Queue

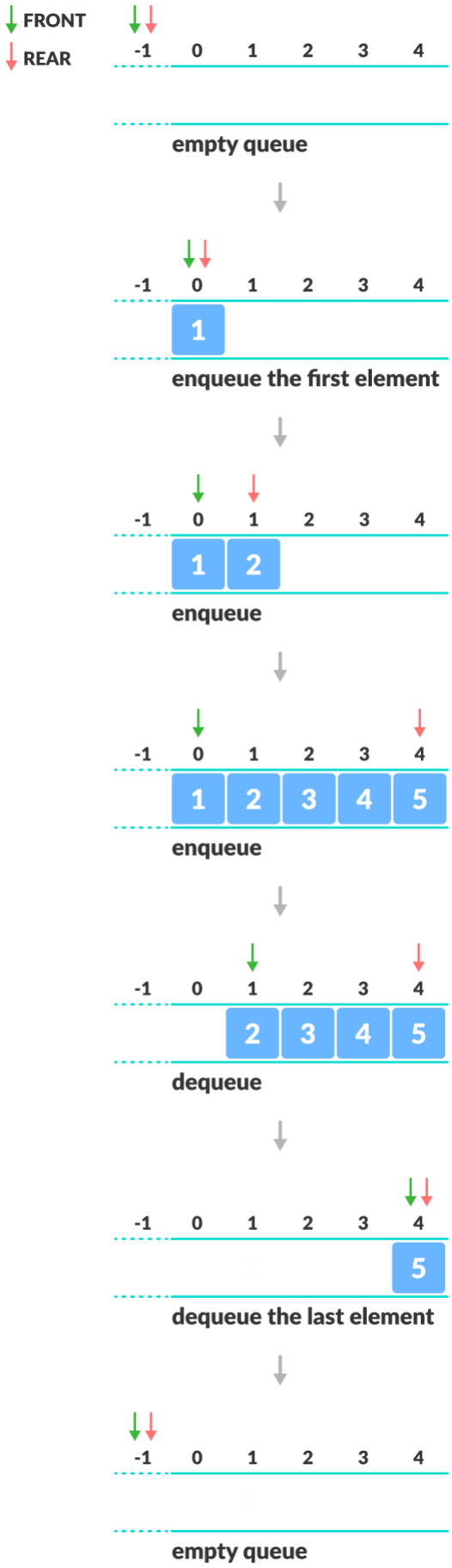
A Queue is a linear structure which follows a particular order in which the operations are performed. The order is First In First Out (FIFO).

Unlike stacks, a queue is open at both its ends. One end is always used to insert data (enqueue) and the other is used to remove data (dequeue).

Operations on Queue

There are two fundamental operations performed on a Queue:

- **Enqueue:** The enqueue operation is used to insert the element at the rear end of the queue. It returns void.
- **Dequeue:** The dequeue operation performs the deletion from the front-end of the queue. It also returns the element which has been removed from the front-end. It returns an integer value. The dequeue operation can also be designed to void.
- **Peek:** This is the third operation that returns the element, which is pointed by the front pointer in the queue but does not delete it.
- **Queue overflow (isfull):** When the Queue is completely full, then it shows the overflow condition.
- **Queue underflow (isempty):** When the Queue is empty, i.e., no elements are in the Queue then it throws the underflow condition.


FRONT
REAR
-1 0 1 2 3 4
empty queue
1
enqueue the first element
2
enqueue
3 4 5
enqueue
dequeue
dequeue the last element
empty queue

Algorithm to insert any element in a queue

Algorithm

Step 1: IF REAR = MAX - 1

Write OVERFLOW

Go to step

[END OF IF]

Step 2: IF FRONT = -1 and REAR = -1

SET FRONT = REAR = 0

ELSE

SET REAR = REAR + 1

[END OF IF]

Step 3: Set QUEUE[REAR] = NUM

Step 4: EXIT

Algorithm to delete an element from the queue

Step 1: IF FRONT = -1 or FRONT > REAR

Write UNDERFLOW

ELSE

SET VAL = QUEUE[FRONT]

SET FRONT = FRONT + 1

[END OF IF]

Step 2: EXIT

Array implementation of Queue

```
#include <iostream>
using namespace std;
int queue[100], n = 100, front = - 1, rear = - 1;
void Insert() {
   int val;
   if (rear == n - 1)
   cout<<"Queue Overflow"<<endl;
   else {
      if (front == - 1)
      front = 0;
      cout<<"Insert the element in queue : "<<endl;
      cin>>val;
      rear++;
      queue[rear] = val;
   }
}
void Delete() {
   if (front == - 1 || front > rear) {
      cout<<"Queue Underflow ";
      return ;
   } else {
      cout<<"Element deleted from queue is : "<< queue[front]
<<endl;
      front++;;
   }
}
void Display() {
   if (front == - 1)
   cout<<"Queue is empty"<<endl;
   else {
      cout<<"Queue elements are : ";
      for (int i = front; i <= rear; i++)
      cout<<queue[i]<<" ";
```

```
         cout<<endl;
   }
}
int main() {
   int ch;
   cout<<"1) Insert element to queue"<<endl;
   cout<<"2) Delete element from queue"<<endl;
   cout<<"3) Display all the elements of queue"<<endl;
   cout<<"4) Exit"<<endl;
   do {
      cout<<"Enter your choice : "<<endl;
      cin<<ch;
      switch (ch) {
         case 1: Insert();
         break;
         case 2: Delete();
         break;
         case 3: Display();
         break;
         case 4: cout<<"Exit"<<endl;
         break;
         default: cout<<"Invalid choice"<<endl;
      }
   } while(ch!=4);
   return 0;
}
```

The output of the above program is as follows

```
1) Insert element to queue
2) Delete element from queue
3) Display all the elements of queue
4) Exit
Enter your choice : 1
Insert the element in queue : 4
Enter your choice : 1
```

```
Insert the element in queue : 3
Enter your choice : 1
Insert the element in queue : 5
Enter your choice : 2
Element deleted from queue is : 4
Enter your choice : 3
Queue elements are : 3 5
Enter your choice : 7
Invalid choice
Enter your choice : 4
Exit
```

```cpp
// Circular Queue implementation in C++

#include <iostream>
#define SIZE 5 /* Size of Circular Queue */

using namespace std;

class Queue {
   private:
  int items[SIZE], front, rear;

   public:
  Queue() {
    front = -1;
    rear = -1;
  }
  // Check if the queue is full
  bool isFull() {
    if (front == 0 && rear == SIZE - 1) {
      return true;
    }
    if (front == rear + 1) {
      return true;
    }
    return false;
  }
  // Check if the queue is empty
  bool isEmpty() {
    if (front == -1)
      return true;
    else
      return false;
  }
  // Adding an element
  void enQueue(int element) {
    if (isFull()) {
      cout << "Queue is full";
    } else {
      if (front == -1) front = 0;
      rear = (rear + 1) % SIZE;
      items[rear] = element;
      cout << endl
         << "Inserted " << element << endl;
```

```
    }
  }
  // Removing an element
  int deQueue() {
    int element;
    if (isEmpty()) {
      cout << "Queue is empty" << endl;
      return (-1);
    } else {
      element = items[front];
      if (front == rear) {
        front = -1;
        rear = -1;
      }
      // Q has only one element,
      // so we reset the queue after deleting it.
      else {
        front = (front + 1) % SIZE;
      }
      return (element);
    }
  }

  void display() {
    // Function to display status of Circular Queue
    int i;
    if (isEmpty()) {
      cout << endl
         << "Empty Queue" << endl;
    } else {
      cout << "Front -> " << front;
      cout << endl
         << "Items -> ";
      for (i = front; i != rear; i = (i + 1) % SIZE)
        cout << items[i];
      cout << items[i];
      cout << endl
         << "Rear -> " << rear;
    }
  }
};

int main() {
```

```
  Queue q;

  // Fails because front = -1
  q.deQueue();

  q.enQueue(1);
  q.enQueue(2);
  q.enQueue(3);
  q.enQueue(4);
  q.enQueue(5);

  // Fails to enqueue because front == 0 && rear == SIZE - 1
  q.enQueue(6);

  q.display();

  int elem = q.deQueue();

  if (elem != -1)
    cout << endl
       << "Deleted Element is " << elem;

  q.display();

  q.enQueue(7);

  q.display();

  // Fails to enqueue because front == rear + 1
  q.enQueue(8);

  return 0;
}
```

```
#include <iostream>
using namespace std;
struct node {
   int data;
   struct node *next;
};
struct node* front = NULL;
struct node* rear = NULL;
struct node* temp;
void Insert() {
   int val;
   cout<<"Insert the element in queue : "<<endl;
   cin>>val;
   if (rear == NULL) {
      rear = (struct node *)malloc(sizeof(struct node));
      rear->next = NULL;
      rear->data = val;
      front = rear;
   } else {
      temp=(struct node *)malloc(sizeof(struct node));
      rear->next = temp;
      temp->data = val;
      temp->next = NULL;
      rear = temp;
   }
}
void Delete() {
   temp = front;
   if (front == NULL) {
      cout<<"Underflow"<<endl;
      return;
   }
   else
   if (temp->next != NULL) {
      temp = temp->next;
      cout<<"Element deleted from queue is : "<<front-
>data<<endl;
      free(front);
      front = temp;
   } else {
      cout<<"Element deleted from queue is : "<<front-
>data<<endl;
```

```
      free(front);
      front = NULL;
      rear = NULL;
   }
}
void Display() {
   temp = front;
   if ((front == NULL) && (rear == NULL)) {
      cout<<"Queue is empty"<<endl;
      return;
   }
   cout<<"Queue elements are: ";
   while (temp != NULL) {
      cout<<temp->data<<" ";
      temp = temp->next;
   }
   cout<<endl;
}
int main() {
   int ch;
   cout<<"1) Insert element to queue"<<endl;
   cout<<"2) Delete element from queue"<<endl;
   cout<<"3) Display all the elements of queue"<<endl;
   cout<<"4) Exit"<<endl;
   do {
      cout<<"Enter your choice : "<<endl;
      cin>>ch;
      switch (ch) {
         case 1: Insert();
         break;
         case 2: Delete();
         break;
         case 3: Display();
         break;
         case 4: cout<<"Exit"<<endl;
         break;
         default: cout<<"Invalid choice"<<endl;
      }
   } while(ch!=4);
   return 0;
}
```

Output

The output of the above program is as follows

```
1) Insert element to queue
2) Delete element from queue
3) Display all the elements of queue
4) Exit
Enter your choice : 1
Insert the element in queue : 4
Enter your choice : 1
Insert the element in queue : 3
Enter your choice : 1
Insert the element in queue : 5
Enter your choice : 2
Element deleted from queue is : 4
Enter your choice : 3
Queue elements are : 3 5
Enter your choice : 7
Invalid choice
Enter your choice : 4
Exit
```