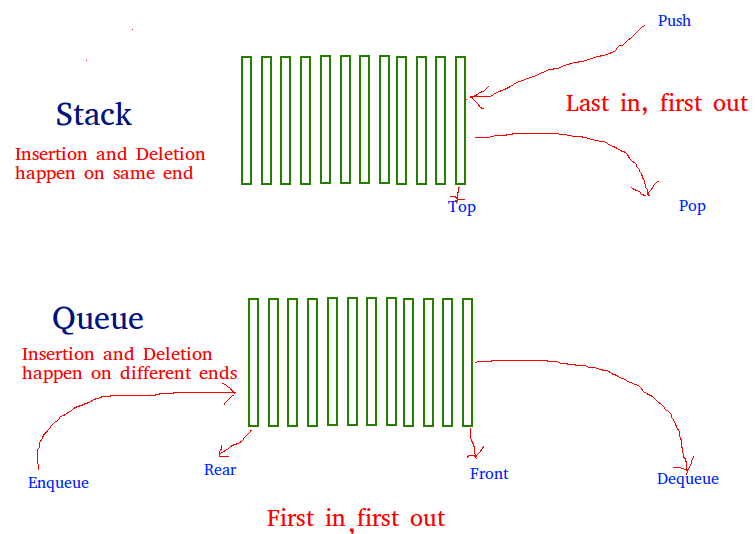
Queue using Stacks

The problem is opposite of [this](https://tutorialspoint.dev/slugresolver/implement-stack-using-queue/) post. We are given a stack data structure with push and pop operations, the task is to implement a queue using instances of stack data structure and operations on them.



A queue can be implemented using two stacks. Let queue to be implemented be q and stacks used to implement q be stack1 and stack2. q can be implemented in two ways:

**Method 1 (By making enQueue operation costly)** This method makes sure that oldest entered element is always at the top of stack 1, so that deQueue operation just pops from stack1. To put the element at top of stack1, stack2 is used.

enQueue(q, x)

1) While stack1 is not empty, push everything from stack1 to stack2.

2) Push x to stack1 (assuming size of stacks is unlimited).

3) Push everything back to stack1.

Here time complexity will be O(n)

deQueue(q)

1) If stack1 is empty then error

2) Pop an item from stack1 and return it

Here time complexity will be O(1)

**C++**

|  |
| --- |
| // CPP program to implement Queue using  // two stacks with costly enQueue()  #include <bits/stdc++.h>  **using** **namespace** std;    **struct** Queue {      stack<**int**> s1, s2;    **void** enQueue(**int** x)      {          // Move all elements from s1 to s2  **while** (!s1.empty()) {              s2.push(s1.top());              s1.pop();          }            // Push item into s1          s1.push(x);            // Push everything back to s1  **while** (!s2.empty()) {              s1.push(s2.top());              s2.pop();          }      }        // Dequeue an item from the queue  **int** deQueue()      {          // if first stack is empty  **if** (s1.empty()) {              cout << "Q is Empty";  **exit**(0);          }            // Return top of s1  **int** x = s1.top();          s1.pop();  **return** x;      }  };    // Driver code  **int** main()  {      Queue q;      q.enQueue(1);      q.enQueue(2);      q.enQueue(3);        cout << q.deQueue() << '  ';      cout << q.deQueue() << '  ';      cout << q.deQueue() << '  ';    **return** 0;  } |

**Java**

|  |
| --- |
| // Java program to implement Queue using  // two stacks with costly enQueue()  **import** java.util.\*;    **class** GFG  {  **static** **class** Queue  {  **static** Stack<Integer> s1 = **new** Stack<Integer>();  **static** Stack<Integer> s2 = **new** Stack<Integer>();    **static** **void** enQueue(**int** x)      {          // Move all elements from s1 to s2  **while** (!s1.isEmpty())          {              s2.push(s1.pop());              //s1.pop();          }            // Push item into s1          s1.push(x);            // Push everything back to s1  **while** (!s2.isEmpty())          {              s1.push(s2.pop());              //s2.pop();          }      }        // Dequeue an item from the queue  **static** **int** deQueue()      {          // if first stack is empty  **if** (s1.isEmpty())          {              System.out.println("Q is Empty");              System.exit(0);          }            // Return top of s1  **int** x = s1.peek();          s1.pop();  **return** x;      }  };    // Driver code  **public** **static** **void** main(String[] args)  {      Queue q = **new** Queue();      q.enQueue(1);      q.enQueue(2);      q.enQueue(3);        System.out.println(q.deQueue());      System.out.println(q.deQueue());      System.out.println(q.deQueue());  }  } |

Python3

# Python3 program to implement Queue using  
# two stacks with costly enQueue()

class Queue:  
def \_\_init\_\_(self):  
self.s1 = []  
self.s2 = []

def enQueue(self, x):

# Move all elements from s1 to s2  
while len(self.s1) != 0:  
self.s2.append(self.s1[-1])  
self.s1.pop()

# Push item into self.s1  
self.s1.append(x)

# Push everything back to s1  
while len(self.s2) != 0:  
self.s1.append(self.s2[-1])  
self.s2.pop()

# Dequeue an item from the queue  
def deQueue(self):

# if first stack is empty  
if len(self.s1) == 0:  
print(“Q is Empty”)

# Return top of self.s1  
x = self.s1[-1]  
self.s1.pop()  
return x

# Driver code  
if \_\_name\_\_ == ‘\_\_main\_\_’:  
q = Queue()  
q.enQueue(1)  
q.enQueue(2)  
q.enQueue(3)

print(q.deQueue())  
print(q.deQueue())  
print(q.deQueue())

**C#**

|  |
| --- |
| // C# program to implement Queue using  // two stacks with costly enQueue()  **using** System;  **using** System.Collections;    **class** GFG  {    **public** **class** Queue  {  **public** Stack s1 = **new** Stack();  **public** Stack s2 = **new** Stack();    **public** **void** enQueue(**int** x)      {          // Move all elements from s1 to s2  **while** (s1.Count > 0)          {              s2.Push(s1.Pop());              //s1.Pop();          }            // Push item into s1          s1.Push(x);            // Push everything back to s1  **while** (s2.Count > 0)          {              s1.Push(s2.Pop());              //s2.Pop();          }      }        // Dequeue an item from the queue  **public** **int** deQueue()      {          // if first stack is empty  **if** (s1.Count == 0)          {              Console.WriteLine("Q is Empty");            }            // Return top of s1  **int** x = (**int**)s1.Peek();          s1.Pop();  **return** x;      }  };    // Driver code  **public** **static** **void** Main()  {      Queue q = **new** Queue();      q.enQueue(1);      q.enQueue(2);      q.enQueue(3);        Console.Write(q.deQueue()+" ");      Console.Write(q.deQueue()+" ");      Console.Write(q.deQueue());  }  } |

**Output:**

1

2

3

**Method 2 (By making deQueue operation costly)**In this method, in en-queue operation, the new element is entered at the top of stack1. In de-queue operation, if stack2 is empty then all the elements are moved to stack2 and finally top of stack2 is returned.

enQueue(q, x)

1) Push x to stack1 (assuming size of stacks is unlimited).

Here time complexity will be O(1)

deQueue(q)

1) If both stacks are empty then error.

2) If stack2 is empty

While stack1 is not empty, push everything from stack1 to stack2.

3) Pop the element from stack2 and return it.

Here time complexity will be O(n)

Method 2 is definitely better than method 1.  
Method 1 moves all the elements twice in enQueue operation, while method 2 (in deQueue operation) moves the elements once and moves elements only if stack2 empty.  
Implementation of method 2:

**C++**

|  |
| --- |
| // CPP program to implement Queue using  // two stacks with costly deQueue()  #include <bits/stdc++.h>  **using** **namespace** std;    **struct** Queue {      stack<**int**> s1, s2;        // Enqueue an item to the queue  **void** enQueue(**int** x)      {          // Push item into the first stack          s1.push(x);      }        // Dequeue an item from the queue  **int** deQueue()      {          // if both stacks are empty  **if** (s1.empty() && s2.empty()) {              cout << "Q is empty";  **exit**(0);          }            // if s2 is empty, move          // elements from s1  **if** (s2.empty()) {  **while** (!s1.empty()) {                  s2.push(s1.top());                  s1.pop();              }          }            // return the top item from s2  **int** x = s2.top();          s2.pop();  **return** x;      }  };    // Driver code  **int** main()  {      Queue q;      q.enQueue(1);      q.enQueue(2);      q.enQueue(3);        cout << q.deQueue() << '  ';      cout << q.deQueue() << '  ';      cout << q.deQueue() << '  ';    **return** 0;  } |

**C**

|  |
| --- |
| /\* C Program to implement a queue using two stacks \*/  #include <stdio.h>  #include <stdlib.h>    /\* structure of a stack node \*/  **struct** sNode {  **int** data;  **struct** sNode\* next;  };    /\* Function to push an item to stack\*/  **void** push(**struct** sNode\*\* top\_ref, **int** new\_data);    /\* Function to pop an item from stack\*/  **int** pop(**struct** sNode\*\* top\_ref);    /\* structure of queue having two stacks \*/  **struct** queue {  **struct** sNode\* stack1;  **struct** sNode\* stack2;  };    /\* Function to enqueue an item to queue \*/  **void** enQueue(**struct** queue\* q, **int** x)  {      push(&q->stack1, x);  }    /\* Function to deQueue an item from queue \*/  **int** deQueue(**struct** queue\* q)  {  **int** x;        /\* If both stacks are empty then error \*/  **if** (q->stack1 == NULL && q->stack2 == NULL) {  **printf**("Q is empty");  **getchar**();  **exit**(0);      }        /\* Move elements from stack1 to stack 2 only if         stack2 is empty \*/  **if** (q->stack2 == NULL) {  **while** (q->stack1 != NULL) {              x = pop(&q->stack1);              push(&q->stack2, x);          }      }        x = pop(&q->stack2);  **return** x;  }    /\* Function to push an item to stack\*/  **void** push(**struct** sNode\*\* top\_ref, **int** new\_data)  {      /\* allocate node \*/  **struct** sNode\* new\_node = (**struct** sNode\*)**malloc**(**sizeof**(**struct** sNode));  **if** (new\_node == NULL) {  **printf**("Stack overflow  ");  **getchar**();  **exit**(0);      }        /\* put in the data \*/      new\_node->data = new\_data;        /\* link the old list off the new node \*/      new\_node->next = (\*top\_ref);        /\* move the head to point to the new node \*/      (\*top\_ref) = new\_node;  }    /\* Function to pop an item from stack\*/  **int** pop(**struct** sNode\*\* top\_ref)  {  **int** res;  **struct** sNode\* top;        /\*If stack is empty then error \*/  **if** (\*top\_ref == NULL) {  **printf**("Stack underflow  ");  **getchar**();  **exit**(0);      }  **else** {          top = \*top\_ref;          res = top->data;          \*top\_ref = top->next;  **free**(top);  **return** res;      }  }    /\* Driver function to test anove functions \*/  **int** main()  {      /\* Create a queue with items 1 2 3\*/  **struct** queue\* q = (**struct** queue\*)**malloc**(**sizeof**(**struct** queue));      q->stack1 = NULL;      q->stack2 = NULL;      enQueue(q, 1);      enQueue(q, 2);      enQueue(q, 3);        /\* Dequeue items \*/  **printf**("%d ", deQueue(q));  **printf**("%d ", deQueue(q));  **printf**("%d ", deQueue(q));    **return** 0;  } |

**Java**

|  |
| --- |
| /\* Java Program to implement a queue using two stacks \*/  // Note that Stack class is used for Stack implementation    **import** java.util.Stack;    **public** **class** GFG {      /\* class of queue having two stacks \*/  **static** **class** Queue {          Stack<Integer> stack1;          Stack<Integer> stack2;      }        /\* Function to push an item to stack\*/  **static** **void** push(Stack<Integer> top\_ref, **int** new\_data)      {          // Push the data onto the stack          top\_ref.push(new\_data);      }        /\* Function to pop an item from stack\*/  **static** **int** pop(Stack<Integer> top\_ref)      {          /\*If stack is empty then error \*/  **if** (top\_ref.isEmpty()) {              System.out.println("Stack Underflow");              System.exit(0);          }            // pop the data from the stack  **return** top\_ref.pop();      }        // Function to enqueue an item to the queue  **static** **void** enQueue(Queue q, **int** x)      {          push(q.stack1, x);      }        /\* Function to deQueue an item from queue \*/  **static** **int** deQueue(Queue q)      {  **int** x;            /\* If both stacks are empty then error \*/  **if** (q.stack1.isEmpty() && q.stack2.isEmpty()) {              System.out.println("Q is empty");              System.exit(0);          }            /\* Move elements from stack1 to stack 2 only if          stack2 is empty \*/  **if** (q.stack2.isEmpty()) {  **while** (!q.stack1.isEmpty()) {                  x = pop(q.stack1);                  push(q.stack2, x);              }          }          x = pop(q.stack2);  **return** x;      }        /\* Driver function to test above functions \*/  **public** **static** **void** main(String args[])      {          /\* Create a queue with items 1 2 3\*/          Queue q = **new** Queue();          q.stack1 = **new** Stack<>();          q.stack2 = **new** Stack<>();          enQueue(q, 1);          enQueue(q, 2);          enQueue(q, 3);            /\* Dequeue items \*/          System.out.print(deQueue(q) + " ");          System.out.print(deQueue(q) + " ");          System.out.println(deQueue(q) + " ");      }  } |

**Output:**

1 2 3

**Queue can also be implemented using one user stack and one Function Call Stack.**Below is modified Method 2 where recursion (or Function Call Stack) is used to implement queue using only one user defined stack.

*enQueue(x)*

1) Push *x* to *stack1*.

*deQueue:*

1) If *stack1* is empty then error.

2) If *stack1* has only one element then return it.

3) Recursively pop everything from the stack1, store the popped item

in a variable *res*, push the *res* back to stack1 and return *res*

The step 3 makes sure that the last popped item is always returned and since the recursion stops when there is only one item in *stack1* (step 2), we get the last element of *stack1*in deQueue() and all other items are pushed back in step

**3. Implementation of method 2 using Function Call Stack:**

**C++**

|  |
| --- |
| // CPP program to implement Queue using  // one stack and recursive call stack.  #include <bits/stdc++.h>  **using** **namespace** std;    **struct** Queue {      stack<**int**> s;        // Enqueue an item to the queue  **void** enQueue(**int** x)      {          s.push(x);      }        // Dequeue an item from the queue  **int** deQueue()      {  **if** (s.empty()) {              cout << "Q is empty";  **exit**(0);          }            // pop an item from the stack  **int** x = s.top();          s.pop();            // if stack becomes empty, return          // the popped item  **if** (s.empty())  **return** x;            // recursive call  **int** item = deQueue();            // push popped item back to the stack          s.push(x);            // return the result of deQueue() call  **return** item;      }  };    // Driver code  **int** main()  {      Queue q;      q.enQueue(1);      q.enQueue(2);      q.enQueue(3);        cout << q.deQueue() << '  ';      cout << q.deQueue() << '  ';      cout << q.deQueue() << '  ';    **return** 0;  } |

**C**

|  |
| --- |
| /\* Program to implement a queue using one user defined stack  and one Function Call Stack \*/  #include <stdio.h>  #include <stdlib.h>    /\* structure of a stack node \*/  **struct** sNode {  **int** data;  **struct** sNode\* next;  };    /\* structure of queue having two stacks \*/  **struct** queue {  **struct** sNode\* stack1;  };    /\* Function to push an item to stack\*/  **void** push(**struct** sNode\*\* top\_ref, **int** new\_data);    /\* Function to pop an item from stack\*/  **int** pop(**struct** sNode\*\* top\_ref);    /\* Function to enqueue an item to queue \*/  **void** enQueue(**struct** queue\* q, **int** x)  {      push(&q->stack1, x);  }    /\* Function to deQueue an item from queue \*/  **int** deQueue(**struct** queue\* q)  {  **int** x, res;        /\* If both stacks are empty then error \*/  **if** (q->stack1 == NULL) {  **printf**("Q is empty");  **getchar**();  **exit**(0);      }  **else** **if** (q->stack1->next == NULL) {  **return** pop(&q->stack1);      }  **else** {          /\* pop an item from the stack1 \*/          x = pop(&q->stack1);            /\* store the last deQueued item \*/          res = deQueue(q);            /\* push everything back to stack1 \*/          push(&q->stack1, x);  **return** res;      }  }    /\* Function to push an item to stack\*/  **void** push(**struct** sNode\*\* top\_ref, **int** new\_data)  {      /\* allocate node \*/  **struct** sNode\* new\_node = (**struct** sNode\*)**malloc**(**sizeof**(**struct** sNode));    **if** (new\_node == NULL) {  **printf**("Stack overflow  ");  **getchar**();  **exit**(0);      }        /\* put in the data \*/      new\_node->data = new\_data;        /\* link the old list off the new node \*/      new\_node->next = (\*top\_ref);        /\* move the head to point to the new node \*/      (\*top\_ref) = new\_node;  }    /\* Function to pop an item from stack\*/  **int** pop(**struct** sNode\*\* top\_ref)  {  **int** res;  **struct** sNode\* top;        /\*If stack is empty then error \*/  **if** (\*top\_ref == NULL) {  **printf**("Stack underflow  ");  **getchar**();  **exit**(0);      }  **else** {          top = \*top\_ref;          res = top->data;          \*top\_ref = top->next;  **free**(top);  **return** res;      }  }    /\* Driver function to test above functions \*/  **int** main()  {      /\* Create a queue with items 1 2 3\*/  **struct** queue\* q = (**struct** queue\*)**malloc**(**sizeof**(**struct** queue));      q->stack1 = NULL;        enQueue(q, 1);      enQueue(q, 2);      enQueue(q, 3);        /\* Dequeue items \*/  **printf**("%d ", deQueue(q));  **printf**("%d ", deQueue(q));  **printf**("%d ", deQueue(q));    **return** 0;  } |

**Java**

|  |
| --- |
| // Java Program to implement a queue using one stack    **import** java.util.Stack;    **public** **class** QOneStack {      // class of queue having two stacks  **static** **class** Queue {          Stack<Integer> stack1;      }        /\* Function to push an item to stack\*/  **static** **void** push(Stack<Integer> top\_ref, **int** new\_data)      {          /\* put in the data \*/          top\_ref.push(new\_data);      }        /\* Function to pop an item from stack\*/  **static** **int** pop(Stack<Integer> top\_ref)      {          /\*If stack is empty then error \*/  **if** (top\_ref == **null**) {              System.out.println("Stack Underflow");              System.exit(0);          }          // return element from stack  **return** top\_ref.pop();      }        /\* Function to enqueue an item to queue \*/  **static** **void** enQueue(Queue q, **int** x)      {          push(q.stack1, x);      }        /\* Function to deQueue an item from queue \*/  **static** **int** deQueue(Queue q)      {  **int** x, res = 0;          /\* If the stacks is empty then error \*/  **if** (q.stack1.isEmpty()) {              System.out.println("Q is Empty");              System.exit(0);          }          // Check if it is a last element of stack  **else** **if** (q.stack1.size() == 1) {  **return** pop(q.stack1);          }  **else** {                /\* pop an item from the stack1 \*/              x = pop(q.stack1);                /\* store the last deQueued item \*/              res = deQueue(q);                /\* push everything back to stack1 \*/              push(q.stack1, x);  **return** res;          }  **return** 0;      }        /\* Driver function to test above functions \*/  **public** **static** **void** main(String[] args)      {          /\* Create a queue with items 1 2 3\*/          Queue q = **new** Queue();          q.stack1 = **new** Stack<>();            enQueue(q, 1);          enQueue(q, 2);          enQueue(q, 3);            /\* Dequeue items \*/          System.out.print(deQueue(q) + " ");          System.out.print(deQueue(q) + " ");          System.out.print(deQueue(q) + " ");      }  } |

**Output:**

1 2 3