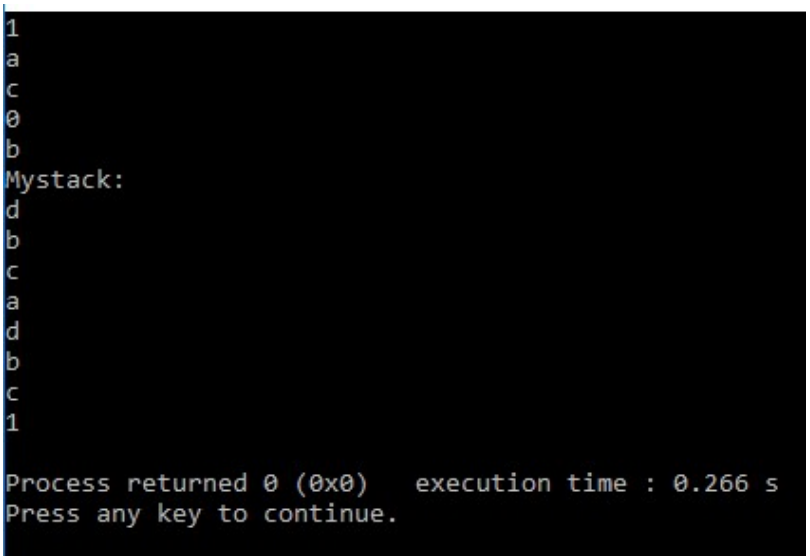
Examples

Example 1:

This code initializes a `Stack` object and pushes characters a, c, b and d, respectively. After each push operation, `Peek` method is called in order to see which one is at the top of the stack. Then, it pops element from the stack twice. Finally, clears the stack just to see the stack is empty. Note that throughout the code, whether the stack is empty or not is checked three times.

```
Stack<char> *Mystack;

Mystack = new Stack<char>();

cout << Mystack->StackEmpty() << endl;

Mystack->Push('a');

cout << Mystack->Peek() << endl;

Mystack->Push('c');

cout << Mystack->Peek() << endl;

cout << Mystack->StackEmpty() << endl;

Mystack->Push('b');

cout << Mystack->Peek() << endl;

Mystack->Push('d');

cout << "Mystack: " << endl;

Mystack->ShowStack();

cout << Mystack->Pop() << endl;

cout << Mystack->Pop() << endl;

cout << Mystack->Peek() << endl;

Mystack->ClearStack();

cout << Mystack->StackEmpty() << endl;
```

Output:



Example 2:

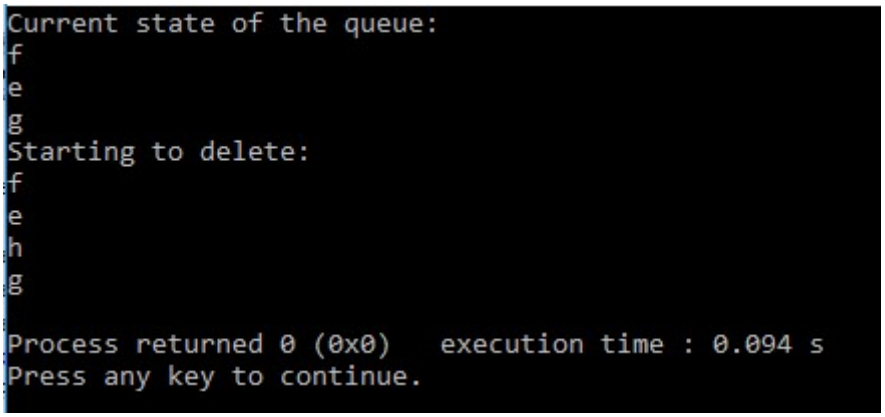
The following code creates four `Element` objects, and a `PQueue` object. Then, inserts the elements to the queue. Finally, deletes all of the elements.

Note: This example also shows what should happen if two elements have the same priority

```
Element<char> *MyElement1, *MyElement2, *MyElement3, *MyElement4;

MyElement1 = new Element<char>('e', 1);

MyElement2 = new Element<char>('f', 0);

MyElement3 = new Element<char>('g', 2);

MyElement4 = new Element<char>('h', 1);

PQueue< Element<char> > *MyPQ;

MyPQ = new PQueue< Element<char> >();

MyPQ->PQInsert(*MyElement1); // queue: e

MyPQ->PQInsert(*MyElement2); // queue: fe

MyPQ->PQInsert(*MyElement3); // queue: feg

cout << "Current state of the queue:" << endl;

MyPQ->ShowPQ();

MyPQ->PQInsert(*MyElement4); // queue: fehg

cout << "Starting to delete:" << endl;

cout << (MyPQ->PQDelete()).ShowData() << endl;

cout << (MyPQ->PQDelete()).ShowData() << endl;

cout << (MyPQ->PQDelete()).ShowData() << endl;
```

```
cout << (MyPQ->PQDelete()).ShowData() << endl;
```

Output:



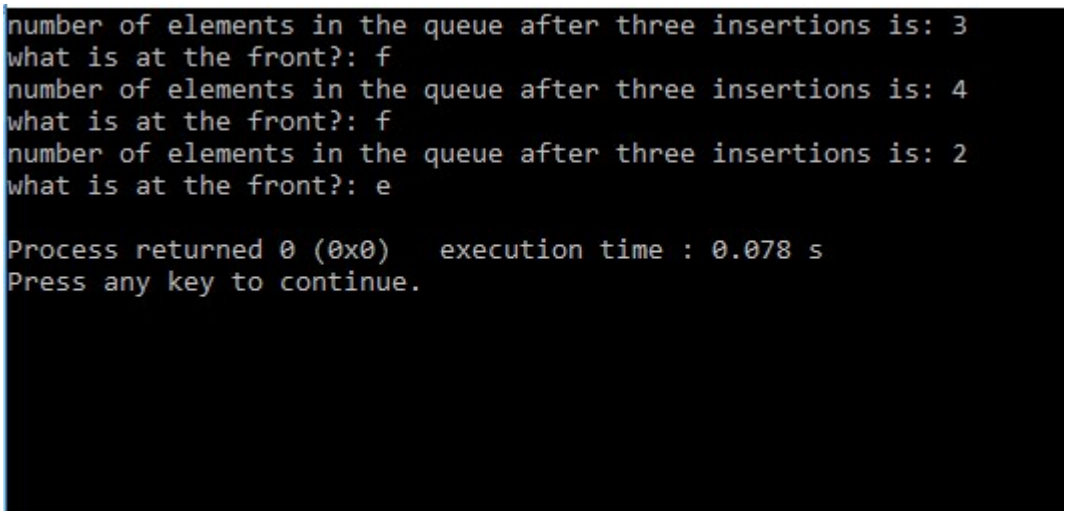
Example 3:

This example is somewhat continuation of the previous one. This time, after inserting some elements in the queue, we check the queue length and the front element. Then, insert one more element and check again. After that, clear the queue and insert two more elements. Finally check the conditions for a last time.

```
Element<char> *MyElement1, *MyElement2, *MyElement3, *MyElement4;

MyElement1 = new Element<char>('e', 1);

MyElement2 = new Element<char>('f', 0);

MyElement3 = new Element<char>('g', 2);

MyElement4 = new Element<char>('h', 1);

PQueue< Element<char> > *MyPQ;

MyPQ = new PQueue< Element<char> >();

MyPQ->PQInsert(*MyElement1); // queue: e

MyPQ->PQInsert(*MyElement2); // queue: fe

MyPQ->PQInsert(*MyElement3); // queue: feg

cout << "number of elements in the queue after three insertions is:
" << MyPQ->PQLength() << endl; // length: 3

cout << "what is at the front?: " << (MyPQ->PQFront()).ShowData() <<
endl; // front: f

MyPQ->PQInsert(*MyElement4); // queue: fheg

cout << "number of elements in the queue after three insertions is:
" << MyPQ->PQLength() << endl; // length: 4
```

```
cout << "what is at the front?: " << (MyPQ->PQFront()).ShowData() << endl; // front: f

MyPQ->PQClear(); // queue:

MyPQ->PQInsert(*MyElement3); // queue: g

MyPQ->PQInsert(*MyElement1); // queue: eg

cout << "number of elements in the queue after three insertions is: " << MyPQ->PQLength() << endl; // length: 2

cout << "what is at the front?: " << (MyPQ->PQFront()).ShowData() << endl; // front: e
```

Output:



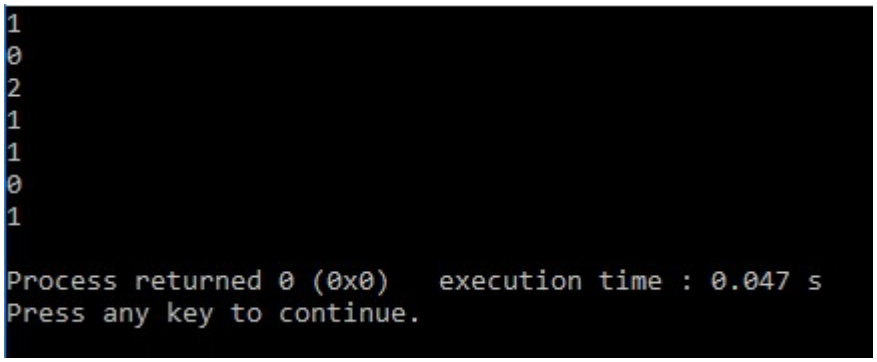
Example 4:

This code creates four element objects, shows their priorities and compares them pairwise.

```
Element<char> *MyElement1, *MyElement2, *MyElement3, *MyElement4;

MyElement1 = new Element<char>('e', 1);

MyElement2 = new Element<char>('f', 0);

MyElement3 = new Element<char>('g', 2);

MyElement4 = new Element<char>('h', 1);

cout << MyElement1->ShowPriority() << endl;

cout << MyElement2->ShowPriority() << endl;

cout << MyElement3->ShowPriority() << endl;

cout << MyElement4->ShowPriority() << endl;

cout << (*MyElement1 < *MyElement2) << endl;
```

```
cout << (*MyElement2 < *MyElement3) << endl;

cout << (*MyElement1 < *MyElement4) << endl;
```

Output:



Regulations:

1. You should insert comments to your source code at appropriate places without including any unnecessary detail. Comments will be graded.

2. Use **Code::Blocks IDE** and choose GNU GCC Compiler while creating your project. Name your project as "e<student_ID>_HW2". Send the whole project folder compressed in a rar or zip file. You will not get full credit if you fail to submit your project folder as required.

3. Your C++ program should follow object oriented principles, including proper class and method usage and should be correctly structured including private and public components. Your work will be graded on its correctness, efficiency and clarity as a whole.

4. Late submissions are welcome, but penalized according to the following policy:

    - 1 day late submission: HW will be evaluated out of 70.
    - 2 days late submission: HW will be evaluated out of 50.
    - 3 days late submission: HW will be evaluated out of 30.
    - 4 or more days late submission: HW will not be evaluated.

Good Luck!