ACADEMY OF TECHNOLOGY

NAME : PRIYADARSHAN GHOSH

COLLEGE ROLL NO: 72

UNIVERSITY ROLL NO: 16900319072

DEPARTMENT: ECE-1(Y)

SEMESTER: 3rd

PAPER CODE : ES-CS391

Special Laboratory *Assignment*

SPECIAL ASSIGNMENT:

#1. A bracket is considered to be any one of the following characters: (,), {, }, [, or]. Two brackets are considered to be a matched pair if the an opening bracket (i.e., (, [, or {) occurs to the left of a closing bracket (i.e.,),], or }) of the exact same type. There are three types of matched pairs of brackets: [ ],

{ },and ( ).

A matching pair of brackets is not balanced if the set of brackets it encloses are not matched. For example,

{[(])} is not balanced because the contents in between { and } are not balanced. The pair of square

brackets encloses a single, unbalanced opening bracket, (, and the pair of parentheses encloses a single,

unbalanced closing square bracket,].

By this logic, we say a sequence of brackets is balanced if the following conditions are met:

It contains no unmatched brackets.

The subset of brackets enclosed within the confines of a matched pair of brackets is also a matched pair

of brackets.

Given n strings of brackets, determine whether each sequence of brackets is balanced. If a string is

balanced, return YES. Otherwise, return NO.

SAMPLE INPUT

3

{[()]}

{[()]}

{{[[(())]]}}

SAMPLE OUTPUT

YES

NO

YES

Ans:

```
#include <stdio.h>
#include <string.h>
#define MAX 100
char stack[MAX];
int top=-1;
int isFull(){
   if(top == MAX-1){
        return 1;
   }
   else
        return 0;
}
int isEmpty(){
   if(top == -1){
        return 1;
   }
   else
        return 0;
}
void push(char ch)
{
        if(isFull() == 1) {
        printf("\n Stack is full");
        return;
```

```
        }
        stack[++top]=ch;
}
char pop()
{
        if (top==-1)
        {
                printf("\nStack is empty");
                return -1;
        }
        return (stack[top--]);
}
char peek(void){
   char x;
   if(top == -1){
        printf("\n Stack is empty");
        return -1;
   }
   x = stack[top];
   return x;
}
int isbalanced(char s[])
{
        char x,ch;
        int i=0,j=0;
        while (s[i]!='\0')
        {
```

```
ch=s[i];
switch(ch){
        case '{':
        case '(':
        case '[':
                push(ch);
                break;
        case '}':
                if(isEmpty()||(peek() != '{')){
                        return 0;
                }
                pop();
                break;
        case ')':
                if(isEmpty()||(peek() != '(')){
                        return 0;
                }
                pop();
                break;
        case ']':
                if(isEmpty()||(peek() != '[')){
                        return 0;
                }
                pop();
                break;
}
i++;
```

```
        }
        return isEmpty();

}
int main()
{
        int x;
        char s[100];
        printf("enter the string : ");
        gets(s);
        x=isbalanced(s);
        if(x==0)
        {
                printf("NO");
        }
        else
        {
                printf("YES");
        }
        return 0;
}
```

OUTPUT =>

```
enter the string : {[()]}
YES
--------------------------------
Process exited after 39.3 seconds with return value 0
Press any key to continue . . .
```

#2. You are given a stack of N integers such that the first element represents the top of the stack and the

last element represents the bottom of the stack. You need to pop at least one element from the stack.

At any one moment, you can convert stack into a queue. The bottom of the stack represents the front of

the queue. You cannot convert the queue back into a stack. Your task is to remove exactly K elements

such that the sum of the K removed elements is maximised.

SAMPLE INPUT

10 5

10 9 1 2 3 4 5 6 7 8

SAMPLE OUTPUT

40

Explanation

Pop two elements from the stack. i.e {10,9}

Then convert the stack into queue and remove first three elements from the queue. i.e {8,7,6}.

The maximum possible sum is 10+9+8+7+6 = 40

Ans:

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 100
int stack[MAX];
int top=-1;
```

```
int isFull(){
   if(top == MAX-1){
        return 1;
   }
   else
        return 0;
}

void push(int x) {
    if(isFull() == 1) {
        printf("\n Stack is full");
        return;
    }
    stack[++top] = x;
}

int isEmpty(){
   if(top == -1){
        return 1;
   }
   else
        return 0;
}

int pop(void){
   int x;
```

```
   if(isEmpty() == 1){
        printf("\n Stack is empty");
        return -1;
   }
   x = stack[top--];
   return x;
}

int peek(void){
   int x;
   if(top == -1){
        printf("\n Stack is empty");
        return -1;
   }
   x = stack[top];
   return x;
}

void display(){
        int i;
        for(i=top;i>=0;i--){
                printf("\n %d",stack[i]);
        }
}
int LQ[MAX];
int rear = -1;
int front = -1;
```

```
void enQueue(int x) {

   if(rear == MAX-1){
      printf("Queue is full");
   }
   if(rear == -1 && front == -1){
      front = rear = 0;
   }
   else
     rear++;
   LQ[rear] = x;
}

int deQueue() {
   int x;
   if(front == -1 || front > rear) {
      printf("Queue is empty");
      return -1;
   }
   x = LQ[front++];
   return x;
}

void displayQ() {
  int i;
  for(i=front; i<= rear; i++)
```

```
    printf("%d ", LQ[i]);
}
int maximisedSum(int arr[],int n,int k)
{
        int x,ch,sum=0;
        for(int i=n-1;i>=0;i--)
        {
                push(arr[i]);
        }
        sum=sum+pop();
        k--;
        while(stack[top]>stack[top-1] && k!=0)
        {
                sum=sum+pop();
                k--;
        }
        for(int i=0;i<=top;i++)
        {
                enQueue(stack[i]);
        }
        while(k>0)
        {
                sum=sum+deQueue();
                k--;
        }
        return sum;
}
```

```
int main()
{
	int res,n,k;
	printf("Enter the number of elements in Stack : ");
	scanf("%d",&n);
	printf("Enter the number of elements to remove : ");
	scanf("%d",&k);
	int arr[n];
	for(int i=0;i<n;i++)
	{
		scanf("%d",&arr[i]);
	}
	res=maximisedSum(arr,n,k);
	printf("%d",res);
}
```

OUTPUT =>

```
Enter the number of elements in Stack : 10 5
Enter the number of elements to remove : 10 9 1 2 3 4 5 6 7 8
40
--------------------------------
Process exited after 101.9 seconds with return value 0
Press any key to continue . . .
```

#3. You are given a stack of N integers. In one operation, you can either pop an element from the stack

or push any popped element into the stack. You need to maximize the top element of the stack after

performing exactly K operations. If the stack becomes empty after performing K operations and there is

no other way for the stack to be non-empty, print -1.

SAMPLE INPUT

6 4

1 2 4 3 3 5

SAMPLE OUTPUT

4

Explanation

In 3 operations, we remove 1,2,4 and in the fourth operation, we push 4 back into the stack. Hence, 4 is

the answer.

Ans:

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 100
int stack[MAX];
int top=-1;

int isFull(){
   if(top == MAX-1){
        return 1;
   }
   else
        return 0;
}
```

```
void push(int x) {
    if(isFull() == 1) {
        printf("\n Stack is full");
        return;
    }
    stack[++top] = x;
}

int isEmpty(){
   if(top == -1){
        return 1;
   }
   else
        return 0;
}

int pop(void){
   int x;
   if(isEmpty() == 1){
        printf("\n Stack is empty");
        return -1;
   }
   x = stack[top--];
   return x;
}

int peek(void){
```

```
   int x;
   if(top == -1){
        printf("\n Stack is empty");
        return -1;
   }
   x = stack[top];
   return x;
}

void display(){
        int i;
        for(i=top;i>=0;i--){
                printf("\n %d",stack[i]);
        }
}
int maximumtop(int n,int p)
{
        int max=peek(),a;
        for(int i=0;i<p-1;i++)
        {
                a=pop();
                max=max>a ? max : a;
        }
        push(max);
        return max;
}
int main()
```

```
{
	int n,p,x,l,res,arr[100];
	printf("Enter the number of elements : ");
	scanf("%d",&n);
	printf("Enter the number of operations : ");
	scanf("%d",&p);
	x=n;
	printf("enter the elements : ");
	for(int i=0;i<n;i++)
	{
		scanf("%d",&arr[i]);
	}
	for(int i=n-1;i>=0;i--)
	{
		push(arr[i]);
	}

	res=maximumtop(n,p);
	printf("%d",res);
	return 0;
}
```

OUTPUT =>

```
Enter the number of elements : 6 4
Enter the number of operations : enter the elements : 1 2 4 3 3 5
4
--------------------------------
Process exited after 11.27 seconds with return value 0
```

Press any key to continue . . .

#4. The Monk is trying to explain to its users that even a single unit of time can be extremely important

and to demonstrate this particular fact he gives them a challenging task.

There are N processes to be completed by you, the chosen one, since you're Monk's favorite student.

All the processes have a unique number assigned to them from 1 to N.

Now, you are given two things:

• The calling order in which all the processes are called.

• The ideal order in which all the processes should have been executed.

Now, let us demonstrate this by an example. Let's say that there are 3 processes, the calling order of

the processes is: 3 - 2 - 1. The ideal order is: 1 - 3 - 2, i.e., process number 3 will only be executed after

process number 1 has been completed; process number 2 will only be executed after process number 3

has been executed.

• Iteration #1: Since the ideal order has process #1 to be executed firstly, the calling ordered is

changed, i.e., the first element has to be pushed to the last place. Changing the position of the

element takes 1 unit of time. The new calling order is: 2 - 1 - 3. Time taken in step #1: 1.

• Iteration #2: Since the ideal order has process #1 to be executed firstly, the calling ordered has

to be changed again, i.e., the first element has to be pushed to the last place. The new calling

order is: 1 - 3 - 2. Time taken in step #2: 1.

• Iteration #3: Since the first element of the calling order is same as the ideal order, that process

will be executed. And it will be thus popped out. Time taken in step #3: 1.

• Iteration #4: Since the new first element of the calling order is same as the ideal order, that

process will be executed. Time taken in step #4: 1.

• Iteration #5: Since the last element of the calling order is same as the ideal order, that process

will be executed. Time taken in step #5: 1.

Total time taken: 5 units.

PS: Executing a process takes 1 unit of time. Changing the position takes 1 unit of time.

SAMPLE INPUT

3

3 2 1

1 3 2

SAMPLE OUTPUT

5

Ans:

```
#include<stdio.h>

#include<stdlib.h>

#define MAX 100

int stack[MAX];

int top;
```

```
int isFull(){
   if(top == MAX-1){
        return 1;
   }
   else
        return 0;
}

void push(int x) {
    if(isFull() == 1) {
        printf("\n Stack overflow!");
        return;
    }
    stack[++top] = x;
}

int isEmpty(){
   if(top == -1){
        return 1;
   }
   else
        return 0;
}

int pop(void){
```

```
   int x;
   if(isEmpty() == 1){
        printf("\n Stack is empty");
        return -1;
   }
   x = stack[top--];
   return x;
}

int peek(void){
   int x;
   if(top == -1){
        printf("\n Stack is empty");
        return -1;
   }
   x = stack[top];
   return x;
}

int main(){
   top = -1;
        int n;
    int c=0;
    int i,j;
    int arr1[20];
```

```
int arr2[20];
scanf("%d",&n);
for (int i = 0; i < n; i++)
{
    scanf("%d",&arr1[i]);

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

    scanf("%d",&arr2[i]);
    push(arr2[i]);
}


for(j =0 ;arr1[j] != arr2[j];j++)
{

    int r = arr1[0];
    for ( i = 0; i < n-1; i++)
    {
        arr1[i] = arr1[i+1];
    }
    arr1[i]=r;
    c++;
```

```
    }
    int k=0;
    while (arr1[k] == arr2[k])
    {
       pop();
       c++;
       k++;
    }

   printf("%d",c);
    return 0;

}
```

OUTPUT =>

```
3
3 2 1
1 3 2
5

...Program finished with exit code 0
Press ENTER to exit console.
```