IT DS 201 LAB

SUBMITTED BY ADITYA SINGH 2K19/EP/005

Program 1 : Write a program to find the minimum element in the stack using extra space.

CODE

```
#include <iostream>
#include <stack>
using namespace std;

class Stack{
    stack<int> s;
    stack<int> aux;

public:
    void push(int x){
        s.push(x);
        if (aux.empty()) {
            aux.push(x);
        }
        else {
            if (aux.top() >= x) {
                aux.push(x);
            }
        }
    }
```

```
int pop(){
    if (empty()){
        cout << "Stack underflow!!" << endl;
        return -1;
    }
    int top = s.top();
    s.pop();
    if (top == aux.top()) {
        aux.pop();
    }
    return top;
}

int top() {
    return s.top();
}

int size() {
    return s.size();
}

bool empty() {
    return s.empty();
}
```

```
 int min(){
     if (aux.empty()){
         cout << "Stack underflow!! ";
         return -1;
     }
     return aux.top();
 }
};
```

ALGORITHM

Push (x) :

1) push x to the first stack (the stack with actual elements)
2) compare x with the top element of the second stack (the auxiliary stack). Let the top element be y.
    a) If x is smaller than y then push x to the auxiliary stack.
    b) If x is greater than y then push y to the auxiliary stack.

Pop() :

1) pop the top element from the auxiliary stack.
2) pop the top element from the actual stack and return it.
Step 1 is necessary to make sure that the auxiliary stack is also updated for future operations.

getMin() :

// returns the minimum element from Special Stack
1) Return the top element of the auxiliary stack.

Operation	Main Stack	Auxiliary Stack	Minimum Number
push (6)	6	6	6
push (7)	6, 7	6	6
push (8)	6, 7, 8	6	6
push (5)	6, 7, 8, 5	6, 5	5
push (3)	6, 7, 8, 5, 3	6, 5, 3	3
pop()	6, 7, 8, 5	6, 5	5
push (10)	6, 7, 8, 5, 10	6, 5	5
pop()	6, 7, 8, 5	6, 5	5
pop()	6, 7, 8	6	6

INPUT/OUTPUT

```cpp
int main(){
    Stack s;
    s.push(6);
    cout << s.min() << " ";
    s.push(7);
    cout << s.min() << " ";
    s.push(8);
    cout << s.min() << " ";
    s.push(5);
    cout << s.min() << " ";
    s.push(3);
    cout << s.min() << " ";
    s.pop();
    cout << s.min() << " ";
    s.push(10);
    cout << s.min() << " ";
    s.pop();
    cout << s.min() << endl;

    return 0;
}
```

```
6 6 6 5 3 5 5 5
[Finished in 658ms]
```

Line 54, Column 3

END

Program 2 : Write a program to implement Queue data structure.

CODE

```
#include <bits/stdc++.h>
using namespace std;

class Queue {
public:
    int front, rear, size;
    unsigned capacity;
    int* array;
};

Queue* createQueue(unsigned capacity){
    Queue* queue = new Queue();
    queue->capacity = capacity;
    queue->front = queue->size = 0;

    queue->rear = capacity - 1;
    queue->array = new int[queue->capacity];
    return queue;
}

void enqueue(Queue* queue, int item){
    if (isFull(queue))
        return;
    queue->rear = (queue->rear + 1) % queue->capacity;
    queue->array[queue->rear] = item;
    queue->size = queue->size + 1;
    cout << item << " enqueued to queue\n";
}
int isFull(Queue* queue){
    return (queue->size == queue->capacity);
}
```

```c
int isEmpty(Queue* queue){
    return (queue->size == 0);
}

int dequeue(Queue* queue){
    if (isEmpty(queue))
        return INT_MIN;
    int item = queue->array[queue->front];
    queue->front = (queue->front + 1) % queue->capacity;
    queue->size = queue->size - 1;
    return item;
}

int front(Queue* queue){
    if (isEmpty(queue))
        return INT_MIN;
    return queue->array[queue->front];
}

int rear(Queue* queue){
    if (isEmpty(queue))
        return INT_MIN;
    return queue->array[queue->rear];
}
```

ALGORITHM

Enqueue (x) :

1. Check if the queue is full or not.
2. If the queue is full, then print an overflow error and exit the program.
3. If the queue is not full, then increment the tail and add the element.

Dequeue () :

1. Check if the queue is empty or not.
2. If the queue is empty, then print an underflow error and exit the program.
3. If the queue is not empty, then print the element at the head and increment the head.

INPUT/OUTPUT

```
int main(){
    Queue* queue = createQueue(1000);

    enqueue(queue, 10);
    enqueue(queue, 20);
    enqueue(queue, 30);
    enqueue(queue, 40);

    cout << dequeue(queue) << " dequeued from queue\n";

    cout << "Front item is " << front(queue) << endl;
    cout << "Rear item is " << rear(queue) << endl;

    return 0;
}
```

```
10 enqueued to queue
20 enqueued to queue
30 enqueued to queue
40 enqueued to queue
10 dequeued from queue
Front item is 20
Rear item is 40
[Finished in 1.2s]
```

END