Implement two stacks in an array

Create a data structure *twoStacks*that represents two stacks. Implementation of *twoStacks*should use only one array, i.e., both stacks should use the same array for storing elements. Following functions must be supported by *twoStacks*.

push1(int x) –> pushes x to first stack  
push2(int x) –> pushes x to second stack

pop1() –> pops an element from first stack and return the popped element  
pop2() –> pops an element from second stack and return the popped element

Implementation of *twoStack*should be space efficient.

**Method 1 (Divide the space in two halves)**  
A simple way to implement two stacks is to divide the array in two halves and assign the half half space to two stacks, i.e., use arr[0] to arr[n/2] for stack1, and arr[(n/2) + 1] to arr[n-1] for stack2 where arr[] is the array to be used to implement two stacks and size of array be n.

The problem with this method is inefficient use of array space. A stack push operation may result in stack overflow even if there is space available in arr[]. For example, say the array size is 6 and we push 3 elements to stack1 and do not push anything to second stack2. When we push 4th element to stack1, there will be overflow even if we have space for 3 more elements in array.

**Method 2 (A space efficient implementation)**  
This method efficiently utilizes the available space. It doesn’t cause an overflow if there is space available in arr[]. The idea is to start two stacks from two extreme corners of arr[]. stack1 starts from the leftmost element, the first element in stack1 is pushed at index 0. The stack2 starts from the rightmost corner, the first element in stack2 is pushed at index (n-1). Both stacks grow (or shrink) in opposite direction. To check for overflow, all we need to check is for space between top elements of both stacks. This check is highlighted in the below code.

**C++**

|  |
| --- |
| #include<iostream>  #include<stdlib.h>    **using** **namespace** std;    **class** twoStacks  {  **int** \*arr;  **int** size;  **int** top1, top2;  **public**:     twoStacks(**int** n)  // constructor     {         size = n;         arr = **new** **int**[n];         top1 = -1;         top2 = size;     }       // Method to push an element x to stack1  **void** push1(**int** x)     {         // There is at least one empty space for new element  **if** (top1 < top2 - 1)         {             top1++;             arr[top1] = x;         }  **else**         {             cout << "Stack Overflow";  **exit**(1);         }     }       // Method to push an element x to stack2  **void** push2(**int** x)     {         // There is at least one empty space for new element  **if** (top1 < top2 - 1)         {             top2--;             arr[top2] = x;         }  **else**         {             cout << "Stack Overflow";  **exit**(1);         }     }       // Method to pop an element from first stack  **int** pop1()     {  **if** (top1 >= 0 )         {  **int** x = arr[top1];            top1--;  **return** x;         }  **else**         {             cout << "Stack UnderFlow";  **exit**(1);         }     }       // Method to pop an element from second stack  **int** pop2()     {  **if** (top2 < size)         {  **int** x = arr[top2];            top2++;  **return** x;         }  **else**         {             cout << "Stack UnderFlow";  **exit**(1);         }     }  };      /\* Driver program to test twStacks class \*/  **int** main()  {      twoStacks ts(5);      ts.push1(5);      ts.push2(10);      ts.push2(15);      ts.push1(11);      ts.push2(7);      cout << "Popped element from stack1 is " << ts.pop1();      ts.push2(40);      cout << "  Popped element from stack2 is " << ts.pop2();  **return** 0;  } |

**Java**

|  |
| --- |
| // Java program to implement two stacks in a  // single array  **class** TwoStacks  {  **int** size;  **int** top1, top2;  **int** arr[];        // Constructor      TwoStacks(**int** n)      {          arr = **new** **int**[n];          size = n;          top1 = -1;          top2 = size;      }        // Method to push an element x to stack1  **void** push1(**int** x)      {          // There is at least one empty space for          // new element  **if** (top1 < top2 - 1)          {              top1++;              arr[top1] = x;          }  **else**          {              System.out.println("Stack Overflow");              System.exit(1);          }      }        // Method to push an element x to stack2  **void** push2(**int** x)      {          // There is at least one empty space for          // new element  **if** (top1 < top2 -1)          {              top2--;              arr[top2] = x;          }  **else**          {              System.out.println("Stack Overflow");              System.exit(1);          }      }        // Method to pop an element from first stack  **int** pop1()      {  **if** (top1 >= 0)          {  **int** x = arr[top1];              top1--;  **return** x;          }  **else**          {              System.out.println("Stack Underflow");              System.exit(1);          }  **return** 0;      }        // Method to pop an element from second stack  **int** pop2()      {  **if**(top2 < size)          {  **int** x =arr[top2];              top2++;  **return** x;          }  **else**          {              System.out.println("Stack Underflow");              System.exit(1);            }  **return** 0;      }        // Driver program to test twoStack class  **public** **static** **void** main(String args[])      {          TwoStacks ts = **new** TwoStacks(5);          ts.push1(5);          ts.push2(10);          ts.push2(15);          ts.push1(11);          ts.push2(7);          System.out.println("Popped element from" +                             " stack1 is " + ts.pop1());          ts.push2(40);          System.out.println("Popped element from" +                           " stack2 is " + ts.pop2());      }  } |

**Python**

|  |
| --- |
| # Python Script to Implement two stacks in a list  **class** twoStacks:    **def** \_\_init\_\_(self, n):     #constructor          self.size **=** n          self.arr **=** [None] **\*** n          self.top1 **=** **-**1          self.top2 **=** self.size        # Method to push an element x to stack1  **def** push1(self, x):            # There is at least one empty space for new element  **if** self.top1 < self.top2 **-** 1 :              self.top1 **=** self.top1 **+** 1              self.arr[self.top1] **=** x    **else**:              print("Stack Overflow ")              exit(1)        # Method to push an element x to stack2  **def** push2(self, x):            # There is at least one empty space for new element  **if** self.top1 < self.top2 **-** 1:              self.top2 **=** self.top2 **-** 1              self.arr[self.top2] **=** x    **else** :  **print**("Stack Overflow ")             exit(1)        # Method to pop an element from first stack  **def** pop1(self):  **if** self.top1 >**=** 0:              x **=** self.arr[self.top1]              self.top1 **=** self.top1 **-**1  **return** x  **else**:              print("Stack Underflow ")              exit(1)        # Method to pop an element from second stack  **def** pop2(self):  **if** self.top2 < self.size:              x **=** self.arr[self.top2]              self.top2 **=** self.top2 **+** 1  **return** x  **else**:              print("Stack Underflow ")              exit()    # Driver program to test twoStacks class  ts **=** twoStacks(5)  ts.push1(5)  ts.push2(10)  ts.push2(15)  ts.push1(11)  ts.push2(7)    print("Popped element from stack1 is " **+** str(ts.pop1()))  ts.push2(40)  print("Popped element from stack2 is " **+** str(ts.pop2())) |

**C#**

|  |
| --- |
| // C# program to implement two  // stacks in a single array  **using** System;    **public** **class** TwoStacks  {  **public** **int** size;  **public** **int** top1, top2;  **public** **int**[] arr;    // Constructor  **public** TwoStacks(**int** n)  {      arr = **new** **int**[n];      size = n;      top1 = -1;      top2 = size;  }    // Method to push an element x to stack1  **public** **virtual** **void** push1(**int** x)  {      // There is at least one empty      // space for new element  **if** (top1 < top2 - 1)      {          top1++;          arr[top1] = x;      }  **else**      {          Console.WriteLine("Stack Overflow");          Environment.Exit(1);      }  }    // Method to push an element x to stack2  **public** **virtual** **void** push2(**int** x)  {      // There is at least one empty      // space for new element  **if** (top1 < top2 - 1)      {          top2--;          arr[top2] = x;      }  **else**      {          Console.WriteLine("Stack Overflow");          Environment.Exit(1);      }  }    // Method to pop an element  // from first stack  **public** **virtual** **int** pop1()  {  **if** (top1 >= 0)      {  **int** x = arr[top1];          top1--;  **return** x;      }  **else**      {          Console.WriteLine("Stack Underflow");          Environment.Exit(1);      }  **return** 0;  }    // Method to pop an element  // from second stack  **public** **virtual** **int** pop2()  {  **if** (top2 < size)      {  **int** x = arr[top2];          top2++;  **return** x;      }  **else**      {          Console.WriteLine("Stack Underflow");          Environment.Exit(1);        }  **return** 0;  }    // Driver Code  **public** **static** **void** Main(**string**[] args)  {      TwoStacks ts = **new** TwoStacks(5);      ts.push1(5);      ts.push2(10);      ts.push2(15);      ts.push1(11);      ts.push2(7);      Console.WriteLine("Popped element from" +                        " stack1 is " + ts.pop1());      ts.push2(40);      Console.WriteLine("Popped element from" +                        " stack2 is " + ts.pop2());  }  } |