College Of Engineering Trivandrum

# Data Structures Lab

Rahul T

S3 CSE Roll No:53

TVE19CS053

Department of Computer Science

September 25, 2020

# Table of contents

click on the title to go to the page

- Table of contents
- 1. Stack using Array
  - 1.1 Problem
  - 1.2 Algorithm
  - 1.3 Code
  - 1.4 Sample Output
  - 1.5 Result
- 2. Queue using Array
  - 2.1 Problem
  - 2.2 Algorithm
  - 2.3 Code
  - 2.4 Sample Output
  - 2.5 Result
- 3. Polynomial using Array
  - 3.1 Problem
  - 3.2 Algorithm
  - 3.3 Code
  - 3.4 Sample Output
  - 3.5 Result
- 4. Sorting
  - 4.1 Problem
  - 4.2 Algorithm
  - 4.3 Code
  - 4.4 Sample Output
  - 4.5 Result
- 5. Employee Details
  - 5.1 Problem
  - 5.2 Algorithm
  - 5.3 Code
  - 5.4 Sample Output
  - 5.5 Result
- 6. Infix to Postfix
  - 6.1 Problem
  - 6.2 Algorithm
  - 6.3 Code
  - 6.4 Sample Output
  - 6.5 Result
- 7. Postfix Evaluation
  - 7.1 Problem

  - 7.2 Algorithm
  - 7.3 Code
  - 7.4 Sample Output
  - 7.5 Result
- 8. Sort and Search Strings
  - 8.1 Problem
  - 8.2 Algorithm
  - 8.3 Code
  - 8.4 Sample Output
  - 8.5 Result
- 9. Priority Queue
  - 9.1 Problem
  - 9.2 Algorithm
  - 9.3 Code
  - 9.4 Sample Output
  - 9.5 Result
- 10. Circular Queue
  - 10.1 Problem
  - 10.2 Algorithm
  - 10.3 Code
  - 10.4 Sample Output
  - 10.5 Result
- 11. Double Ended Queue
  - 11.1 Problem
  - 11.2 Algorithm
  - 11.3 Code
  - 11.4 Sample Output
  - 11.5 Result

# 1. Stack using Array

## 1.1 Problem

Implement a Stack using arrays with the operations:

- Pushing elements to the Stack.
- Popping elements from the Stack
- Check if the stack is empty
- Check if the stack is full

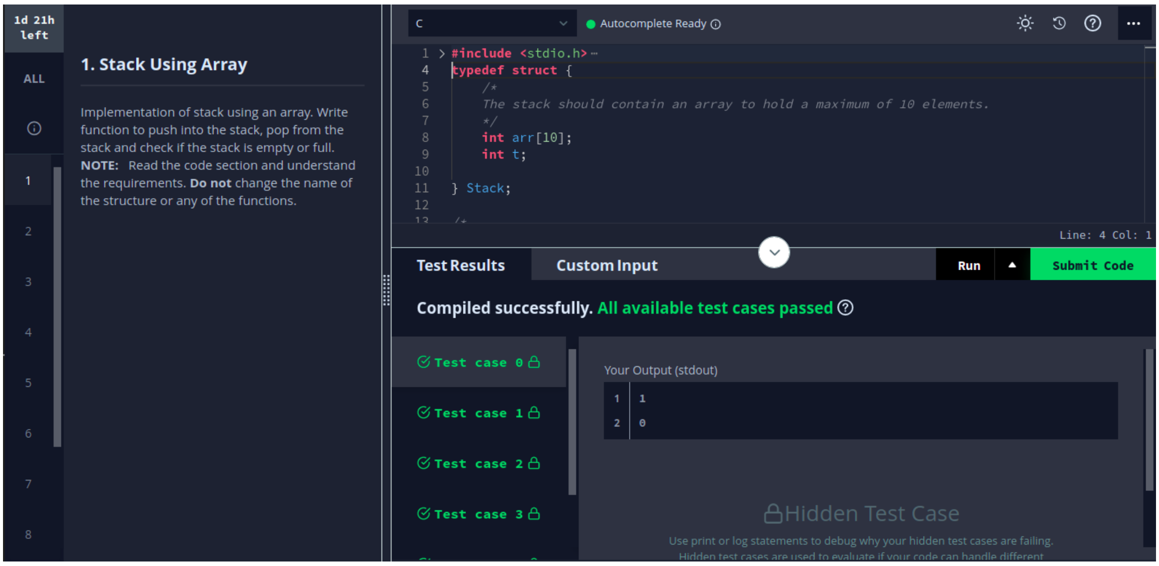
## 1.2 Algorithm

```
Start of struct Stack
    int arr[10]     {10 is the maximum capacity of the stack}
    int top         {top of the stack}
End of the struct Stack
Stack s={.top = -1}     {we initialise the top with -1}
Start of main function
input q
while q > 0 do
    input choice and n
    switch(choice)
        case 0 : push(n)        {function call}
            break
        case 1: print pop() {function call}
            break
        case 2: print isEmpty() {function call}
            break
        case 3: print isFull()  {function call}
            break
    End switch
Endwhile
return 0
End of main function
Start of function push(n)   {n is the argument}
if s.top < 9 then           {when the stack is not full}
    increment s.top
    s.arr[s.top] <-- n
Endif
End of function push
Start of function pop()
    if s.top equal to -1 then
        return -1
    Endif
    else
        a <-- s.arr[top]
        decrement s.top
        return a
    Endelse
End of function pop
Start of function isEmpty()
if s.top == -1 then
    return true
Endif
else
    return false
Endelse
End of function isEmpty
Start of function isFull()
if s.top Equal to 9 then
    return true
Endif
else
    return false
Endelse
End of isFull function
```

## 1.3 Code

```c
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>

typedef struct {
    /*
    The stack should contain an array to hold a maximum of 10 elements.
    */
    int arr[10];
    int t;

} Stack;

/*
Initialising the stack, use this stack variable 's' in your functions.
*/
Stack s={.t = -1};

void push(int n) {
    /*
    Push the integer n into the stack.
    Ignore if the operation is not possible.
    */
    if(s.t < 9){
        s.t++;
        s.arr[s.t] = n;
    }
}

int pop() {
    /*
    Pop the top element in the stack and return that element.
    Return -1 the operation is not possible.
    */
    int a;
    if(s.t == -1){
        return -1;
    }
    else{
        a=s.arr[s.t];
        s.t--;
        return a;
    }
}

bool isEmpty() {
    /*
    Check if the stack is empty or not. Return true/false.
    */
    if(s.t == -1){
        return true;
    }
    else{
        return false;
    }
}

bool isFull() {
    /*
    Check if the stack is full or not. Return true/false.
    */
    if(s.t == 9){
        return true;
    }
    else {
        return false;
    }
}

int main() {
    int q, choice, n;
    scanf("%d", &q);
    while(q--) {
        scanf("%d%d", &choice, &n);
        switch(choice) {
            case 0: push(n);
                    break;
            case 1: printf("%d\n", pop());
                    break;
            case 2: printf("%d\n", isEmpty());
                    break;
            case 3: printf("%d\n", isFull());
                    break;
            // case 4: ;
            //          Stack temp;
            //          pop(&temp);
            //          push(&temp, n);
            //          break;
        }
    }
    return 0;
}
```

## 1.4 Sample Output



**1. Stack Using Array**

Implementation of stack using an array. Write function to push into the stack, pop from the stack and check if the stack is empty or full.
**NOTE:** Read the code section and understand the requirements. **Do not** change the name of the structure or any of the functions.

```
> #include <stdio.h> ⋯
typedef struct {
    /*
    The stack should contain an array to hold a maximum of 10 elements.
    */
    int arr[10];
    int t;

} Stack;

/*
```

Compiled successfully. All available test cases passed

Your Output (stdout)

```
1
0
```

Hidden Test Case

Use print or log statements to debug why your hidden test cases are failing.
Hidden test cases are used to evaluate if your code can handle different

## 1.5 Result

Program submitted and executed successfully in HackerRank Platform via user id @rahulmanoj

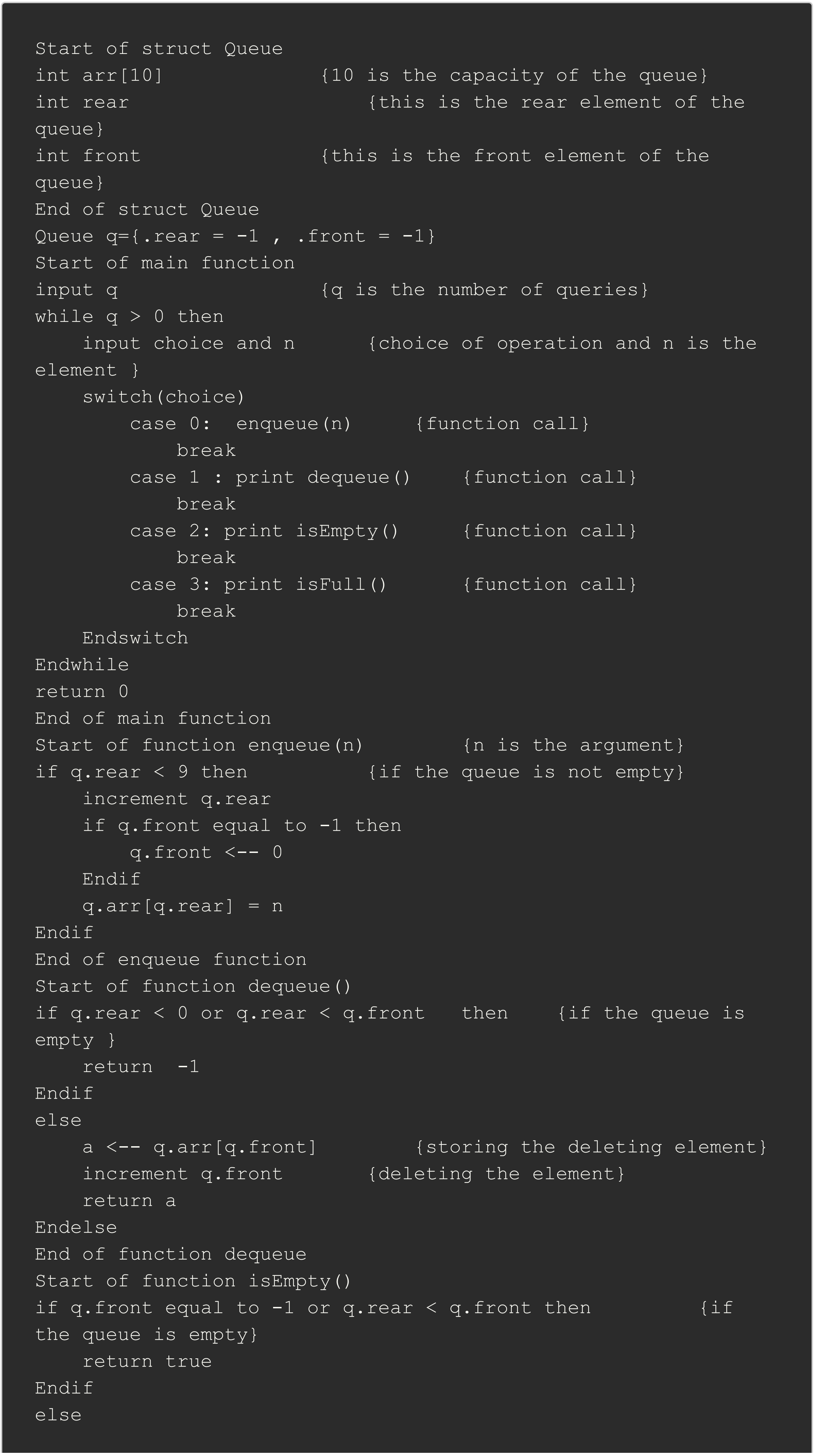
# 2. Queue using Array

## 2.1 Problem

Implement a Queue using arrays with the operations:

- Insert elements to the Queue.
- Delete elements from the Queue.
- Check if the Queue is empty.
- Check if the Queue is full

## 2.2 Algorithm



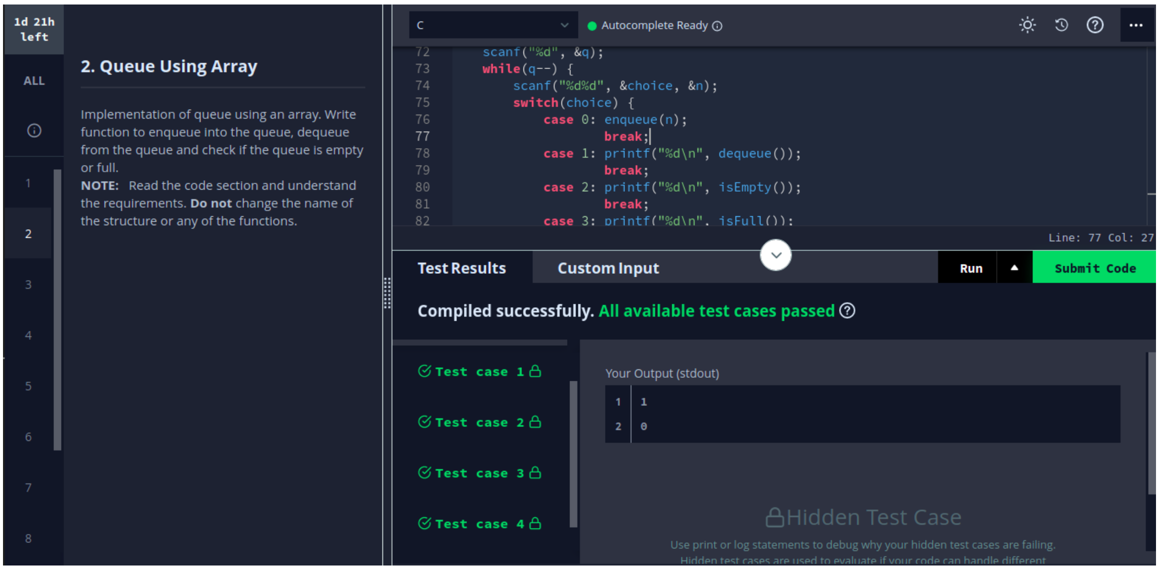
```
Start of struct Queue
int arr[10]             {10 is the capacity of the queue}
int rear                    {this is the rear element of the
queue}
int front               {this is the front element of the
queue}
End of struct Queue
Queue q={.rear = -1 , .front = -1}
Start of main function
input q                 {q is the number of queries}
while q > 0 then
    input choice and n      {choice of operation and n is the
element }
    switch(choice)
        case 0:  enqueue(n)     {function call}
            break
        case 1 : print dequeue()    {function call}
            break
        case 2: print isEmpty()     {function call}
            break
        case 3: print isFull()      {function call}
            break
    Endswitch
Endwhile
return 0
End of main function
Start of function enqueue(n)        {n is the argument}
if q.rear < 9 then          {if the queue is not empty}
    increment q.rear
    if q.front equal to -1 then
        q.front <-- 0
    Endif
    q.arr[q.rear] = n
Endif
End of enqueue function
Start of function dequeue()
if q.rear < 0 or q.rear < q.front   then    {if the queue is
empty }
    return  -1
Endif
else
    a <-- q.arr[q.front]        {storing the deleting element}
    increment q.front       {deleting the element}
    return a
Endelse
End of function dequeue
Start of function isEmpty()
if q.front equal to -1 or q.rear < q.front then         {if
the queue is empty}
    return true
Endif
else
```

```
    return false
Endelse
End of function isEmpty
Start of function isFull()
if q.rear - q.front equal to 9 then      {if the queue is full}
    return true
Endif
else
    return false
Endelse
End of isFull function
```

## 2.3 Code

```
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
typedef struct {
    /*
    The queue should contain an array to hold a maximum of 10 elements.
    */
    int arr[10];
    int r;
    int f;
} Queue;

/*
Initialising the queue, use this queue variable 'q' in your functions.
*/
Queue q={.r = -1,.f = -1};

void enqueue(int n) {
    /*
    Enqueue the integer n into the queue.
    Ignore if the operation is not possible.
    */
    if(q.r -q.f < 9){
        q.r++;
        if(q.f == -1){
            q.f = 0;
        }
        q.arr[q.r] = n;
    }
}

int dequeue() {
    /*
    Dequeue the front element from the queue and return that
element.
    Return -1 the operation is not possible.
    */
    if(q.r < 0||q.r < q.f){
        return -1;
    }
    else{
        int a = q.arr[q.f];
        q.f++;
    return a;
    }
}

bool isEmpty() {
    /*
    Check if the queue is empty or not. Return true/false.
    */
    if(q.f == -1 || q.r < q.f){
        return true;
    }
    else{
        return false;
    }
}

bool isFull() {
    /*
    Check if the queue is full or not. Return true/false.
    */
    if(q.r-q.f == 9){
        return true;
    }
    else{
        return false;
    }
}
int main() {
    int q, choice, n;
    scanf("%d", &q);
    while(q--) {
        scanf("%d%d", &choice, &n);
        switch(choice) {
            case 0: enqueue(n);
                    break;
            case 1: printf("%d\n", dequeue());
                    break;
            case 2: printf("%d\n", isEmpty());
                    break;
            case 3: printf("%d\n", isFull());
                    break;
            // case 4: ;
            //         Stack temp;
            //         pop(&temp);
            //          push(&temp, n);
            //          break;
        }
    }
    return 0;
}
```

## 2.4 Sample Output



**2. Queue Using Array**

Implementation of queue using an array. Write function to enqueue into the queue, dequeue from the queue and check if the queue is empty or full.
**NOTE:** Read the code section and understand the requirements. **Do not** change the name of the structure or any of the functions.

```
    scanf("%d", &q);
    while(q--) {
        scanf("%d%d", &choice, &n);
        switch(choice) {
            case 0: enqueue(n);
                    break;
            case 1: printf("%d\n", dequeue());
                    break;
            case 2: printf("%d\n", isEmpty());
                    break;
            case 3: printf("%d\n", isFull());
```

Compiled successfully. All available test cases passed

✓ Test case 1
✓ Test case 2
✓ Test case 3
✓ Test case 4

Your Output (stdout)

```
1
0
```

Hidden Test Case

Use print or log statements to debug why your hidden test cases are failing.
Hidden test cases are used to evaluate if your code can handle different

## 2.5 Result

Program submitted and executed successfully in HackerRank Platform via user id @rahulmanoj

# 3. Polynomial using Array

## 3.1 Problem

Write a program to read two polynomials and store them in an array. Calculate the sum of the two polynomials and display the first polynomial, second polynomial and the resultant polynomial.

## 3.2 Algorithm

```
START
Input degree of the polynomials n and m
For i <-- 0 to n
    Input coefficients of first polynomial, a[i]
End for
For i<--0 to m
    Input coefficients of second polynomial b[i]
End for
```

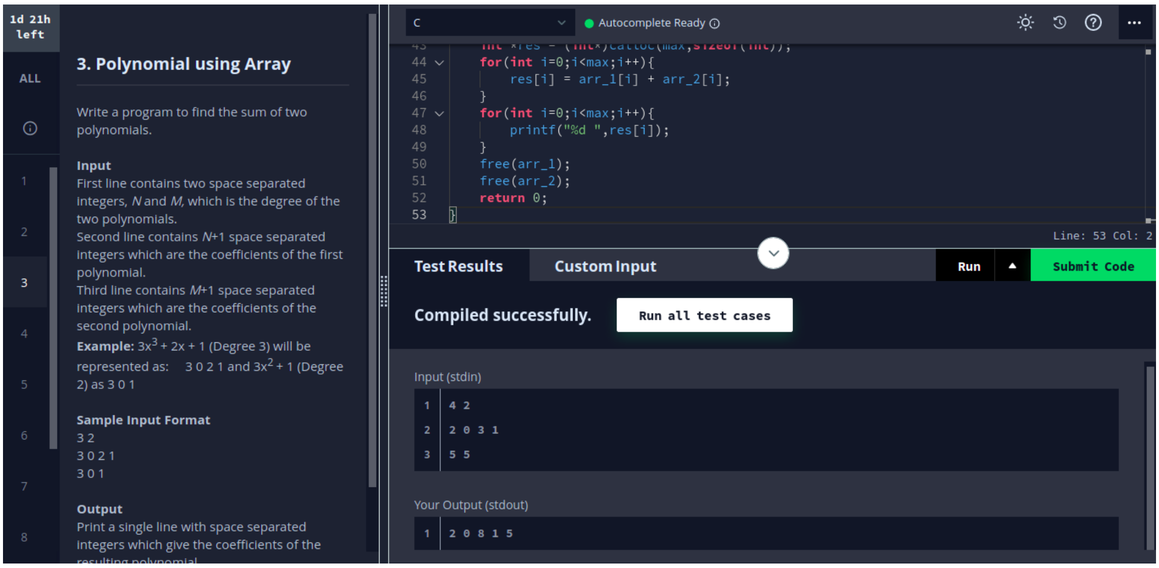
```
If n > m
    For i<--0 to n
        Set polynomial sum s[i] <-- 0
    End for
    For i <-- n to n-m and j<--m to 0
        s[i] <-- a[i] + b[j]
    End for
    For i <-- 0 to n-m-1
        s[i] <-- a[i]
    End for
    For i<--0 to n
        Print coefficients of resulting polynomial, s[i]
    End for
End if
Else
    For i <--0 to m
        Set polynomial sum s[i] <-- 0
    End for
    For i<--m to m-n and j<-- n to 0
        s[i] <-- a[j] + b[i]
    End for
    for i <-- 0 to m-n-1
        s[i]<--b[i]
    End for
    For i <-- 0 to m
        Print coefficients of resulting polynomial, s[i]
    End for
End else
STOP
```

## 3.3 Code

```
#include<stdio.h>
#include<stdlib.h>
int main(){
    int a,b;
    scanf("%d %d",&a,&b);
    int *arr_1,*arr_2;
    a++;
    b++;
    int max;
    if(a>b){
        max = a;
    }
    else{
        max = b;
    }
    arr_1 = (int*)calloc((max),sizeof(int));
    arr_2 = (int*)calloc((max),sizeof(int));
    if(max == a){
        for(int i=0;i<a;i++){
```

```
        scanf(" %d",&arr_1[i]);
    }
        int i;
        for(i=0;i<a-b;i++){
            arr_2[i] = 0;
        }
    for(;i<a;i++){
        scanf(" %d",&arr_2[i]);
    }
    }
    else{
    int i;
    for(i=0;i<b-a;i++){
            arr_1[i] = 0;
        }
    for(;i<b;i++){
        scanf(" %d",&arr_1[i]);
    }
    for(int i=0;i<b;i++){
        scanf(" %d",&arr_2[i]);
    }
    }

    int *res = (int*)calloc(max,sizeof(int));
    for(int i=0;i<max;i++){
        res[i] = arr_1[i] + arr_2[i];
    }
    for(int i=0;i<max;i++){
        printf("%d ",res[i]);
    }
    free(arr_1);
    free(arr_2);
    return 0;
}
```

## 3.4 Sample Output



**3. Polynomial using Array**

Write a program to find the sum of two polynomials.

**Input**
First line contains two space separated integers, *N* and *M,* which is the degree of the two polynomials.
Second line contains *N*+1 space separated integers which are the coefficients of the first polynomial.
Third line contains *M*+1 space separated integers which are the coefficients of the second polynomial.
**Example:** $3x^3 + 2x + 1$ (Degree 3) will be represented as: 3 0 2 1 and $3x^2 + 1$ (Degree 2) as 3 0 1

**Sample Input Format**
3 2
3 0 2 1
3 0 1

**Output**
Print a single line with space separated integers which give the coefficients of the resulting polynomial

```
    int *res = (int*)calloc(max,sizeof(int));
    for(int i=0;i<max;i++){
        res[i] = arr_1[i] + arr_2[i];
    }
    for(int i=0;i<max;i++){
        printf("%d ",res[i]);
    }
    free(arr_1);
    free(arr_2);
    return 0;
}
```

Line: 53 Col: 2

Compiled successfully.

Input (stdin)

```
1  4 2
2  2 0 3 1
3  5 5
```

Your Output (stdout)

```
1  2 0 8 1 5
```

## 3.5 Result

Program submitted and executed successfully in HackerRank Platform via user id @rahulmanoj

# 4. Sorting

## 4.1 Problem

Write a program to read numerical data stored in a file. Implement the following sorting algorithms to sort the numbers in ascending order. Implement each algorithm as a separate function.

- Bubble sort
- Insertion sort
- Selection sort

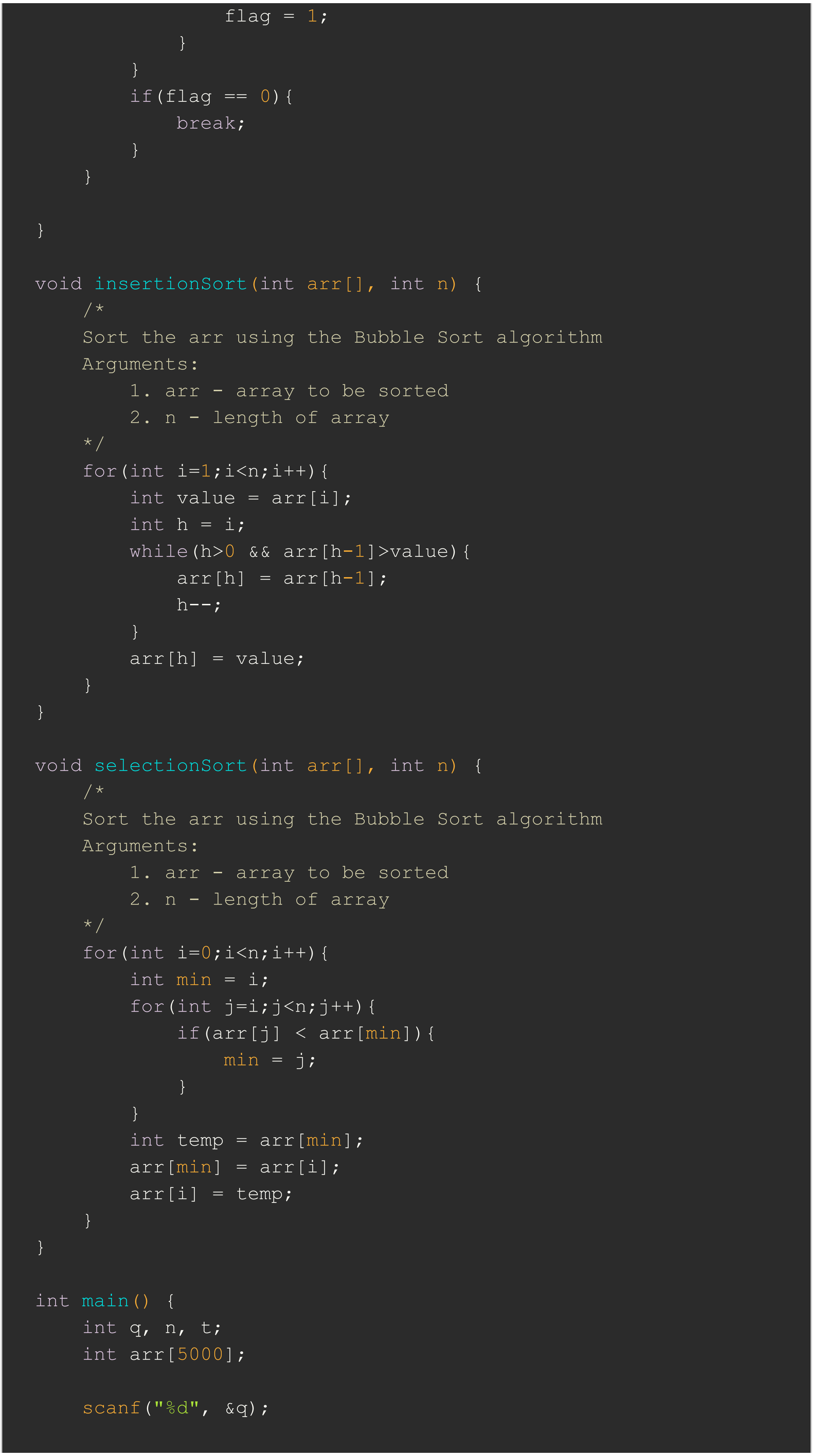
## 4.2 Algorithm

```
Start of main function
input q             {Number of Queries}
while q>0 do
    input t and n       {where t=choice of sorting n=the
number of elements in array}
    input the array arr
    if t equal to 1  then
        bubbleSort(arr,n)   {Function Call}
    Endif
    else if t equal to 2 then
        insertionSort(arr,n)    {Function Call}
    Endif
    else then
        selectionSort(arr,n)    {function call}
    Endelse
    print the array arr
End of main function
Start of function bubbleSort(arr,n) {arr and n are arguments}
for i <-- 0 to n do
    flag <-- 0
    for j <-- 0 to n - i - 1  do
        if arr[j] > arr[j+1] then
            temp <-- arr[j]     {swapping}
            arr[j] <-- arr[j+1]
            arr[j+1] <-- temp
        Endif
    Endfor
    if flag equal to 0
        break;          {to stop the iteration when arr
sorted}
```

```
Endfor
End of bubbleSort function
Start of function insertionSort(arr,n)  {arr and n are
arguments}
    for i<--1 to n do
        value <-- arr[i]
        hole <-- i
        while hole > 0 and arr[hole-1]>value do
            arr[hole] <-- arr[hole-1]
            decrement hole
        Endwhile
        arr[hole] <-- value
    Endfor
End of insertionSort function
Start of function insertionSort(arr,n)      {arr and n are
arguments}
for i <-- 0 to n do
    min <-- i
    for j <-- i to n do
        if arr[j] < arr[min] then           {finding the
smallest  element}
            min <-- j
        Endif
    Endfor
    temp <-- arr[min]               {swapping}
    arr[min] <-- arr[i]
    arr[i] <-- temp
Endfor
End of selectionSort function
```

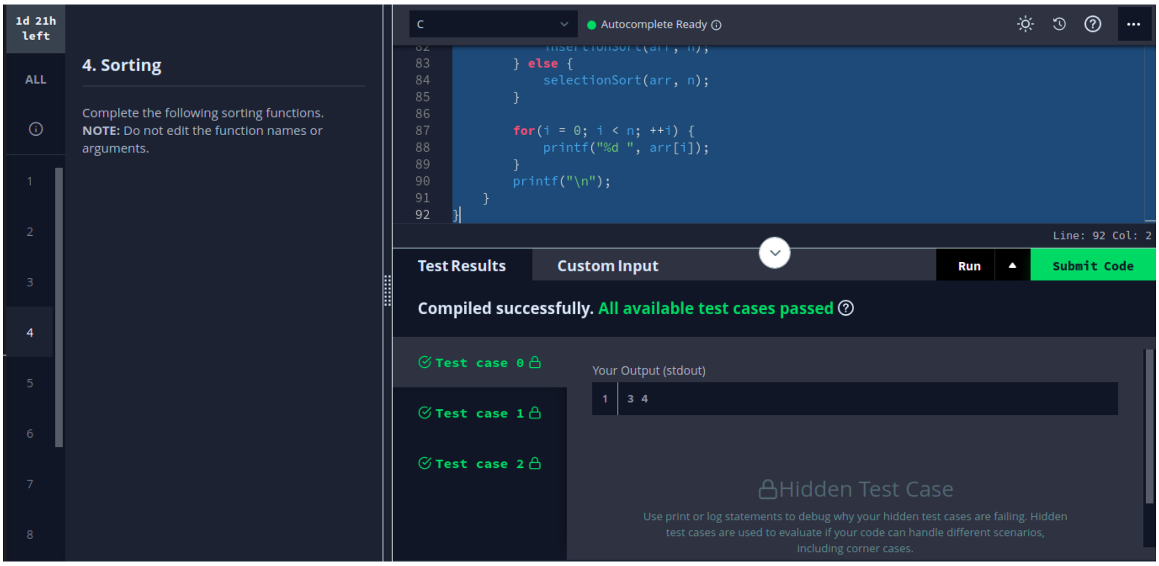
## 4.3 Code

```c
#include <stdio.h>

#include <stdio.h>

void bubbleSort(int arr[], int n) {
    /*
    Sort the arr using the Bubble Sort algorithm
    Arguments:
        1. arr - array to be sorted
        2. n - length of array
    */
    for(int i=0;i<n;i++){
            int flag = 0;
        for(int j=0;j<n-i-1;j++){
            if(arr[j]>arr[j+1]){
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
```



```
                flag = 1;
            }
        }
        if(flag == 0){
            break;
        }
    }

}

void insertionSort(int arr[], int n) {
    /*
    Sort the arr using the Bubble Sort algorithm
    Arguments:
        1. arr - array to be sorted
        2. n - length of array
    */
    for(int i=1;i<n;i++){
        int value = arr[i];
        int h = i;
        while(h>0 && arr[h-1]>value){
            arr[h] = arr[h-1];
            h--;
        }
        arr[h] = value;
    }
}

void selectionSort(int arr[], int n) {
    /*
    Sort the arr using the Bubble Sort algorithm
    Arguments:
        1. arr - array to be sorted
        2. n - length of array
    */
    for(int i=0;i<n;i++){
        int min = i;
        for(int j=i;j<n;j++){
            if(arr[j] < arr[min]){
                min = j;
            }
        }
        int temp = arr[min];
        arr[min] = arr[i];
        arr[i] = temp;
    }
}

int main() {
    int q, n, t;
    int arr[5000];

    scanf("%d", &q);
    while (q--) {
        scanf("%d%d", &t, &n);
        int i;

        for(i = 0; i < n; ++i) {
            scanf("%d", &arr[i]);
        }

        if (t == 1) {
            bubbleSort(arr, n);
        } else if (t == 2) {
            insertionSort(arr, n);
        } else {
            selectionSort(arr, n);
        }

        for(i = 0; i < n; ++i) {
            printf("%d ", arr[i]);
        }
        printf("\n");
    }
}
```

## 4.4 Sample Output



4. Sorting

Complete the following sorting functions.
**NOTE:** Do not edit the function names or arguments.

```
        } else {
            selectionSort(arr, n);
        }

        for(i = 0; i < n; ++i) {
            printf("%d ", arr[i]);
        }
        printf("\n");
    }
}
```

Compiled successfully. All available test cases passed

Test case 0

Test case 1

Test case 2

Your Output (stdout)

```
3 4
```

Hidden Test Case

Use print or log statements to debug why your hidden test cases are failing. Hidden test cases are used to evaluate if your code can handle different scenarios, including corner cases.

## 4.5 Result

Program submitted and executed successfully in HackerRank Platform via user id @rahulmanoj

# 5. Employee Details

## 5.1 Problem

Create a structure Employee with fields EmpId, Name and Salary. Name should contain first name, middle name and last name. Store the details of n employees, dynamically allocating memory for the same. Write a function to implement Linear Search to search for a particular employee, given the EmpId.

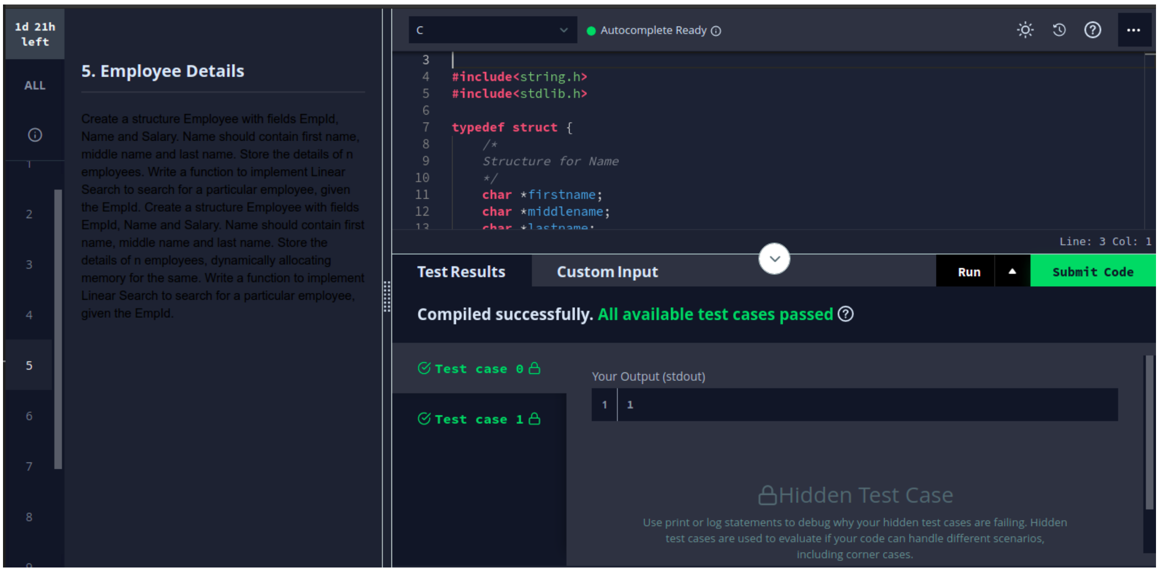
## 5.2 Algorithm

```
Start of Struct Name
    firstname       {the struct has 3 character pointer}
    middlename
    lastname
End of Struct name
 Start of Struct Employee
    EmpId           {the Employee id}
    name            {name of the employee of type struct name}
    salary          {salary of the Employee}
End of Struct Employee
Start of main function
input q                 {the number of Queries}
n <-- 0
while q > 0 do
    input t             {the choice of operation}
    if t is equal to 1 then
        input Empid,salary,firstName,middleName,lastName

addEmployee(E,n,firstName,middleName,lastName,Empid,salary)
{functioncall}
        increment n
    Endif
    else
        input Empid
        print search(E,n,Empid)     {function call}
    Endelse
Endwhile
End of main function
Start of function
addEmployee(E,n,firstname,middlename,lastname,empid,salary)
    if n < 9 then
        {dynamic memory allocation}
        E[n].name.firstname =
(char*)malloc(strlen(firstname)*sizeof(char))
        E[n].name.middlename =
(char*)malloc(strlen(middlename)*sizeof(char))
        E[n].name.lastname =
(char*)malloc(strlen(lastname)*sizeof(char))
        {copying strings}
        strcpy(E[n].name.firstname,firstName)
        strcpy(E[n].name.middlename,middleName)
        strcpy(E[n].name.lastname,lastName)
        E[n].EmpId <-- empid
        E[n].Salary <-- salary
    Endif
    else
        E <-- E + n      {incrementing the pointer}
        {dynamic memory allocation}
        E = (Employee*)malloc((n+1)*sizeof(Employee))
        (E+n)->name.firstname =
(char*)malloc(strlen(firstname)*sizeof(char))
        (E+n)->name.middlename =
(char*)malloc(strlen(middlename)*sizeof(char))
        (E+n)->name.lastname =
(char*)malloc(strlen(lastname)*sizeof(char))
        {copying strings}
        strcpy(E[n].name.firstname,firstName)
        strcpy(E[n].name.middlename,middleName)
        strcpy(E[n].name.lastname,lastName)
        (E+n)->EmpId <-- empId
        (E+n)->Salary <-- salary

End of addEmployee function
Start of function search(E,n,empId)         {E,n,empId are the
arguments}
for i <-- 0 to n do
    if (E+n)->EmpId is equal to empid then  {checking for the
given Empid}
        return true
    Endif
Endfor
return false
End of search function
```

## 5.3 Code

```
#include <stdio.h>
#include<stdbool.h>

#include<string.h>
#include<stdlib.h>

typedef struct {
    /*
    Structure for Name
    */
    char *firstname;
    char *middlename;
    char *lastname;
} Name;

typedef struct {
    /*
    Structure for Employee
    */
    int EmpId;
    Name name;
    float Salary;

} Employee;

void addEmployee(Employee E[], int n, char firstName[], char
middleName[], char lastName[], int empId, float salary) {
    /*
    n - Length of Employee array
    Add employee with Name(firstName, middleName, lastName),
empId, salary to array of employees E
    */
    if(n<9){
        E[n].name.firstname =
(char*)malloc(strlen(firstName)*sizeof(char));
        E[n].name.middlename =
(char*)malloc(strlen(middleName)*sizeof(char));
        E[n].name.lastname =
(char*)malloc(strlen(lastName)*sizeof(char));
        strcpy(E[n].name.firstname,firstName);
        strcpy(E[n].name.middlename,middleName);
        strcpy(E[n].name.lastname,lastName);
        E[n].EmpId = empId;
        E[n].Salary = salary;
    }
    else{
        E = E+n;
        E = (Employee*)malloc((n+1)*sizeof(Employee));
        (E+n)->name.firstname =
(char*)malloc(strlen(firstName)*sizeof(char));
        (E+n)->name.middlename =
(char*)malloc(strlen(middleName)*sizeof(char));
        (E+n)->name.lastname =
(char*)malloc(strlen(lastName)*sizeof(char));
        strcpy((E+n)->name.firstname,firstName);
        strcpy((E+n)->name.middlename,middleName);
        strcpy((E+n)->name.lastname,lastName);
        (E+n)->EmpId = empId;
        (E+n)->Salary = salary;
    }
}

bool search(Employee E[], int n, int empId) {
    /*
    n - Length of Employee array
    Search for employee with empId in array of employees E
    Return true if found, else false
    */
    for(int i=0;i<n;i++){
        if((E+i)->EmpId == empId){
            return true;
        }
    }
    return false;
}
int main() {
    int q, t, n = 0;
    Employee E[10];
    int empId;
    float salary;
    char firstName[30], middleName[30], lastName[30];

    scanf("%d", &q);
    while (q--) {
        scanf("%d", &t);

        if (t == 1) {
            scanf("%d%f%s%s%s", &empId, &salary, firstName,
middleName, lastName);
            addEmployee(E, n, firstName, middleName, lastName,
empId, salary);
            n += 1;
        } else {
            scanf("%d", &empId);
            printf("%d\n", search(E, n, empId));
        }
    }
}
```

## 5.4 Sample Output



5. Employee Details

Create a structure Employee with fields EmpId, Name and Salary. Name should contain first name, middle name and last name. Store the details of n employees. Write a function to implement Linear Search to search for a particular employee, given the EmpId. Create a structure Employee with fields EmpId, Name and Salary. Name should contain first name, middle name and last name. Store the details of n employees, dynamically allocating memory for the same. Write a function to implement Linear Search to search for a particular employee, given the EmpId.

```

#include<string.h>
#include<stdlib.h>

typedef struct {
    /*
    Structure for Name
    */
    char *firstname;
    char *middlename;
    char *lastname;
```

Compiled successfully. All available test cases passed

Test case 0

Test case 1

Your Output (stdout)

```
1
```

Hidden Test Case

Use print or log statements to debug why your hidden test cases are failing. Hidden test cases are used to evaluate if your code can handle different scenarios, including corner cases.

## 5.5 Result

Program submitted and executed successfully in HackerRank Platform via user id @rahulmanoj

# 6. Infix to Postfix

## 6.1 Problem

Using stack do the following:

- Convert an infix expression to a postfix expression
- Evaluate the postfix expression

## 6.2 Algorithm

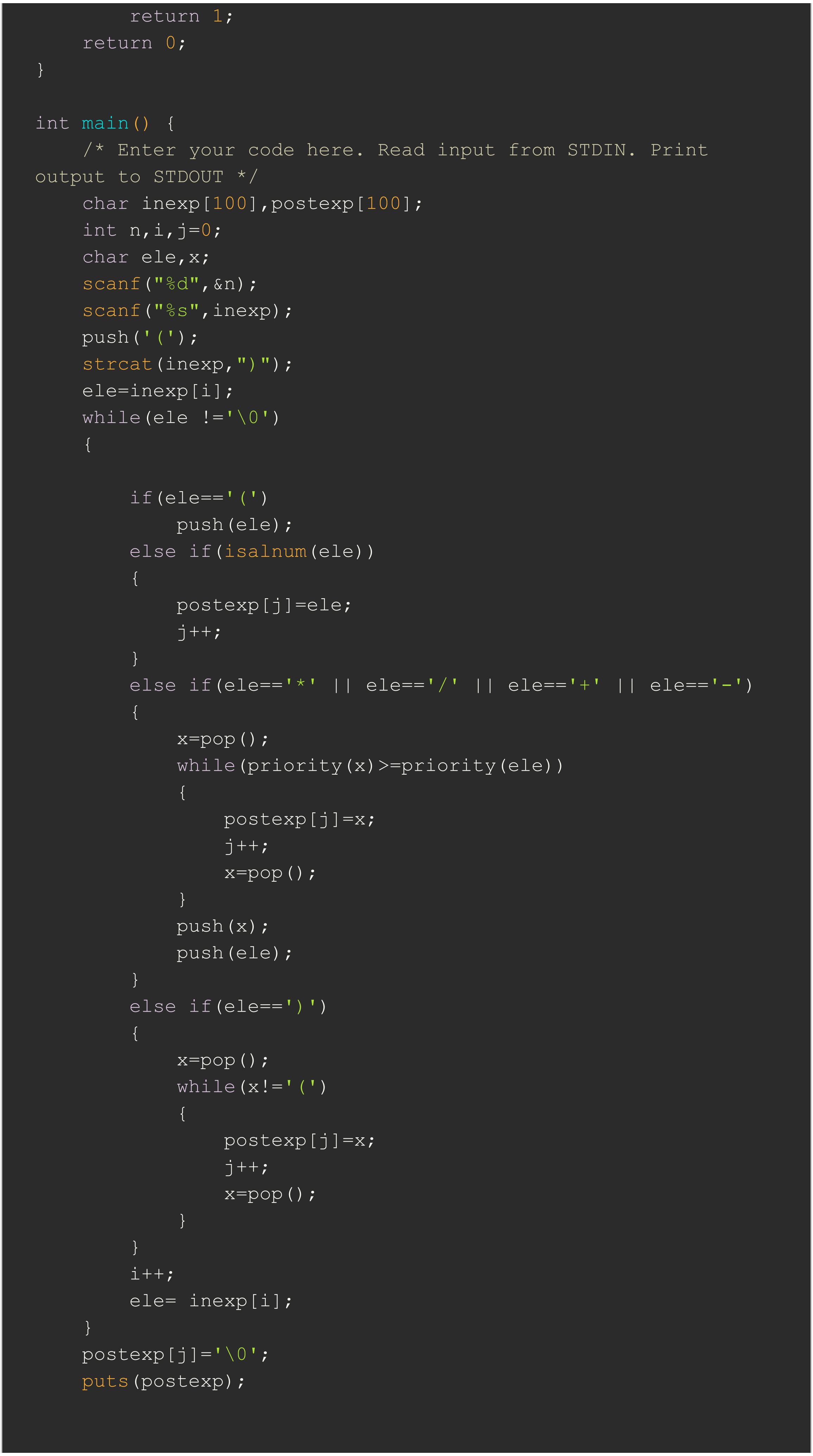
```
Start of Struct stack
char arr[10]            {the capacity of the stack is 10}
int top             {the top element of the stack}
End of Struct stack
Stack s ={.top =  -1}           {initialise the top of the stack with -1}
Start of main function
input len           {the length of the expression}
input exp           {the expression}
infixtopostfix(exp,len)     {function call}
End of main function
Start of function  infixtopostfix(exp,len)      {exp ,len are arguments}
for i <-- 0 to len do
    if isOperand(exp[i]) is true then       {function call}
        res[k] <-- exp[i]
        increment k
    Endif
    else if exp[i] is equal to '(' then
        push(exp[i])            {function call}
    Endelseif
    else if exp[i] is equal to ')' then
        while !isEmpty() and peak() != '(' do       {function call}
            res[k] = pop()      {function call}
            increment k
        Endwhile
        if !isEmpty() and peak() != '(' then        {function call}
            return                  {return void}
        Endif
        else
            pop()                   {function call}
        Endelse
    Endelseif
    else
        while !isEmpty() and position(exp[i]) <= position(peak()) do  {function call}
            res[k] <-- pop()            {function call}
            increment k
        Endwhile
        push(exp[i])                {function call}
    Endelse
```

```
Endfor
while !isEmpty()                    {function call}
    res[k] <-- pop()                    {function call}
    increment k
Endwhile
res[k] <-- '\0'
print res
End of function infixtopostfix
Start of function push(c)       {c is the argument}
if s.top == 9 then          {if the stack is full}
    return              {return void}
Endif
else if s.top equal to -1
    increment s.top
    s.arr[s.top] <-- c
Endelseif
else
    increment s.top
    s.arr[s.top] <-- c
Endelse
End of push function
Start of function pop
if s.top equal to  -1           {if the stack is empty}
    return -1
Endif
else
    temp <-- s.arr[s.top]
    decrement s.top
Endelse
return temp                 {return the popped element}
End of pop function
Start of function peak()
return s.arr[s.top]         {return the top element of the
stack}
End of function peak
Start of function isEmpty()
if s.top == -1 then             {if the stack is empty}
    return 1
Endif
return 0
End of isEmpty function
Start of function isOperand(ch)     {ch is the argument}
if ch >= 'a' and ch <= 'z' or ch >= 'A' and ch <= 'Z'   then
{check if the character is an alphabet}
    return 1
Endif
return 0
End of isOperand function
Start of function position(ch)          {ch is the argument}
if ch equal to '+' or ch equal to '-' then
    return 1
Endif
else if ch equal to '*' or ch equal to '/' then
    return 2
```

```
Endelseif
else if ch equal to  '^'  then
    return 3
Endelseif
return -1
End of position function
```

## 6.3 Code

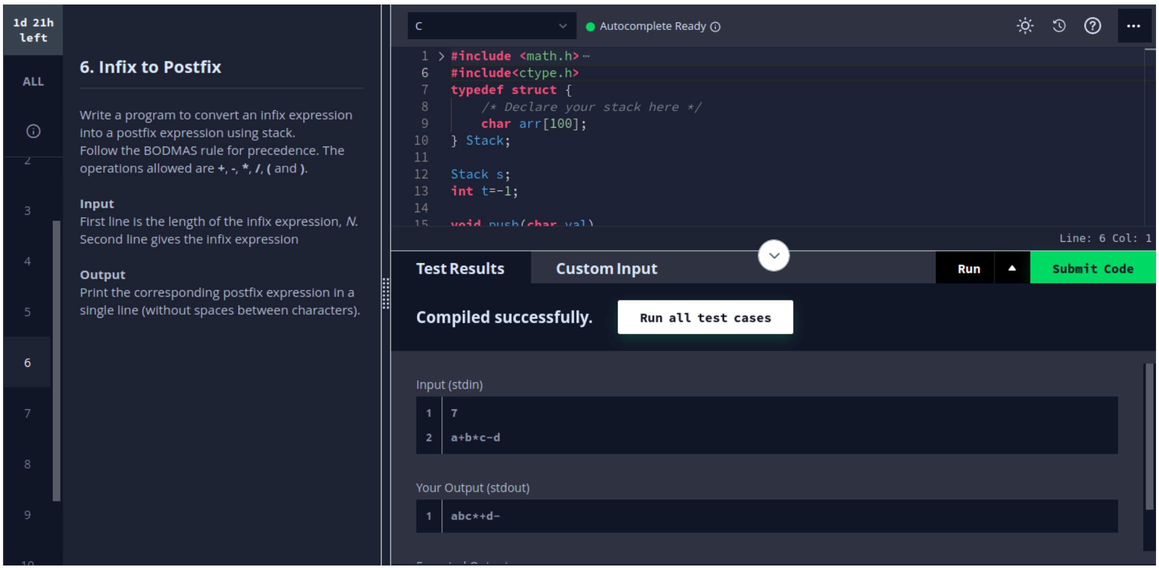
```c
#include <math.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#include<ctype.h>
typedef struct {
    /* Declare your stack here */
    char arr[100];
} Stack;

Stack s;
int t=-1;

void push(char val)
{
    if(t!=99)
    {
        t++;
        s.arr[t]=val;
    }
}

char pop()
{
    char ele;
    if(t<0)
        return -1;
    else
    {
        ele=s.arr[t];
        t--;
    }
    return ele;
}

int priority(char x)
{
    if(x=='*' || x=='/')
        return 2;
    if(x=='+' || x=='-')
```



```
        return 1;
    return 0;
}

int main() {
    /* Enter your code here. Read input from STDIN. Print 
output to STDOUT */
    char inexp[100],postexp[100];
    int n,i,j=0;
    char ele,x;
    scanf("%d",&n);
    scanf("%s",inexp);
    push('(');
    strcat(inexp,")");
    ele=inexp[i];
    while(ele !='\0')
    {

        if(ele=='(')
            push(ele);
        else if(isalnum(ele))
        {
            postexp[j]=ele;
            j++;
        }
        else if(ele=='*' || ele=='/' || ele=='+' || ele=='-')
        {
            x=pop();
            while(priority(x)>=priority(ele))
            {
                postexp[j]=x;
                j++;
                x=pop();
            }
            push(x);
            push(ele);
        }
        else if(ele==')')
        {
            x=pop();
            while(x!='(')
            {
                postexp[j]=x;
                j++;
                x=pop();
            }
        }
        i++;
        ele= inexp[i];
    }
    postexp[j]='\0';
    puts(postexp);
```

```
        return 0;
    }
```

## 6.4 Sample Output



6. Infix to Postfix

Write a program to convert an infix expression into a postfix expression using stack.
Follow the BODMAS rule for precedence. The operations allowed are **+**, **-**, *****, **/**, **(** and **)**.

**Input**
First line is the length of the infix expression, *N*.
Second line gives the infix expression

**Output**
Print the corresponding postfix expression in a single line (without spaces between characters).

```
#include <math.h>
#include<ctype.h>
typedef struct {
    /* Declare your stack here */
    char arr[100];
} Stack;

Stack s;
int t=-1;

void push(char val)
```

Line: 6 Col: 1

Compiled successfully.

Input (stdin)

```
7
a+b*c-d
```

Your Output (stdout)

```
abc*+d-
```

## 6.5 Result

Program submitted and executed successfully in HackerRank Platform via user id @rahulmanoj

---

# 7. Postfix Evaluation

## 7.1 Problem

Using stack do the following:

- Convert an infix expression to a postfix expression
- Evaluate the postfix expression

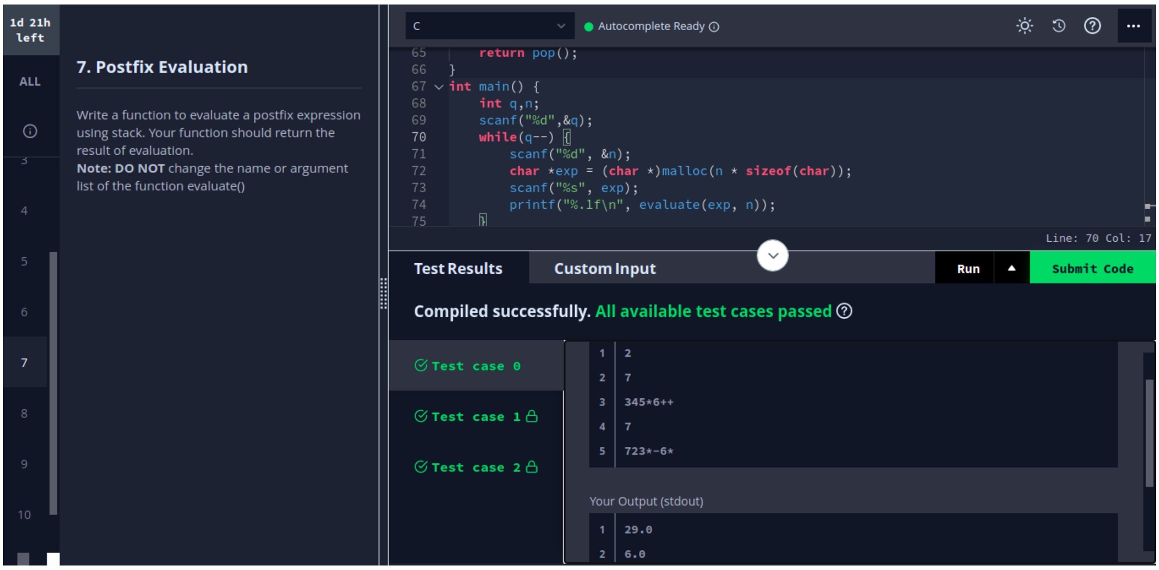
## 7.2 Algorithm

```
Start of Struct stack
char arr[10]            {the capacity of the stack is 10}
int top             {the top element of the stack}
End of Struct stack
Stack s ={.top =  -1}            {initialise the top of the
stack with -1}
Start of  main function
input q                      {the number of Queries}
while q > 0 do
    input n             {length of the expression}
    *exp = (char *)malloc(n * sizeof(char))    {dynamic memory
allocation}
    input exp           { the expression }
    print evaluate(exp,n)       {function call}
Endwhile
return 0
End of main function
Start of function evaluate(expression,len)  {expression,len
are arguments}
for i <-- 0 to len do
    if isdigit(expression[i])           {function call}
        push(expression[i] - '0')       {function call}
    Endif
    else
        a <-- pop()         {function call}
        b <-- pop()         {function call}
        switch expression[i]
            case '+'        {addition}
                push(b+a)   {function call}
                break
            case '-'        {subtraction}
                push(b-a)   function call
                break
            case '*'
                push(b*a)   function call
                break
            case '/'
                push(b/a)
                break
        End switch
    End else
Endfor
End of evaluate function
Start of function push(n)           {n is the argument}
if s.top Equal to 9  then               {if the stack to full}
    return                  {return void}
Endif
else
    increment s.top
    s.arr[s.top] <-- n
Endelse
End of push function
Start of function pop()
if s.top equal to  -1           {if the stack is empty}
    return -1
Endif
else
    temp <-- s.arr[s.top]
    decrement s.top
    return temp         {return the popped element}
Endelse
End of pop function
```

## 7.3 Code

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <stdbool.h>
#include <ctype.h>

typedef struct {
    /* Declare the stack */
    float arr[10];
    int t;
} Stack;
Stack s={.t = -1};

/*
Complete the evaluate function which takes the postfix
expression
and the length of expression as arguments and returns the
result.
*/
void push(float n){
    if(s.t == 9){
        return;
    }
    else{
        s.t++;
        s.arr[s.t] = n;
    }
}

float pop(){
    if(s.t == -1){
        return -1;
    }
    else{
        float a = s.arr[s.t];
        s.t--;
        return a;
    }

}

float evaluate(char expression[], int len) {
    for(int i=0;i<len;i++){
        if(isdigit(expression[i])){
            push(expression[i] - '0');
        }
        else{
            float x=pop();
            float y=pop();
            switch(expression[i]){
            case '+':
                push(y+x);
                break;
            case '-':
                push(y-x);
                break;
            case '*':
                push(y*x);
                break;
            case '/':
                push(y/x);
                break;
            }
        }
    }
    return pop();
}
int main() {
```

## 7.4 Sample Output



**7. Postfix Evaluation**

Write a function to evaluate a postfix expression using stack. Your function should return the result of evaluation.
**Note: DO NOT** change the name or argument list of the function evaluate()

```
    return pop();
}
int main() {
    int q,n;
    scanf("%d",&q);
    while(q--) {
        scanf("%d", &n);
        char *exp = (char *)malloc(n * sizeof(char));
        scanf("%s", exp);
        printf("%.1f\n", evaluate(exp, n));
    }
```

Compiled successfully. All available test cases passed

✓ Test case 0

✓ Test case 1

✓ Test case 2

```
2
7
345*6++
7
723*-6*
```

Your Output (stdout)

```
29.0
6.0
```

## 7.5 Result

Program submitted and executed successfully in HackerRank Platform via user id @rahulmanoj

# 8. Sort and Search Strings

## 8.1 Problem

Write a program to read string data stored in a file. Sort the strings in alphabetical order.Implement Binary Search to search for a given string.Implement sort and search routines as separate functions.

## 8.2 Algorithm

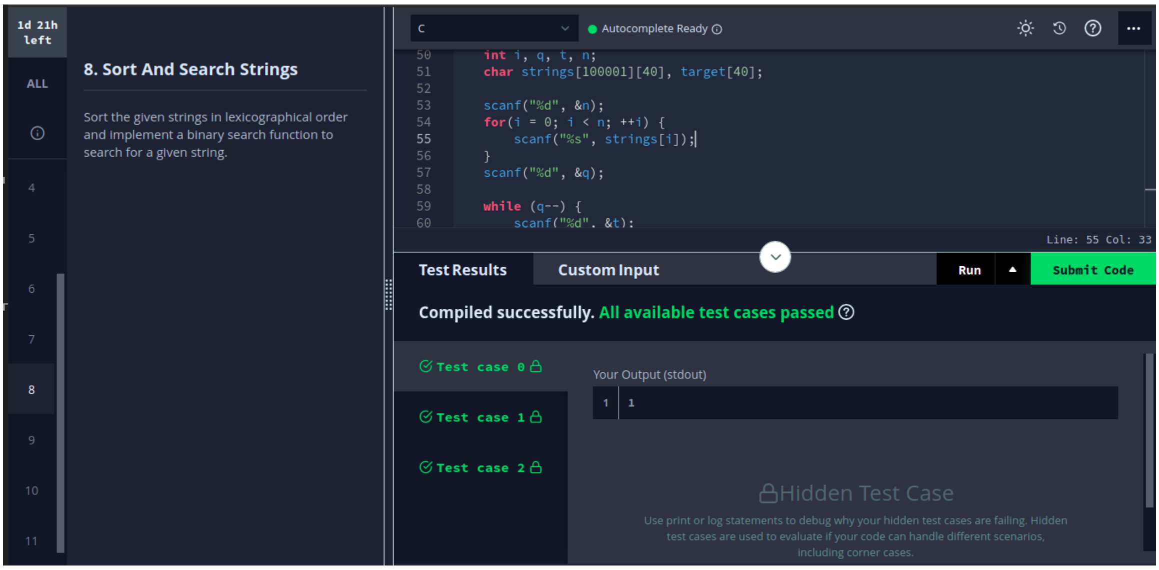
```
Start of main function
input n                     {n is the number of strings}
input all the strings to the array strings
input q                         {number of Queries}
while q > 0 do
    if t is equal to 1   then
        sort(strings,n)         {function call}
    Endif
    else
        input target
        print search(strings,n,target)  {function call}
    Endelse
End of main function
Start of function sort(strings,n)           {strings and n are
arguments}
for i <-- 0 to n-1 do
    for i <-- i+1 to n do
        if strcmp(strings[j-1],strings[j]>0) then      {check
whether the strings are equal}
            {swap strings string[j-1] and strings[j]}
            strcpy(str,strings[j-1])
                        strcpy(strings[j-1],strings[j])
                        strcpy(strings[j],str)
        Endif
    Endfor
Endfor
End of function sort
Start of function search(strings,n,target)
start <-- 0
end <-- n
while start <= end do
    mid <-- (start + end)/2
    if(strcmp(target,strings[mid])==0)  then    {if the string
is found}
        return true
    Endif
    else if(strcmp(target,strings[mid])<0)  then {if target
string is smaller than the string}
        end  = mid - 1          {we neglet the middle to end
portion}
    Endelseif
    else if(strcmp(target,strings[mid])>0) then     {if target
is larger than the string}
        start = mid + 1             {we neglet the front to
middle portion}
    Endelseif
Endwhile
return false
```

```
End of search function
```

## 8.3 Code

```c
#include <stdio.h>
#include <stdbool.h>
#include <string.h>

void sort(char strings[][40], int n) {
    /*
    Sort the given array of strings
    n - Number of strings

    NOTE: strings dimensions are n x 30 (strings[n][30])
    */
    for(int i=0;i<n-1;i++){
        for(int j=i+1;j<n;j++){
            if(strcmp(strings[j-1],strings[j])>0){
                char str[40];
                strcpy(str,strings[j-1]);
                strcpy(strings[j-1],strings[j]);
                strcpy(strings[j],str);
            }
        }
    }
}

bool search(char strings[][40], int n, char target[40]) {
    /*
    Binary Search for target string in strings array
    Return true if found, else false

    NOTE: strings array here can be assumed as sorted
    */
    int s = 0;
    int e = n;
    while(s <= e){
        int m = (s + e)/2;
        if(strcmp(target,strings[m])==0){
            return true;
        }
        else if(strcmp(target,strings[m])<0){
            e = m-1;
        }
        else if(strcmp(target,strings[m])>0){
            s = m + 1;
        }

    }
    return false;
}
int main() {
    int i, q, t, n;
    char strings[100001][40], target[40];

    scanf("%d", &n);
    for(i = 0; i < n; ++i) {
        scanf("%s", strings[i]);
    }
    scanf("%d", &q);

    while (q--) {
        scanf("%d", &t);
        if (t == 1) {
            sort(strings, n);
        } else {
            scanf("%s", target);
            printf("%d \n", search(strings, n, target));
        }
    }
}
```

## 8.4 Sample Output



8. Sort And Search Strings

Sort the given strings in lexicographical order and implement a binary search function to search for a given string.

```c
    int i, q, t, n;
    char strings[100001][40], target[40];

    scanf("%d", &n);
    for(i = 0; i < n; ++i) {
        scanf("%s", strings[i]);
    }
    scanf("%d", &q);

    while (q--) {
        scanf("%d", &t);
```

Compiled successfully. All available test cases passed

Test case 0
Test case 1
Test case 2

Your Output (stdout)

```
1
```

Hidden Test Case

Use print or log statements to debug why your hidden test cases are failing. Hidden test cases are used to evaluate if your code can handle different scenarios, including corner cases.

## 8.5 Result

Program submitted and executed successfully in HackerRank Platform via user id @rahulmanoj

# 9. Priority Queue

## 9.1 Problem

Implement a Priority Queue using arrays with the operations:

- Insert elements to the Priority Queue.
- Delete elements from the Priority Queue.

## 9.2 Algorithm

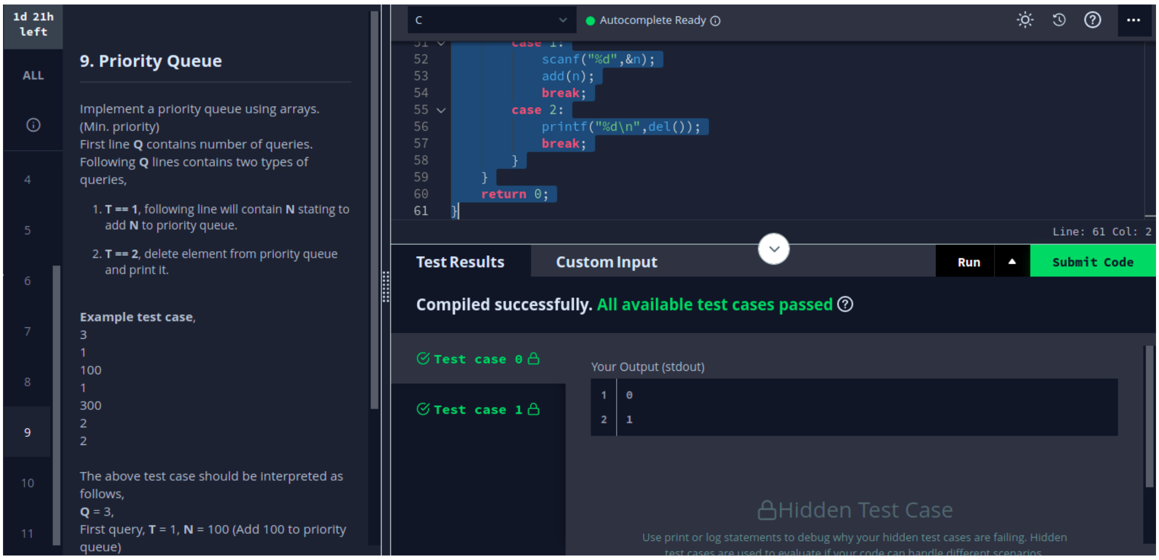
```
Start of struct Queue
int arr[10]             {10 is the capacity of the queue}
int rear                    {this is the rear element of the
queue}
int front               {this is the front element of the
queue}
End of struct Queue
Queue q={.rear = -1 , .front = -1}
Start of main function
input q                 {the number of queries}
while q > 0 do
    input  T                {T is the choice}
    switch(T)
        case 1: input n     {n is the element to be inserted}
            add(n)      {function call}
            break
        case 2: print  del()    {function call}
            break
    Endswitch
Endwhile
return 0
End of main function
Start of function add(n)            {n is the argument}
if q.front equal to -1 and q.rear equal to -1 then
    increment q.front
    increment q.rear
    q.arr[q.rear] <-- n
Endif
else if q.rear - q.front < 9 then       {if the queue is not
empty}
    increment q.rear
    for i <-- q.front to q.rear
        if q.arr[i] > n then
            for j <-- q.rear j>i decrement j    {shifting
elements}
                q.arr[j] <-- q.arr[j-1]
            Endfor
        q.arr[i] <-- n
        return
    Endif
Endfor
q.arr[q.rear] <-- n
Endelseif
End of add function
start of function del()
```

```
if q.front not equal to -1 and q.front <= q.rear then
{if the queue is empty}
    temp <-- q.arr[q.front]
    increment q.front
    return temp
Endif
else
    return -1
Endelse
End of function del
```

## 9.3 Code

```c
#include<stdio.h>

typedef struct{
    int arr[10];
    int f;
    int r;
}Queue;

Queue q={.f = -1,.r = -1};
void add(int n){
    if(q.f == -1 && q.r == -1){
        q.f++;
        q.r++;
        q.arr[q.r] = n;
    }
    else if(q.r-q.f<9){
        int i;
        q.r++;
        for(i=q.f;i<q.r;i++){
            if(q.arr[i]>n){
                for(int j=q.r;j>i;j--){
                    q.arr[j] = q.arr[j-1];
                }
                q.arr[i] = n;
                return;
            }
        }
        q.arr[q.r] = n;
    }
}

int del(){
    if(q.f != -1 && q.f <= q.r){
        int t = q.arr[q.f];
        q.f++;
        return t;
    }
    else{
        return -1;
    }
}

int main(){
    int Q;
    int T;
    int n;
    scanf("%d",&Q);
    while(Q--){
        scanf("%d",&T);
        switch(T){
        case 1:
            scanf("%d",&n);
            add(n);
            break;
        case 2:
            printf("%d\n",del());
            break;
        }
    }
    return 0;
}
```

## 9.4 Sample Output



**9. Priority Queue**

Implement a priority queue using arrays. (Min. priority)
First line **Q** contains number of queries.
Following **Q** lines contains two types of queries,

1. **T == 1**, following line will contain **N** stating to add **N** to priority queue.
2. **T == 2**, delete element from priority queue and print it.

**Example test case**,
3
1
100
1
300
2
2

The above test case should be interpreted as follows,
**Q** = 3,
First query, **T** = 1, **N** = 100 (Add 100 to priority queue)

```
            case 1:
                scanf("%d",&n);
                add(n);
                break;
            case 2:
                printf("%d\n",del());
                break;
            }
        }
        return 0;
    }
```

Compiled successfully. All available test cases passed

✓ Test case 0

✓ Test case 1

Your Output (stdout)

```
0
1
```

Hidden Test Case

Use print or log statements to debug why your hidden test cases are failing. Hidden test cases are used to evaluate if your code can handle different scenarios

## 9.5 Result

Program submitted and executed successfully in HackerRank Platform via user id @rahulmanoj

# 10. Circular Queue

## 10.1 Problem

Implement a circular queue using arrays with the operations:

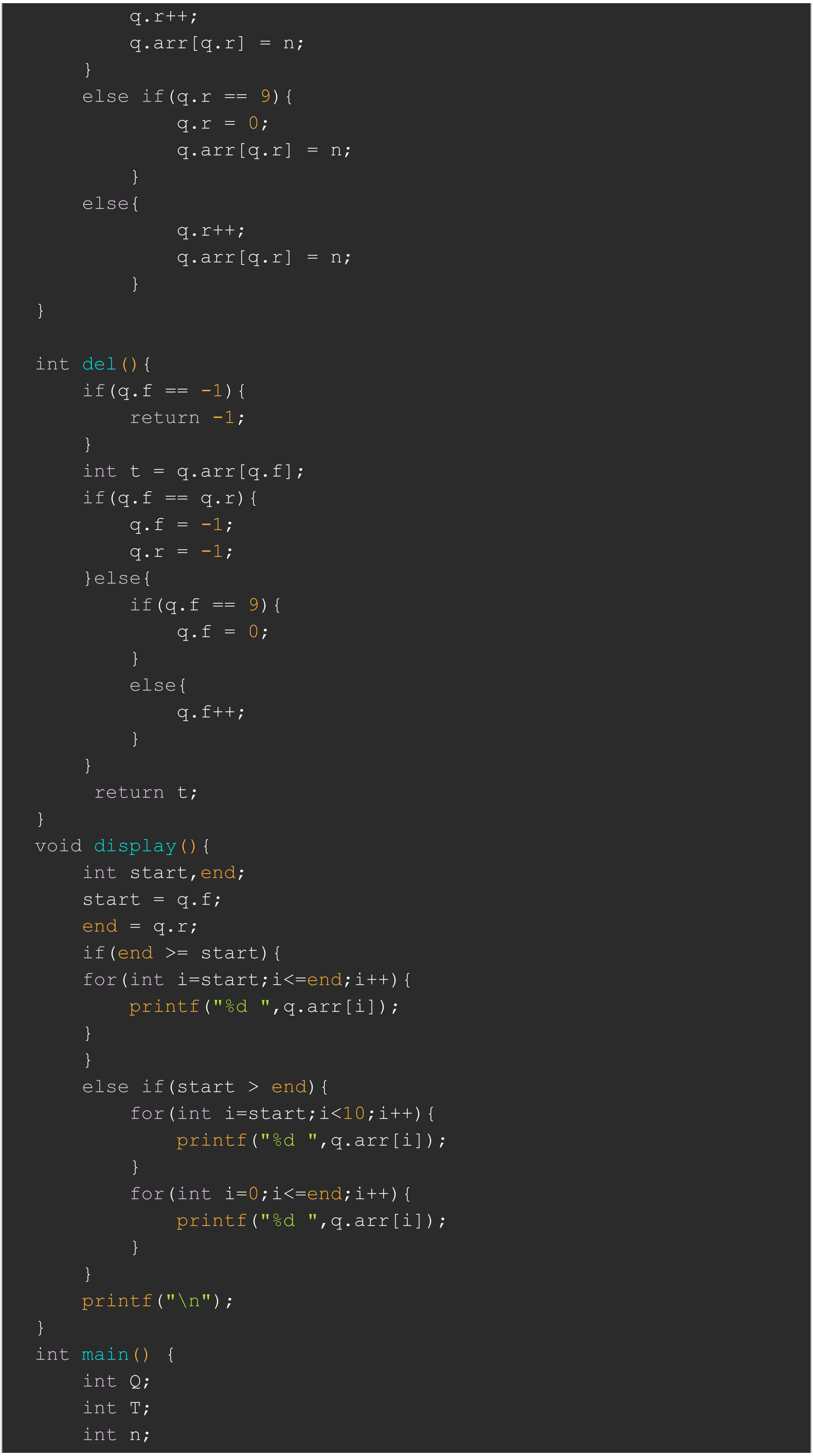
- Insert an element to the queue.
- Delete an element from the queue.
- Display the contents of the queue after each operation.

## 10.2 Algorithm

```
Start of struct Queue
int arr[10]             {10 is the capacity of the queue}
int rear                    {this is the rear element of the
queue}
int front               {this is the front element of the
queue}
End of struct Queue
Queue q={.rear = -1 , .front = -1}
Start of main function
input Q                 {the number of queries}
while Q > 0 do
    input  T            {the choice}
    switch(T)
        case 1: intput n
            insert(n)       {function call}
            break
        case 2: print del()     {function call}
            break
        case 3: display()       {function call}
            break
    Endswitch
Endwhile
return 0
End of main function
Start of function insert(n)     {n is the argument}
if (q.rear+1)%n equal to q.front then   {if the queue is full}
    return
Endif
else if q.front equal to -1 and q.rear equal to -1 then
    increment q.front
    increment q.rear
    q.arr[q.rear] <-- n
Endelseif
else
    increment q.rear
    q.arr[q.rear] <-- n
Endelse
End of insert function
Start of the function del()
if q.front equal to -1 then
    return -1
Endif
temp <-- q.arr[q.front]
if q.front equal to q.rear then             {the last element
in the queue}
    q.front <-- -1
    q.rear <-- -1
Endif
else
    if q.front equal to 9 then
        q.front = 0
    Endif
    else
        increment q.front
    Endelse
Endelse
return temp
End of del function
Start of function display()
start <-- q.front
end <-- q.rear
if end >= start then
    for i <-- start to end do
        print q.arr[i]
    Endfor
Endif
else if start > end then
    for i <-- start to 10  do
        print q.arr[i]
    Endfor
    for i <-- 0 to end do
        print q.arr[i]
    Endfor
Endelseif
print '\n'
End of display function
```

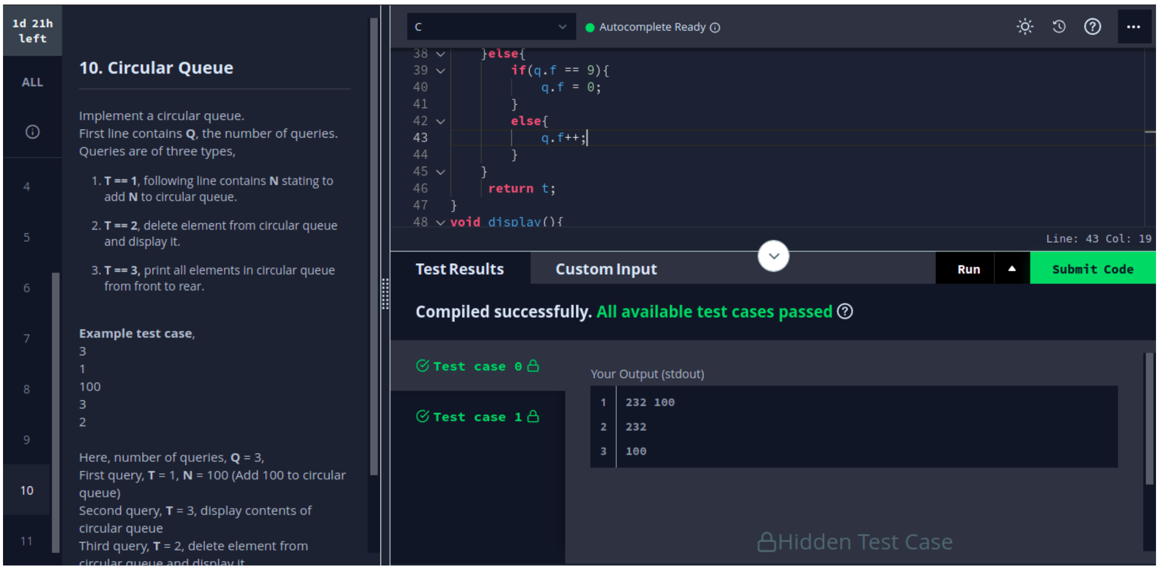
## 10.3 Code

```
#include <stdio.h>

typedef struct{
    int arr[10];
    int f;
    int r;
}Queue;

Queue q = {.f = -1,.r = -1};

void insert(int n){
    if((q.f == 0 && q.r == 9) || (q.f == q.r + 1)){
        return;
    }
    else if(q.f == -1 && q.r == -1){
        q.f++;
```



```
        q.r++;
        q.arr[q.r] = n;
    }
    else if(q.r == 9){
            q.r = 0;
            q.arr[q.r] = n;
        }
    else{
            q.r++;
            q.arr[q.r] = n;
        }
}

int del(){
    if(q.f == -1){
        return -1;
    }
    int t = q.arr[q.f];
    if(q.f == q.r){
        q.f = -1;
        q.r = -1;
    }else{
        if(q.f == 9){
            q.f = 0;
        }
        else{
            q.f++;
        }
    }
     return t;
}
void display(){
    int start,end;
    start = q.f;
    end = q.r;
    if(end >= start){
    for(int i=start;i<=end;i++){
        printf("%d ",q.arr[i]);
    }
    }
    else if(start > end){
        for(int i=start;i<10;i++){
            printf("%d ",q.arr[i]);
        }
        for(int i=0;i<=end;i++){
            printf("%d ",q.arr[i]);
        }
    }
    printf("\n");
}
int main() {
    int Q;
    int T;
    int n;
    scanf("%d",&Q);
    while(Q--){
        scanf("%d",&T);
        switch(T){
        case 1:
            scanf("%d",&n);
            insert(n);
            break;
        case 2:
            printf("%d\n",del());
            break;
        case 3:
            display();
            break;
        }
    }
    /* Enter your code here. Read input from STDIN. Print
output to STDOUT */
    return 0;
}
```

## 10.4 Sample Output



**10. Circular Queue**

Implement a circular queue.
First line contains **Q**, the number of queries.
Queries are of three types,

1. **T == 1**, following line contains **N** stating to add **N** to circular queue.
2. **T == 2**, delete element from circular queue and display it.
3. **T == 3,** print all elements in circular queue from front to rear.

**Example test case**,
3
1
100
3
2

Here, number of queries, **Q** = 3,
First query, **T** = 1, **N** = 100 (Add 100 to circular queue)
Second query, **T** = 3, display contents of circular queue
Third query, **T** = 2, delete element from circular queue and display it

```
    }else{
        if(q.f == 9){
            q.f = 0;
        }
        else{
            q.f++;
        }
    }
    return t;
}
void display(){
```

Compiled successfully. All available test cases passed

✓ Test case 0

✓ Test case 1

Your Output (stdout)

```
232 100
232
100
```

## 10.5 Result

Program submitted and executed successfully in HackerRank Platform via user id @rahulmanoj

# 11. Double Ended Queue

## 11.1 Problem

Implement a Double-Ended Queue (DEQUEUE) with the operations:

- Insert elements to the Front of the queue.
- Insert elements to the Rear of the queue
- Delete elements from the Front of the queue.
- Delete elements from the Rear of the que

## 11.2 Algorithm

```
Start of struct Queue
int arr[10]             {10 is the capacity of the queue}
int rear                    {this is the rear element of the
queue}
int front               {this is the front element of the
queue}
End of struct Queue
Queue q={.rear = -1 , .front = -1}
Start of main function
input Q                 {the number of queries}
while Q > 0 do
    input T                 {choice of operation}
    switch(T)
        case 1: input n
            insertfront(n)      {function call}
            break
        case 2: input n
            insertrear(n)       {function call}
            break
        case 3: print delfront()
            break
        case 4: print delrear()
            break
        case 5: traverse()
            break
    Endswitch
Endwhile
return 0
End of main function
Start of function insertrear(n)         {n is the argument}
if (q.rear + 1)%n equal to q.front then
    return
Endif
else if q.front equal to -1 then
    q.front <-- 0
    q.rear <-- 0
    q.arr[q.rear] <-- n
Endelseif
else if q.rear equal to 9 then
    q.rear <-- 0
    q.arr[q.rear] <-- n
Endelseif
else
    increment q.rear
    q.arr[q.rear] <-- n
```

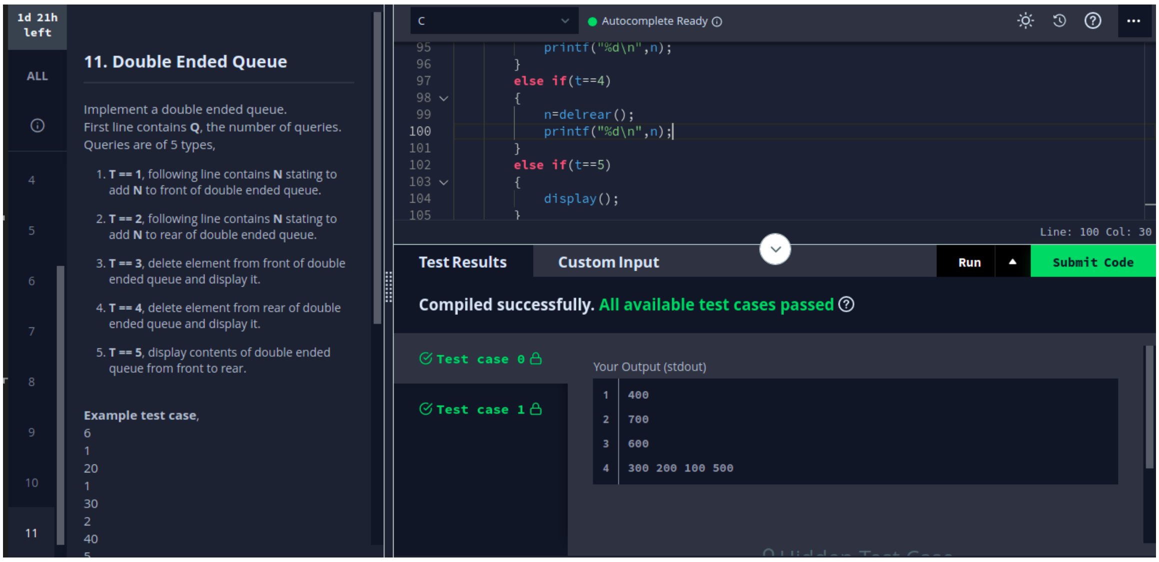
```
Endelse
End of insertrear function
Start of function insertfront(n)            {n is the
argument}
if q.rear equal to q.front + 1 then
    return
Endif
else if q.front equal to -1 then
    q.front <-- 0
    q.rear <-- 0
    q.arr[q.front] <-- n
Endelseif
else if q.front equal to 0 then
    q.front <-- 9
    q.arr[a.front] <-- n
Endelseif
else
    decrement q.front
    q.arr[a.front] <-- n
Endelse
End of insertfront function
Start of function delfront()
if q.front equal to -1 then
    return -1
Endif
else if q.front equal to q.rear then            {the last
element in the queue}
    temp <-- q.arr[q.front]
    q.front <-- -1
    q.rear <-- -1
Endelseif
else if q.front equal to 9 then
    temp <-- q.arr[q.front]
    q.front <-- 0
Endelseif
return temp
End of delfront function
Start of function delrear()
if q.rear equal to -1 then              {if the queue is
empty}
    return -1
Endif
else if q.front equal to q.rear then            {the last
element in the queue}
    temp <-- q.arr[q.rear]
    q.front <-- -1
    q.rear <-- -1
Endelseif
else if q.rear equal to 0 then
    temp <-- q.arr[q.rear]
    q.rear <-- 9
Endelseif
else
    temp <-- q.arr[q.rear]
    decrement q.rear
Endelse
return temp
End of delrear function
Start of function traverse()
start <-- q.front
end <-- q.rear
if end >= start then
    for i <-start to end do
        print q.arr[i]
    Endfor
Endif
else if start > end then
    for i < start to 10 do
        print q.arr[i]
    Endfor
    for  i <-- 0 to end do
        print q.arr[i]
    Endfor
Endelseif
print '\n'
End of traverse function
```

## 11.3 Code

```c
#include <stdio.h>
#define max 10
typedef struct{
int arr[max];
}queue;
queue q;
int f=-1, r=-1;

void addfront(int val)
{
    if(f==-1)
        f=r=0;
    else if(f==0)
        f=max-1;
    else
        f=f-1;
    q.arr[f]=val;
}

void addrear(int val)
{
    if(f==-1)
        f=r=0;
    else if(r==max-1)
        r=0;
    else
```

```
        r=r+1;
    q.arr[r]=val;
}

int delfront()
{
    int ele=q.arr[f];
    if(f==r)
        f=r=-1;
    else {
        if(f==max-1)
            f=0;
        else
            f=f+1;
    }
    return ele;
}
int delrear()
{
    int ele=q.arr[r];
    if(f==r)
        f=r=-1;
    else if (r==0) {
        r=max-1;
    }
    else {
    r=r-1;
    }
    return ele;
}
void display()
{
   int i;
   if(r>=f)
    {
        for(i=f;i<=r;i++)
            printf("%d",q.arr[i]);
        printf("\n");
    }
    else {
        for(i=f;i<max;i++)
            printf("%d ",q.arr[i]);
        for(i=0;i<=r;i++)
            printf("%d ",q.arr[i]);
        printf("\n");
    }
}

int main() {
    /* Enter your code here. Read input from STDIN. Print
output to STDOUT */
    int q,t,n;
    scanf("%d",&q);
    while(q--)
    {
        scanf("%d",&t);
        if(t==1)
        {
            scanf("%d",&n);
            addfront(n);
        }
        else if(t==2)
        {
            scanf("%d",&n);
            addrear(n);
        }
        else if(t==3)
        {
            n=delfront();
            printf("%d\n",n);
        }
        else if(t==4)
        {
            n=delrear();
            printf("%d\n",n);
        }
        else if(t==5)
        {
            display();
        }
    }
    return 0;
}
```

## 11.4 Sample Output



**11. Double Ended Queue**

Implement a double ended queue.
First line contains **Q**, the number of queries.
Queries are of 5 types,

1. **T == 1**, following line contains **N** stating to add **N** to front of double ended queue.
2. **T == 2**, following line contains **N** stating to add **N** to rear of double ended queue.
3. **T == 3**, delete element from front of double ended queue and display it.
4. **T == 4**, delete element from rear of double ended queue and display it.
5. **T == 5**, display contents of double ended queue from front to rear.

**Example test case,**
6
1
20
1
30
2
40
5

```
            printf("%d\n",n);
        }
        else if(t==4)
        {
            n=delrear();
            printf("%d\n",n);
        }
        else if(t==5)
        {
            display();
        }
```

Line: 100 Col: 30

Compiled successfully. All available test cases passed

✓ Test case 0
✓ Test case 1

Your Output (stdout)

```
400
700
600
300 200 100 500
```

## 11.5 Result

Program submitted and executed successfully in HackerRank Platform via user id @rahulmanoj