

JASKIRAT SINGH (2020CSC1008)

CPP code to illustrate Queue in Standard Template Library (STL)

```
#include <iostream>

#include <queue>

using namespace std;

// Print the queue
void print(queue<int> gq)
{
    queue<int> g = gq;
    while (!g.empty()) {
        cout << '\t' << g.front();
        g.pop();
    }
    cout << '\n';
}

// Driver Code
int main()
{
    queue<int> que;

    //push() function adds the element 'g' at the end of the queue.
    que.push(50);
    que.push(60);
```

```
que.push(70);
```

```
cout << "The queue is : ";
```

```
print(que);
```

```
//Returns the size of the queue.
```

```
cout << "\nQUEUE.size() : " << que.size();
```

```
//front() function returns a reference to the first element of the queue.
```

```
cout << "\nQUEUE.front() : " << que.front();
```

```
//back() function returns a reference to the last element of the queue.
```

```
cout << "\nQUEUE.back() : " << que.back();
```

```
//pop() function deletes the first element of the queue.
```

```
cout << "\nQUEUE.pop() : ";
```

```
que.pop();
```

```
print(que);
```

```
queue<int> que1;
```

```
que1.push(10);
```

```
    que1.push(20);
```

```
    que1.push(30);
```

```
que1.push(40);
```

```
cout<<"\nQUEUE2 : ";
```

```
print(que1);
```

```
//Exchange the contents of two queues but the queues must be of same type, although sizes may differ.
```

```
que.swap(que1);
```

```
cout<<"\nQueues after swap : ";
```

```
cout<<"\nQUEUE1 : ";
```

```

    print(que);

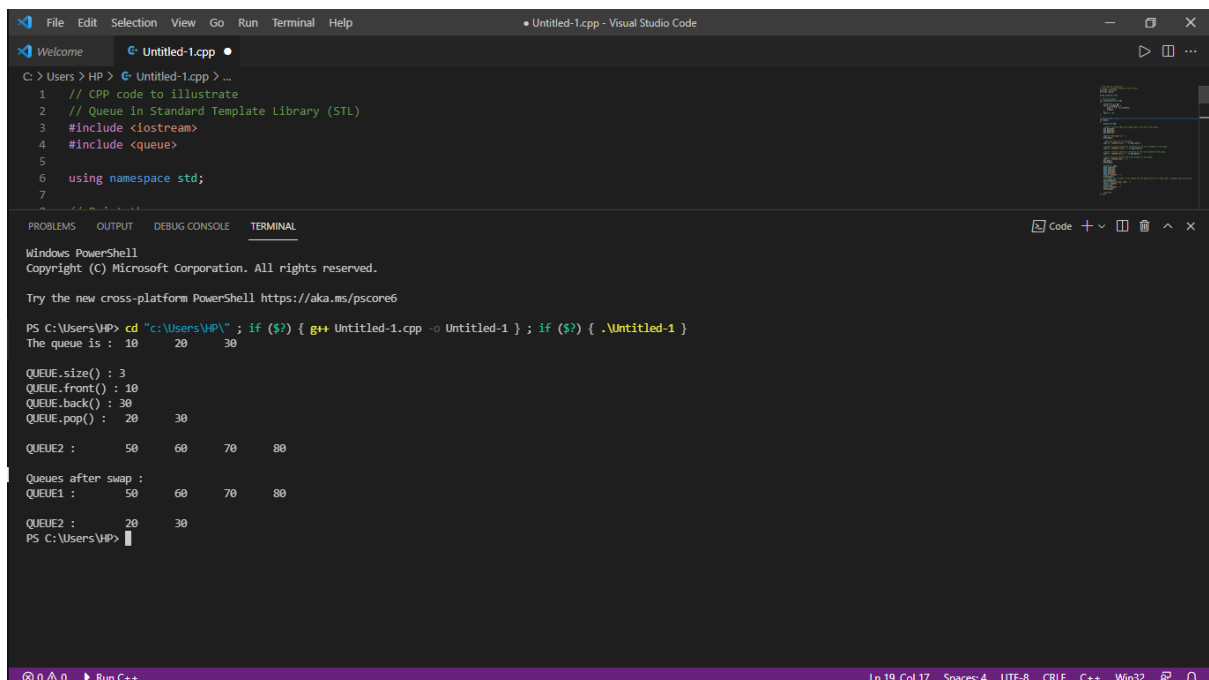
    cout<<"\nQUEUE2 : ";

    print(que1);


    return 0;
}

```

output



The screenshot shows a Visual Studio Code window with a C++ file named 'Untitled-1.cpp'. The code in the editor is as follows:

```

1 // CPP code to illustrate
2 // Queue in Standard Template Library (STL)
3 #include <iostream>
4 #include <queue>
5
6 using namespace std;
7
8 int main()
9 {
10     queue<int> que;
11     que.push(10);
12     que.push(20);
13     que.push(30);
14
15     cout<<"\nQueue size : " << que.size() << endl;
16     cout<<"\nQueue front : " << que.front() << endl;
17     cout<<"\nQueue back : " << que.back() << endl;
18     cout<<"\nQueue pop : " << que.pop() << endl;
19
20     queue<int> que1;
21     que1.push(50);
22     que1.push(60);
23     que1.push(70);
24     que1.push(80);
25
26     cout<<"\nQueues after swap : " << endl;
27     cout<<"\nQueue1 : " << que1.front() << " " << que1.back() << endl;
28     cout<<"\nQueue2 : " << que2.front() << " " << que2.back() << endl;
29
30     return 0;
31 }

```

The terminal output shows the execution of the program:

```

PS C:\Users\HP> cd "C:\Users\HP\" ; if ($?) { g++ Untitled-1.cpp -o Untitled-1 } ; if ($?) { .\Untitled-1 }
The queue is : 10 20 30

QUEUE.size() : 3
QUEUE.front() : 10
QUEUE.back() : 30
QUEUE.pop() : 20 30

QUEUE2 : 50 60 70 80

Queues after swap :
QUEUE1 : 50 60 70 80
QUEUE2 : 20 30
PS C:\Users\HP>

```

Implement a templated Queue using Array.

```

#include <iostream>

using namespace std;

```

```

// A class to represent a queue

```

```

template <class X>
class queue
{
    X *arr;    // array to store queue elements
    int capacity; // maximum capacity of the queue
    int front;  // front points to the front element in the queue (if any)
    int rear;   // rear points to the last element in the queue

```

```

public:

```

```

    queue(int size); // constructor
    ~queue() { delete[] arr; } //destructor

```

```

    void dequeue();
    void enqueue(X x);
    X peek();
    int size();
    bool isEmpty();
    bool isFull();

```

```

};

```

```

// Constructor to initialize a queue

```

```

template <class X>
queue<X>::queue(int size)
{
    arr = new X[size];
    capacity = size;
    front = -1;
    rear = -1;
}

```

```

// Template function to dequeue the front element

```

```

template <class X>
void queue<X>::dequeue()
{
    // check for queue underflow
    if (isEmpty())
    {
        cout << "Underflow";
        exit(EXIT_FAILURE);
    }

    front++;
    cout << "Removing " << arr[front] << endl;
}

// Template function to add an item to the queue
template <class X>
void queue<X>::enqueue(X item)
{
    // check for queue overflow
    if (isFull())
    {
        cout << "Overflow";
        exit(EXIT_FAILURE);
    }

    cout << "Inserting " << item << endl;

    rear++;
    arr[rear] = item;
}

```

```
// Template function to check if the queue is empty or not
```

```
template <class X>
```

```
bool queue<X>::isEmpty()
```

```
{
```

```
    if (rear == front)
```

```
        return true;
```

```
    else
```

```
        return false;
```

```
}
```

```
// Template function to check if the queue is full or not
```

```
template <class X>
```

```
bool queue<X>::isFull()
```

```
{
```

```
    if (rear == capacity - 1)
```

```
        return true;
```

```
    else
```

```
        return false;
```

```
}
```

```
int main()
```

```
{
```

```
    // create a queue of capacity 4
```

```
    queue<string> q(4);
```

```
    q.enqueue("p");
```

```
    q.enqueue("q");
```

```
    q.enqueue("r");
```

```
    q.dequeue();
```

```
q.enqueue("s");

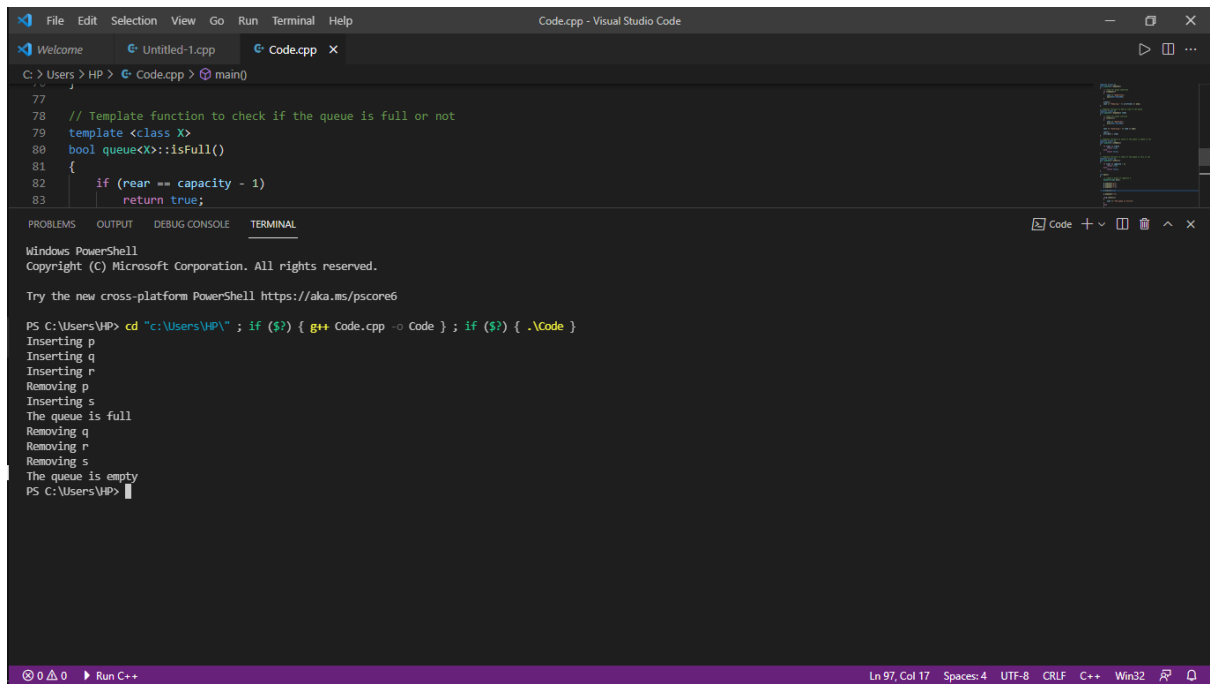
if(q.isFull())
{
    cout << "The queue is full\n";
}
else
{
    cout << "The queue is not full\n";
}

q.dequeue();
q.dequeue();
q.dequeue();

if (q.isEmpty())
{
    cout << "The queue is empty\n";
}
else
{
    cout << "The queue is not empty\n";
}

return 0;
}
```

output



The image shows a Visual Studio Code window with a C++ file named `Code.cpp` open. The code in the editor is as follows:

```
77  
78 // Template function to check if the queue is full or not  
79 template <class X>  
80 bool queue<X>::isFull()  
81 {  
82     if (rear == capacity - 1)  
83         return true;
```

The terminal window at the bottom shows the output of a C++ program. The prompt is `PS C:\Users\HP>`. The first command executed is `cd "C:\Users\HP\" ; if ($?) { g++ Code.cpp -o Code } ; if ($?) { .\Code }`. The output of the program is:

```
Inserting p  
Inserting q  
Inserting r  
Removing p  
Inserting s  
The queue is full  
Removing q  
Removing r  
Removing s  
The queue is empty  
PS C:\Users\HP>
```

The status bar at the bottom indicates the file is at line 97, column 17, with 4 spaces, UTF-8 encoding, CRLF line endings, C++ language, and Win32 architecture.