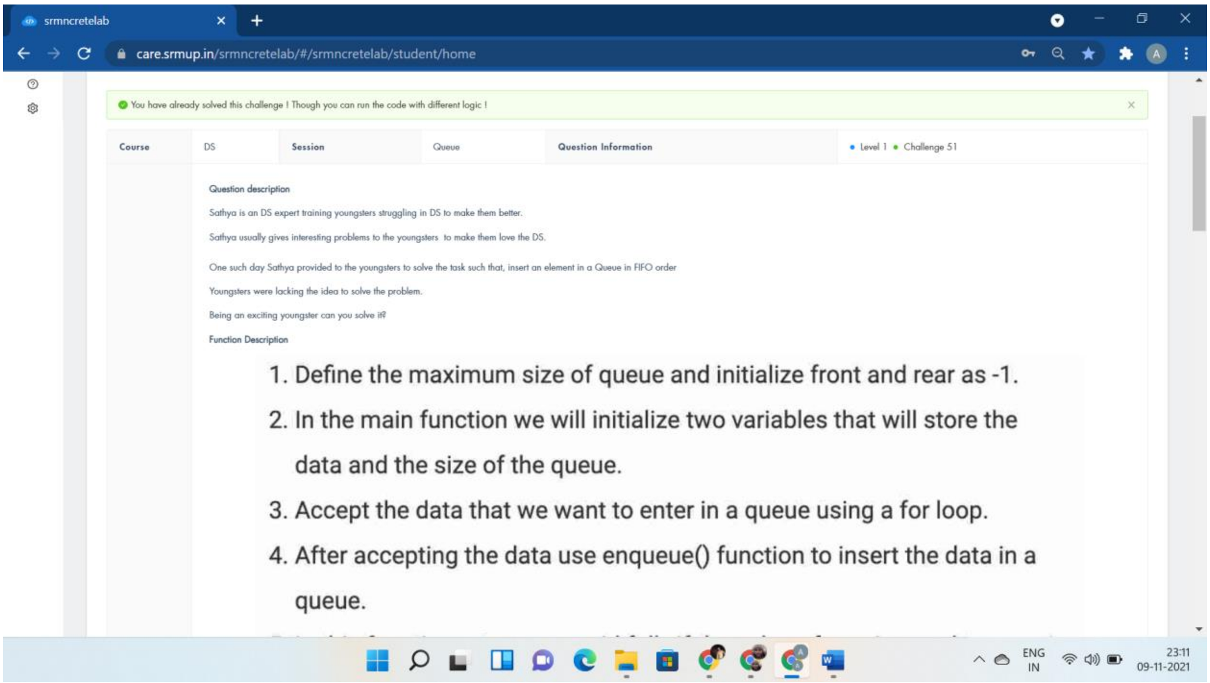
```
  cout<<getInfix(exp);


            return 0;
}
```

QUEUES:-



You have already solved this challenge ! Though you can run the code with different logic !

Course	DS	Session	Queue	Question Information	• Level 1 • Challenge 51

Question description

Sathya is an DS expert training youngsters struggling in DS to make them better.

Sathya usually gives interesting problems to the youngsters to make them love the DS.

One such day Sathya provided to the youngsters to solve the task such that, insert an element in a Queue in FIFO order

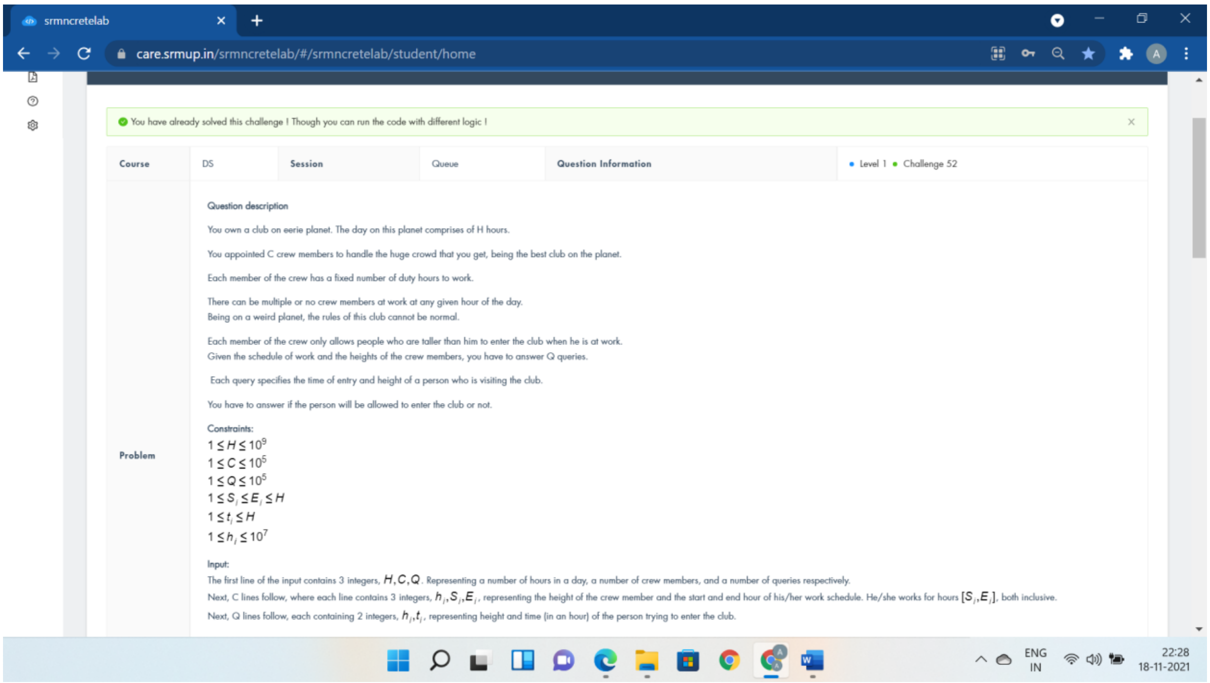
Youngsters were lacking the idea to solve the problem.

Being an exciting youngster can you solve it?

Function Description

1. Define the maximum size of queue and initialize front and rear as -1.
2. In the main function we will initialize two variables that will store the data and the size of the queue.
3. Accept the data that we want to enter in a queue using a for loop.
4. After accepting the data use enqueue() function to insert the data in a queue.

```
#include <stdio.h>
#define SIZE 100
void enqueue(int);
void display();
int items[SIZE], front = -1, rear = -1;
int main() {
  int n,data,i;
  scanf("%d",&n);
  for(i=0;i<n;i++)
  {
     scanf("%d",&data);
     enqueue(data);
     display();
  }
  return 0;
}
void enqueue(int data) {
  if (rear == SIZE - 1)
    printf("Queue is Full!!");
  else {
    if (front == -1)
      front = 0;
    rear++;
    items[rear] = data;
    printf("Enqueuing %d\n", data);
  }
}
void display() {
  if (rear == -1)
    printf("\nQueue is Empty!!!");
  else {
    int i;
    for(i=front;i<=rear;i++)
      printf("%d ", items[i]);
  }
}
```



You have already solved this challenge ! Though you can run the code with different logic !

Course	DS	Session	Queue	Question Information	• Level 1 • Challenge 52

Problem

Question description

You own a club on eerie planet. The day on this planet comprises of H hours.

You appointed C crew members to handle the huge crowd that you get, being the best club on the planet.

Each member of the crew has a fixed number of duty hours to work.

There can be multiple or no crew members at work at any given hour of the day.
Being on a weird planet, the rules of this club cannot be normal.

Each member of the crew only allows people who are taller than him to enter the club when he is at work.
Given the schedule of work and the heights of the crew members, you have to answer Q queries.

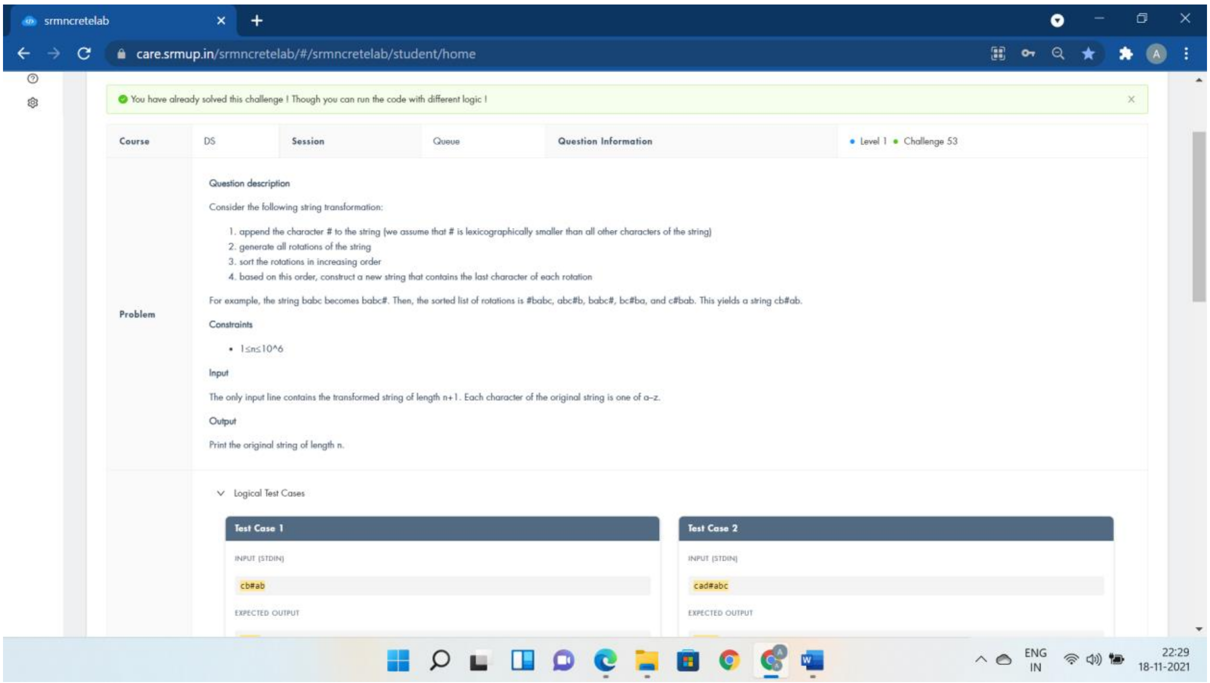
Each query specifies the time of entry and height of a person who is visiting the club.

You have to answer if the person will be allowed to enter the club or not.

Constraints:
$1 \leq H \leq 10^9$
$1 \leq C \leq 10^5$
$1 \leq Q \leq 10^5$
$1 \leq S_i \leq E_i \leq H$
$1 \leq t_i \leq H$
$1 \leq h_i \leq 10^7$

Input:
The first line of the input contains 3 integers, H,C,Q. Representing a number of hours in a day, a number of crew members, and a number of queries respectively.
Next, C lines follow, where each line contains 3 integers, h_i,S_i,E_i, representing the height of the crew member and the start and end hour of his/her work schedule. He/she works for hours $[S_i,E_i]$, both inclusive.
Next, Q lines follow, each containing 2 integers, h_i,t_i, representing height and time (in an hour) of the person trying to enter the club.

```
#include<stdio.h>

int main()

{

 long long int i,j,t,H,C,height,Q,S[100000],E[100000],h[100000];

 long long int nc=0,val=0,flag=0,maximum_height=0;

 scanf("%lld%lld%lld",&H,&C,&Q);



 for(i=0;i<C;i++)

 {

  scanf("%lld%lld%lld",&h[i],&S[i],&E[i]);

  if(h[i]>maximum_height)

   maximum_height=h[i];

 }



 for(i=0;i<Q;i++)

 {

  scanf("%lld%lld",&height,&t);

  if(height>maximum_height)

    printf("YES\n");

  else{

  val=0;

  nc=0;
  flag=0;
  for(j=0;j<C;j++)
  {
   if(t>=S[j] && t<=E[j])
   {
    nc++;
    if(height<=h[j])
    {
     printf("NO\n");
     flag=1;
     break;
    }
    else
     val++;
   }
  }

  if(nc==val)
    printf("YES\n");
  else
   if(flag==0)
    printf("NO\n");
  }


 }
 return 0;
 printf("void enqueue(long long h,long long start,long long end) while(c--)");
}
```



You have already solved this challenge ! Though you can run the code with different logic !

Course	DS	Session	Queue	Question Information	• Level 1 • Challenge 53

Problem

Question description

Consider the following string transformation:

1. append the character # to the string (we assume that # is lexicographically smaller than all other characters of the string)
2. generate all rotations of the string
3. sort the rotations in increasing order
4. based on this order, construct a new string that contains the last character of each rotation

For example, the string babc becomes babc#. Then, the sorted list of rotations is #babc, abc#b, babc#, bc#ba, and c#bab. This yields a string cb#ab.

Constraints

- 1≤n≤10^6

Input

The only input line contains the transformed string of length n+1. Each character of the original string is one of a–z.

Output

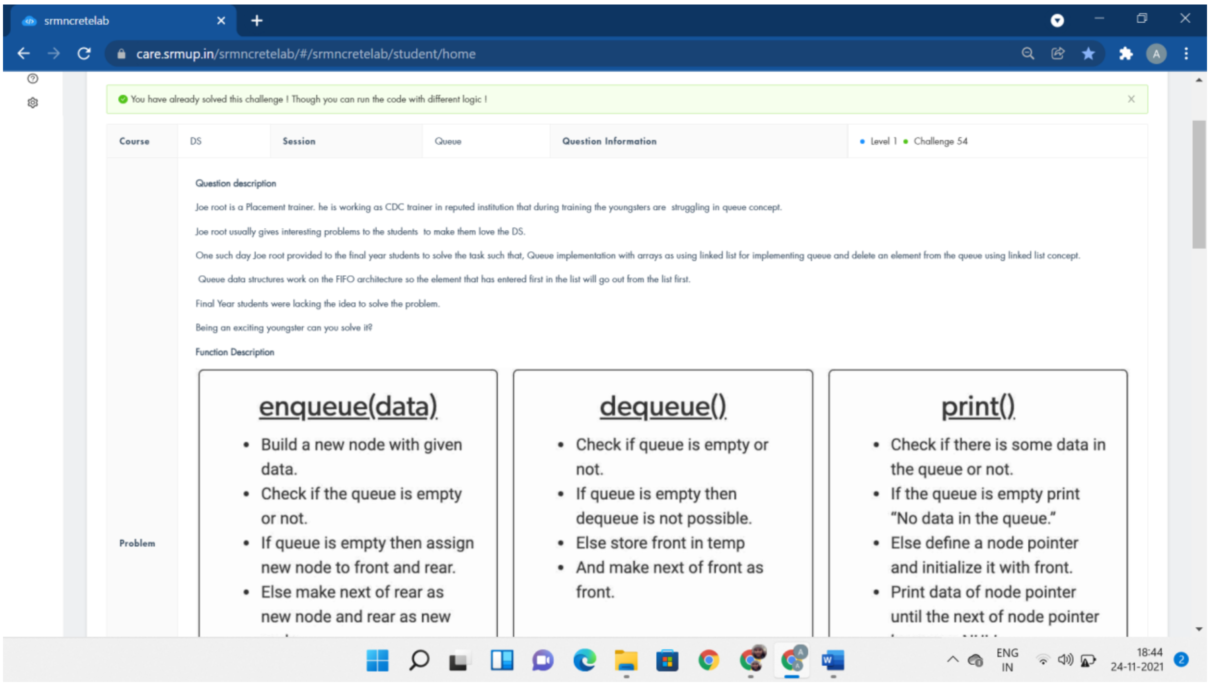
Print the original string of length n.

Logical Test Cases

Test Case 1	Test Case 2
INPUT (STDIN)	INPUT (STDIN)
cb#ab	cad#abc
EXPECTED OUTPUT	EXPECTED OUTPUT

```
#include<bits/stdc++.h>

using namespace std;



int main() {

   int i;

   string s; cin>>s;

   vector<int> v;

   vector<int> a[26];

   int n= s.size();

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

      if (s[i] == '#')

         v.push_back(i);

      else

         a[s[i]-'a'].push_back(i);

   }

   for (int i = 0; i < 26; i++) {

      for (auto j: a[i])

         v.push_back(j);

   }

   string ans;

   int j = v[v[0]];
```

```
    while(s[j] != '#') {
        ans += s[j];
        j = v[j];
    }
    cout<<ans;
    return 0;
}
```



You have already solved this challenge ! Though you can run the code with different logic !

Course	DS	Session	Queue	Question Information	Level 1 • Challenge 54

Problem

Question description

Joe root is a Placement trainer. he is working as CDC trainer in reputed institution that during training the youngsters are struggling in queue concept.

Joe root usually gives interesting problems to the students to make them love the DS.

One such day Joe root provided to the final year students to solve the task such that, Queue implementation with arrays as using linked list for implementing queue and delete an element from the queue using linked list concept.

Queue data structures work on the FIFO architecture so the element that has entered first in the list will go out from the list first.

Final Year students were lacking the idea to solve the problem.

Being an exciting youngster can you solve it?

Function Description

enqueue(data)

- Build a new node with given data.
- Check if the queue is empty or not.
- If queue is empty then assign new node to front and rear.
- Else make next of rear as new node and rear as new

dequeue()

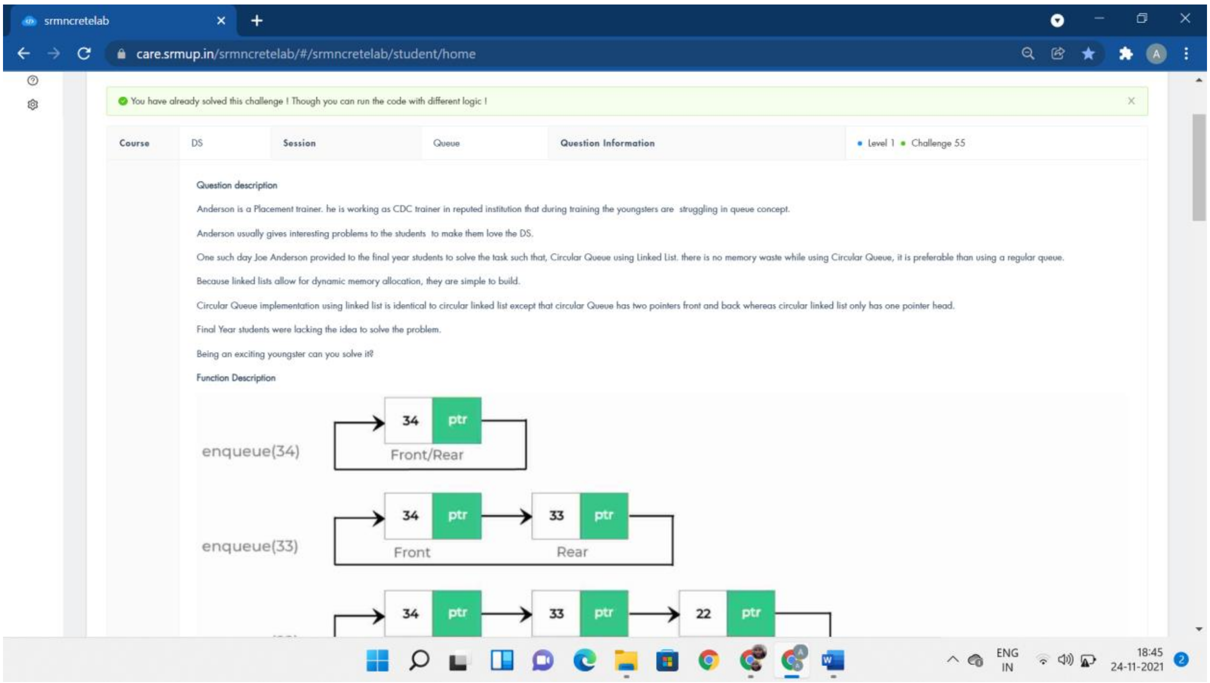
- Check if queue is empty or not.
- If queue is empty then dequeue is not possible.
- Else store front in temp
- And make next of front as front.

print()

- Check if there is some data in the queue or not.
- If the queue is empty print "No data in the queue."
- Else define a node pointer and initialize it with front.
- Print data of node pointer until the next of node pointer

```
#include <stdio.h>
#include <stdlib.h>
struct node *front = NULL;
struct node *rear = NULL;
struct node
{
    int data;
    struct node *next;
};
void linkedListTraversal(struct node *ptr)
{
    //printf("Printing the elements of this linked list\n");
    while (ptr != NULL)
    {
        printf("%d ", ptr->data);
        ptr = ptr->next;
    }
}
void enqueue(int d)
{
    struct node* new_n;
    new_n = (struct node*)malloc(sizeof(struct node));
    if(new_n==NULL){
        printf("Queue is Full");
    }
    else{
        new_n->data = d;
        new_n->next = NULL;
        if(front==NULL){
            front=rear=new_n;
        }
        else{
            rear->next = new_n;
            rear=new_n;
        }
    }
}
int dequeue()
{
    int val = -1;
    struct node *ptr = front;
    if(front==NULL){
        printf("Queue is Empty\n");
    }
    else{
        front = front->next;
        val = ptr->data;
        free(ptr);
    }
    return val;
```

```
}
int main()
{
   int n,i,t;
   scanf("%d",&n);
   for(i=0;i<n;i++)
   {
      scanf("%d",&t);
      enqueue(t);
   }
   linkedListTraversal(front);
   dequeue();
   printf("\n");
   linkedListTraversal(front);
   return 0;
}
```



You have already solved this challenge ! Though you can run the code with different logic !

Course	DS	Session	Queue	Question Information	• Level 1 • Challenge 55

Question description

Anderson is a Placement trainer. he is working as CDC trainer in reputed institution that during training the youngsters are struggling in queue concept.

Anderson usually gives interesting problems to the students to make them love the DS.

One such day Joe Anderson provided to the final year students to solve the task such that, Circular Queue using Linked List. there is no memory waste while using Circular Queue, it is preferable than using a regular queue.

Because linked lists allow for dynamic memory allocation, they are simple to build.

Circular Queue implementation using linked list is identical to circular linked list except that circular Queue has two pointers front and back whereas circular linked list only has one pointer head.

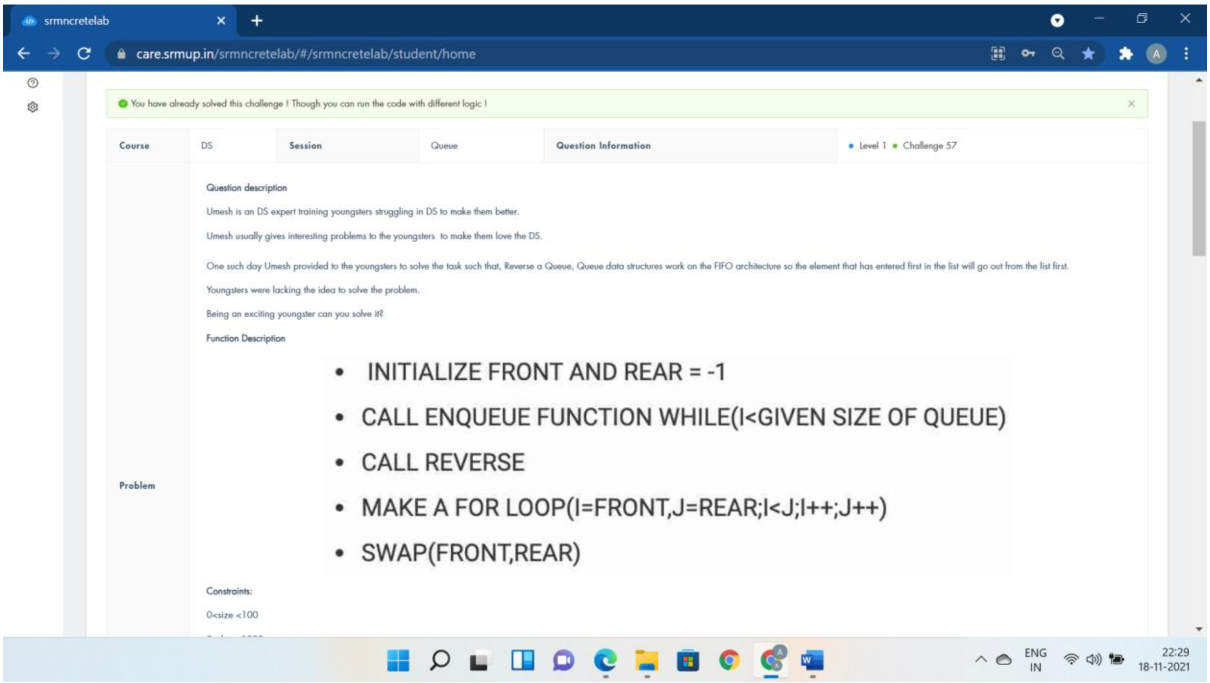
Final Year students were lacking the idea to solve the problem.

Being an exciting youngster can you solve it?

Function Description

```
#include <stdio.h>
#include <stdlib.h>
struct node *f = NULL;
struct node *r = NULL;
struct node
```

```
{
    int data;
    struct node* next;
};
void enqueue(int d)
{
    struct node *n;
    n = (struct node*)malloc(sizeof(struct node));
    if(n==NULL){
        printf("Queue is Full");
    }
    else{
        n->data = d;
        n->next = NULL;
        if(f==NULL){
            f=r=n;
        }
        else{
            r->next = n;
            r=n;
        }
    }
}
int dequeue()
{
    int val = -1;
    struct node* t;
    t = f;
    if(f==NULL){
        printf("Queue is Empty\n");
    }
    else{
        f = f->next;
        val = t->data;
        free(t);
    }
    return val;
}
int main()
{
    int n,i,t;
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        scanf("%d",&t);
        enqueue(t);
    }
    for(i=0;i<n;i++){
        printf("%d\n",dequeue());
    }
    return 0;
}
```



You have already solved this challenge ! Though you can run the code with different logic !

Course	DS	Session	Queue	Question Information	• Level 1 • Challenge 57

Problem

Question description

Umesh is an DS expert training youngsters struggling in DS to make them better.

Umesh usually gives interesting problems to the youngsters to make them love the DS.

One such day Umesh provided to the youngsters to solve the task such that, Reverse a Queue, Queue data structures work on the FIFO architecture so the element that has entered first in the list will go out from the list first.

Youngsters were lacking the idea to solve the problem.

Being an exciting youngster can you solve it?

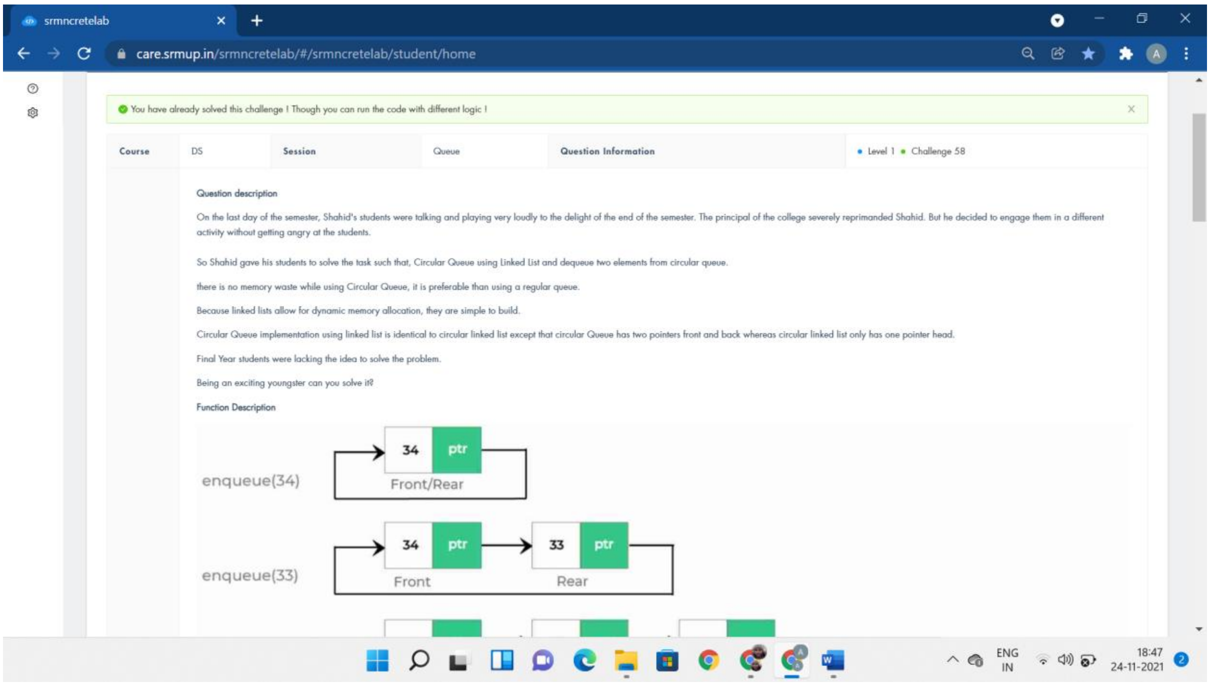
Function Description

- INITIALIZE FRONT AND REAR = -1
- CALL ENQUEUE FUNCTION WHILE(I<GIVEN SIZE OF QUEUE)
- CALL REVERSE
- MAKE A FOR LOOP(I=FRONT,J=REAR;I<J;I++;J++)
- SWAP(FRONT,REAR)

Constraints:

0<size <100

```
#include <stdio.h>
#define SIZE 100
void enqueue(int,int);
void display();
void reverse();

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

int main() {

 int n,t,i;

 scanf("%d",&n);

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

 {

    scanf("%d",&t);

    enqueue(t,n);

 }

 printf("Queue:");

 display();

 reverse();

 printf("\nReversed Queue:");

 display();

 return 0;

}

void reverse(){

   int i,j,temp;

   for(i=front,j=rear;i<j;i++,j--){

      temp=items[i];

      items[i]=items[j];

      items[j]=temp;

   }

}

void enqueue(int data,int l) {

 if (rear == l - 1)

   printf("Queue is Full!!");

 else {

   if (front == -1)

     front = 0;

   rear++;

   items[rear] = data;

  // printf("Enqueuing %d\n", data);

 }
}
void display() {
 if (rear == -1)
   printf("\nQueue is Empty!!!");
 else {
   int i;
   for(i=front;i<=rear;i++)
     printf("%d ", items[i]);
 }
}
```



You have already solved this challenge ! Though you can run the code with different logic !

Course	DS	Session	Queue	Question Information	• Level 1 • Challenge 58

Question description

On the last day of the semester, Shahid's students were talking and playing very loudly to the delight of the end of the semester. The principal of the college severely reprimanded Shahid. But he decided to engage them in a different activity without getting angry at the students.

So Shahid gave his students to solve the task such that, Circular Queue using Linked List and dequeue two elements from circular queue.

there is no memory waste while using Circular Queue, it is preferable than using a regular queue.

Because linked lists allow for dynamic memory allocation, they are simple to build.

Circular Queue implementation using linked list is identical to circular linked list except that circular Queue has two pointers front and back whereas circular linked list only has one pointer head.

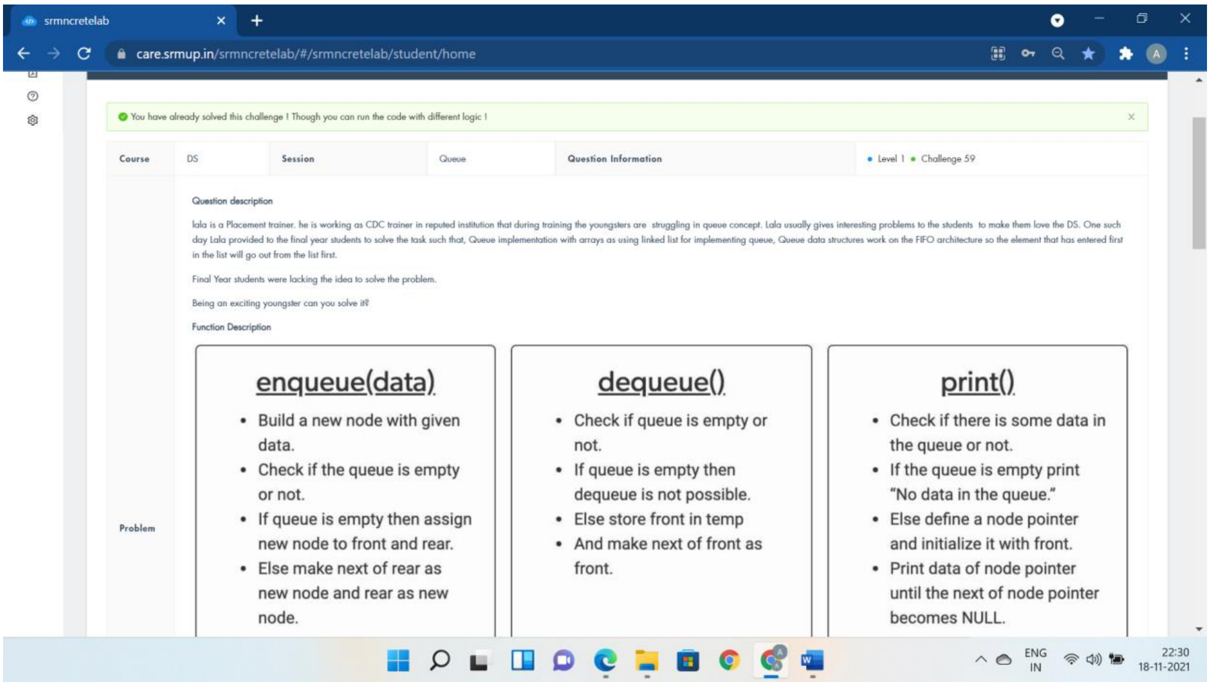
Final Year students were lacking the idea to solve the problem.

Being an exciting youngster can you solve it?

Function Description

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

struct node *f = NULL;
struct node *r = NULL;

struct node
{
   int data;
   struct node* next;
};
```

```
void linkedListTraversal(struct node *ptr)
{
    //printf("Printing the elements of this linked list\n");
    while (ptr != NULL)
    {
        printf("%d ", ptr->data);
        ptr = ptr->next;
    }
}

void enqueue(int d)
{
    struct node *n;
    n = (struct node*)malloc(sizeof(struct node));
    if(n==NULL){
        printf("Queue is Full");
    }
    else{
        n->data = d;
        n->next = NULL;
        if(f==NULL){
            f=r=n;
        }
        else{
            r->next = n;
            r=n;
        }
    }
}

int dequeue()
{
    int val = -1;
    struct node* t;
    t = f;
    if(f==NULL){
        printf("Queue is Empty\n");
    }
    else{
        f = f->next;
        val = t->data;
        free(t);
    }
    return val;
}

int main()
{
    int n,i,t;
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        scanf("%d",&t);
        enqueue(t);
    }
    linkedListTraversal(f);
    for(i=0;i<2;i++){
        dequeue();
        printf("\n");
        linkedListTraversal(f);
    }
    return 0;
}
```



You have already solved this challenge ! Though you can run the code with different logic !

Course	DS	Session	Queue	Question Information	• Level 1 • Challenge 59

Problem

Question description

lala is a Placement trainer. he is working as CDC trainer in reputed institution that during training the youngsters are struggling in queue concept. Lala usually gives interesting problems to the students to make them love the DS. One such day Lala provided to the final year students to solve the task such that, Queue implementation with arrays as using linked list for implementing queue, Queue data structures work on the FIFO architecture so the element that has entered first in the list will go out from the list first.

Final Year students were lacking the idea to solve the problem.

Being an exciting youngster can you solve it?

Function Description

enqueue(data)

- Build a new node with given data.
- Check if the queue is empty or not.
- If queue is empty then assign new node to front and rear.
- Else make next of rear as new node and rear as new node.

dequeue()

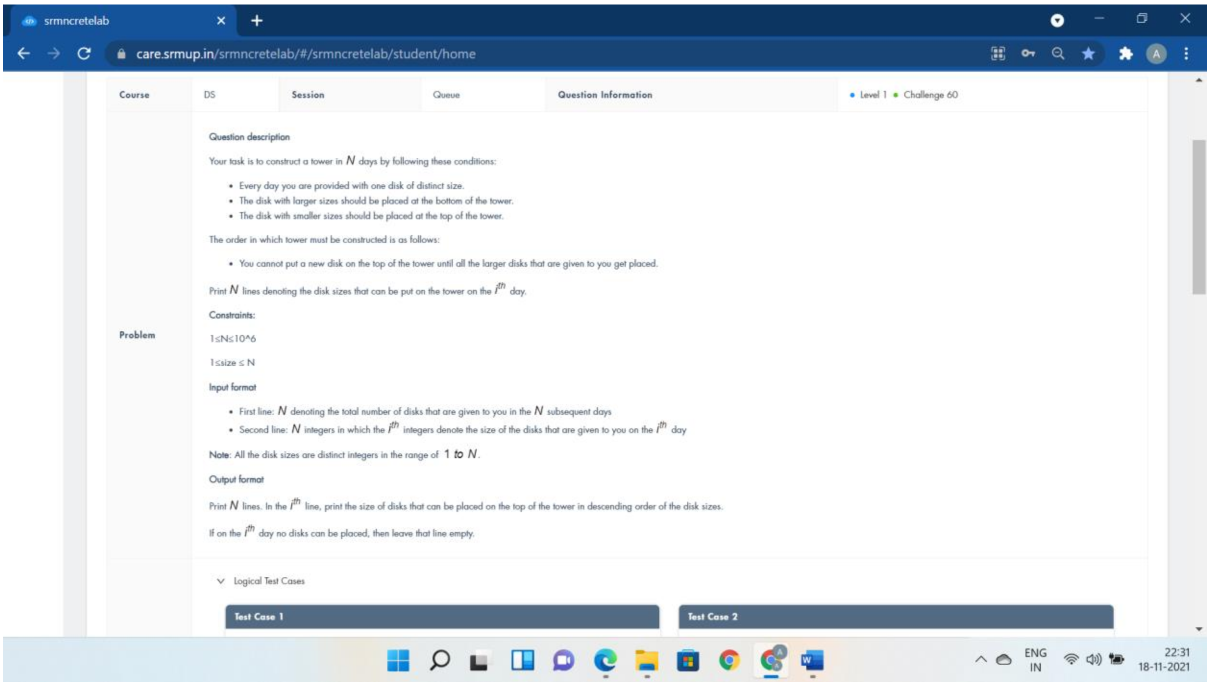
- Check if queue is empty or not.
- If queue is empty then dequeue is not possible.
- Else store front in temp
- And make next of front as front.

print()

- Check if there is some data in the queue or not.
- If the queue is empty print "No data in the queue."
- Else define a node pointer and initialize it with front.
- Print data of node pointer until the next of node pointer becomes NULL.

```
#include <stdio.h>

#include <stdlib.h>

struct node *front = NULL;

struct node *rear = NULL;

struct node

{

   int data;

   struct node *next;

};

void linkedListTraversal(struct node *ptr)

{

   //printf("Printing the elements of this linked list\n");

   while (ptr != NULL)

   {

      printf("%d ", ptr->data);

      ptr = ptr->next;

   }

}

void enqueue(int d)

{

   struct node* new_n;

   new_n = (struct node*)malloc(sizeof(struct node));
```

```
    if(new_n==NULL){
        printf("Queue is Full");
    }
    else{
        new_n->data = d;
        new_n->next = NULL;
        if(front==NULL){
            front=rear=new_n;
        }
        else{
            rear->next = new_n;
            rear=new_n;
        }
    }
}
int main()
{
    int n,i,t;
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        scanf("%d",&t);
        enqueue(t);
    }
    linkedListTraversal(front);
    return 0;
}
```



Course	DS	Session	Queue	Question Information	• Level 1 • Challenge 60

Problem

Question description

Your task is to construct a tower in N days by following these conditions:

- Every day you are provided with one disk of distinct size.
- The disk with larger sizes should be placed at the bottom of the tower.
- The disk with smaller sizes should be placed at the top of the tower.

The order in which tower must be constructed is as follows:

- You cannot put a new disk on the top of the tower until all the larger disks that are given to you get placed.

Print N lines denoting the disk sizes that can be put on the tower on the i^{th} day.

Constraints:

1≤N≤10^6

1≤size ≤ N

Input format

- First line: N denoting the total number of disks that are given to you in the N subsequent days
- Second line: N integers in which the i^{th} integers denote the size of the disks that are given to you on the i^{th} day

Note: All the disk sizes are distinct integers in the range of 1 *to* N.

Output format

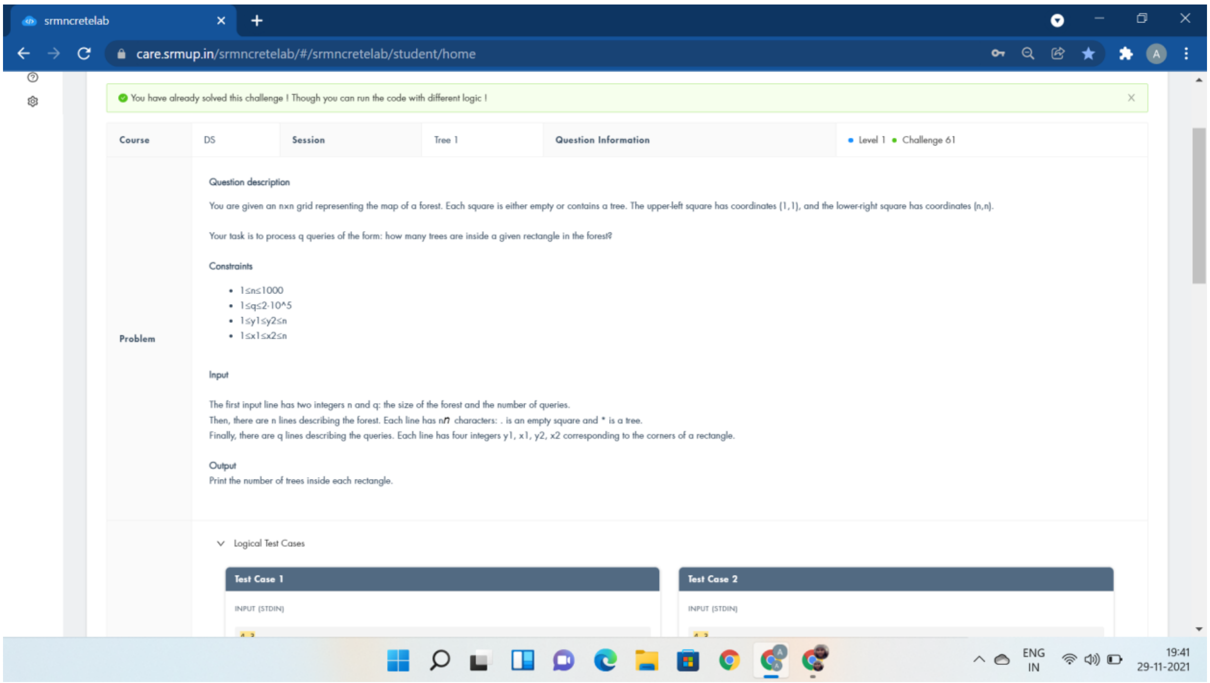
Print N lines. In the i^{th} line, print the size of disks that can be placed on the top of the tower in descending order of the disk sizes.

If on the i^{th} day no disks can be placed, then leave that line empty.

```
#include<stdio.h>

int main()

{

   long int disk,temp[1000000]={0},size,i,max;

   scanf("%ld",&disk);

   max=disk;

   for(i=0;i<disk;i++)

   {

      scanf("%ld",&size);

      temp[size]=size;

      if(size==max)

      {

         while(temp[size])

         {

            printf("%ld ",temp[size]);

            size--;

         }

         max=size;

         printf("\n");

      }

      else

       printf("\n");
```

```
    }
    return 0;
}
```

TREE 1:-



You have already solved this challenge ! Though you can run the code with different logic !

Course	DS	Session	Tree 1	Question Information	● Level 1 ● Challenge 61

Problem

Question description

You are given an nxn grid representing the map of a forest. Each square is either empty or contains a tree. The upper-left square has coordinates (1,1), and the lower-right square has coordinates (n,n).

Your task is to process q queries of the form: how many trees are inside a given rectangle in the forest?

Constraints

- 1≤n≤1000
- 1≤q≤2·10^5
- 1≤y1≤y2≤n
- 1≤x1≤x2≤n

Input

The first input line has two integers n and q: the size of the forest and the number of queries.
Then, there are n lines describing the forest. Each line has nn characters: . is an empty square and * is a tree.
Finally, there are q lines describing the queries. Each line has four integers y1, x1, y2, x2 corresponding to the corners of a rectangle.

Output
Print the number of trees inside each rectangle.

Logical Test Cases

Test Case 1

INPUT (STDIN)

Test Case 2

INPUT (STDIN)

```
#include<bits/stdc++.h>

using namespace std;

#define rep(i,a,b) for (int i=a; i<b; ++i)

int dp[1005][1005];

int main(){

    int n,m; cin>>n>>m;

    rep(i,1,n+1){

        rep(j,1,n+1){

            char x; cin>>x;

            dp[i][j] = (dp[i-1][j] - dp[i-1][j-1]) + dp[i][j-1] + (x=='*');

        }

    }

    while(m--){

        int y1 , x1, y2, x2; cin>>y1>>x1>>y2>>x2;
```