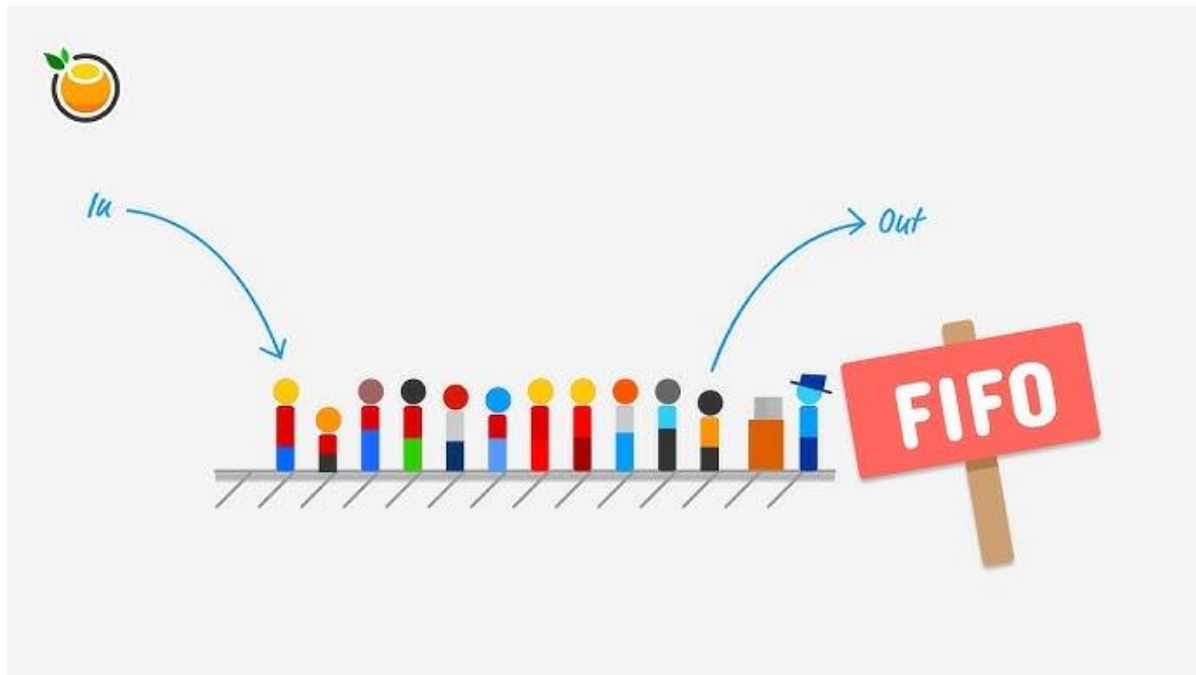


Introduction to Queue Data Structure



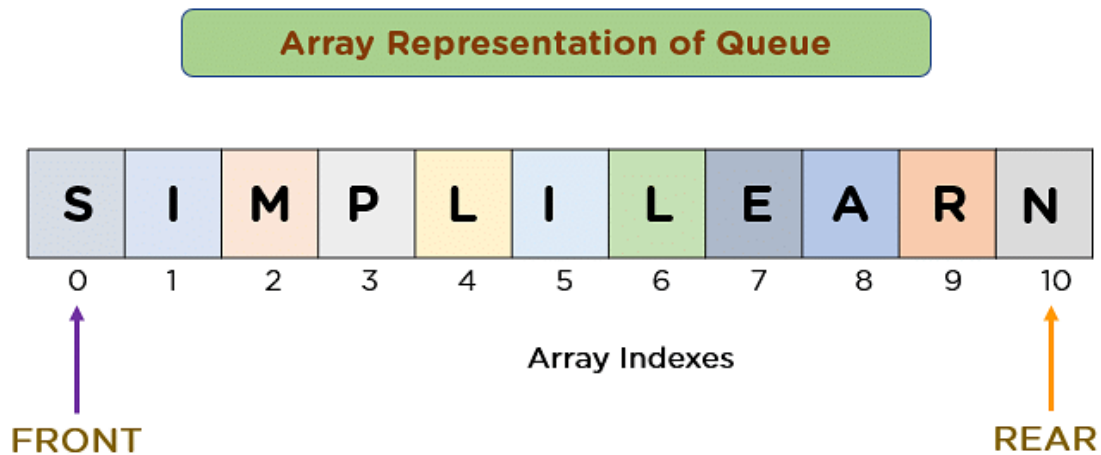
A **Queue Data Structure** is a fundamental concept in computer science used for storing and managing data in a specific order. It follows the principle of "**First in, First out**" (**FIFO**), where the first element added to the queue is the first one to be removed. Queues are commonly used in various algorithms and applications for their simplicity and efficiency in managing data flow. **Queue** is a non-primitive linear data structure that permits insertion of an element at one end and deletion of an element at the other end. The end at which the deletion of an element take place is called **front**, and the end at which insertion of a new element can take place is called **rear**. The deletion or insertion of elements can take place only at the front and rear end of the list respectively.

How to Implement Queue Using Arrays?

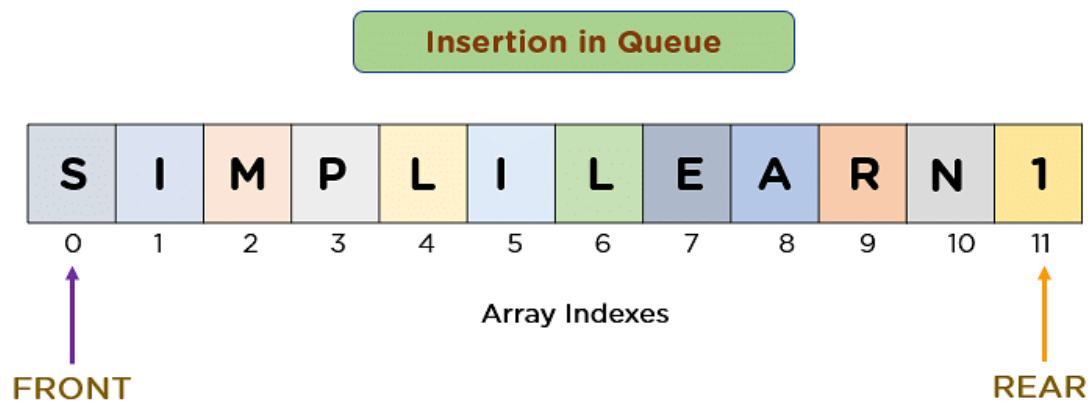
The queue is a linear collection of distinct entities like an array. The fact that the queue possesses some restrictions while performing insertion and deletion is vital. In the case of a queue, it can perform both insertion and deletion only at specific ends, i.e., rear and front nodes. Whereas arrays do not follow any order for these operations. You can illustrate this dissimilarity between a queue and an array using pointers in C++.

You can represent queues in the form of an array using pointers: front and rear.

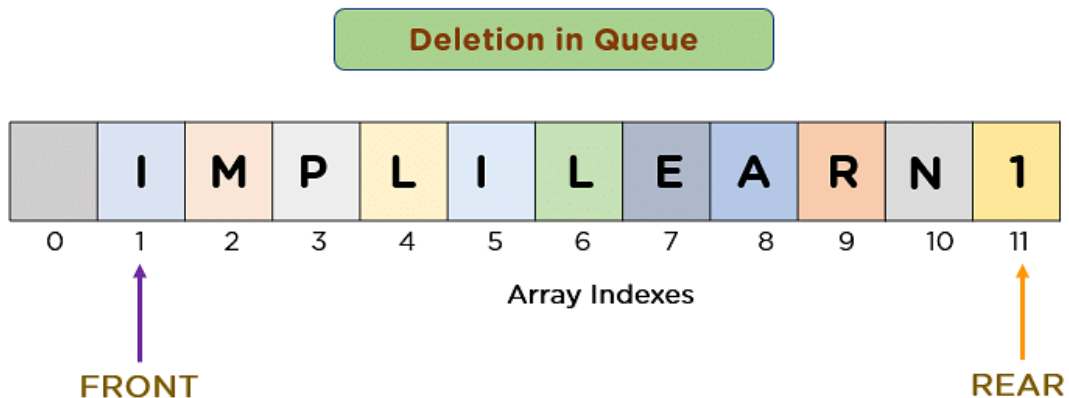
For example, the array shown in the diagram below consists of 11 characters having S at the front and N at the rear node.



The figure above shows the queue of characters forming the word “SIMPLILEARN”. And since N is the last inserted element, its position becomes the rear node. However, if you want to insert more values into the queue, the rear pointer will also increase one by one every time. After you perform an insertion operation on the queue shown in the figure above, the queue will look like below:



The value of the rear pointer becomes 11, whereas the front pointer remains the same. After deleting an element from the queue, the value of the front pointer will change from 0 to 1. The queue will look like below:



Now that you have understood how to represent a queue using an array, implement the supportive queue operations.

Execution of Supportive Queue Operations

The three supportive queue operations that check the state of a queue are `isFull()`, `isEmpty()`, and `Peek()`. These functions do not depend on the number of elements inside the queue or the size of the queue; that is why they take constant execution time, i.e., $O(1)$ [time-complexity]. Here you will implement all the following supportive functions.

`isFull()` Operation

This function checks if the queue is full or not. If the queue is full, then the insertion of elements is not possible in a queue. This condition is known as overflow error. You need to perform the following steps while carrying this operation:

Pseudocode:

Function `isFull()`

 If `Rear == Maxsize - 1`:

 Return "Queue is Full"

 End IF

End `isFull()`

Function in C++:

```
4
5 void isFull(){
6     if(rear == n-1){
7         cout<<"Queue is Full!"<<endl;
8     }
9 }
10
```

`isEmpty()` Operation

This function validates if the queue is empty. If both the front and rear nodes are pointing to null memory space (-1), then you can consider the queue as empty. The pseudocode for this operation is:

Pseudocode:

Function `isEmpty`

 If `Rear == -1 && Front == -1`:

 Return "Queue is Empty"

 End IF

End `isEmpty()`

Function in C++:

```
4
5 void isEmpty(){
6     if(rear == -1 && front == -1){
7         cout<<"Queue is Empty!"<<endl;
8     }
9 }
```

Peek () Operation

This function helps in extracting the data element where the front is pointing without removing it from the queue. The pseudocode for Peek() function is as follows -

Pseudocode:

Function Peek

 If Rear == -1 && Front == -1:

 Return "Queue is Empty"

 Else

 Return queue[front]

 End if

End isFull()

Code in C++:

```
4
5 void Peek(){
6     if(front == -1 && rear == -1){
7         cout<<"The queue is empty!"<<endl;
8     }
9     else{
10        cout<<"The element at the front node is :"<<queue[front]<<endl;
11    }
12 }
13
```

Implementation of Enqueue Operation

The process of inserting elements into the queue is known as Enqueue operation. You perform this operation at the rear node of the queue. The pseudocode for this operation is as follows:

Pseudocode:

Function Enqueue()

 If Rear = MAXSIZE -1:

 Return "Overflow Error"

 ElseIF (Front = -1 and Rear = -1):

 Set Front = Rear = 0

 Else

 Set Rear = Rear + 1

 End if

Set Queue[Rear] = NUM

End Enqueue()

Code in C++:

```
4
5 void Enqueue(){
6     int element;
7     if(rear== n-1){
8         cout<<"Overflow error"<<endl;
9     }
10    else{
11        if(front == -1){
12            front = 0;
13        }
14        cout<<"Enter the element for insertion : "<<endl;
15        cin>>element;
16        rear++;
17        queue[rear] = element;
18    }
19 }
20
```

Implementation of Dequeue Operation

The Dequeue operation is a process of removing elements from the queue. It can only delete an element when there is at least an element to delete, i.e., $Rear > 0$ (queue shouldn't be empty). This function follows the steps listed below while deleting an element from the queue:

Pseudocode:

Function Dequeue()

 If $Rear == -1 \ \&\& \ Front == -1$:

 Return "Underflow Error"

 Else

 Set $Rem = Queue[Front]$

 Set $Front = Front + 1$

 End IF

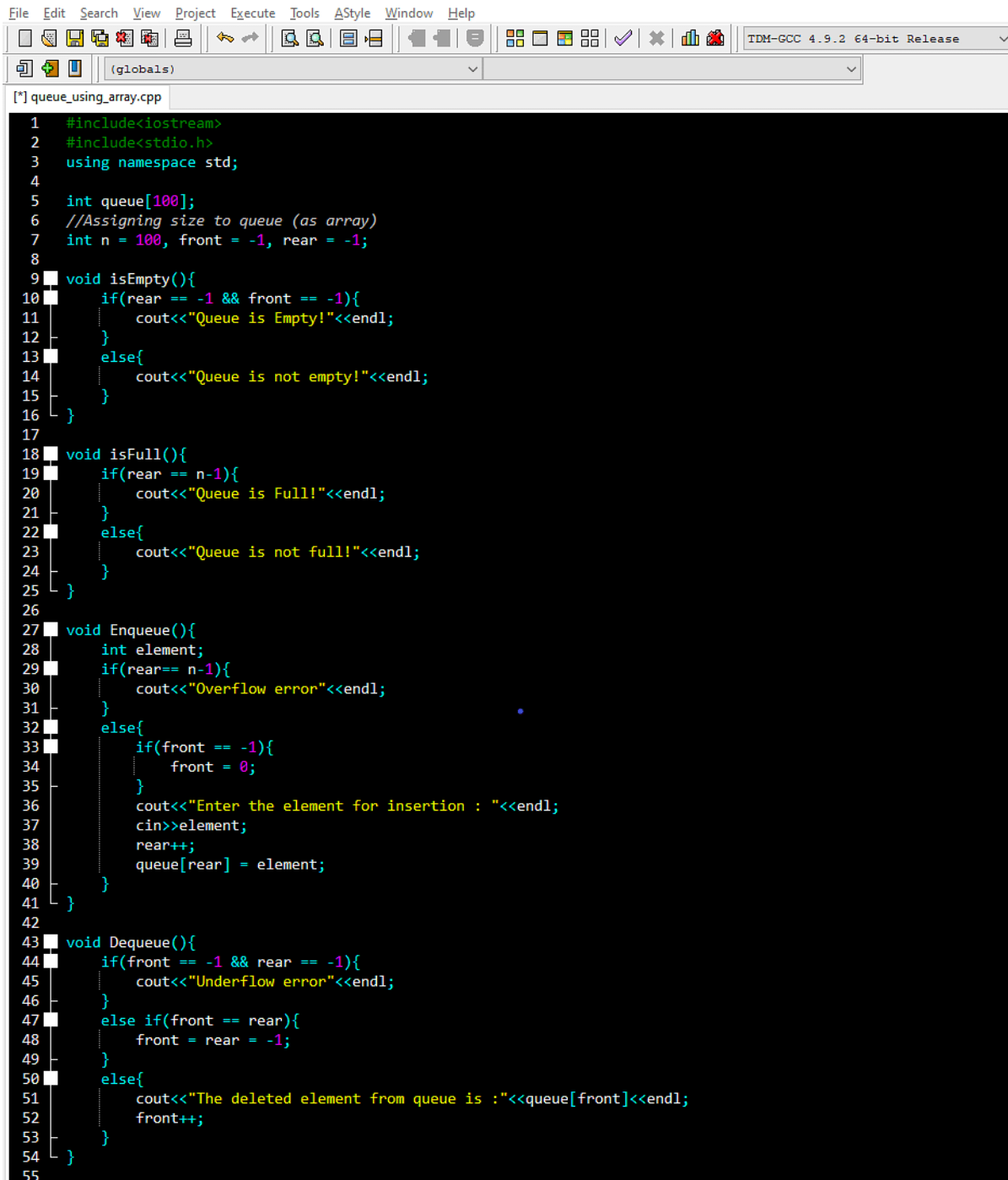
End Dequeue()

Code in C++:

```
4
5 void Dequeue(){
6     if(front == -1 && rear == -1){
7         cout<<"Underflow error"<<endl;
8     }
9     else if(front == rear){
10        front = rear = -1;
11    }
12    else{
13        cout<<"The deleted element from queue is : "<<queue[front]<<endl;
14        front++;
15    }
16 }
17
```

Menu-Driven Program for Queue Implementation Using Array

Now that you are clear with the building blocks of queue operation, it's time to dive in further and formulate a menu-driven C++ program to visualize a queue using an array data structure. You are going to use the switch statement to take input from the user and control the flow of the program.



```
1  #include<iostream>
2  #include<stdio.h>
3  using namespace std;
4
5  int queue[100];
6  //Assigning size to queue (as array)
7  int n = 100, front = -1, rear = -1;
8
9  void isEmpty(){
10     if(rear == -1 && front == -1){
11         cout<<"Queue is Empty!"<<endl;
12     }
13     else{
14         cout<<"Queue is not empty!"<<endl;
15     }
16 }
17
18 void isFull(){
19     if(rear == n-1){
20         cout<<"Queue is Full!"<<endl;
21     }
22     else{
23         cout<<"Queue is not full!"<<endl;
24     }
25 }
26
27 void Enqueue(){
28     int element;
29     if(rear== n-1){
30         cout<<"Overflow error"<<endl;
31     }
32     else{
33         if(front == -1){
34             front = 0;
35         }
36         cout<<"Enter the element for insertion : "<<endl;
37         cin>>element;
38         rear++;
39         queue[rear] = element;
40     }
41 }
42
43 void Dequeue(){
44     if(front == -1 && rear == -1){
45         cout<<"Underflow error"<<endl;
46     }
47     else if(front == rear){
48         front = rear = -1;
49     }
50     else{
51         cout<<"The deleted element from queue is : "<<queue[front]<<endl;
52         front++;
53     }
54 }
55
```

```

56 void Peek(){
57     if(front == -1 && rear == -1){
58         cout<<"The queue is empty!"<<endl;
59     }
60     else{
61         cout<<"The element at the front node is :"<<queue[front]<<endl;
62     }
63 }
64
65
66 void Display() {
67     if (front == - 1)
68         cout<<"Queue is empty"<<endl;
69     else {
70         cout<<"Queue elements are : ";
71         for (int i = front; i <= rear; i++)
72             cout<<queue[i]<<" ";
73         cout<<endl;
74     }
75 }
76
77 int main() {
78     int choice;
79     cout<<"*****Main Menu*****\n";
80     cout<<"\n===== \n";
81     cout<<"1) Insert element to queue"<<endl;
82     cout<<"2) Delete element from queue"<<endl;
83     cout<<"3) Display all the elements of queue"<<endl;
84     cout<<"4) Display element at the front node without deletion"<<endl;
85     cout<<"5) Display if the queue is full or not"<<endl;
86     cout<<"6) Display if the queue is empty or not"<<endl;
87     cout<<"7) Quit"<<endl;
88     do {
89         cout<<"Enter your choice : "<<endl;
90         cin>>choice;
91         switch (choice) {
92             case 1: Enqueue();
93             break;
94             case 2: Dequeue();
95             break;
96             case 3: Display();
97             break;
98             case 4: Peek();
99             break;
100            case 5: isFull();
101            break;
102            case 6: isEmpty();
103            break;
104            case 7: exit(1);
105            default: cout<<"Invalid choice"<<endl;
106        }
107    } while(choice!=7);
108    return 0;
109 }
110

```

Have a look at the results of the menu-driven program for queue implementation using an array. Here, you must insert 3 elements (12,112,45) into a queue with the help of case- 1 And later you will display them with the function display(). Shown below is the output.

```
C:\Users\A\Downloads\queue_using_array.exe
*****Main Menu*****
=====
1) Insert element to queue
2) Delete element from queue
3) Display all the elements of queue
4) Display element at the front node without deletion
5) Display if the queue is full or not
6) Display if the queue is empty or not
7) Quit
Enter your choice :
1
Enter the element for insertion :
12
Enter your choice :
1
Enter the element for insertion :
112
Enter your choice :
1
Enter the element for insertion :
45
Enter your choice :
3
Queue elements are : 12 112 45
Enter your choice :
7

-----
Process exited after 61.07 seconds with return value 1
Press any key to continue . . .
```

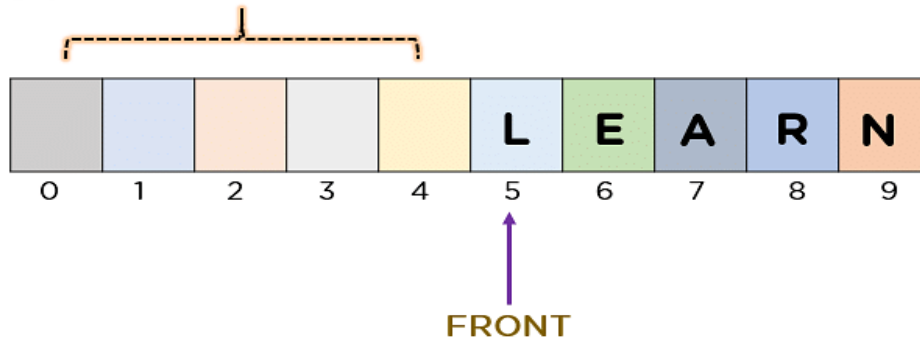
Drawbacks of Queue Implementation Using Array

Although this method of creating a queue using an array is easy, some drawbacks make this method vulnerable. Here, you will explore the drawbacks of queue implementation using array one-by-one:

- **Memory Wastage:** The memory space utilized by the array to store the queue elements can never be re-utilized to store new queue elements. As you can only insert the elements at the front end and the value of the front might be quite high, it can never reuse the blank space before that.

For example, consider the array shown in the figure above. The size of the queue is 10, and the front pointer has already reached location 5, thus wasting newly created empty spaces.

Empty Space created after Deletion of elements!



- **Deciding the array size:** In this method, you have to predetermine the size of an array. And the fact that you can extend the queue at the runtime depending upon the problem makes this method unresistant, as it is almost impossible to extend an array size at runtime.

APPLICATIONS OF QUEUES :

1. Round Robin technique for processor scheduling is implemented using queues.
2. All types of customer service (like railway ticket reservation) center software's are designed using queues to store customers information.

Printer server routines are designed using queues. A number of users share a printer using printer server (a dedicated computer to which a printer is connected), the printer server then spools all the jobs from all the users, to the server's hard disk in a queue. From here jobs are printed one-by-one according to their number in the queue.