Experiment no. 4

→ Aim. Queues are frequently used in computer programming and a typical en is the creation of a job queue by an operating system. If the operating system does not use priorities, then the job are processed in the order they enter the system. Write C++ program for simulating job queue. Write functions to add job & delete job from queue.

Pre-requisite:

Basics of Queue
Different operations that can be performed on queue

Objective:

To perform addition & deletion operations on queue.

Input:

Size of & queue element in queue

Outcome:

Result of addition of job operation on queue
Result of deletion of job operation on queue

→ Theory

· Queue

Queue is an abstract data structure, similar to Stacks. Unlike stacks, a queue is open at both its ends. One end is always used to insert data & the other is used to remove data. Queue follows First in First Out methodology, i.e the data item stored first will be accessed first.

As in stacks, a queue can also be implemented using Array, linked-list, Pointer & structures.

- Basic Operations

  enqueue() - Add (store) an items to the queue.
  dequeue() - remove (access) an items from the queue
  peek() - Gets the element at the front
  isfull() - Checks if the queue is full.
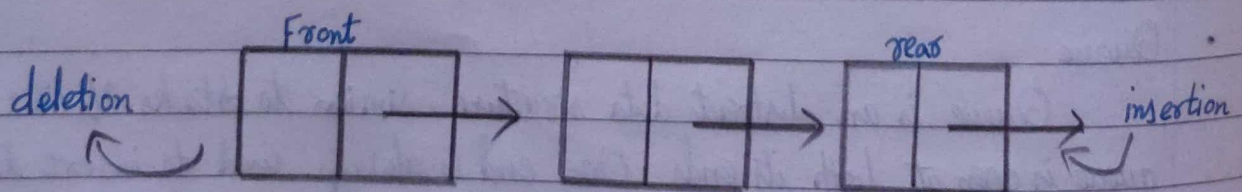 isempty() - Checks if the queue is empty.

- Types of Queue

  There are 4 different types of Queue

  1 Simple Queue
  2 Circular Queue
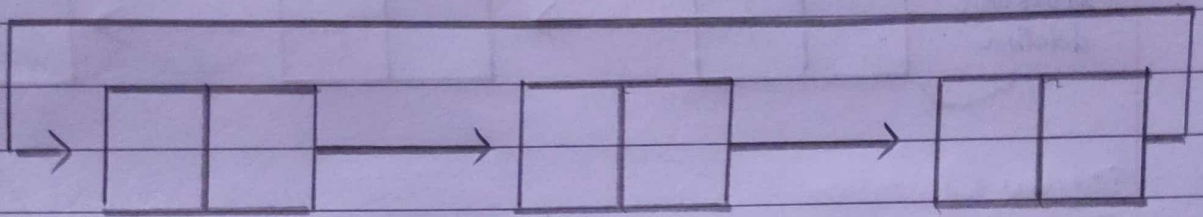 3 Priority Queue
  4 Double Ended Queue

- Simple Queue

 In a Simple Queue, inserted takes places at the rear & removal occurs at the fount. it strictly follows FIFO rule.

- Circular Queue

In a circular Queue, the last element points to the first element making a circular link. The main advantage of a circular queue over a simple queue is better memory utilization. If the last position is full & the first position is empty. We can insert an element in the first position.

- Priority Queue

A Priority Queue is a special type of queue in which element is associated with a priority and is served according to its priority. If element with the same priority occur, they are served according to their order in the queue
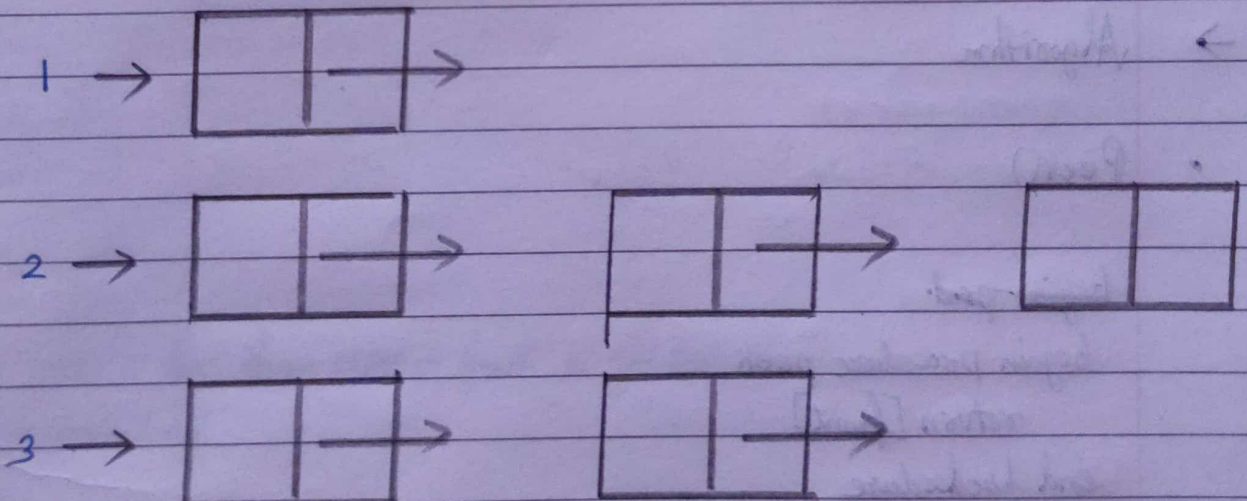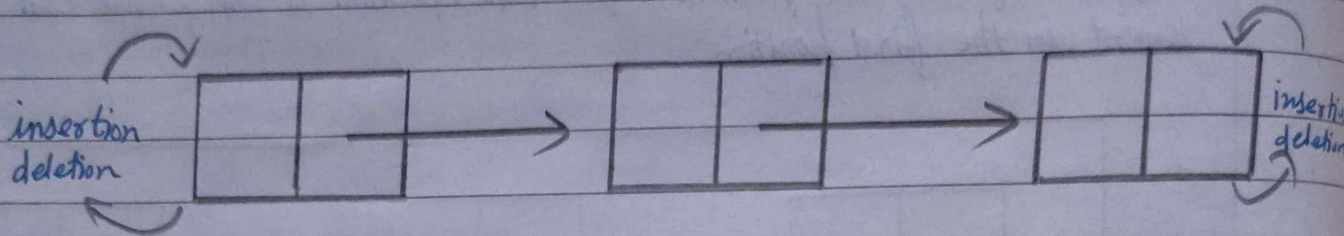

Insertion occurs based on the arrival of the values and removal occurs based on priority

- **Deque (Double Ended Queue)**

In a double ended queue, insertion and removal of elements can be performed from either from the front or rear. Thus it does not follow FIFO rule.

insertion deletion

insertion deletion

Queues have the advantages of being able to handle multiple data types & they are both flexible & flexibility & fast. Moreover, queues can be of potentially infinite length compared with the use of fixed-length arrays.

A major disadvantage of classical queue is the a new element can only be inserted when all of the element are deleted from the queue.

→ **Algorithm**

- **Peek()**

~~begin peek.~~
begin procedure peek
   return [front]
end procedure

ex→
  int peek() {
    return queue [front];
  }

- Isfull ()

begin produre isfull

   if rear equals to MAX SIZE
      return true
   else
      return False
   endif
end procedure

Ex→

```
bool isFull () {
  if (rear == MAX SIZE -1)
    return true;
  else
    return False;
}
```

- Is empty ()

begin

   If Front is less than MIN or front is greater than rear
      return true
   else
      return False
   endif

end.

ex

```
bool isempty (){
if (front < o || Front > rear)
    return true;
else
    return False;
}
```

- Enqueue Operation

Queues maintain two data pointers, front & rear. Therefore, its operator are comparatively difficult to implement than that of stacks

Procedure enqueue (data)

```
If queue is full
    return 0;


rear = rear + 1
queue[rear] = data


return 1;
end procedure
```

- Dequeue

Procedure dequeue

```
int dequeue () {

    if (isempty())
        return underflow
end if


    data = queue [Front];
    front = front + 1;
    return true


    end.
```

Program:

```cpp
#include <iostream>
#define MAX 10
using namespace std;
struct queue
{
    int data[MAX];
    int front, rear;
};
class Queue
{
    struct queue q;

public:
    Queue() { q.front = q.rear = -1; }
    int IsEmpty();
    int IsFull();
    void Enqueue(int);
    int Delqueue();
    void Display();
};
int Queue::IsEmpty()
{
    return (q.front == q.rear) ? 1 : 0;
}
int Queue::IsFull()
{
    return (q.rear == MAX - 1) ? 1 : 0;
}
void Queue::Enqueue(int x)
{
    q.data[++q.rear] = x;
}
int Queue::Delqueue()
{
    return q.data[++q.front];
}
void Queue::Display()
{
    int i;
    cout << "\n";
    for (i = q.front + 1; i <= q.rear; i++)
        cout << q.data[i] << ", ";
}
int main()
{
    Queue obj;
    int ch, x;
    do
    {
        cout << "\n* * * * * * * * *";
        cout << "\n* 1.Insert Job  *";
        cout << "\n* 2.Delete Job  *";
```

```cpp
        cout << "\n* 3.Display     *";
        cout << "\n* 4.Exit        *";
        cout << "\n* * * * * * * * *";
        cout << "\nEnter your choice: ";
        cin >> ch;
        switch (ch)
        {
        case 1:
            if (!obj.IsFull())
            {
                cout << "\n Enter Data: ";
                cin >> x;
                obj.Enqueue(x);
            }
            else
                cout << "Queue is overflow";
            break;
        case 2:
            if (!obj.IsEmpty())
                cout << "\n Deleted Element= " << obj.Delqueue();
            else
            {
                cout << "\n Queue is underflow";
            }
            cout << "\nRemaining jobs :";
            obj.Display();
            break;
        case 3:
            if (!obj.IsEmpty())
            {
                cout << "\n Queue contains:";
                obj.Display();
            }
            else
                cout << "\n Queue is empty";
            break;
        case 4:
            cout << "\n Exit";
        }
    } while (ch != 4);
    return 0;
}
```

Output :


```
orion@OMEN-15:/mnt/d/College/2 Second year/SY SEM 3/Data Structures and Algorithms (DSA)/Lab manual/assign 4$ ./Assignment4

* * * * * * * * *
* 1.Insert Job  *
* 2.Delete Job  *
* 3.Display     *
* 4.Exit        *
* * * * * * * * *
Enter your choice: 1

 Enter Data: 2

* * * * * * * * *
* 1.Insert Job  *
* 2.Delete Job  *
* 3.Display     *
* 4.Exit        *

* * * * * * * * *
* 1.Insert Job  *
* 2.Delete Job  *
* 3.Display     *
* 4.Exit        *
* * * * * * * * *
Enter your choice: 1

 Enter Data: 10

* * * * * * * * *
* 1.Insert Job  *
* 2.Delete Job  *
* 3.Display     *
* 4.Exit        *
* * * * * * * * *
Enter your choice: 3

 Queue contains:
2, 5, 10,
* * * * * * * * *
* 1.Insert Job  *
* 2.Delete Job  *
```



```
* * * * * * * * *
* 1.Insert Job  *
* 2.Delete Job  *
* 3.Display     *
* 4.Exit        *
* * * * * * * * *
Enter your choice: 1

 Enter Data: 10

* * * * * * * * *
* 1.Insert Job  *
* 2.Delete Job  *
* 3.Display     *
* 4.Exit        *
* * * * * * * * *
Enter your choice: 3

 Queue contains:
2, 5, 10,
* * * * * * * * *
* 1.Insert Job  *
* 2.Delete Job  *
* 3.Display     *
* 4.Exit        *
* * * * * * * * *
Enter your choice: 2

 Deleted Element= 2
Remaining jobs :
5, 10,
* * * * * * * * *
* 1.Insert Job  *
* 2.Delete Job  *
* 3.Display     *
* 4.Exit        *
* * * * * * * * *
Enter your choice: 4

orion@OMEN-15:/mnt/d/College/2 Second year/SY SEM 3/Data Structures and Algorithms (DSA)/Lab manual/assign 4$
```