QUEUES

NATIONAL UNIVERSITY OF TECHNOLOGY (NUTECH)

DR. SAMAN RIAZ

LECTURE # 10

QUEUES

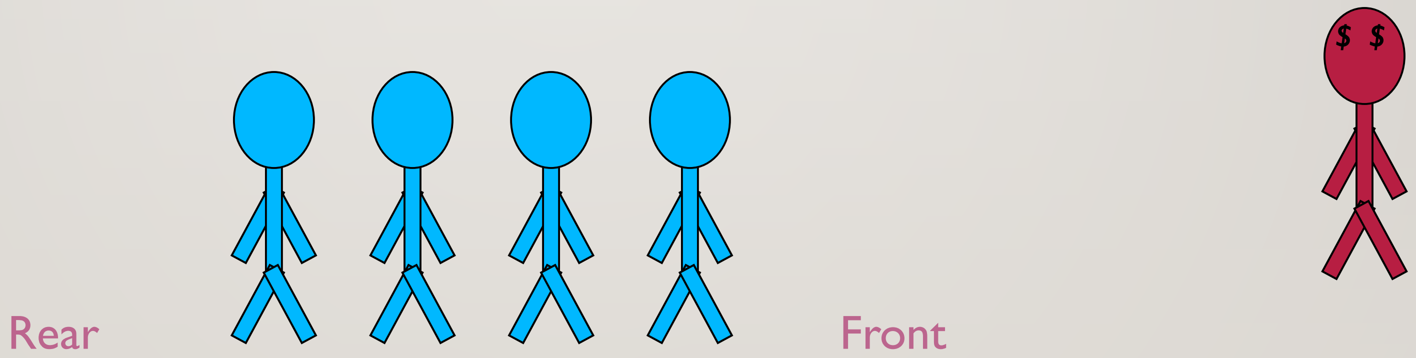
“A ***Queue*** is a special kind of list, where items are inserted at one end (***the rear***) And deleted at the other end (***the front***)”

Other Name:

- First In First Out (FIFO)

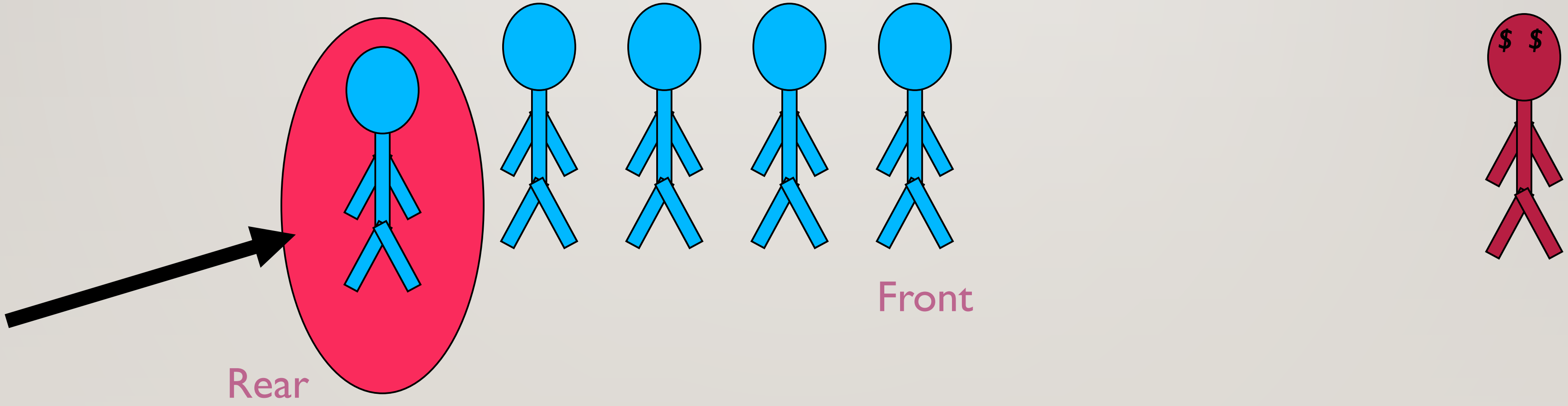
QUEUES

- A queue is like a line of people waiting for a bank teller. The queue has a **front** and a **rear**.



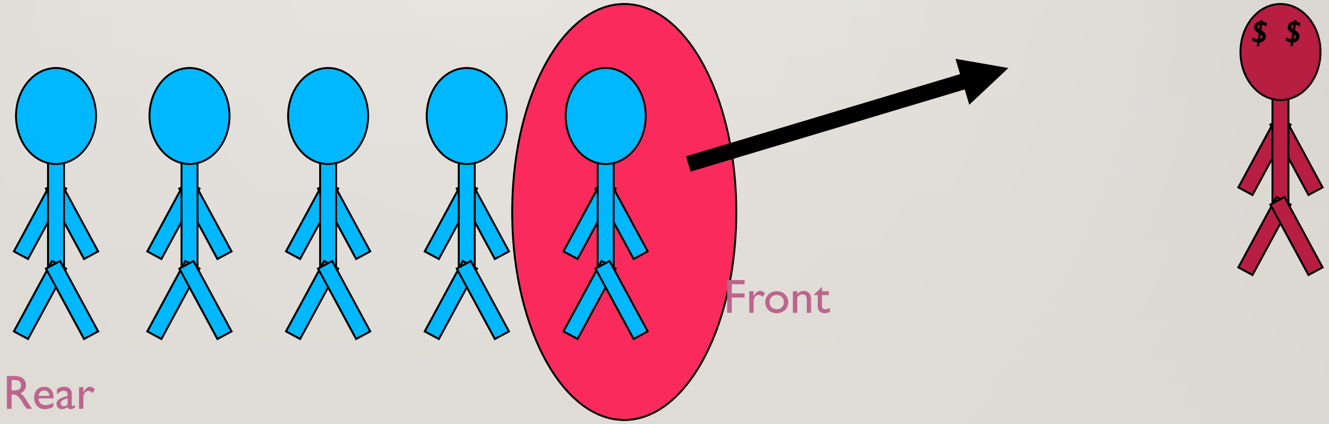
QUEUES

- □New people must enter the queue at the rear.



QUEUES

- When an item is taken from the queue, it always comes from the front.



SOME EXAMPLES

- Billing counter
  - Booking movie tickets
  - Queue for paying bills

- A print queue
- Vehicles on toll-tax bridge
- Luggage checking machine

- Some others?

APPLICATIONS OF QUEUES

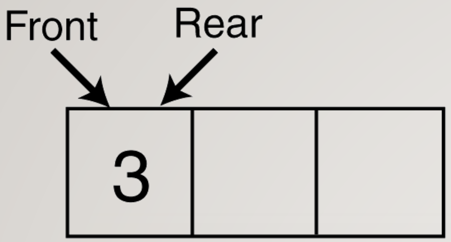
- Operating system
  - multi-user/multitasking environments, where several users or task may be requesting the same resource simultaneously.

- Communication Software
  - queues to hold *information* received over networks and dial up connections. (Information can be transmitted faster than it can be processed, so is placed in a queue waiting to be processed)

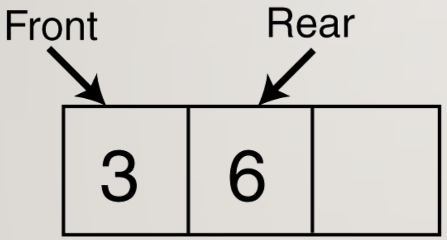
COMMON OPERATIONS (QUEUE ADT)

1. **MAKENULL(Q):** Makes Queue Q be an empty list.
2. **FRONT(Q):** Returns the first element on Queue Q.
3. **ENQUEUE(*x*,Q):** Inserts element x at the end of Queue Q.
4. **DEQUEUE(Q):** Deletes the first element of *Q*.
5. **EMPTY(Q):** Returns true if and only if Q is an empty queue.

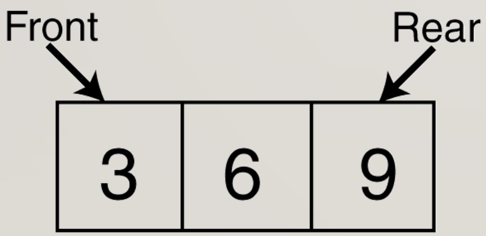
Enqueue(3);



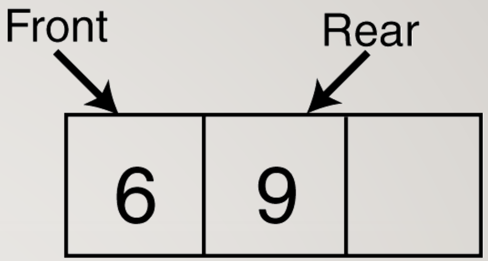
Enqueue(6);



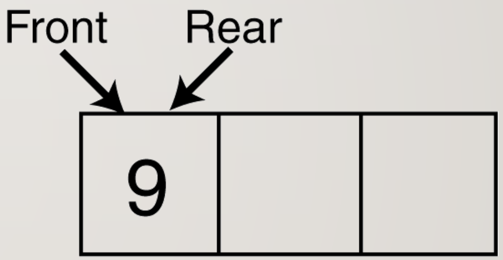
Enqueue(9);



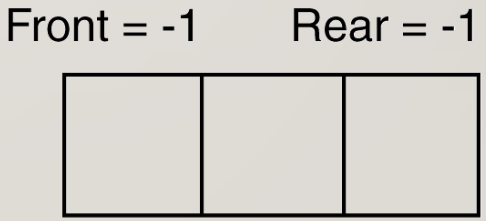
Dequeue();



Dequeue();



Dequeue();



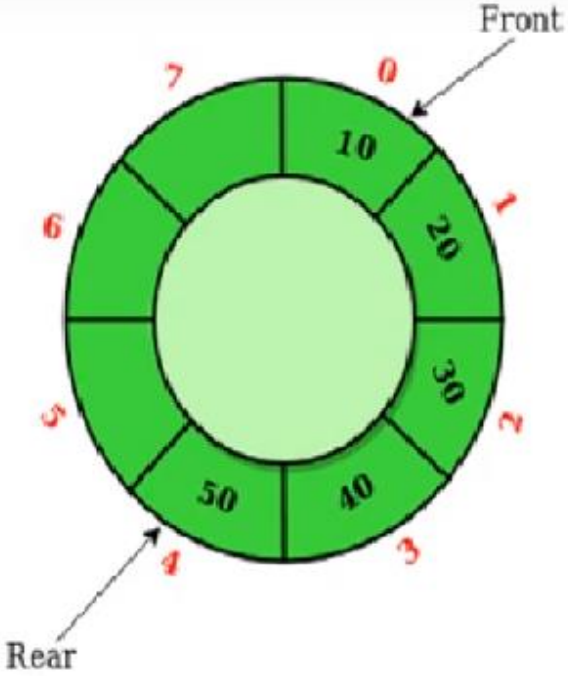
IMPLEMENTATION

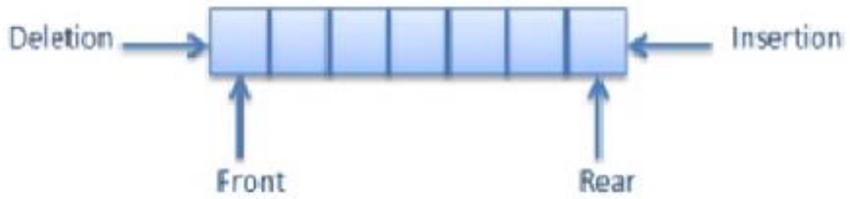
- Static
  - Queue is implemented by an array, and size of queue remains fix

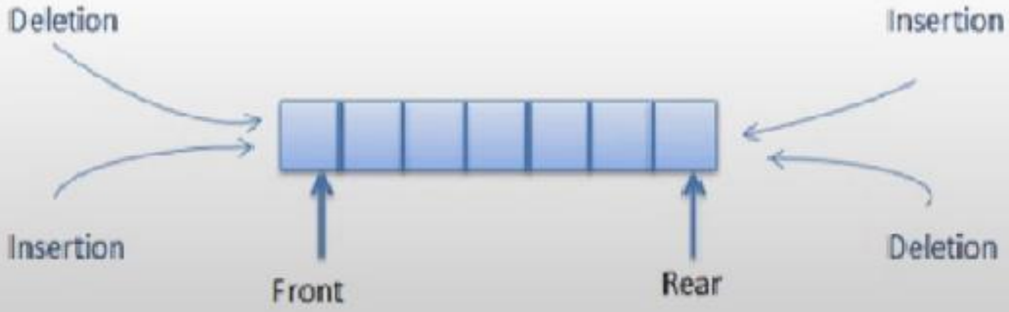
- Dynamic
  - A **queue** can be **implemented** as a **linked list**, and *expand* or *shrink* with each *enqueue* or *dequeue* operation.

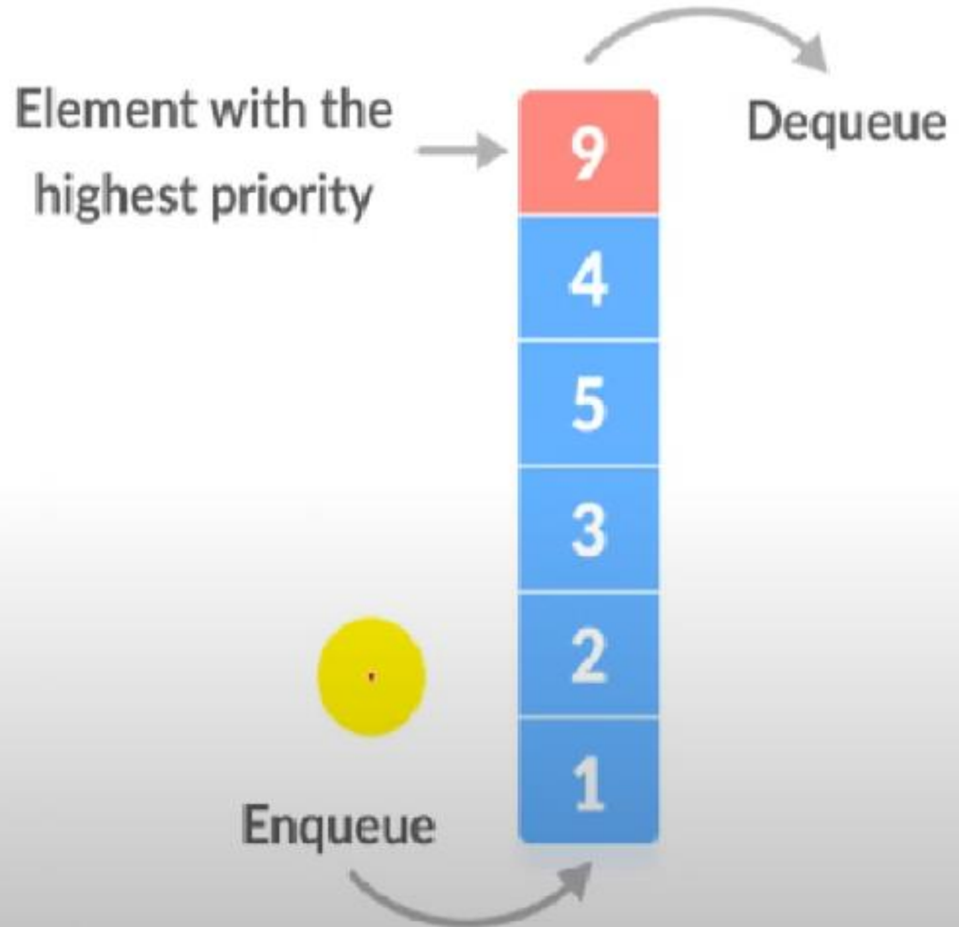
TYPE OF QUEUE DATA STRUCTURE

- Simple **Queue**
- Circular **Queue**
- Priority **Queue**
- Doubly Ended **Queue**
  - **Input Restricted Deque**
  - **Output Restricted Deque**







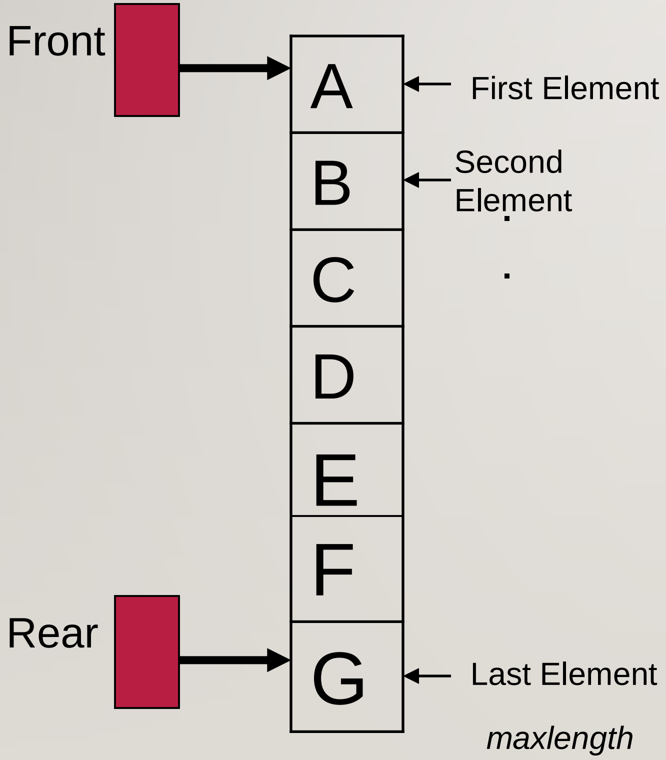


ARRAY IMPLEMENTATION

- Signify *zero* index as front.
- Dequeue
  - Shift elements to the left
  - Expensive!
- Equeue
  - Need to save index of last item inserted
    - On Enqueue, increment index
    - On Dequeue, decrement index

ALTERNATIVE ARRAY IMPLEMENTATION

- Use two counters that signify rear and front

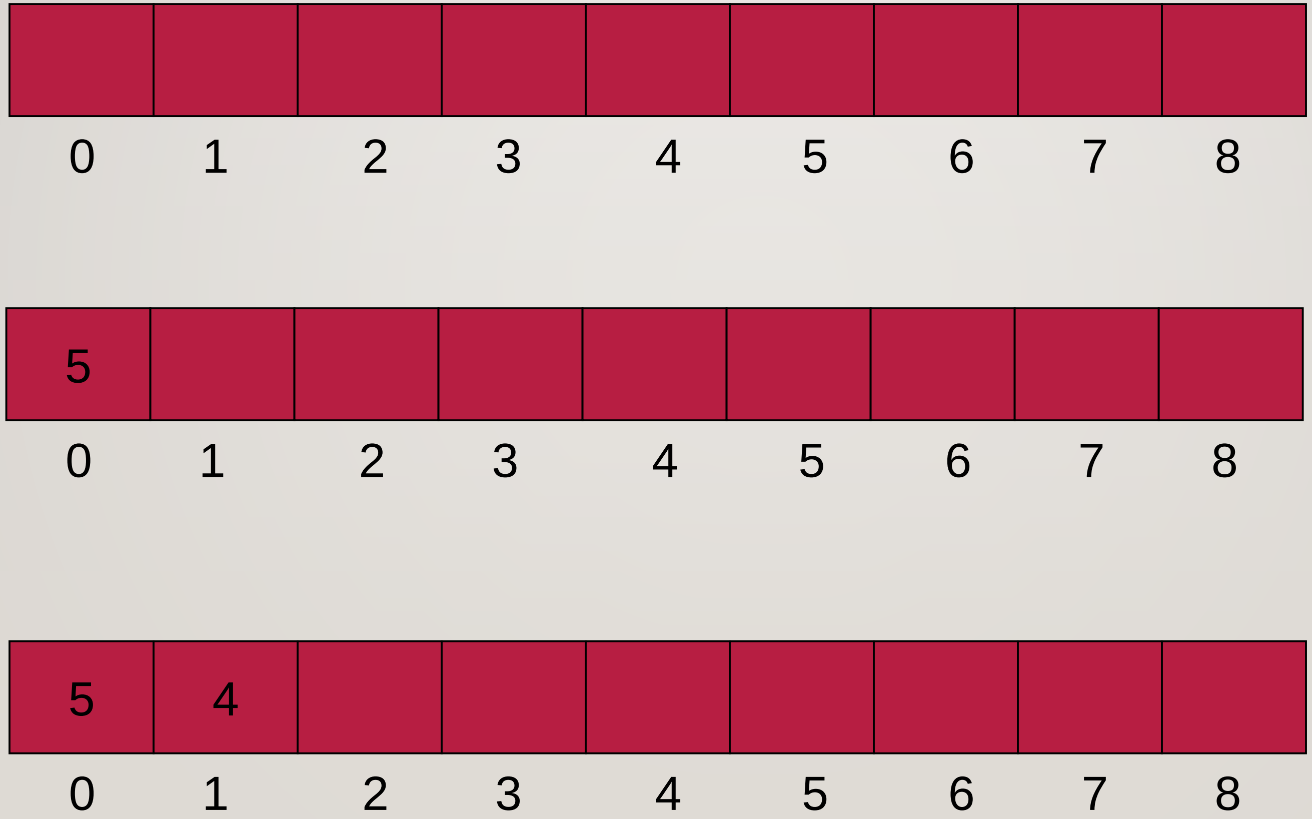


When queue is empty both front and rear are set to -1

While enqueueing increment rear by 1, and while dequeueing increment front by 1

When there is only one value in the Queue, both rear and front have same index

ARRAY IMPLEMENTATION



0	1	2	3	4	5	6	7	8

Front= -1
Rear = -1

5								
0	1	2	3	4	5	6	7	8

Front= 0
Rear = 0

5	4							
0	1	2	3	4	5	6	7	8

Front= 0
Rear = 1

ARRAY IMPLEMENTATION

5	4	6	7	8	7	6		
0	1	2	3	4	5	6	7	8

Front=0
Rear=6

				8	7	6		
0	1	2	3	4	5	6	7	8

Front=4
Rear=6

ARRAY IMPLEMENTATION

					7	6	12	67
0	1	2	3	4	5	6	7	8

Front=5
Rear=8

How can we insert more elements? Rear index can not move beyond the last element….

SOLUTION: USING CIRCULAR QUEUE

- Allow rear to wrap around the array.

```
if(rear == queueSize-1)
            rear = 0;
else
            rear++;
```

- Or use module arithmetic

```
rear = (rear + 1) % queueSize;
```

					7	6	12	67
0	1	2	3	4	5	6	7	8

Front=5
Rear=8

Enqueue 39 Rear=(Rear+1) mod Queue Size = (8+1) mod 9 = 0

39					7	6	12	67
0	1	2	3	4	5	6	7	8

Front=5
Rear=0

HOW TO DETERMINE EMPTY AND FULL QUEUES?

- It can be somewhat tricky
- Number of approaches
  - A counter indicating number of values in the queue can be used (we will use this approach)
  - Later, we will see another approach

IMPLEMENTATION

```
class IntQueue
{
private:
    int *queueArray;
    int queueSize;
    int front;
    int rear;
    int numItems;
public:
    IntQueue(int);
    ~IntQueue(void);
    void enqueue(int);
    int dequeue(void);
    bool isEmpty(void);
    bool isFull(void);
    void clear(void);
};
```

Note, the member function clear, which clears the queue by resetting the front and rear indices, and setting the numItems to 0.

```
IntQueue::IntQueue(int s) //constructor
{
     queueArray = new int[s];
     queueSize = s;
     front = -1;
     rear = -1;
     numItems = 0;
}


IntQueue::~IntQueue(void) //destructor
{
      delete [] queueArray;
}
```

```
//*********************************************
// Function isEmpty returns true if the queue *
// is empty, and false otherwise.             *
//*********************************************

bool IntQueue::isEmpty(void)
{
   if (numItems)
      return false;
   else
      return true;
}
```

```
//********************************************
// Function isFull returns true if the queue *
// is full, and false otherwise.             *
//********************************************

bool IntQueue::isFull(void)
{
   if (numItems < queueSize)
      return false;
   else
      return true;
}
```

```
//********************************************
// Function enqueue inserts the value in num *
// at the rear of the queue.                 *
//********************************************

void IntQueue::enqueue(int num)
{
   if (isFull())
      cout << "The queue is full.\n";
   else
   {
      // Calculate the new rear position
      rear = (rear + 1) % queueSize;
      // Insert new item
      queueArray[rear] = num;
      // Update item count
      numItems++;
   }
}
```

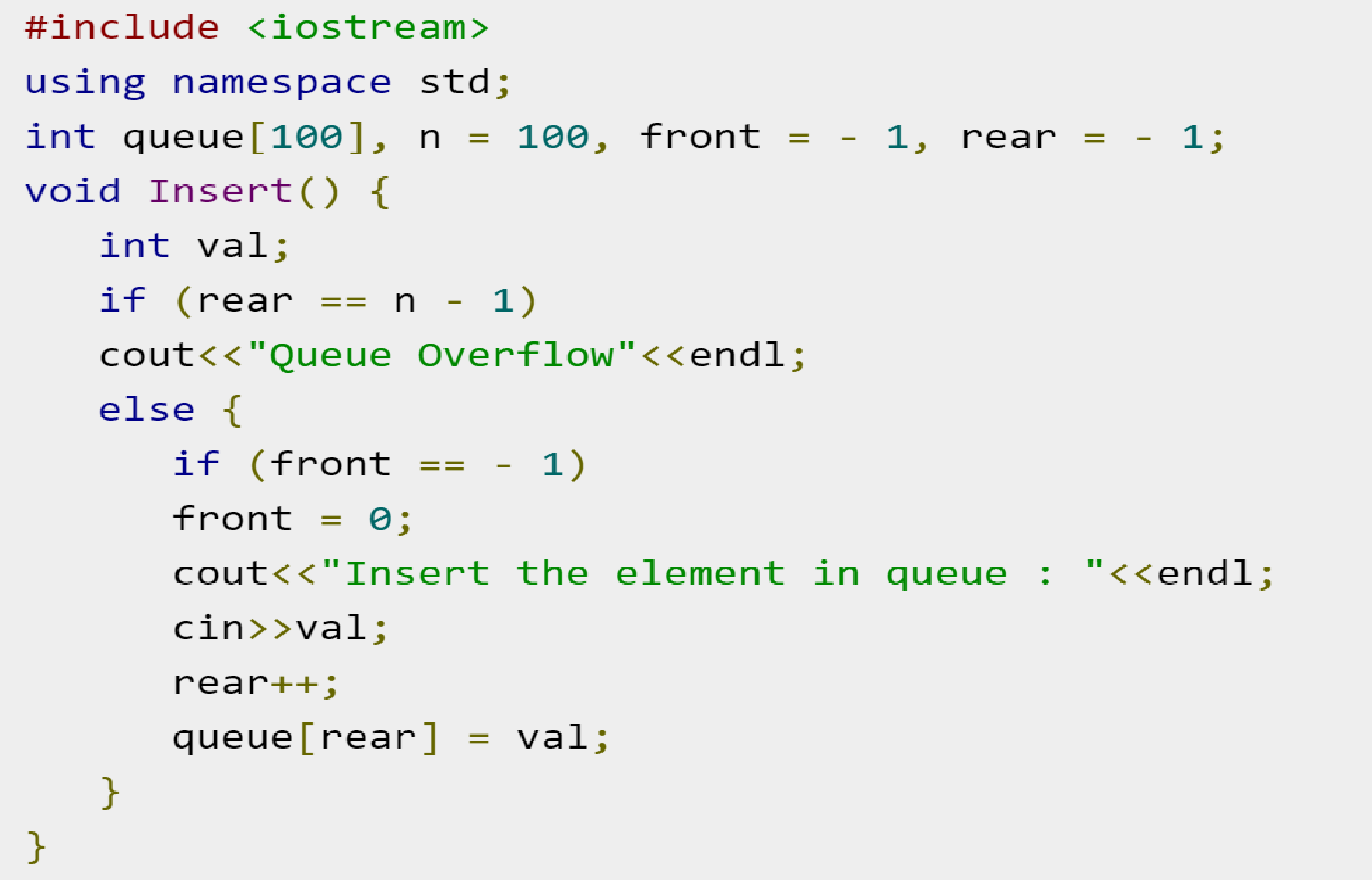
```
//*********************************************
// Function dequeue removes the value at the  *
// front of the queue, and copies it into num.*
//*********************************************

bool IntQueue::dequeue(int &num)
{
   if (isEmpty())
   {
      cout << "The queue is empty.\n";
      return false;
   }
   // Retrieve the front item
   num = queueArray[front];
   // Move front
   front = (front + 1) % queueSize;
   // Update item count
   numItems--;
   if (numItems == 0) front=rear=-1;
   return true;
}
```

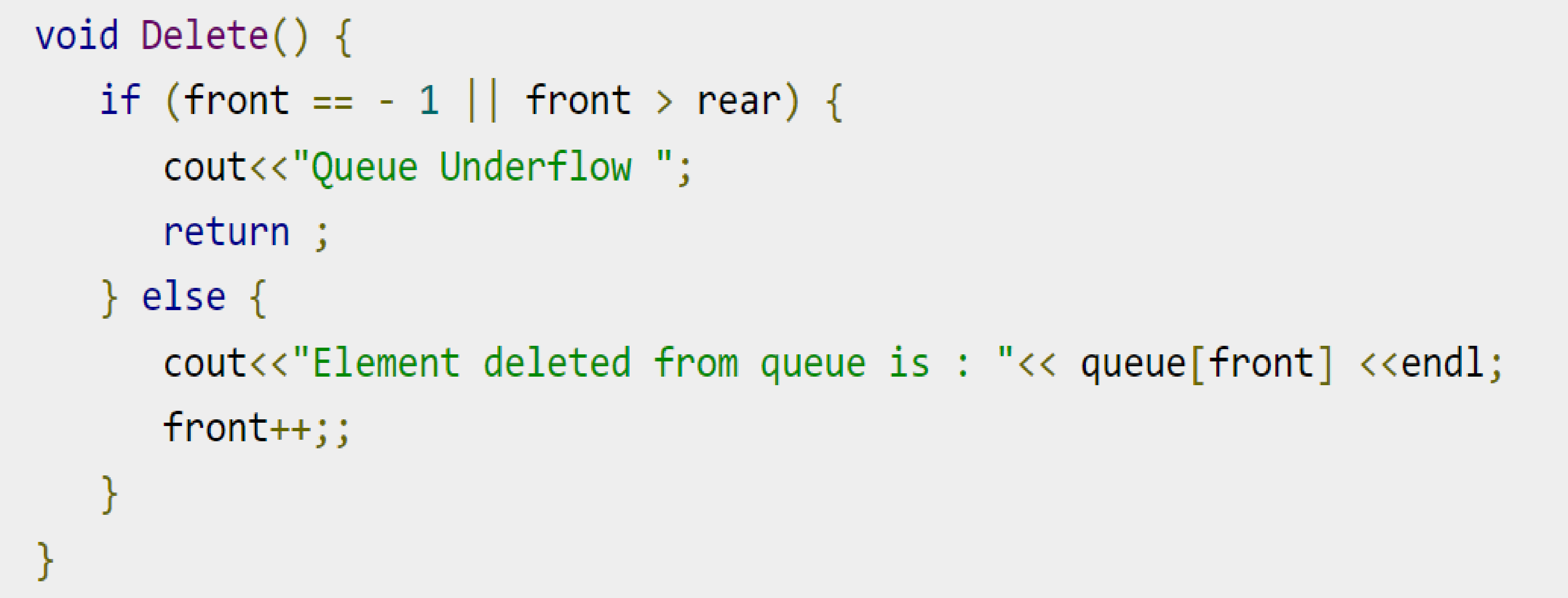
```
//*******************************************
// Function clear resets the front and rear *
// indices, and sets numItems to 0.         *
//*******************************************

void IntQueue::clear(void)
{
   front = - 1;
   rear = - 1;
   numItems = 0;
}
```

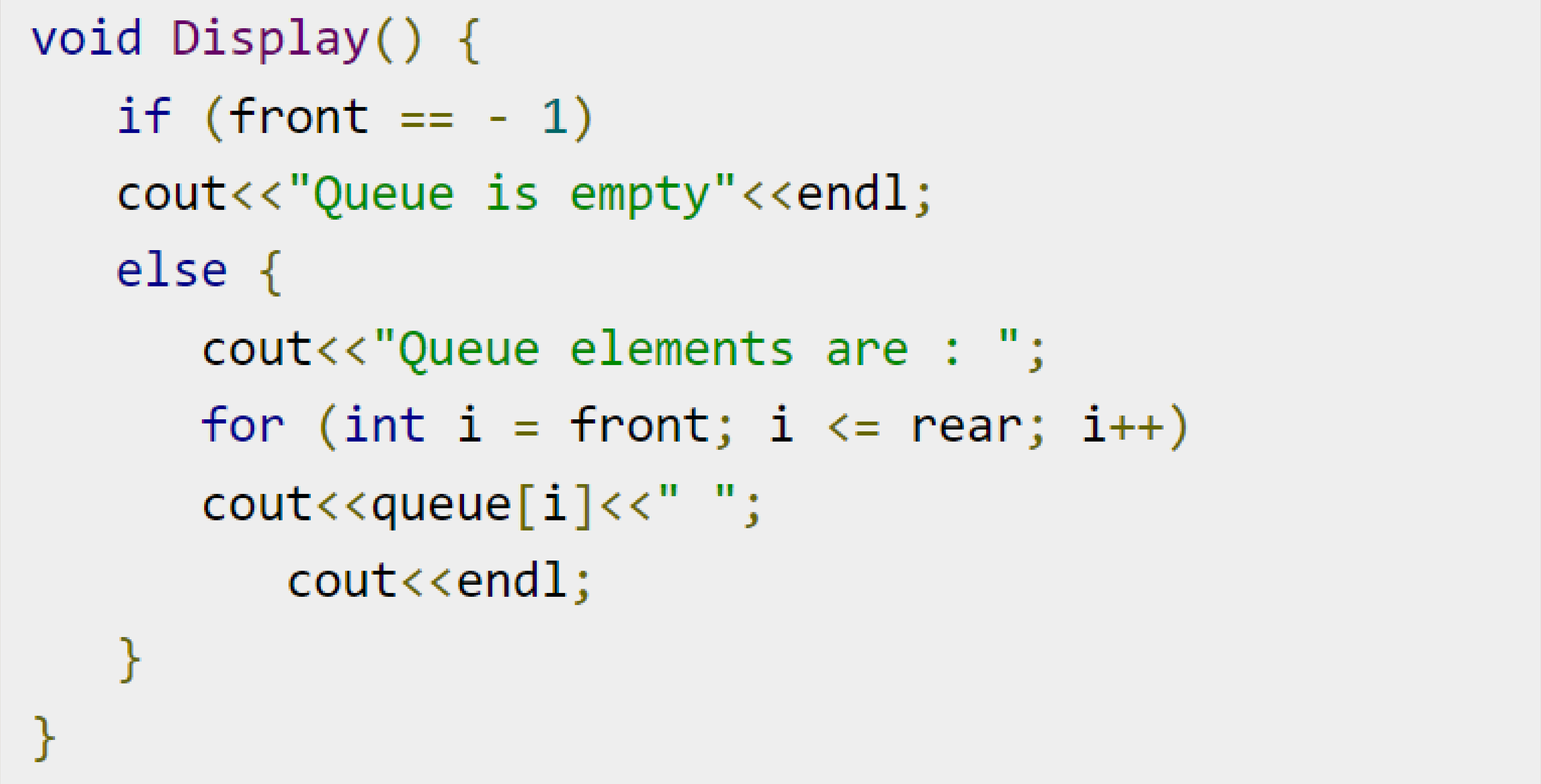
EXAMPLE:



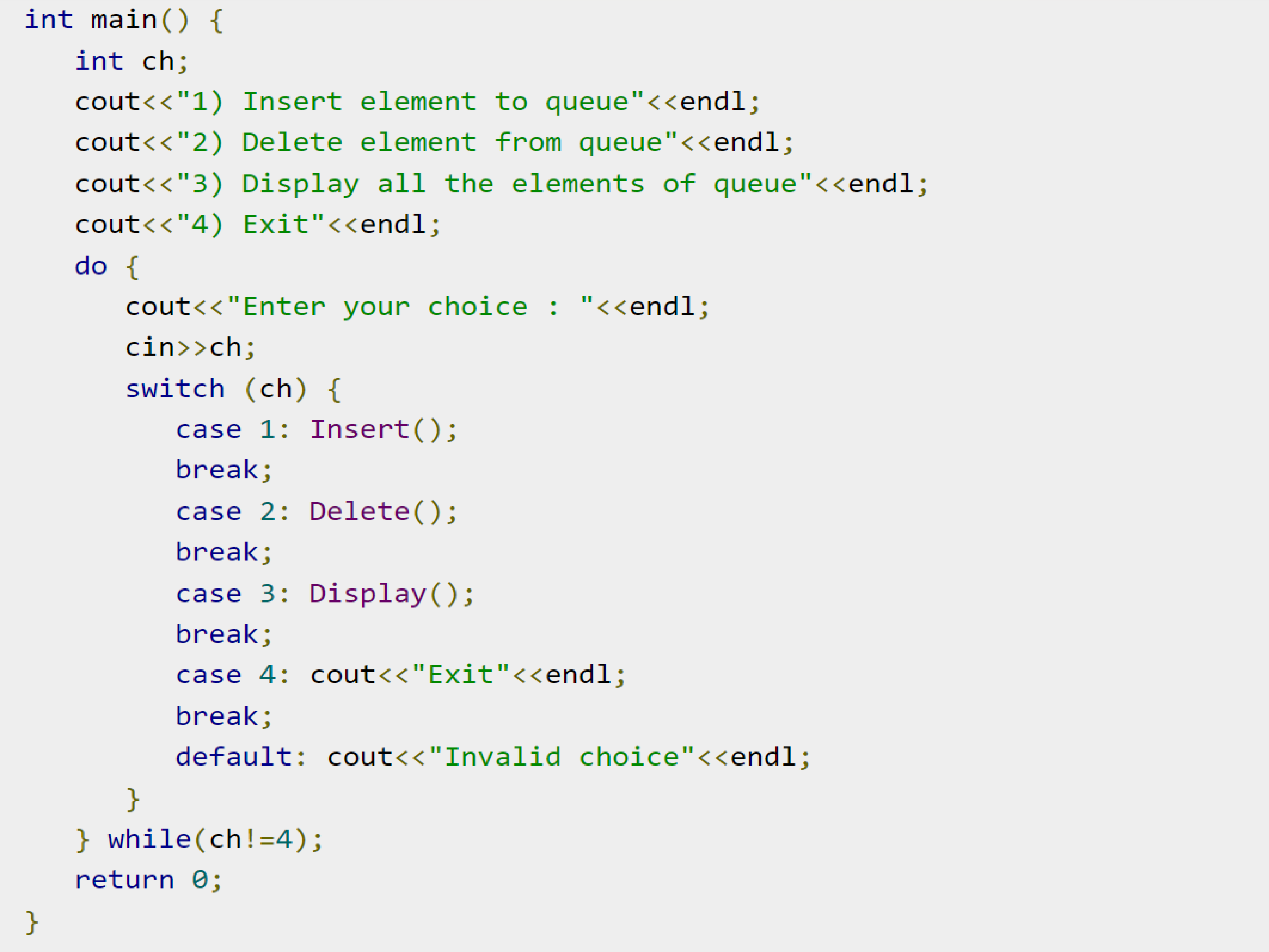
```cpp
#include <iostream>
using namespace std;
int queue[100], n = 100, front = - 1, rear = - 1;
void Insert() {
   int val;
   if (rear == n - 1)
   cout<<"Queue Overflow"<<endl;
   else {
      if (front == - 1)
      front = 0;
      cout<<"Insert the element in queue : "<<endl;
      cin>>val;
      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++;;
   }
}
```

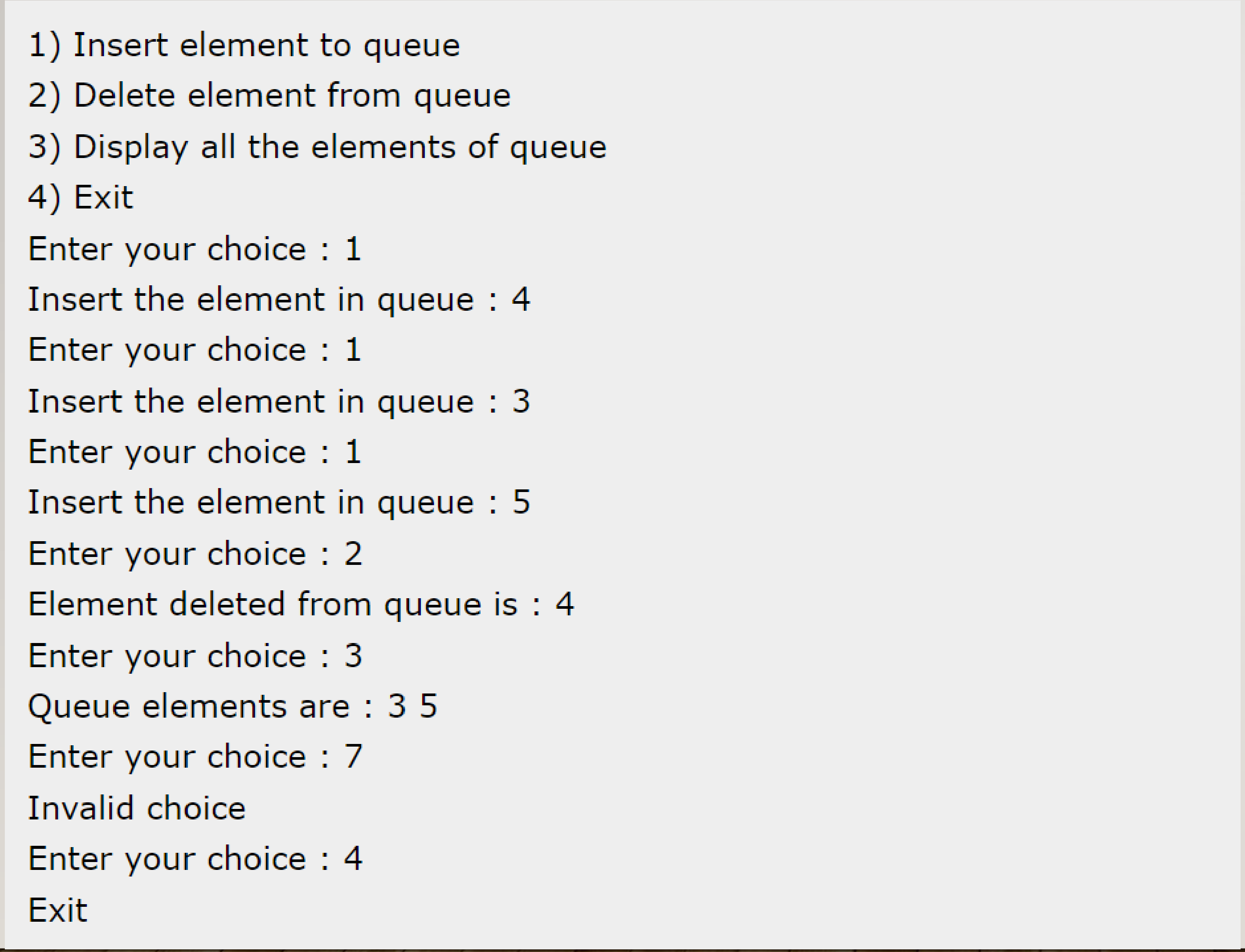


```
void Display() {
   if (front == - 1)
   cout<<"Queue is empty"<<endl;
   else {
      cout<<"Queue elements are : ";
      for (int i = front; i <= rear; i++)
      cout<<queue[i]<<" ";
         cout<<endl;
   }
}
```



```
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;
}
```

OUTPUT:



```
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
```