Lab Sheet 5

"Computers make it easier to do a lot of things,
but most of the things they make it easier to do
don't need to be done."
-- Andy Rooney

1. A *Queue* is a particular kind of ADT in which the entities in the collection are kept *in-order* and the principal operations on the collections are *addition* of entities to the *rear* position of the queue (called as *enqueue* operation) and the *removal* of the entities from the *front* position of the queue (called as *dequeue* operation). This makes the queue a First-In-First-Out (*FIFO*) data structure.

   Write a program to implement a *circular queue* of integers using *arrays*. Name of the class should be `Queue_Int`. Details of the class are as follows:

   Data types:

Name of data type	**Description**
`size_type`	Data type of queue size; equivalent to `size_t`.
`value_type`	Data type of queue value; equivalent to `int`.

   Data Members:

Name of data member	**Description**
`static const size_t queue_size = 5`	To hold the size of the array.
`data`	Array of size `queue_size` to store the data elements. Type is `value_type`.
`queue_size`	Size of the queue. Type is `size_type`.
`front_queue`	To store the front of the queue.
`back_queue`	To store the back of the queue.

   Constructor:

Constructor	**Description**
`Queue_Int()`	Default constructor. Set `queue_size` to 0. Front and rear of the queue is set to -1. Initialize the data elements of the queue to 0.

   Member functions:

Function name	**Description**
`bool empty() const;`	Returns true if queue is empty.
`void push(const int& x);`	Insert an element to the rear of the queue.
`void pop();`	Delete an element from the front of the queue.
`size_type size() const;`	To get the size of the element.
`value_type& front();` `const value_type& front() const;`	To get the data from the front of the queue.
`value_type& back();` `const value_type& back() const;`	To get the data from the back of the queue.

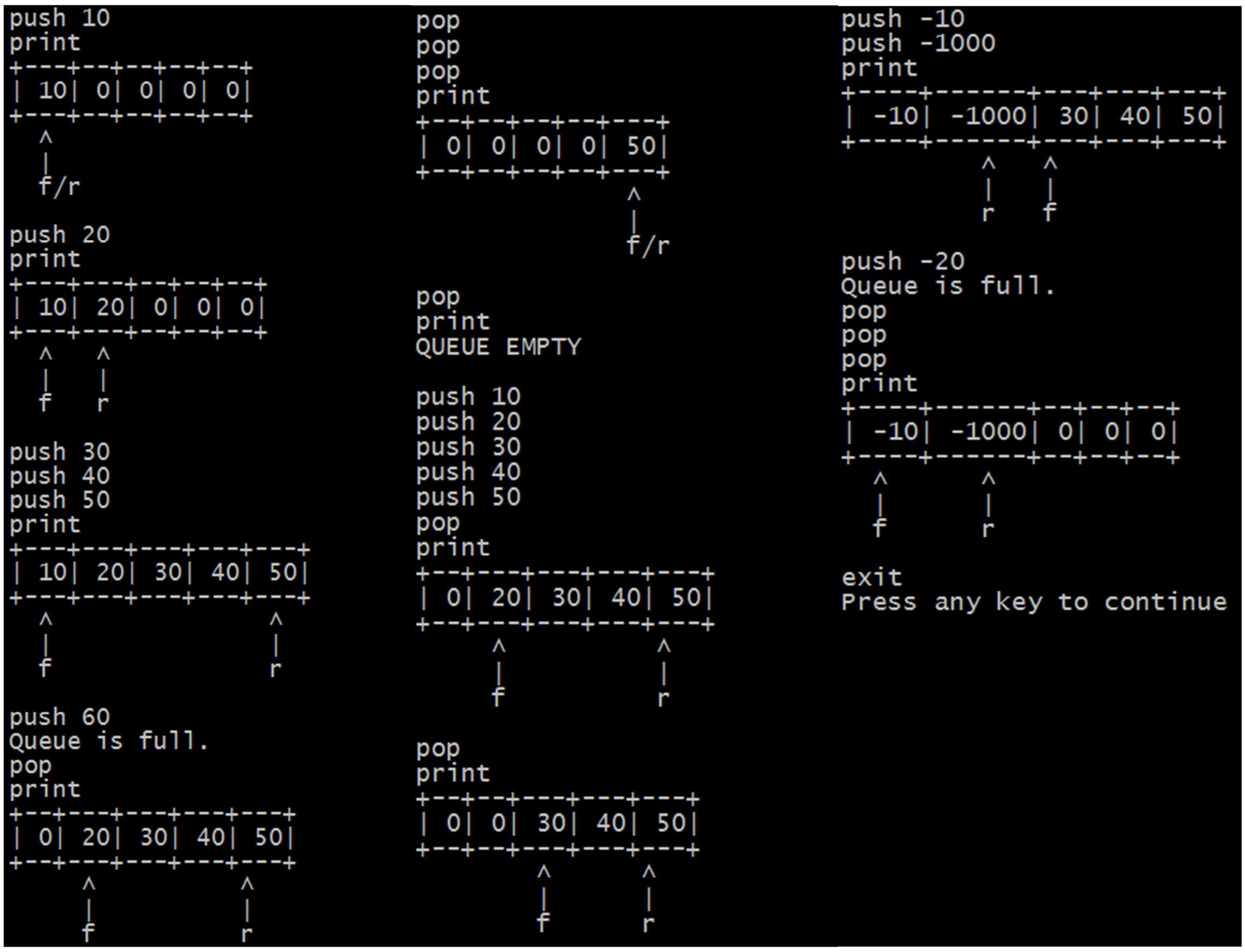
   Friend functions:

Function name	**Description**
`ostream& operator << (ostream& out, const Queue_Int& q);`	To print the elements in the queue using the output operator.

   - Is it possible to create a *non-circular* queue using *arrays*?
   - What is the advantages/disadvantages of queue implementation using *arrays* over queue implementation using *linked-lists*?

Make a driver for the `Queue_Int` class that behaves as follows. It should accept *commands* and do the requested operation on the queue. The command-set must include `push`, `pop`, `print`, `front`, `back` and `exit`.

Sample run for the program: (The size of the queue is set to 5)



```
push 10
print
+---+--+--+--+--+
| 10| 0| 0| 0| 0|
+---+--+--+--+--+
  ^
  |
  f/r

push 20
print
+---+---+--+--+--+
| 10| 20| 0| 0| 0|
+---+---+--+--+--+
  ^   ^
  |   |
  f   r

push 30
push 40
push 50
print
+---+---+---+---+---+
| 10| 20| 30| 40| 50|
+---+---+---+---+---+
  ^               ^
  |               |
  f               r

push 60
Queue is full.
pop
print
+--+---+---+---+---+
| 0| 20| 30| 40| 50|
+--+---+---+---+---+
     ^           ^
     |           |
     f           r
```

```
pop
pop
pop
print
+--+--+--+--+---+
| 0| 0| 0| 0| 50|
+--+--+--+--+---+
              ^
              |
              f/r

pop
print
QUEUE EMPTY

push 10
push 20
push 30
push 40
push 50
pop
print
+--+---+---+---+---+
| 0| 20| 30| 40| 50|
+--+---+---+---+---+
     ^           ^
     |           |
     f           r

pop
print
+--+--+---+---+---+
| 0| 0| 30| 40| 50|
+--+--+---+---+---+
        ^      ^
        |      |
        f      r
```

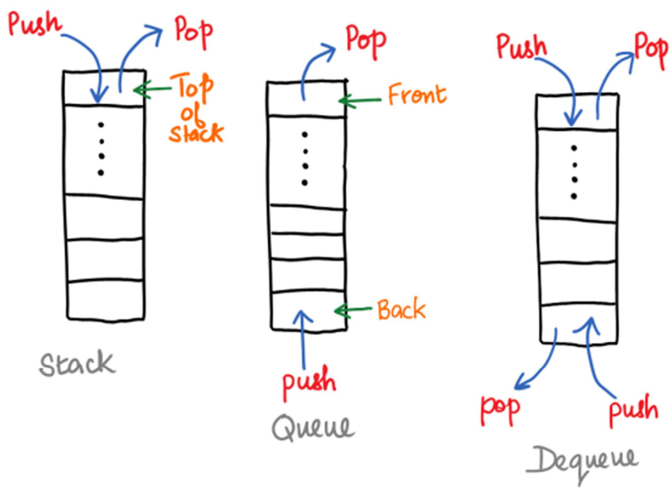
```
push -10
push -1000
print
+----+-------+---+---+---+
| -10| -1000| 30| 40| 50|
+----+-------+---+---+---+
         ^   ^
         |   |
         r   f

push -20
Queue is full.
pop
pop
pop
print
+----+-------+--+--+--+
| -10| -1000| 0| 0| 0|
+----+-------+--+--+--+
  ^      ^
  |      |
  f      r

exit
Press any key to continue
```

2. Write a program to implement a *queue* of integers using *doubly linked list*. You may name the class as `Queue_doubly_Linked_List`. The interface of the class remains the same as above (default constructor, empty, push, pop, size, front, back).

   Make use of the doubly linked list template class that we have with us (`List_doubly_linked`) to implement this queue class.

3. A *double-ended queue* (*dequeue*, often abbreviated to *deque*, pronounced *deck*) is an ADT that generalizes a queue, in which the elements can be added to or removed from either the *front* (head) or the *back* (tail).

   Write a program to implement double-ended queue using doubly linked list that we have with us (`List_doubly_linked`).


Push
Pop
Top of Stack
Stack
Pop
Front
Back
push
Queue
Push
Pop
pop
push
Dequeue