5. Queue Data Structure

What is Queue?

Queue is a data structure which is used to handle data in a first-in-first-out (FIFO) method. That is we can remove the element which has been added earlier from the queue first.

Common operations of Queue are:

initializeQueue() – initializes the queue as empty queue.
enQueue()- adds an element at the rear of the queue.
deQueue()-removes and returns the front element from the queue.
frontElt()-returns the front element without removing it.
isEmpty() - returns true if the queue has no elements and false otherwise.
isFull() - returns true if the queue is full of elements and false otherwise.
displayQueue() - displays all elements from front to rear.

Graphical Representation of <u>Queue Operation</u>:

1. initializeQueue()						
2. p=isEmpty()						p = true
3. enQueue(5)	5					
4. enQueue(9) enQueue(7)	5	9	7			
5. x=deQueue()	9	7				x = 5
6. enQueue(2) enQueue(6)	9	7	2	6		
7. q = isFull()	9	7	2	6		q = false
8. enQueue(3)	9	7	2	6	3	
9. r = isFull() y = deQueue()	7	2	6	3		r = true y = 9

Static (Array based) <u>Implementation of Queue Operations</u> [Graphical Representation]:

1. initializeQueue()

0	1	2	3	4

front	-1
rear	-1
size	0

2. p=isEmpty()

0	1	2	3	4

p = true

front	-1
rear	-1
size	0

3. enQueue(5)

0	1	2	3	4
5				

front	-1
rear	0
size	1

4. enQueue(9)
   enQueue(7)

0	1	2	3	4
5	9	7		

front	-1
rear	2
size	3

5. x=deQueue()

0	1	2	3	4
~~5~~	9	7		

x = 5

front	0
rear	2
size	2

6. enQueue(2)
   enQueue(6)

0	1	2	3	4
~~5~~	9	7	2	6

front	0
rear	4
size	4

7. q = isFull()

0	1	2	3	4
~~5~~	9	7	2	6

q = false

front	0
rear	4
size	4

8. enQueue(3)

0	1	2	3	4
3	9	7	2	6

front	0
rear	0
size	5

9. r = isFull()
   y = deQueue()

0	1	2	3	4
3	~~9~~	7	2	6

front	1
rear	0
size	4

r = true
y = 9

Static (Array based) Implementation of Stack Operations [C++ Code]:

```
#include<iostream.h>
#include<conio.h>

const Q_SIZE=5;

class Queue
{
private:
  int front, rear, size;
  int que[Q_SIZE];

public:
  Queue();
  void initializeQueue();
  void enQueue(int);
  int deQueue();
  int frontElt();
  int isEmpty();
  int isFull();
  void displayQueue();
}

Queue::Queue()
{
  front=(-1);
  rear=(-1);
  size=0;
}

void Queue::initializeQueue()
{
  front=(-1);
  rear=(-1);
  size=0;
}

void Queue::enQueue(int elt)
{
  if (size < Q_SIZE)
  {
    rear=(rear+1)%Q_SIZE;
    que[rear]=elt;
    size++;
  }                       //Else cout<<"Queue is full"
```

```
}

int Queue::deQueue()
{
  if (size > 0)
  {
    front=(front+1)%Q_SIZE;
    size--;
    return que[front];
  }
  else
    return 999; //Some invalid integer should be returned or cout<<”Queue is empty”
}

int Queue::frontElt()
{
  if (size>0)
  {
    return que[(front+1)%Q_SIZE];
  }
  else
    return 999; //Some invalid integer should be returned or cout<<”Queue is empty”
}

int Queue::isEmpty()
{
  return (size == 0);
}

int Queue::isFull()
{
  return (size == Q_SIZE);
}

void Queue::displayQueue()
{
  int i=front;
  for (int j=1;j<=size;j++)
  {
    i=(i+1)%Q_SIZE;
    cout<<que[i]<<”\t”;
  }
}

void main()
{
  clrscr();
  Queue q;
  q.enQueue(5);
  q.enQueue(9);
  q.enQueue(7);
  int x=q.deQueue();
  q.enQueue(2);
  q.enQueue(6);
  q.enQueue(3);
  int y=q.deQueue();
  int z=q.frontElt();
  cout<<”x=”<<x<<”\t” <<”y=”<<y<<”\t” <<”z=”<<z<<”\n”;
  cout<<"Current queue elements:"<<endl;
  q.displayQueue();
}
```

Output:

```
x=5     y=9     z=7
Current stack elements:
7        2        6        3
```

Dynamic (Linked List based) Implementation of Queue Operations:

```
#include<iostream.h>
#include<conio.h>

struct node
{
  int data;
  node *next;
};

class Queue
{
  private:
    node *rear,*front;
public:
    Queue();
    void initializeQueue();
    void enQueue(int);
    int deQueue();
    int frontElt();
    int isEmpty();
    int isFull();
    void displayQueue();
};

Queue::Queue()
{
  rear=NULL;
  front=NULL;
}

void Queue::initializeQueue()
{
  rear=NULL;
  front=NULL;
}

void Queue::enQueue(int elt)
{
  node *newNode;
  newNode = new node;
  newNode->data = elt;
  newNode->next = NULL;
  if(rear==NULL)
  {
    rear=newNode;
    front=newNode;
  }
  else
  {
        rear->next = newNode;
        rear = newNode;
    }
}

int Queue::deQueue()
{
  if (front != NULL)
  {
     int num = front->data;
     front = front->next;
     if(front==NULL) rear = NULL;
     return num;
   }
  else
     return 999;
}

int Queue::frontElt()
{
  if (front!=NULL)
     return front->data;
  else
     return 999;
}

int Queue::isEmpty()
{
  return (front == NULL);
}

int QueueisFull()
{
  return 0;
}

void Queue::displayQueue()
{
  node *temp=front;
  while (temp!=NULL)
  {
    cout<<temp->data<<"\t";
    temp=temp->next;
  }
  cout<<"\n";
}

//Using the above class Queue

void main()
{
  Queue que;
  char opt;
  int n;
  clrscr();
  do
  {
    cout<< "Enter 'i' to insert, 'd' to delete, 's' to show elements and 'q' to quit: ";
    cin>>opt;
    switch(opt)
    {
        case 'i' :
             cout<<"Enter an integer to insert: ";
          cin>>n;
          que.enQueue(n);
          break;
        case 'd' :
             cout<<"The element "<<que.deQueue()<<" is deleted.\n";
          break;
        case 's' :
             cout<<"Queue elements are: ";
             que.displayQueue();
    }
  }
  while (opt != 'q');
}
```

Advantages of Queue:
First-in-first-out access

Disadvantages of Queue:
Difficult to access other items