**############################################################**

**1- Evaluation of Postfix Expression**

Given string **S** representing a postfix expression, the task is to evaluate the expression and find the final value. Operators will only include the basic arithmetic operators like **\*, /, + and -**.

**Example 1:**

**Input**: S = "231\*+9-"

**Output**: -4

**Explanation**:

After solving the given expression,

we have -4 as result.

**Example 2:**

**Input**: S = "123+\*8-"

**Output**: -3

**Explanation**:

After solving the given postfix

expression, we have -3 as result.

**Your Task:**  
You do not need to read input or print anything. Complete the function**evaluatePostfixExpression()**that takes the string S denoting the expression as input parameter and returnsthe evaluated value.

**Expected Time Complexity**: O(|S|)  
**Expected Auixilliary Space**: O(|S|)

**Constraints:**  
1 ≤ |S| ≤ 105

int evaluatePostfix(string s)

{ stack<int> stk;

for(int i = 0; i < s.length(); i++)

{

if(isdigit(s[i]))

stk.push((int)(s[i] - '0'));

else

{ int b = stk.top();

stk.pop();

int a = stk.top();

stk.pop();

switch(s[i])

{ case '+': stk.push(a + b);

break;

case '-': stk.push(a - b);

break;

case '\*': stk.push(a \* b);

break;

case '/': stk.push(a / b);

break;

}

}

}

return stk.top();

}

**############################################################**

**2- Reverse a string using Stack**

You are given a string **S**, the task is to reverse the string using stack.

**Example 1:**

**Input:** S="GeeksforGeeks"

**Output:** skeeGrofskeeG

**Your Task:**  
You don't need to read input or print anything. Your task is to complete the function**reverse()** which takes the string **S**as an input parameter and returns the reversed string.

**Expected Time Complexity:** O(N)  
**Expected Auxiliary Space:** O(N)

**Constraints:**  
1 ≤ length of the string ≤ 100

char\* reverse(char \*s, int len)

{

stack<char> stk;

for(int i = 0; i < len; i++)

stk.push(s[i]);

int i = 0;

while(!stk.empty())

{

s[i++] = stk.top();

stk.pop();

}

return s;

}

**############################################################**

**3- Sort a stack**

Given a stack, the task is to sort it such that the top of the stack has the greatest element.

**Example 1:**

**Input:**

Stack: 3 2 1

**Output:** 3 2 1

**Example 2:**

**Input:**

Stack: 11 2 32 3 41

**Output:** 41 32 11 3 2

**Your Task:**  
You don't have to read input or print anything. Your task is to complete the function **sort()**which sorts the elements present in the given stack. (The sorted stack is printed by the driver's code by popping the elements of the stack.)

**Expected Time Complexity** : O(N\*N)  
**Expected Auixilliary Space** : O(N) recursive.

**Constraints:**  
1<=N<=100  
  
**Note:**The **Input/Ouput** format and **Example** given are used for system's internal purpose, and should be used by a user for **Expected Output** only. As it is a function problem, hence a user should not read any input from stdin/console. The task is to complete the function specified, and not to write the full code.

void update(stack<int> &s, int data)

{

if(s.size() == 0 || data >= s.top())

{ s.push(data);

return;

}

int temp = s.top();

s.pop();

update(s, data);

s.push(temp);

return;

}

void SortedStack :: sort()

{

if(s.size() == 1)

return;

int temp = s.top();

s.pop();

sort();

update(s, temp);

return;

}

**############################################################**

**4. Get minimum element from stack**

You are given **N** elements and your task is to Implement a Stack in which you can get minimum element in O(1) time.

**Example 1:**

**Input:**

push(2)

push(3)

pop()

getMin()

push(1)

getMin()

**Output:** 3 2 1

**Explanation:** In the first test case for

query

push(2)  the stack will be {2}

push(3)  the stack will be {2 3}

pop() poped element will be 3 the

  stack will be {2}

getMin() min element will be 2

push(1) the stack will be {2 1}

getMin() min element will be 1

**Your Task:**  
You are required to complete the three methods **push()** which take one argument an integer **'x'** to be pushed into the stack, **pop()** which returns a integer poped out from the stack and **getMin()** which returns the min element from the stack. (-1 will be returned if for **pop() and getMin()**the stack is empty.)

**Expected Time Complexity** : O(1) for all the 3 methods.  
**Expected Auixilliary Space** : O(1) for all the 3 methods.

**Constraints:**  
1 <= Number of queries <= 100  
1 <= values of the stack <= 100

int \_stack :: getMin()

{

if(s.empty())

return -1;

return minEle;

}

/\*returns poped element from stack\*/

int \_stack ::pop()

{

if(s.empty())

return -1;

else if(minEle > s.top())

{ int top = minEle;

minEle = (minEle \* 2) - s.top();

s.pop();

return top;

}

else

{ int top = s.top();

s.pop();

return top;

}

}

/\*push element x into the stack\*/

void \_stack::push(int x)

{

if(s.empty())

{ s.push(x);

minEle = x;

return;

}

else if(minEle > x)

{ s.push((2 \* x) - minEle);

minEle = x;

return;

}

else

{ s.push(x);

return;

}

}

**############################################################**

**5. Parenthesis Checker**

Given an expression string **x**. Examine whether the pairs and the orders of “{“,”}”,”(“,”)”,”[“,”]” are correct in exp.  
For example, the function should return 'true' for exp = “[()]{}{[()()]()}” and 'false' for exp = “[(])”.

**Example 1:**

**Input**:

{([])}

**Output**:

true

**Explanation**:

{ ( [ ] ) }. Same colored brackets can form

balaced pairs, with 0 number of

unbalanced bracket.

**Example 2:**

**Input**:

()

**Output**:

true

**Explanation**:

(). Same bracket can form balanced pairs,

and here only 1 type of bracket is

present and in balanced way.

**Example 3:**

**Input**:

([]

**Output**:

false

**Explanation**:

([]. Here square bracket is balanced but

the small bracket is not balanced and

Hence , the output will be unbalanced.

**Your Task:**  
This is a **function**problem. You only need to complete the function **ispar()**that takes a **string**as a **parameter**and returns a boolean value **true**if **brackets**are **balanced**else **returns false**. The **printing**is done **automatically**by the **driver code**.  
  
**Expected Time Complexity**: O(|x|)  
**Expected Auixilliary Space**: O(|x|)  
  
**Constraints:**  
1 ≤ |x| ≤ 32000

**Note**: The drive code prints "balanced" if function return true, otherwise it prints "not balanced".

bool ispar(string s)

{

stack<char> stk;

for(int i = 0; i < s.length(); i++)

{

if(stk.empty())

stk.push(s[i]);

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

if(stk.top() == '[')

stk.pop();

else

return false;

else if(s[i] == '}')

if(stk.top() == '{')

stk.pop();

else

return false;

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

if(stk.top() == '(')

stk.pop();

else

return false;

else

stk.push(s[i]);

}

if(stk.empty() == true)

return true;

else

return false;

}

**############################################################**

**Max rectangle**

Given a binary matrix. Find the maximum area of a rectangle formed only of 1s in the given matrix.

**Example 1:**

**Input:**

n = 4, m = 4

M[][] = {{0 1 1 0},

{1 1 1 1},

{1 1 1 1},

{1 1 0 0}}

**Output:** 8

**Explanation:** For the above test case the

matrix will look like

0 1 1 0

1 1 1 1

1 1 1 1

1 1 0 0

the max size rectangle is

1 1 1 1

1 1 1 1

and area is 4 \*2 = 8.

**Your Task:**  
Your task is to complete the function **maxArea** which returns the maximum size rectangle area in a binary-sub-matrix with all 1’s. The function takes 3 arguments the first argument is the Matrix M[ ] [ ] and the next two are two integers n and m which denotes the size of the matrix M.

**Expected Time Complexity** : O(n\*m)  
**Expected Auixiliary Space** : O(m)

**Constraints:**  
1<=n,m<=1000  
0<=M[][]<=1  
  
**Note:**The **Input/Ouput** format and **Example** given are used for system's internal purpose, and should be used by a user for **Expected Output** only. As it is a function problem, hence a user should not read any input from stdin/console. The task is to complete the function specified, and not to write the full code.

int getsum(vector<int> v, int n)

{

vector<int> l(n), r(n);

stack<int> stk;

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

{

if(stk.empty())

{ l[i] = 0;

stk.push(i);

}

else

{ while(!stk.empty() && v[i] <= v[stk.top()])

stk.pop();

l[i] = stk.empty()? 0: stk.top() + 1;

stk.push(i);

}

}

while(!stk.empty())

stk.pop();

for(int i = n-1; i >= 0; i--)

{

if(stk.empty())

{ r[i] = n-1;

stk.push(i);

}

else

{ while(!stk.empty() && v[i] <= v[stk.top()])

stk.pop();

r[i] = stk.empty()? n-1: stk.top() - 1;

stk.push(i);

}

}

int sum = 0;

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

sum = max(sum, (r[i] - l[i] + 1) \* v[i]);

return sum;

}

int maxArea(int M[MAX][MAX], int n, int m)

{

vector<int> v;

for(int i = 0; i < m; i++)

v.push\_back(M[0][i]);

int sum = getsum(v, m);

for(int i = 1; i < n; i++)

{

for(int j = 0; j < m; j++)

{

if(M[i][j])

v[j]++;

else

v[j] = 0;

}

sum = max(sum, getsum(v, m));

}

return sum;

}

**############################################################**

**7. Maximum Rectangular Area in a Histogram**

Find the largest rectangular area possible in a given histogram where the largest rectangle can be made of a number of contiguous bars. For simplicity, assume that all bars have the same width and the width is**1 unit**.

**Example 1:**

**Input:**

N = 7

arr[] = {6,2,5,4,5,1,6}

**Output:** 12

**Explanation:**



**Example 2:**

**Input:**

N = 8

arr[] = {7 2 8 9 1 3 6 5}

**Output:** 16

**Explanation:** Maximum size of the histogram

will be 8  and there will be 2 consecutive

histogram. And hence the area of the

histogram will be 8x2 = 16.

**Your Task:**  
The task is to complete the function **getMaxArea**() which takes the array arr[] and its size N as inputs and finds the largest rectangular area possible and **returns** the answer.

**Expected Time Complxity** : O(N)  
**Expected Auxilliary Space** : O(N)

**Constraints:**  
1 ≤ N ≤ 106  
1 ≤ arr[i] ≤ 1012

long long getMaxArea(long long arr[], int n)

{

vector<long long int> l(n), r(n);

stack<long long int> stk;

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

{

if(stk.empty())

{ stk.push(i);

l[i] = 0;

}

else

{ while(!stk.empty() && arr[i] <= arr[stk.top()])

stk.pop();

l[i] = stk.empty()? 0: stk.top() + 1;

stk.push(i);

}

}

while(!stk.empty())

stk.pop();

for(int i = n-1; i >= 0; i--)

{

if(stk.empty())

{ stk.push(i);

r[i] = n-1;

}

else

{ while(!stk.empty() && arr[i] <= arr[stk.top()])

stk.pop();

r[i] = stk.empty()? n-1: stk.top() - 1;

stk.push(i);

}

}

long long int sum = 0;

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

sum = max(sum, (r[i] - l[i] + 1) \* arr[i]);

return sum;

}

**############################################################**

**8. Stock span problem**

The stock span problem is a financial problem where we have a series of **n** daily price quotes for a stock and we need to calculate the span of stock’s price for all **n** days.   
The span **Si** of the stock’s price on a given day **i** is defined as the maximum number of consecutive days just before the given day, for which the price of the stock on the current day is less than or equal to its price on the given day.  
For example, if an array of 7 days prices is given as {100, 80, 60, 70, 60, 75, 85}, then the span values for corresponding 7 days are {1, 1, 1, 2, 1, 4, 6}.

**Example 1:**

**Input**:

N = 7, price[] = [100 80 60 70 60 75 85]

**Output**:

1 1 1 2 1 4 6

**Explanation**:

Traversing the given input span for 100

will be 1, 80 is smaller than 100 so the

span is 1, 60 is smaller than 80 so the

span is 1, 70 is greater than 60 so the

span is 2 and so on. Hence the output will

be 1 1 1 2 1 4 6.

**Example 2:**

**Input**:

N = 6, price[] = [10 4 5 90 120 80]

**Output**:

1 1 2 4 5 1

**Explanation**:

Traversing the given input span for 10

will be 1, 4 is smaller than 10 so the

span will be 1, 5 is greater than 4 so

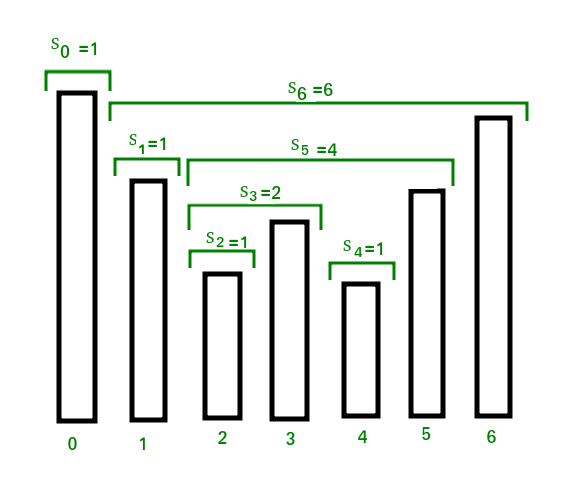
the span will be 2 and so on. Hence, the

output will be 1 1 2 4 5 1.

**User Task:**  
The task is to complete the function **calculateSpan**() which takes two parameters, an array**price[]**denoting the price of stocks, and an integer **N**denoting the size of the array and number of days. This function finds the span of stock's price for all N days and returns an array of length **N** denoting the span for the i-th day.

**Expected Time Complexity:**O(N).  
**Expected Auxiliary Space:**O(N).

**Constraints:**  
1 ≤ N ≤ 105  
1 ≤ C[i] ≤ 105



vector <int> calculateSpan(int arr[], int n)

{

stack<int> stk;

vector<int> v(n);

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

{

if(stk.empty())

{ stk.push(i);

v[i] = 1;

}

else

{ while(!stk.empty() && arr[i] >= arr[stk.top()])

stk.pop();

v[i] = stk.empty()? (i + 1): (i - stk.top());

stk.push(i);

}

}

return v;

}

**############################################################**

**9. Infix to Postfix**

Given an infix expression in the form of string **str**. Convert this infix expression to postfix expression.

* **Infix expression:** The expression of the form a **op** b. When an operator is in-between every pair of operands.
* **Postfix expression:** The expression of the form a b **op**. When an operator is followed for every pair of operands.  
  â€‹**Note:** The order of precedence is: ^ **greater than** \* **equals to** / **greater than** + **equals to** -.

**Example 1:**

**Input**: str = "a+b\*(c^d-e)^(f+g\*h)-i"

**Output**: abcd^e-fgh\*+^\*+i-

**Explanation**:

After converting the infix expression

into postfix expression, the resultant

expression will be abcd^e-fgh\*+^\*+i-

**Example 2:**

**Input**: str = "A\*(B+C)/D"

**Output**: ABC+\*D/

**Explanation**:

After converting the infix expression

into postfix expression, the resultant

expression will be ABC+\*D/

**Your Task:**  
This is a **function**problem. You only need to complete the function**infixToPostfix()**that takes a **string**(Infix Expression) as a **parameter** and **returns**a **string(**postfix expression**)**. The **printing**is done **automatically**by the **driver code**.

**Expected Time Complexity:**O(|str|).  
**Expected Auxiliary Space:**O(|str|).

**Constraints:**  
1 ≤ |str| ≤ 105

int priority(char ch)

{

if(ch == '^')

return 3;

else if(ch == '\*' || ch == '/')

return 2;

else if(ch == '+' || ch == '-')

return 1;

else

return -1;

}

string infixToPostfix(string s)

{

stack<char> stk;

string ans = "";

for(int i = 0; i < s.length(); i++)

{

if(isalnum(s[i]))

ans += s[i];

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

stk.push('(');

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

{ while(!stk.empty() && stk.top() != '(')

{ ans += stk.top();

stk.pop();

}

stk.pop();

}

else

{ while(!stk.empty() && priority(s[i]) <= priority(stk.top()))

{ ans += stk.top();

stk.pop();

}

stk.push(s[i]);

}

}

while(!stk.empty())

{ ans += stk.top();

stk.pop();

}

return ans;

}

**############################################################**

**10. Next Greater Element**

Given an array **arr[ ]** of size **N** having distinct elements, the task is to find the next greater element for each element of the array in order of their appearance in the array.  
Next greater element of an element in the array is the nearest element on the right which is greater than the current element.  
If there does not exist next greater of current element, then next greater element for current element is -1. For example, next greater of the last element is always -1.

**Example 1:**

**Input**:

N = 4, arr[] = [1 3 2 4]

**Output**:

3 4 4 -1

**Explanation**:

In the array, the next larger element

to 1 is 3 , 3 is 4 , 2 is 4 and for 4 ?

since it doesn't exist, it is -1.

**Example 2:**

**Input**:

N = 5, arr[] [6 8 0 1 3]

**Output**:

8 -1 1 3 -1

**Explanation**:

In the array, the next larger element to

6 is 8, for 8 there is no larger elements

hence it is -1, for 0 it is 1 , for 1 it

is 3 and then for 3 there is no larger

element on right and hence -1.

**Your Task:**  
This is a **function**problem. You only need to complete the function **nextLargerElement()**that takes list of integers **arr[ ]**and**N** as input parametersand returns list of integers of length N denoting the next greater elements for all the corresponding elements in the input array.

**Expected Time Complexity** : O(N)  
**Expected Auxilliary Space** : O(N)

**Constraints:**  
1 ≤ N ≤ 106  
1 ≤ Ai ≤ 1018

vector<long long> nextLargerElement(vector<long long> arr, int n)

{ vector<long long> v;

stack<long long> stk;

for(int i = n-1; i >= 0; i--)

{ if(stk.empty())

v.push\_back(-1);

else

{ while(!stk.empty() && stk.top() <= arr[i])

stk.pop();

v.push\_back(stk.empty()? -1: stk.top() );

}

stk.push(arr[i]);

}

reverse(v.begin(), v.end());

return v;

}

**############################################################**

**11. Maximum of minimum for every window size**

Given an integer array. The task is to find the maximum of the minimum of every window size in the array.  
**Note:** Window size varies from 1 to the size of the Array.

**Example 1:**

**Input:**

N = 7

arr[] = {10,20,30,50,10,70,30}

**Output:** 70 30 20 10 10 10 10

**Explanation:** First element in output

indicates maximum of minimums of all

windows of size 1. Minimums of windows

of size 1 are {10}, {20}, {30}, {50},

{10}, {70} and {30}. Maximum of these

minimums is 70.

Second element in output indicates

maximum of minimums of all windows of

size 2. Minimums of windows of size 2

are {10}, {20}, {30}, {10}, {10}, and

{30}. Maximum of these minimums is 30

Third element in output indicates

maximum of minimums of all windows of

size 3. Minimums of windows of size 3

are {10}, {20}, {10}, {10} and {10}.

Maximum of these minimums is 20.

Similarly other elements of output are

computed.

**Example 2:**

**Input:**

N = 3

arr[] = {10,20,30}

**Output:** 30 20 10

**Explanation:** First element in output

indicates maximum of minimums of all

windows of size 1.Minimums of windows

of size 1 are {10} , {20} , {30}.

Maximum of these minimums are 30 and

similarly other outputs can be computed

**Your Task:**  
The task is to complete the function **maxOfMin**() which takes the array arr[] and its size N as inputs and finds the maximum of minimum of every window size and returns an array containing the result.

**Expected Time Complxity** : O(N)  
**Expected Auxilliary Space** : O(N)

**Constraints:**  
1 <= N <= 105  
1 <= arr[i] <= 106

vector <int> maxOfMin(int arr[], int n)

{

stack<int> stk;

vector<int> minright(n), minleft(n);

for(int i = n-1; i >= 0; i--)

{

if(stk.empty())

{ minright[i] = n;

stk.push(i);

}

else

{ while(!stk.empty() && arr[stk.top()] >= arr[i])

stk.pop();

minright[i] = stk.empty()? n: stk.top();

stk.push(i);

}

}

while(!stk.empty())

stk.pop();

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

{

if(stk.empty())

{ minleft[i] = -1;

stk.push(i);

}

else

{ while(!stk.empty() && arr[stk.top()] >= arr[i])

stk.pop();

minleft[i] = stk.empty()? -1: stk.top();

stk.push(i);

}

}

vector<int> ans(n + 1);

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

{ int j = minright[i] - minleft[i] - 1;

ans[j] = max(ans[j], arr[i]);

}

for(int i = n-1; i >= 0; i--)

ans[i] = max(ans[i], ans[i + 1]);

ans.erase(ans.begin());

return ans;

}

**############################################################**