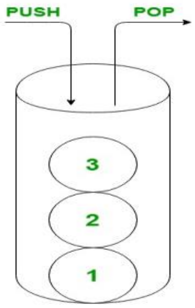
Stack and Queue in C#

A Stack represents a last-in, first-out collection of objects. It is used when you need last-in, first-out access to items.

Key Operations of a Stack



Push: Adds an item to the top of the stack.

Pop: Removes and returns the item from the top of the stack.

Peek: Returns the item at the top of the stack without removing it.

Count: Returns the number of elements in the stack.

Creating and Using a Stack in C#

```
using System.Collections.generic;

namespace stack

{   internal class Program

  {

    static void Main()

    {

      // Creating a stack of integers

      Stack<int> S1 = new Stack<int>();

      // Push operation

      S1.Push(10);

      S1.Push(20);

      S1.Push(30);

      Console.WriteLine("After pushing \n");

      PrintStack(S1);

      // Peek operation

      var topItem = S1.Peek();

      Console.WriteLine($"\nTop item after peek: {topItem}"); // 30

       // Pop operation

      var removedItem = S1.Pop();

      Console.WriteLine($"\nPopped item: {removedItem}");

      Console.WriteLine("\nStack after pop:\n");

      PrintStack(S1); //20,10

       // Count operation

      Console.WriteLine($"\nNumber of items in stack: {S1.Count} \n"); //2

      Console.WriteLine(S1.Contains("30"));
```

```
        ////////////////////////////////////////////////////////
       Console.WriteLine("\nPrinting Stack1 current \n");
        PrintStack(S1);

        Stack S2 = S1; // shallow copy
        Console.WriteLine("\nPrinting Stack2\n");
        PrintStack(S2);


        Stack S3 = (Stack)S1.Clone(); // deep copy
        Console.WriteLine("\nPrinting Stack3 \n");
        PrintStack(S3);


        S1.Pop();
        Console.WriteLine("\nPrinting Stack2 after pop stack 1\n");
        PrintStack(S2);
        Console.WriteLine("\nPrinting Stack3 after pop stack 1 \n");
        PrintStack(S3);
    }
    static void PrintStack(Stack stack)
    {
        foreach (var number in stack)
        { Console.WriteLine(number); }
    }
}}
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

A **Queue** represents a **first-in, first-out (FIFO)** collection of objects. It is used when you need to process elements in the exact order they were added.

Key Operations of a Queue

- **Enqueue**: Adds an item to the end of the queue.
- **Dequeue**: Removes and returns the object at the beginning of the queue.
- **Peek**: Returns the object at the beginning of the queue without removing it.
- **Count**: Returns the number of elements in the queue.

Creating and Using a Queue in C#

```
using System.Collections;

namespace queue_example

{

  class Program

  {

    static void Main()

    {

      // Creating a queue of objects

      Queue<int> Q1 = new Queue<int>();

      // Enqueue operation

      Q1.Enqueue(100);

      Q1.Enqueue(200);

      Console.WriteLine("After enqueueing:\n");

      PrintQueue(Q1);

      // Peek operation

      var frontItem = Q1.Peek();


      Console.WriteLine($"\nFront item after peek: {frontItem}"); // 100

      // Dequeue operation

      var removedItem = Q1.Dequeue();


      Console.WriteLine($"\nDequeued item: {removedItem}");

      Console.WriteLine("\nQueue after dequeue:\n");

      PrintQueue(Q1);
```

```
            // Count operation
            Console.WriteLine($"\nNumber of items in queue: {Q1.Count} \n");
            Console.WriteLine(Q1.Contains("300"));

            ////////////////////////////////////////////////////////
            Console.WriteLine("\nPrinting Queue1 current:\n");
            PrintQueue(Q1);

            Queue Q2 = Q1; // shallow copy
            Console.WriteLine("\nPrinting Queue2:\n");
            PrintQueue(Q2);

            Queue Q3 = (Queue)Q1.Clone(); // deep copy
            Console.WriteLine("\nPrinting Queue3:\n");
            PrintQueue(Q3);

            Q1.Dequeue();
            Console.WriteLine("\nPrinting Queue2 after dequeue from Queue1:\n");
            PrintQueue(Q2);

            Console.WriteLine("\nPrinting Queue3 after dequeue from Queue1:\n");
            PrintQueue(Q3);
        }
        static void PrintQueue(Queue queue)
        {
            foreach (var item in queue)
            {     Console.WriteLine(item); }
        }
    }
}
```

Summary of Key Real-Life Uses:

- **Undo/Redo** in applications.
- **Function Call Stack** for recursion and nested functions.
- **Expression Evaluation** for mathematical calculations.
- **Syntax Parsing** (e.g., matching parentheses).
- **Browser Navigation** (Back/Forward history).
- **Memory Management** (local variables stored in the stack).
- **String/Array Reversal**.

Summary of Key Real-Life Queue Use Cases:

- **Task Scheduling**: CPU scheduling tasks in an operating system.
- **Networking**: Managing data packets in routers.
- **Call Center**: Handling customer service requests in the order they arrive.
- **Message Queues**: Decoupling distributed systems.
- **Simulating Waiting Lines**: Optimizing customer flow in service environments.
- **Real-Time Data Streaming**: Processing events in real-time systems.