**BLOODY BUGS**

**// Bubble sort in C++**

**#include <iostream>**

**using namespace std;**

**// perform bubble sort**

**void bubbleSort(int array[], int size) {**

**// loop to access each array element**

**for (int step = 0; step < size; ++step) {**

**// loop to compare array elements**

**for (int i = 0; i < size - step; ++i) {**

**// compare two adjacent elements**

**// change > to < to sort in descending order**

**if (array[i] > array[i + 1]) {**

**// swapping elements if elements**

**// are not in the intended order**

**int temp = array[i];**

**array[i] = array[i + 1];**

**array[i + 1] = temp;**

**}**

**}**

**}**

**}**

**// print array**

**void printArray(int array[], int size) {**

**for (int i = 0; i < size; ++i) {**

**cout << " " << array[i];**

**}**

**cout << "\n";**

**}**

**int main() {**

**int data[] = {-2, 45, 0, 11, -9};**

**// find array's length**

**int size = sizeof(data) / sizeof(data[0]);**

**bubbleSort(data, size);**

**cout << "Sorted Array in Ascending Order:\n";**

**printArray(data, size);**

**}**

**# Bubble sort in Python**

**def bubbleSort(array):**

**# loop to access each array element**

**for i in range(len(array)):**

**# loop to compare array elements**

**for j in range(0, len(array) - i - 1):**

**# compare two adjacent elements**

**# change > to < to sort in descending order**

**if array[j] > array[j + 1]:**

**# swapping elements if elements**

**# are not in the intended order**

**temp = array[j]**

**array[j] = array[j+1]**

**array[j+1] = temp**

**data = [-2, 45, 0, 11, -9]**

**bubbleSort(data)**

**print('Sorted Array in Ascending Order:')**

**print(data)**

**// Optimized Bubble sort in C**

**#include**

**// perform the bubble sort**

**void bubbleSort(int array[], int size) {**

**// loop to access each array element**

**for (int step = 0; step < size - 1; ++step) {**

**// check if swapping occurs**

**int swapped = 0;**

**// loop to compare array elements**

**for (int i = 0; i < size - step - 1; ++i) {**

**// compare two array elements**

**// change > to < to sort in descending order**

**if (array[i] > array[i + 1]) {**

**// swapping occurs if elements**

**// are not in the intended order**

**int temp = array[i];**

**array[i] = array[i + 1];**

**array[i + 1] = temp;**

**swapped = 1;**

**}**

**}**

**// no swapping means the array is already sorted**

**// so no need for further comparison**

**if (swapped == 0) {**

**break;**

**}**

**}**

**}**

**// print array**

**void printArray(int array[], int size) {**

**for (int i = 0; i < size; ++i) {**

**printf("%d ", array[i]);**

**}**

**printf("\n");**

**}**

**// main method**

**int main() {**

**int data[] = {-2, 45, 0, 11, -9};**

**// find the array's length**

**int size = sizeof(data) / sizeof(data[0]);**

**bubbleSort(data, size);**

**printf("Sorted Array in Ascending Order:\n");**

**printArray(data, size);**

**}**

**// Optimized Bubble sort in Java**

**import java.util.Arrays;**

**class Main {**

**// perform the bubble sort**

**static void bubbleSort(int array[]) {**

**int size = array.length;**

**// loop to access each array element**

**for (int i = 0; i < (size-1); i++) {**

**// check if swapping occurs**

**boolean swapped = false;**

**// loop to compare adjacent elements**

**for (int j = 0; j < (size-i-1); j++) {**

**// compare two array elements**

**// change > to < to sort in descending order**

**if (array[j] > array[j + 1]) {**

**// swapping occurs if elements**

**// are not in the intended order**

**int temp = array[j];**

**array[j] = array[j + 1];**

**array[j + 1] = temp;**

**swapped = true;**

**}**

**}**

**// no swapping means the array is already sorted**

**// so no need for further comparison**

**if (!swapped)**

**break;**

**}**

**}**

**public static void main(String args[]) {**

**int[] data = { -2, 45, 0, 11, -9 };**

**// call method using the class name**

**Main.bubbleSort(data);**

**System.out.println("Sorted Array in Ascending Order:");**

**System.out.println(Arrays.toString(data));**

**}**

**}**

**# Insertion sort in Python**

**def insertionSort(array):**

**for step in range(1, len(array)):**

**key = array[step]**

**j = step - 1**

**# Compare key with each element on the left of it until an element smaller than it is found**

**# For descending order, change key<array[j] to key>array[j].**

**while j >= 0 and key < array[j]:**

**array[j + 1] = array[j]**

**j = j - 1**

**# Place key at after the element just smaller than it.**

**array[j + 1] = key**

**data = [9, 5, 1, 4, 3]**

**insertionSort(data)**

**print('Sorted Array in Ascending Order:')**

**print(data)**

**// Insertion sort in Java**

**import java.util.Arrays;**

**class InsertionSort {**

**void insertionSort(int array[]) {**

**int size = array.length;**

**for (int step = 1; step < size; step++) {**

**int key = array[step];**

**int j = step - 1;**

**// Compare key with each element on the left of it until an element smaller than**

**// it is found.**

**// For descending order, change key<array[j] to key>array[j].**

**while (j >= 0 && key < array[j]) {**

**array[j + 1] = array[j];**

**--j;**

**}**

**// Place key at after the element just smaller than it.**

**array[j + 1] = key;**

**}**

**}**

**# Linear Search in Python**

**def linearSearch(array, n, x):**

**# Going through array sequencially**

**for i in range(0, n):**

**if (array[i] == x):**

**return i**

**return -1**

**array = [2, 4, 0, 1, 9]**

**x = 1**

**n = len(array)**

**result = linearSearch(array, n, x)**

**if(result == -1):**

**print("Element not found")**

**else:**

**print("Element found at index: ", result)**

**// Driver code**

**public static void main(String args[]) {**

**int[] data = { 9, 5, 1, 4, 3 };**

**InsertionSort is = new InsertionSort();**

**is.insertionSort(data);**

**System.out.println("Sorted Array in Ascending Order: ");**

**System.out.println(Arrays.toString(data));**

**}**

**}**

**// Insertion sort in C**

**#include <stdio.h>**

**// Function to print an array**

**void printArray(int array[], int size) {**

**for (int i = 0; i < size; i++) {**

**printf("%d ", array[i]);**

**}**

**printf("\n");**

**}**

**void insertionSort(int array[], int size) {**

**for (int step = 1; step < size; step++) {**

**int key = array[step];**

**int j = step - 1;**

**// Compare key with each element on the left of it until an element smaller than**

**// it is found.**

**// For descending order, change key<array[j] to key>array[j].**

**while (key < array[j] && j >= 0) {**

**array[j + 1] = array[j];**

**--j;**

**}**

**array[j + 1] = key;**

**}**

**}**

**// Driver code**

**int main() {**

**int data[] = {9, 5, 1, 4, 3};**

**int size = sizeof(data) / sizeof(data[0]);**

**insertionSort(data, size);**

**printf("Sorted array in ascending order:\n");**

**printArray(data, size);**

**}**

**// Insertion sort in C++**

**#include <iostream>**

**using namespace std;**

**// Function to print an array**

**void printArray(int array[], int size) {**

**for (int i = 0; i < size; i++) {**

**cout << array[i] << " ";**

**}**

**cout << endl;**

**}**

**void insertionSort(int array[], int size) {**

**for (int step = 1; step < size; step++) {**

**int key = array[step];**

**int j = step - 1;**

**// Compare key with each element on the left of it until an element smaller than**

**// it is found.**

**// For descending order, change key<array[j] to key>array[j].**

**while (key < array[j] && j >= 0) {**

**array[j + 1] = array[j];**

**--j;**

**}**

**array[j + 1] = key;**

**}**

**}**

**// Driver code**

**int main() {**

**int data[] = {9, 5, 1, 4, 3};**

**int size = sizeof(data) / sizeof(data[0]);**

**insertionSort(data, size);**

**cout << "Sorted array in ascending order:\n";**

**printArray(data, size);**

**}**

**# MergeSort in Python**

**def mergeSort(array):**

**if len(array) > 1:**

**# r is the point where the array is divided into two subarrays**

**r = len(array)//2**

**L = array[:r]**

**M = array[r:]**

**# Sort the two halves**

**mergeSort(L)**

**mergeSort(M)**

**i = j = k = 0**

**# Until we reach either end of either L or M, pick larger among**

**# elements L and M and place them in the correct position at A[p..r]**

**while i < len(L) and j < len(M):**

**if L[i] < M[j]:**

**array[k] = L[i]**

**i += 1**

**else:**

**array[k] = M[j]**

**j += 1**

**k += 1**

**# When we run out of elements in either L or M,**

**# pick up the remaining elements and put in A[p..r]**

**while i < len(L):**

**array[k] = L[i]**

**i += 1**

**k += 1**

**while j < len(M):**

**array[k] = M[j]**

**j += 1**

**k += 1**

**# Print the array**

**def printList(array):**

**for i in range(len(array)):**

**print(array[i], end=" ")**

**print()**

**# Driver program**

**if \_\_name\_\_ == '\_\_main\_\_':**

**array = [6, 5, 12, 10, 9, 1]**

**mergeSort(array)**

**print("Sorted array is: ")**

**printList(array)**

**// Merge sort in Java**

**class MergeSort {**

**// Merge two subarrays L and M into arr**

**void merge(int arr[], int p, int q, int r) {**

**// Create L ← A[p..q] and M ← A[q+1..r]**

**int n1 = q - p + 1;**

**int n2 = r - q;**

**int L[] = new int[n1];**

**int M[] = new int[n2];**

**for (int i = 0; i < n1; i++)**

**L[i] = arr[p + i];**

**for (int j = 0; j < n2; j++)**

**M[j] = arr[q + 1 + j];**

**// Maintain current index of sub-arrays and main array**

**int i, j, k;**

**i = 0;**

**j = 0;**

**k = p;**

**// Until we reach either end of either L or M, pick larger among**

**// elements L and M and place them in the correct position at A[p..r]**

**while (i < n1 && j < n2) {**

**if (L[i] <= M[j]) {**

**arr[k] = L[i];**

**i++;**

**} else {**

**arr[k] = M[j];**

**j++;**

**}**

**k++;**

**}**

**// When we run out of elements in either L or M,**

**// pick up the remaining elements and put in A[p..r]**

**while (i < n1) {**

**arr[k] = L[i];**

**i++;**

**k++;**

**}**

**while (j < n2) {**

**arr[k] = M[j];**

**j++;**

**k++;**

**}**

**}**

**// Divide the array into two subarrays, sort them and merge them**

**void mergeSort(int arr[], int l, int r) {**

**if (l < r) {**

**// m is the point where the array is divided into two subarrays**

**int m = (l + r) / 2;**

**mergeSort(arr, l, m);**

**mergeSort(arr, m + 1, r);**

**// Merge the sorted subarrays**

**merge(arr, l, m, r);**

**}**

**}**

**// Print the array**

**static void printArray(int arr[]) {**

**int n = arr.length;**

**for (int i = 0; i < n; ++i)**

**System.out.print(arr[i] + " ");**

**System.out.println();**

**}**

**// Driver program**

**public static void main(String args[]) {**

**int arr[] = { 6, 5, 12, 10, 9, 1 };**

**MergeSort ob = new MergeSort();**

**ob.mergeSort(arr, 0, arr.length - 1);**

**System.out.println("Sorted array:");**

**printArray(arr);**

**}**

**}**

**// Merge sort in C**

**#include <stdio.h>**

**// Merge two subarrays L and M into arr**

**void merge(int arr[], int p, int q, int r) {**

**// Create L ← A[p..q] and M ← A[q+1..r]**

**int n1 = q - p + 1;**

**int n2 = r - q;**

**int L[n1], M[n2];**

**for (int i = 0; i < n1; i++)**

**L[i] = arr[p + i];**

**for (int j = 0; j < n2; j++)**

**M[j] = arr[q + 1 + j];**

**// Maintain current index of sub-arrays and main array**

**int i, j, k;**

**i = 0;**

**j = 0;**

**k = p;**

**// Until we reach either end of either L or M, pick larger among**

**// elements L and M and place them in the correct position at A[p..r]**

**while (i < n1 && j < n2) {**

**if (L[i] <= M[j]) {**

**arr[k] = L[i];**

**i++;**

**} else {**

**arr[k] = M[j];**

**j++;**

**}**

**k++;**

**}**

**// When we run out of elements in either L or M,**

**// pick up the remaining elements and put in A[p..r]**

**while (i < n1) {**

**arr[k] = L[i];**

**i++;**

**k++;**

**}**

**while (j < n2) {**

**arr[k] = M[j];**

**j++;**

**k++;**

**}**

**}**

**// Divide the array into two subarrays, sort them and merge them**

**void mergeSort(int arr[], int l, int r) {**

**if (l < r) {**

**// m is the point where the array is divided into two subarrays**

**int m = l + (r - l) / 2;**

**mergeSort(arr, l, m);**

**mergeSort(arr, m + 1, r);**

**// Merge the sorted subarrays**

**merge(arr, l, m, r);**

**}**

**}**

**// Print the array**

**void printArray(int arr[], int size) {**

**for (int i = 0; i < size; i++)**

**printf("%d ", arr[i]);**

**printf("\n");**

**}**

**// Driver program**

**int main() {**

**int arr[] = {6, 5, 12, 10, 9, 1};**

**int size = sizeof(arr) / sizeof(arr[0]);**

**mergeSort(arr, 0, size - 1);**

**printf("Sorted array: \n");**

**printArray(arr, size);**

**}**

**// Merge sort in C++**

**#include <iostream>**

**using namespace std;**

**// Merge two subarrays L and M into arr**

**void merge(int arr[], int p, int q, int r) {**

**// Create L ← A[p..q] and M ← A[q+1..r]**

**int n1 = q - p + 1;**

**int n2 = r - q;**

**int L[n1], M[n2];**

**for (int i = 0; i < n1; i++)**

**L[i] = arr[p + i];**

**for (int j = 0; j < n2; j++)**

**M[j] = arr[q + 1 + j];**

**// Maintain current index of sub-arrays and main array**

**int i, j, k;**

**i = 0;**

**j = 0;**

**k = p;**

**// Until we reach either end of either L or M, pick larger among**

**// elements L and M and place them in the correct position at A[p..r]**

**while (i < n1 && j < n2) {**

**if (L[i] <= M[j]) {**

**arr[k] = L[i];**

**i++;**

**} else {**

**arr[k] = M[j];**

**j++;**

**}**

**k++;**

**}**

**// When we run out of elements in either L or M,**

**// pick up the remaining elements and put in A[p..r]**

**while (i < n1) {**

**arr[k] = L[i];**

**i++;**

**k++;**

**}**

**while (j < n2) {**

**arr[k] = M[j];**

**j++;**

**k++;**

**}**

**}**

**// Divide the array into two subarrays, sort them and merge them**

**void mergeSort(int arr[], int l, int r) {**

**if (l < r) {**

**// m is the point where the array is divided into two subarrays**

**int m = l + (r - l) / 2;**

**mergeSort(arr, l, m);**

**mergeSort(arr, m + 1, r);**

**// Merge the sorted subarrays**

**merge(arr, l, m, r);**

**}**

**}**

**// Print the array**

**void printArray(int arr[], int size) {**

**for (int i = 0; i < size; i++)**

**cout << arr[i] << " ";**

**cout << endl;**

**}**

**// Driver program**

**int main() {**

**int arr[] = {6, 5, 12, 10, 9, 1};**

**int size = sizeof(arr) / sizeof(arr[0]);**

**mergeSort(arr, 0, size - 1);**

**cout << "Sorted array: \n";**

**printArray(arr, size);**

**return 0;**

**}**

**# Binary Search in python**

**def binarySearch(array, x, low, high):**

**# Repeat until the pointers low and high meet each other**

**while low <= high:**

**mid = low + (high - low)//2**

**if array[mid] == x:**

**return mid**

**elif array[mid] < x:**

**low = mid + 1**

**else:**

**high = mid - 1**

**return -1**

**array = [3, 4, 5, 6, 7, 8, 9]**

**x = 4**

**result = binarySearch(array, x, 0, len(array)-1)**

**if result != -1:**

**print("Element is present at index " + str(result))**

**else:**

**print("Not found")**

**// Binary Search in Java**

**class BinarySearch {**

**int binarySearch(int array[], int x, int low, int high) {**

**// Repeat until the pointers low and high meet each other**

**while (low <= high) {**

**int mid = low + (high - low) / 2;**

**if (array[mid] == x)**

**return mid;**

**if (array[mid] < x)**

**low = mid + 1;**

**else**

**high = mid - 1;**

**}**

**return -1;**

**}**

**public static void main(String args[]) {**

**BinarySearch ob = new BinarySearch();**

**int array[] = { 3, 4, 5, 6, 7, 8, 9 };**

**int n = array.length;**

**int x = 4;**

**int result = ob.binarySearch(array, x, 0, n - 1);**

**if (result == -1)**

**System.out.println("Not found");**

**else**

**System.out.println("Element found at index " + result);**

**}**

**}**

**// Binary Search in C**

**#include <stdio.h>**

**int binarySearch(int array[], int x, int low, int high) {**

**// Repeat until the pointers low and high meet each other**

**while (low <= high) {**

**int mid = low + (high - low) / 2;**

**if (array[mid] == x)**

**return mid;**

**if (array[mid] < x)**

**low = mid + 1;**

**else**

**high = mid - 1;**

**}**

**return -1;**

**}**

**int main(void) {**

**int array[] = {3, 4, 5, 6, 7, 8, 9};**

**int n = sizeof(array) / sizeof(array[0]);**

**int x = 4;**

**int result = binarySearch(array, x, 0, n - 1);**

**if (result == -1)**

**printf("Not found");**

**else**

**printf("Element is found at index %d", result);**

**return 0;**

**}**

**// Binary Search in C++**

**#include <iostream>**

**using namespace std;**

**int binarySearch(int array[], int x, int low, int high) {**

**// Repeat until the pointers low and high meet each other**

**while (low <= high) {**

**int mid = low + (high - low) / 2;**

**if (array[mid] == x)**

**return mid;**

**if (array[mid] < x)**

**low = mid + 1;**

**else**

**high = mid - 1;**

**}**

**return -1;**

**}**

**int main(void) {**

**int array[] = {3, 4, 5, 6, 7, 8, 9};**

**int x = 4;**

**int n = sizeof(array) / sizeof(array[0]);**

**int result = binarySearch(array, x, 0, n - 1);**

**if (result == -1)**

**printf("Not found");**

**else**

**printf("Element is found at index %d", result);**

**}**

**// Linear Search in Java**

**class LinearSearch {**

**public static int linearSearch(int array[], int x) {**

**int n = array.length;**

**// Going through array sequencially**

**for (int i = 0; i < n; i++) {**

**if (array[i] == x)**

**return i;**

**}**

**return -1;**

**}**

**public static void main(String args[]) {**

**int array[] = { 2, 4, 0, 1, 9 };**

**int x = 1;**

**int result = linearSearch(array, x);**

**if (result == -1)**

**System.out.print("Element not found");**

**else**

**System.out.print("Element found at index: " + result);**

**}**

**}**

**// Linear Search in C**

**#include <stdio.h>**

**int search(int array[], int n, int x) {**

**// Going through array sequencially**

**for (int i = 0; i < n; i++)**

**if (array[i] == x)**

**return i;**

**return -1;**

**}**

**int main() {**

**int array[] = {2, 4, 0, 1, 9};**

**int x = 1;**

**int n = sizeof(array) / sizeof(array[0]);**

**int result = search(array, n, x);**

**(result == -1) ? printf("Element not found") : printf("Element found at index: %d", result);**

**}**

**// Linear Search in C++**

**#include <iostream>**

**using namespace std;**

**int search(int array[], int n, int x) {**

**// Going through array sequencially**

**for (int i = 0; i < n; i++)**

**if (array[i] == x)**

**return i;**

**return -1;**

**}**

**int main() {**

**int array[] = {2, 4, 0, 1, 9};**

**int x = 1;**

**int n = sizeof(array) / sizeof(array[0]);**

**int result = search(array, n, x);**

**(result == -1) ? cout << "Element not found" : cout << "Element found at index: " << result;**

**}**

**# Stack implementation in python**

**# Creating a stack**

**def create\_stack():**

**stack = []**

**return stack**

**# Creating an empty stack**

**def check\_empty(stack):**

**return len(stack) == 0**

**# Adding items into the stack**

**def push(stack, item):**

**stack.append(item)**

**print("pushed item: " + item)**

**# Removing an element from the stack**

**def pop(stack):**

**if (check\_empty(stack)):**

**return "stack is empty"**

**return stack.pop()**

**stack = create\_stack()**

**push(stack, str(1))**

**push(stack, str(2))**

**push(stack, str(3))**

**push(stack, str(4))**

**print("popped item: " + pop(stack))**

**print("stack after popping an element: " + str(stack))**

**// Stack implementation in Java**

**class Stack {**

**private int arr[];**

**private int top;**

**private int capacity;**

**// Creating a stack**

**Stack(int size) {**

**arr = new int[size];**

**capacity = size;**

**top = -1;**

**}**

**// Add elements into stack**

**public void push(int x) {**

**if (isFull()) {**

**System.out.println("OverFlow\nProgram Terminated\n");**

**System.exit(1);**

**}**

**System.out.println("Inserting " + x);**

**arr[++top] = x;**

**}**

**// Remove element from stack**

**public int pop() {**

**if (isEmpty()) {**

**System.out.println("STACK EMPTY");**

**System.exit(1);**

**}**

**return arr[top--];**

**}**

**// Utility function to return the size of the stack**

**public int size() {**

**return top + 1;**

**}**

**// Check if the stack is empty**

**public Boolean isEmpty() {**

**return top == -1;**

**}**

**// Check if the stack is full**

**public Boolean isFull() {**

**return top == capacity - 1;**

**}**

**public void printStack() {**

**for (int i = 0; i <= top; i++) {**

**System.out.println(arr[i]);**

**}**

**}**

**public static void main(String[] args) {**

**Stack stack = new Stack(5);**

**stack.push(1);**

**stack.push(2);**

**stack.push(3);**

**stack.push(4);**

**stack.pop();**

**System.out.println("\nAfter popping out");**

**stack.printStack();**

**}**

**}**

**// Stack implementation in C**

**#include <stdio.h>**

**#include <stdlib.h>**

**#define MAX 10**

**int count = 0;**

**// Creating a stack**

**struct stack {**

**int items[MAX];**

**int top;**

**};**

**typedef struct stack st;**

**void createEmptyStack(st \*s) {**

**s->top = -1;**

**}**

**// Check if the stack is full**

**int isfull(st \*s) {**

**if (s->top == MAX - 1)**

**return 1;**

**else**

**return 0;**

**}**

**// Check if the stack is empty**

**int isempty(st \*s) {**

**if (s->top == -1)**

**return 1;**

**else**

**return 0;**

**}**

**// Add elements into stack**

**void push(st \*s, int newitem) {**

**if (isfull(s)) {**

**printf("STACK FULL");**

**} else {**

**s->top++;**

**s->items[s->top] = newitem;**

**}**

**count++;**

**}**

**// Remove element from stack**

**void pop(st \*s) {**

**if (isempty(s)) {**

**printf("\n STACK EMPTY \n");**

**} else {**

**printf("Item popped= %d", s->items[s->top]);**

**s->top--;**

**}**

**count--;**

**printf("\n");**

**}**

**// Print elements of stack**

**void printStack(st \*s) {**

**printf("Stack: ");**

**for (int i = 0; i < count; i++) {**

**printf("%d ", s->items[i]);**

**}**

**printf("\n");**

**}**

**// Driver code**

**int main() {**

**int ch;**

**st \*s = (st \*)malloc(sizeof(st));**

**createEmptyStack(s);**

**push(s, 1);**

**push(s, 2);**

**push(s, 3);**

**push(s, 4);**

**printStack(s);**

**pop(s);**

**printf("\nAfter popping out\n");**

**printStack(s);**

**}**

**// Stack implementation in C++**

**#include <stdlib.h>**

**#include <iostream>**

**using namespace std;**

**#define MAX 10**

**int size = 0;**

**// Creating a stack**

**struct stack {**

**int items[MAX];**

**int top;**

**};**

**typedef struct stack st;**

**void createEmptyStack(st \*s) {**

**s->top = -1;**

**}**

**// Check if the stack is full**

**int isfull(st \*s) {**

**if (s->top == MAX - 1)**

**return 1;**

**else**

**return 0;**

**}**

**// Check if the stack is empty**

**int isempty(st \*s) {**

**if (s->top == -1)**

**return 1;**

**else**

**return 0;**

**}**

**// Add elements into stack**

**void push(st \*s, int newitem) {**

**if (isfull(s)) {**

**cout << "STACK FULL";**

**} else {**

**s->top++;**

**s->items[s->top] = newitem;**

**}**

**size++;**

**}**

**// Remove element from stack**

**void pop(st \*s) {**

**if (isempty(s)) {**

**cout << "\n STACK EMPTY \n";**

**} else {**

**cout << "Item popped= " << s->items[s->top];**

**s->top--;**

**}**

**size--;**

**cout << endl;**

**}**

**// Print elements of stack**

**void printStack(st \*s) {**

**printf("Stack: ");**

**for (int i = 0; i < size; i++) {**

**cout << s->items[i] << " ";**

**}**

**cout << endl;**

**}**

**// Driver code**

**int main() {**

**int ch;**

**st \*s = (st \*)malloc(sizeof(st));**

**createEmptyStack(s);**

**push(s, 1);**

**push(s, 2);**

**push(s, 3);**

**push(s, 4);**

**printStack(s);**

**pop(s);**

**cout << "\nAfter popping out\n";**

**printStack(s);**

**}**

**# Queue implementation in Python**

**class Queue:**

**def \_\_init\_\_(self):**

**self.queue = []**

**# Add an element**

**def enqueue(self, item):**

**self.queue.append(item)**

**# Remove an element**

**def dequeue(self):**

**if len(self.queue) < 1:**

**return None**

**return self.queue.pop(0)**

**# Display the queue**

**def display(self):**

**print(self.queue)**

**def size(self):**

**return len(self.queue)**

**q = Queue()**

**q.enqueue(1)**

**q.enqueue(2)**

**q.enqueue(3)**

**q.enqueue(4)**

**q.enqueue(5)**

**q.display()**

**q.dequeue()**

**print("After removing an element")**

**q.display()**

**// Queue implementation in Java**

**public class Queue {**

**int SIZE = 5;**

**int items[] = new int[SIZE];**

**int front, rear;**

**Queue() {**

**front = -1;**

**rear = -1;**

**}**

**boolean isFull() {**

**if (front == 0 && rear == SIZE - 1) {**

**return true;**

**}**

**return false;**

**}**

**boolean isEmpty() {**

**if (front == -1)**

**return true;**

**else**

**return false;**

**}**

**void enQueue(int element) {**

**if (isFull()) {**

**System.out.println("Queue is full");**

**} else {**

**if (front == -1)**

**front = 0;**

**rear++;**

**items[rear] = element;**

**System.out.println("Inserted " + element);**

**}**

**}**

**int deQueue() {**

**int element;**

**if (isEmpty()) {**

**System.out.println("Queue is empty");**

**return (-1);**

**} else {**

**element = items[front];**

**if (front >= rear) {**

**front = -1;**

**rear = -1;**

**} /\* Q has only one element, so we reset the queue after deleting it. \*/**

**else {**

**front++;**

**}**

**System.out.println("Deleted -> " + element);**

**return (element);**

**}**

**}**

**void display() {**

**/\* Function to display elements of Queue \*/**

**int i;**

**if (isEmpty()) {**

**System.out.println("Empty Queue");**

**} else {**

**System.out.println("\nFront index-> " + front);**

**System.out.println("Items -> ");**

**for (i = front; i <= rear; i++)**

**System.out.print(items[i] + " ");**

**System.out.println("\nRear index-> " + rear);**

**}**

**}**

**public static void main(String[] args) {**

**Queue q = new Queue();**

**// deQueue is not possible on empty queue**

**q.deQueue();**

**// enQueue 5 elements**

**q.enQueue(1);**

**q.enQueue(2);**

**q.enQueue(3);**

**q.enQueue(4);**

**q.enQueue(5);**

**// 6th element can't be added to because the queue is full**

**q.enQueue(6);**

**q.display();**

**// deQueue removes element entered first i.e. 1**

**q.deQueue();**

**// Now we have just 4 elements**

**q.display();**

**}**

**}**

**// Queue implementation in C**

**#include <stdio.h>**

**#define SIZE 5**

**void enQueue(int);**

**void deQueue();**

**void display();**

**int items[SIZE], front = -1, rear = -1;**

**int main() {**

**//deQueue is not possible on empty queue**

**deQueue();**

**//enQueue 5 elements**

**enQueue(1);**

**enQueue(2);**

**enQueue(3);**

**enQueue(4);**

**enQueue(5);**

**// 6th element can't be added to because the queue is full**

**enQueue(6);**

**display();**

**//deQueue removes element entered first i.e. 1**

**deQueue();**

**//Now we have just 4 elements**

**display();**

**return 0;**

**}**

**void enQueue(int value) {**

**if (rear == SIZE - 1)**

**printf("\nQueue is Full!!");**

**else {**

**if (front == -1)**

**front = 0;**

**rear++;**

**items[rear] = value;**

**printf("\nInserted -> %d", value);**

**}**

**}**

**void deQueue() {**

**if (front == -1)**

**printf("\nQueue is Empty!!");**

**else {**

**printf("\nDeleted : %d", items[front]);**

**front++;**

**if (front > rear)**

**front = rear = -1;**

**}**

**}**

**// Function to print the queue**

**void display() {**

**if (rear == -1)**

**printf("\nQueue is Empty!!!");**

**else {**

**int i;**

**printf("\nQueue elements are:\n");**

**for (i = front; i <= rear; i++)**

**printf("%d ", items[i]);**

**}**

**printf("\n");**

**}**

**// Queue implementation in C++**

**#include <iostream>**

**#define SIZE 5**

**using namespace std;**

**class Queue {**

**private:**

**int items[SIZE], front, rear;**

**public:**

**Queue() {**

**front = -1;**

**rear = -1;**

**}**

**bool isFull() {**

**if (front == 0 && rear == SIZE - 1) {**

**return true;**

**}**

**return false;**

**}**

**bool isEmpty() {**

**if (front == -1)**

**return true;**

**else**

**return false;**

**}**

**void enQueue(int element) {**

**if (isFull()) {**

**cout << "Queue is full";**

**} else {**

**if (front == -1) front = 0;**

**rear++;**

**items[rear] = element;**

**cout << endl**

**<< "Inserted " << element << endl;**

**}**

**}**

**int deQueue() {**

**int element;**

**if (isEmpty()) {**

**cout << "Queue is empty" << endl;**

**return (-1);**

**} else {**

**element = items[front];**

**if (front >= rear) {**

**front = -1;**

**rear = -1;**

**} /\* Q has only one element, so we reset the queue after deleting it. \*/**

**else {**

**front++;**

**}**

**cout << endl**

**<< "Deleted -> " << element << endl;**

**return (element);**

**}**

**}**

**void display() {**

**/\* Function to display elements of Queue \*/**

**int i;**

**if (isEmpty()) {**

**cout << endl**

**<< "Empty Queue" << endl;**

**} else {**

**cout << endl**

**<< "Front index-> " << front;**

**cout << endl**

**<< "Items -> ";**

**for (i = front; i <= rear; i++)**

**cout << items[i] << " ";**

**cout << endl**

**<< "Rear index-> " << rear << endl;**

**}**

**}**

**};**

**int main() {**

**Queue q;**

**//deQueue is not possible on empty queue**

**q.deQueue();**

**//enQueue 5 elements**

**q.enQueue(1);**

**q.enQueue(2);**

**q.enQueue(3);**

**q.enQueue(4);**

**q.enQueue(5);**

**// 6th element can't be added to because the queue is full**

**q.enQueue(6);**

**q.display();**

**//deQueue removes element entered first i.e. 1**

**q.deQueue();**

**//Now we have just 4 elements**

**q.display();**

**return 0;**

**}**