SNS COLLEGE OF TECHNOLOGY
(An Autonomous Institution)
COIMBATORE- 641 035

Department of Computer Science and Engineering

23ITT101 - PROGRAMMING IN C AND DATA STRUCTURES

UNIT IV STACK AND QUEUE

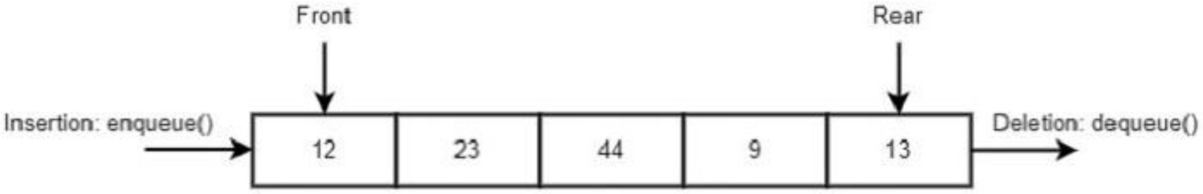
Queue ADT

Queue, like Stack, is also an abstract data structure. The thing that makes queue different from stack is that a queue is open at both its ends. Hence, it follows FIFO (First-In-First-Out) structure, i.e. the data item inserted first will also be accessed first. The data is inserted into the queue through one end and deleted from it using the other end.

A real-world example of queue can be a single-lane one-way road, where the vehicle enters first, exits first. More real-world examples can be seen as queues at the ticket windows and bus-stops.

Representation of Queues

Similar to the stack ADT, a queue ADT can also be implemented using arrays, linked lists, or pointers. As a small example in this tutorial, we implement queues using a one-dimensional array.



Queue: FIFO Operation

Basic Operations

Queue operations also include initialization of a queue, usage and permanently deleting the data from the memory.

The most fundamental operations in the queue ADT include: enqueue(), dequeue(), peek(), isFull(), isEmpty(). These are all built-in operations to carry out data manipulation and to check the status of the queue.

Queue uses two pointers − **front** and **rear**. The front pointer accesses the data from the front end (helping in enqueueing) while the rear pointer accesses data from the rear end (helping in dequeuing).

PROGRAM:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#define max 6
int intarray[max];
int front = 0;
int rear = -1;
int itemcount = 0;
int peek() {
  return intarray[front];
}
bool isempty() {
  return itemcount == 0;
}
bool isfull() {
  return itemcount == max;
}
int size() {
  return itemcount;
}
void insert(int data) {
  if(!isfull()) {
if(rear == max-1) {
      rear = -1;
    }
intarray[++rear] = data;
itemcount++;
  }
}
int removedata() {
int data = intarray[front++];
```

```
if(front == max) {
    front = 0;
  }
itemcount--;
  return data;
}

int main() {
  /* insert 5 items */
insert(3);
insert(5);
insert(9);
insert(1);
insert(12);
  // front : 0
  // rear  : 4
  // ------------------
  // index : 0 1 2 3 4
  // ------------------
  // queue : 3 5 9 1 12
insert(15);
  // front : 0
  // rear  : 5
  // ---------------------
  // index : 0 1 2 3 4  5
  // ---------------------
  // queue : 3 5 9 1 12 15
  if(isfull()) {
printf(“queue is full!\n”);
  }

  // remove one item
intnum = removedata();
printf(“element removed: %d\n”,num);
  // front : 1
  // rear  : 5
  // -------------------
  // index : 1 2 3 4  5
  // -------------------
  // queue : 5 9 1 12 15
  // insert more items
insert(16);
  // front : 1
  // rear  : -1
  // ----------------------
  // index : 0  1 2 3 4  5
  // ----------------------
  // queue : 16 5 9 1 12 15
  // as queue is full, elements will not be inserted.
insert(17);
insert(18);
  // ----------------------
  // index : 0  1 2 3 4  5
  // ----------------------
  // queue : 16 5 9 1 12 15
printf("element at front: %d\n",peek());
printf("----------------------\n");
printf("index : 5 4 3 2  1  0\n");
printf("----------------------\n");
printf("queue:  ");
  while(!isempty()) {
int n = removedata();
printf("%d ",n);
  }
}
```

OUTPUT:

```
Queue is full!
Element removed: 3
Element at front: 5
----------------------
Index : 5 4 3 2 1 0
----------------------
Queue: 5 9 1 12 15 16
```

Implementation of Queue

```
#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<stdbool.h>

#define MAX 6

intintArray[MAX];

int front =0;

int rear =-1;

intitemCount=0;

intpeek(){

returnintArray[front];

}
```

```
bool isEmpty(){
returnitemCount==0;
}
bool isFull(){
returnitemCount== MAX;
}
intsize(){
returnitemCount;
}
voidinsert(int data){
if(!isFull()){
if(rear == MAX-1){
      rear =-1;
}
intArray[++rear]= data;
itemCount++;
}
}
intremoveData(){
int data =intArray[front++];
if(front == MAX){
    front =0;
}
itemCount--;
return data;
}
intmain(){

/* insert 5 items */
insert(3);
insert(5);
insert(9);
insert(1);
insert(12);

// front : 0
// rear : 4
// ------------------
// index : 0 1 2 3 4
// ------------------
// queue : 3 5 9 1 12
insert(15);

// front : 0
// rear : 5
// ---------------------
// index : 0 1 2 3 4 5
// ---------------------
// queue : 3 5 9 1 12 15
if(isFull()){
printf("Queue is full!\n");
}

// remove one item
intnum=removeData();
printf("Element removed: %d\n",num);

// front : 1
// rear : 5
// -------------------
// index : 1 2 3 4 5
// -------------------
// queue : 5 9 1 12 15
// insert more items
```

```
insert(16);


// front : 1

// rear : -1

// ----------------------

// index : 0 1 2 3 4 5

// ----------------------

// queue : 16 5 9 1 12 15

// As queue is full, elements will not be inserted.

insert(17);

insert(18);


// ----------------------

// index : 0 1 2 3 4 5

// ----------------------

// queue : 16 5 9 1 12 15

printf("Element at front: %d\n",peek());

printf("----------------------\n");

printf("index : 5 4 3 2 1 0\n");

printf("----------------------\n");

printf("Queue: ");

while(!isEmpty()){

int n =removeData();

printf("%d ",n);

}

}
```

Output

```
Queue is full!
Element removed: 3
Element at front: 5
----------------------
index : 5 4 3 2 1 0
----------------------
Queue: 5 9 1 12 15 16
```

Insertion operation: enqueue()

The *enqueue()* is a data manipulation operation that is used to insert elements into the stack. The following algorithm describes the enqueue() operation in a simpler way.

Algorithm

```
1 − START
2 – Check if the queue is full.
3 − If the queue is full, produce overflow error and exit.
4 − If the queue is not full, increment rear pointer to point the next empty space.
5 − Add data element to the queue location, where the rear is pointing.
6 − return success.
7 – END
```

Example

Following are the implementations of this operation

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<stdbool.h>
#define MAX 6
intintArray[MAX];
int front =0;
int rear =-1;
intitemCount=0;
bool isFull(){
returnitemCount== MAX;
}
bool isEmpty(){
returnitemCount==0;
}
intremoveData(){
int data =intArray[front++];
if(front == MAX){
    front =0;
}
itemCount--;
return data;
}
voidinsert(int data){
if(!isFull()){
if(rear == MAX-1){
      rear =-1;
}
intArray[++rear]= data;
itemCount++;
}
}
intmain(){
insert(3);
insert(5);
insert(9);
insert(1);
insert(12);
insert(15);
printf("Queue: ");
while(!isEmpty()){
int n =removeData();
printf("%d ",n);
}
}
```

Output

```
Queue: 3 5 9 1 12 15
```

Deletion Operation: dequeue()

The *dequeue()* is a data manipulation operation that is used to remove elements from the stack. The following algorithm describes the dequeue() operation in a simpler way.

Algorithm

```
1 – START
2 − Check if the queue is empty.
3 − If the queue is empty, produce underflow error and exit.
```

```
4 − If the queue is not empty, access the data where front is pointing.
5 − Increment front pointer to point to the next available data element.
6 − Return success.
7 – END
```

Example

Following are the implementations of this operation

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<stdbool.h>
#define MAX 6
intintArray[MAX];
int front =0;
int rear =-1;
intitemCount=0;
bool isFull(){
returnitemCount== MAX;
}
bool isEmpty(){
returnitemCount==0;
}
voidinsert(int data){
if(!isFull()){
if(rear == MAX-1){
        rear =-1;
}
intArray[++rear]= data;
itemCount++;
}
}
intremoveData(){
int data =intArray[front++];
if(front == MAX){
     front =0;
}
itemCount--;
return data;
}
intmain(){
inti;

/* insert 5 items */
insert(3);
insert(5);
insert(9);
insert(1);
insert(12);
insert(15);
printf("Queue: ");
for(i=0;i< MAX;i++)
printf("%d ",intArray[i]);

// remove one item
intnum=removeData();
printf("\nElement removed: %d\n",num);
printf("Updated Queue: ");
while(!isEmpty()){
int n =removeData();
printf("%d ",n);
}
}
```

Output

```
Queue: 3 5 9 1 12 15
Element removed: 3
Updated Queue: 5 9 1 12 15
```

The peek() Operation

The peek() is an operation which is used to retrieve the frontmost element in the queue, without deleting it. This operation is used to check the status of the queue with the help of the pointer.

Algorithm

```
1 – START
2 – Return the element at the front of the queue
3 – END
```

Example

Following are the implementations of this operation

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<stdbool.h>
#define MAX 6
intintArray[MAX];
int front =0;
int rear =-1;
intitemCount=0;
intpeek(){
returnintArray[front];
}
bool isFull(){
returnitemCount== MAX;
}
voidinsert(int data){
if(!isFull()){
if(rear == MAX-1){
        rear =-1;
}
intArray[++rear]= data;
itemCount++;
}
}
```

```
intmain(){

inti;


/* insert 5 items */

insert(3);

insert(5);

insert(9);

insert(1);

insert(12);

insert(15);

printf("Queue: ");

for(i=0;i< MAX;i++)

printf("%d ",intArray[i]);

printf("\nElement at front: %d\n",peek());

}
```

Output

```
Queue: 3 5 9 1 12 15
Element at front: 3
```

The isFull() Operation

The isFull() operation verifies whether the stack is full.

Algorithm

```
1 – START
2 – If the count of queue elements equals the queue size, return true
3 – Otherwise, return false
4 – END
```

Example

Following are the implementations of this operation

```
#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<stdbool.h>

#define MAX 6
```

```
intintArray[MAX];
int front =0;
int rear =-1;
intitemCount=0;
bool isFull(){
returnitemCount== MAX;
}
voidinsert(int data){
if(!isFull()){
if(rear == MAX-1){
        rear =-1;
}
intArray[++rear]= data;
itemCount++;
}
}
intmain(){
inti;
/* insert 5 items */
insert(3);
insert(5);
insert(9);
insert(1);
insert(12);
insert(15);
printf("Queue: ");
for(i=0;i< MAX;i++)
printf("%d ",intArray[i]);
printf("\n");
if(isFull()){
printf("Queue is full!\n");
}
}
```

Output

```
Queue: 3 5 9 1 12 15
Queue is full!
```

The isEmpty() operation

The isEmpty() operation verifies whether the stack is empty. This operation is used to check the status of the stack with the help of top pointer.

Algorithm

```
1 – START
2 – If the count of queue elements equals zero, return true
3 – Otherwise, return false
4 – END
```

Example

Following are the implementations of this operation

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<stdbool.h>
#define MAX 6
intintArray[MAX];
int front =0;
int rear =-1;
intitemCount=0;
bool isEmpty(){
returnitemCount==0;
}
intmain(){
inti;
printf("Queue: ");
for(i=0;i< MAX;i++)
printf("%d ",intArray[i]);
printf("\n");
if(isEmpty()){
```

```
printf("Queue is Empty!\n");

}

}
```

Output

```
Queue: 0 0 0 0 0 0
Queue is Empty!
```