STACKS AND QUEUE

C++ And Data Structure Mini Project

By

—

Pratiksha Phadtare - 22070123084
Sanika Desai- 22070123098
Sanskriti Jha - 22070123101

—

Dr. Snehal Bhosale

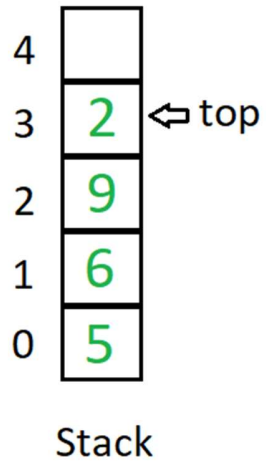
AIM

Write a C++ program to implement stack and queue for the following situation.
Food items, Stationery items and Sports items are arriving continuously in the supermarket randomly but in a queue. Sort and store the items in the three different stacks and update and display the individual count. Also, display the contents of individual stacks.

THEORY

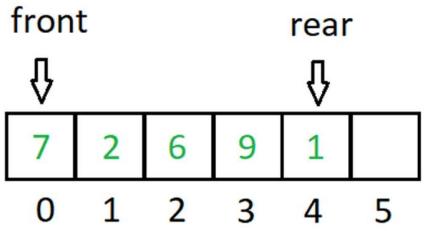
C++ is one of the common programming languages that is mainly used in the development of high-performance applications, Operating Systems, and games. C++ is a powerful and efficient language that provides a wide range of Data Structures and Algorithms for complex data processing tasks. C++ Data Structures and Algorithms (DSA) is a part of Computer Science in which there is the study of different Algorithms and Data Structures using the C++ programming language. Data Structures are the basic block of Computer Science Technology. Data Structures are used to store the data in such a way that we can access them in a very efficient way. There are a lot of Data Structures used in the C++ Programming language Arrays, Stacks, Queues, Linked-lists, Trees, and Graphs.

Stack: A stack is a linear data structure in which elements can be inserted and deleted only from one side of the list, called the top. A stack follows the LIFO (Last In First Out) principle, i.e., the element inserted at the last is the first element to come out. The insertion of an element into the stack is called push operation, and the deletion of an element from the stack is called pop operation. In stack, we always keep track of the last element present in the list with a pointer called top.



Stack

Queue is a linear data structure in which elements can be inserted only from one side of the list called rear, and the elements can be deleted only from the other side called the front**. The queue data structure follows the FIFO (First In First Out) principle, i.e. the element inserted at first in the list, is the first element to be removed from the list. The insertion of an element in a queue is called an enqueue operation and the deletion of an element is called a dequeue operation. In queue, we always maintain two pointers, one pointing to the element which was inserted at the first and still present in the list with the** front **pointer and the second pointer pointing to the element inserted at the last with the rear pointer.**



Queue

ALGORITHM

1.START
2.Intializing stack and queue
3.While loop for selecting type of item or display.
4.Input item in its respective stack.
5.Displaying error message for wrong choice.
6.Displaying the stack and the items stored in it.
7.END

Difference between Stack and Queue Data Structures are as follows:

Stacks	Queues
A stack is a data structure that stores a collection of elements, with operations to push (add) and pop (remove) elements from the top of the stack.	A queue is a data structure that stores a collection of elements, with operations to enqueue (add) elements at the back of the queue, and dequeue (remove) elements from the front of the queue.
Stacks are based on the LIFO principle, i.e., the element inserted at the last, is the first element to come out of the list.	Queues are based on the FIFO principle, i.e., the element inserted at the first, is the first element to come out of the list.
Stacks are often used for tasks that require backtracking, such as parsing expressions or implementing undo functionality.	Queues are often used for tasks that involve processing elements in a specific order, such as handling requests or scheduling tasks.
Insertion and deletion in stacks takes place only from one end of the list called the top.	Insertion and deletion in queues takes place from the opposite ends of the list. The insertion takes place at the rear of the list and the deletion takes place from the front of the list.
Insert operation is called push operation.	Insert operation is called enqueue operation.
Stacks are implemented using an array or linked list data structure.	Queues are implemented using an array or linked list data structure.

Stacks	Queues
Delete operation is called pop operation.	Delete operation is called dequeue operation.
In stacks we maintain only one pointer to access the list, called the top, which always points to the last element present in the list.	In queues we maintain two pointers to access the list. The front pointer always points to the first element inserted in the list and is still present, and the rear pointer always points to the last inserted element.
Stack is used in solving problems works on recursion.	Queue is used in solving problems having sequential processing.
Stacks are often used for recursive algorithms or for maintaining a history of function calls.	Queues are often used in multithreaded applications, where tasks are added to a queue and executed by a pool of worker threads.
Stack does not have any types.	Queue is of three types – 1. Circular Queue 2. Priority queue 3. double-ended queue.
Can be considered as a vertical collection visual.	Can be considered as a horizontal collection visual.
Examples of stack-based languages include PostScript and Forth.	Examples of queue-based algorithms include Breadth-First Search (BFS) and printing a binary tree level-by-level.

CODE

```
#include <iostream>
#include <queue>
#include <stack>
#include <string>
using namespace std;

int main() {
//Intializing stack and queue
   queue<string> itemsQueue;
   stack<string> foodStack, stationeryStack, sportsStack;


//While loop for selecting type of item or display
   while (true)
{
       int choice, situation;
       string item;

       cout << "Enter your choice : "<<endl;
       cout<<"Press 1 for Food \nPress 2 for Stationery \nPress 3 for Sports \nPress 4
to display your itemized list\n";
       cout<<"No space allowed in item names, use underscore instead"<<endl;
       cin >> situation;
       cout<<"\n";

       if (situation == 4)
```

```
{
            break;
        }

        cout << "Enter the item: ";
        cin >> item;
        cout<<"\n";
        itemsQueue.push(item); //inputing item in its respective stack
if (situation == 1)
{
            foodStack.push(item);
        }
else if (situation == 2)
{
            stationeryStack.push(item);
        }
 else if (situation == 3)
{
            sportsStack.push(item);
        }
        else
        {
         //Displaying error message for wrong choice
         cout<<"Error in choice\n";
}
    }

//Displaying the stack and the items stored in it
    cout << "Items in Food stack: " << foodStack.size() << endl;
    while (!foodStack.empty())
{
        cout << foodStack.top() << endl;
        foodStack.pop();

    }
```

```
   cout << "Items in Stationery stack: " << stationeryStack.size() << endl;
   while (!stationeryStack.empty())
{
      cout << stationeryStack.top() << endl;
      stationeryStack.pop();

   }

   cout << "Items in Sports stack: " << sportsStack.size() << endl;
   while (!sportsStack.empty())
{
      cout << sportsStack.top() << endl;
      sportsStack.pop();

   }

   return 0;
}
```

OUTPUT

```
Enter your choice :
Press 1 for Food
Press 2 for Stationery
Press 3 for Sports
Press 4 to display your itemized list
No space allowed in item names, use underscore instead
1

Enter the item: Apple

Enter your choice :
Press 1 for Food
Press 2 for Stationery
Press 3 for Sports
Press 4 to display your itemized list
No space allowed in item names, use underscore instead
1

Enter the item: Orange

Enter your choice :
Press 1 for Food
Press 2 for Stationery
Press 3 for Sports
Press 4 to display your itemized list
No space allowed in item names, use underscore instead
1
```

```
Enter your choice :
Press 1 for Food
Press 2 for Stationery
Press 3 for Sports
Press 4 to display your itemized list
No space allowed in item names, use underscore instead
1

Enter the item: Apple

Enter your choice :
Press 1 for Food
Press 2 for Stationery
Press 3 for Sports
Press 4 to display your itemized list
No space allowed in item names, use underscore instead
1

Enter the item: Orange

Enter your choice :
Press 1 for Food
Press 2 for Stationery
Press 3 for Sports
Press 4 to display your itemized list
No space allowed in item names, use underscore instead
1

Enter the item: Bread

Enter your choice :
Press 1 for Food
Press 2 for Stationery
Press 3 for Sports
Press 4 to display your itemized list
No space allowed in item names, use underscore instead
1

Enter the item: Milk
Enter your choice :
```

```
Press 1 for Food
Press 2 for Stationery
Press 3 for Sports
Press 4 to display your itemized list
No space allowed in item names, use underscore instead
1

Enter the item: Oreo

Enter your choice :
Press 1 for Food
Press 2 for Stationery
Press 3 for Sports
Press 4 to display your itemized list
No space allowed in item names, use underscore instead
2

Enter the item: Pen

Enter your choice :
Press 1 for Food
Press 2 for Stationery
Press 3 for Sports
Press 4 to display your itemized list
No space allowed in item names, use underscore instead
2

Enter the item: Notebook

Enter your choice :
Press 1 for Food
Press 4 to display your itemized list
No space allowed in item names, use underscore instead
3

Enter the item: Tennis ball

Enter your choice :
Press 1 for Food
```

Press 2 for Stationery
Press 3 for Sports

Press 4 to display your itemized list
No space allowed in item names, use underscore instead
3

Enter the item: Bat

Enter your choice :
Press 1 for Food
Press 2 for Stationery
Press 3 for Sports
Press 4 to display your itemized list
No space allowed in item names, use underscore instead
4

Items in Food stack: 5
Oreo
Milk
Bread
Orange
Apple
Items in Stationery stack: 3
Eraser
Notebook
Pen
Items in Sports stack: 3
Bat
Tennis ball
Basketball

CONCLUSION

Here we have written a C++ program to implement stack and queue for the following situation.
Food items, Stationery items and Sports items are arriving continuously in the supermarket randomly but in a queue. Sort and store the items in the three different stacks and update and display the individual count. Also, display the contents of individual stacks.