**Questions:**

1. Stack Implementation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98 |  |  |  | *#include<iostream>*  **using** **namespace** std;  *#define n 100*  **template**<**class** **T**>  **class** **Stack**{  T\* arr;  **int** top;  public:  Stack(){  arr = **new** T[n];  top = -1;  }  ~Stack(){  **delete** [] arr;  }  **void** push(T x){  **if** (top == n-1){  cout<<*"Stack Overflow"*<<endl;  **return**;  }  top++;  arr[top] = x;  }  **void** pop(){  **if** (top == -1){  cout<<*"No elements in stack to pop"*<<endl;  **return**;  }  top--;  }  **void** peek(){  **if** (top == -1){  cout<<*"Stack is empty, nothing to peek"*<<endl;  **return**;  }  cout<<*"Element at the top: "*<<arr[top]<<endl;  }  **bool** isEmpty(){  **return** top == -1;  }  **void** display\_stack(){  **if** (top == -1){  cout<<*"Stack is empty"*<<endl;  **return**;  }    cout<<*"All Stack Elements: "*;  **for** (**int** i=0; i<=top; i++){  cout<<arr[i]<<*' '*;  }  cout<<endl;  }  };  **int** main(){  Stack<**int**> st;  st.push(11);  st.push(22);  st.push(33);  st.peek();  st.pop();  st.peek();  cout<<st.isEmpty()<<endl;  st.peek();  cout<<st.isEmpty()<<endl;  st.push(111);  st.push(222);  st.push(333);  st.display\_stack();  *// Using Stack with string type*  Stack<**double**> st\_double;  st\_double.push(1.1);  st\_double.push(2.2);  st\_double.push(3.3);  st\_double.display\_stack();  *// Using Stack with string type*  Stack<string> st\_string;  st\_string.push(*"Stack"*);  st\_string.push(*"Implementation"*);  st\_string.push(*"using Template Class"*);  st\_string.display\_stack();  } |

Output:

Element at the top: 33

Element at the top: 22

0

Element at the top: 22

0

All Stack Elements: 11 22 111 222 333

All Stack Elements: 1.1 2.2 3.3

All Stack Elements: Stack Implementation using Template Class

2. Infix to Postfix

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59 | *#include<iostream>*  *#include<stack>*  **using** **namespace** std;  **int** precedence(**char** c){  **if** (c == *'^'*){  **return** 3;  }  **else** **if**(c == *'\*'* || c == *'/'*){  **return** 2;  }  **else** **if**(c == *'+'* || c == *'-'*){  **return** 1;  }  **else**{  **return** -1;  }  }  string Convert(string s){  stack<**char**> st;  string res;  **for** (**int** i=0; i< s.length(); i++){  **if** (s[i] >= *'a'* && s[i] <= *'z'* || s[i] >= *'A'* && s[i] <= *'Z'*){  res = res + s[i];  }  **else** **if**(s[i] == *'('*){  st.push(s[i]);  }  **else** **if**(s[i] == *')'*){  **while**( !st.empty() && st.top() != *'('*){  res += st.top();  st.pop();  }  **if**(!st.empty()){  st.pop();  }  }  **else**{  **while**(!st.empty() && ( precedence(st.top()) > precedence(s[i]))){  res = res + st.top();  st.pop();  }  st.push(s[i]);  }  }  **while**(!st.empty()){  res = res + st.top();  st.pop();  }  **return** res;  }  **int** main(){  cout<<”Equivalent Postfix: ”<<Convert(*"(A-B/C)\*(P/Q-R)"*)<<endl;  **return** 0;  } |

Output:

Equivalent Postfix: ABC/-PQ/R-\*

3. Infix to Prefix

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58 | *#include<iostream>*  *#include<stack>*  *#include <bits/stdc++.h>*  **using** **namespace** std;  **int** precedence(**char** c){  **if** (c == *'^'*){  **return** 3;  }  **else** **if**(c == *'\*'* || c == *'/'*){  **return** 2;  }  **else** **if**(c == *'+'* || c == *'-'*){  **return** 1;  }  **else**{  **return** -1;  }  }  string Convert(string s){  stack<**char**> st;  string res;  reverse(s.begin(), s.end());  **for** (**int** i=0; i< s.length(); i++){  **if** (s[i] >= *'a'* && s[i] <= *'z'* || s[i] >= *'A'* && s[i] <= *'Z'*){  res = res + s[i];  }  **else** **if**(s[i] == *')'*){  st.push(s[i]);  }  **else** **if**(s[i] == *'('*){  **while**( !st.empty() && st.top() != *')'*){  res += st.top();  st.pop();  }  **if**(!st.empty()){  st.pop();  }  }  **else**{  **while**(!st.empty() && ( precedence(st.top()) > precedence(s[i]))){  res = res + st.top();  st.pop();  }  st.push(s[i]);  }  }  **while**(!st.empty()){  res = res + st.top();  st.pop();  }  reverse(res.begin(), res.end());  **return** res;  }  **int** main(){  cout<<”Equivalent Prefix: ”<<Convert(*"(A-B/C)\*(P/Q-R)"*)<<endl;  **return** 0;  } |

Output:

Equivalent Prefix: \*-A/BC-/PQR

4. Postfix Evaluator

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38 | *#include<iostream>*  *#include<stack>*  *#include<string>*  **using** **namespace** std;  **int** main(){  stack<**int**> st;  string s = *"363/-63/1-\*"* ;  **for**(**int** i=0;i<s.length();i++){  **if**(isdigit(s[i])){  st.push(s[i] - *'0'*);  }  **else**{  **int** y = st.top();  st.pop();  **int** x = st.top();  st.pop();  **switch**(s[i]){  **case** *'+'*:  st.push(x+y);  **break**;  **case** *'-'*:  st.push(x-y);  **break**;  **case** *'\*'*:  st.push(x\*y);  **break**;  **case** *'/'*:  st.push(x/y);  **break**;  }  }  }  cout<<*"Result: "*<<st.top()<<endl;  **return** 0;  } |

Output:

Result: 1

5. Prefix Evaluator

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40 | *#include<iostream>*  *#include<stack>*  *#include<string>*  *#include <bits/stdc++.h>*  **using** **namespace** std;  **int** main(){  stack<**int**> st;  string s = *"\*-3/63-/631"*;  reverse(s.begin(),s.end());  **for**(**int** i=0;i<s.length();i++){  **if**(isdigit(s[i])){  st.push(s[i] - *'0'*);  }  **else**{  **int** x = st.top();  st.pop();  **int** y = st.top();  st.pop();  **switch**(s[i]){  **case** *'+'*:  st.push(x+y);  **break**;  **case** *'-'*:  st.push(x-y);  **break**;  **case** *'\*'*:  st.push(x\*y);  **break**;  **case** *'/'*:  st.push(x/y);  **break**;  }  }  }  cout<<*"Result: "*<<st.top()<<endl;  **return** 0;  } |

Output:

Result: 1

**Algorithms:**

**1. Stack Implementation**

**Push Operation**

1. START

2. Initialize Top to point to the last element in the stack, top=-1

3. Initialize size of stack N

4. If top == N-1, then

Print “Stack Overflow”

Exit

Else,

Increment top, top = top+1

Add element to the stack, stack[top] = element

Print "Element added successfully"

5. END

**Pop Operation**

1. START

2. Initialize Top to point to the last element in the stack, top=-1

3. Initialize size of stack N

4. If top == -1, then

Print "Stack Underflow"

Exit

Else,

Get the element from the top (element = stack[top])

Decrement top (top = top - 1)

Print "Element removed successfully"

Return element

**2. Infix to Postfix**

**Precedence function:**

1. START

2. Get character c from main program

3. If c == '^', then return 3

Else if c == '\*' or c == '/', then return 2

Else if c == '+' or c == '-',then return 1

Else, return -1 (non-operator)

4. RETURN

**Main Program**

1. START

2. Decalre string s representing infix expression

3. Initialize stack st (empty) and string res (empty)

4. If s[i] is an operand (A-Z, a-z), then

Append s[i] to res

Else if s[i] == '(', then

Push '(' onto stack

Else if s[i] == ')', then

While stack is not empty and stack.top() != '(':

Pop from stack and append to res

Pop '(' from stack

Else,

While stack is not empty and precedence(stack.top()) >= precedence(s[i]):

Pop from stack and append to res

Push s[i] onto stack

5. Repeat step 4 for each character in string s

5. While stack is not empty, pop from stack and append to res

6. Display res as equivalent postfix expression

7. END

**3. Infix to prefix**

**Precedence function:**

1. START

2. Get character c from main program

3. If c == '^', then return 3

Else if c == '\*' or c == '/', then return 2

Else if c == '+' or c == '-',then return 1

Else, return -1 (non-operator)

4. RETURN

**Main Program**

1. START

2. Decalre string s representing infix expression

3. Reverse the expression s to get s'

4. Replace all '(' with ')' and all ')' with '(' in s'

3. Initialize stack st (empty) and string res (empty)

4. If s''[i] is an operand (A-Z, a-z), then

Append s''[i] to res

Else if s''[i] == '(', then

Push '(' onto stack

Else if s''[i] == ')', then

While stack is not empty and stack.top() != '(':

Pop from stack and append to res

Pop '(' from stack

Else,

While stack is not empty and precedence(stack.top()) >= precedence(s''[i]):

Pop from stack and append to res

Push s''[i] onto stack

5. While stack is not empty, pop from stack and append to res

6. Reverse res to get the final prefix expression

7. Display res as equivalent prefix expression

8. END

**4. Postfix Evaluation**

1. START

2. Initialize an empty stack St

3. Initialize a string s representing a postfix expression (e.g., "363/-63/1-\*")

4. For each character c in the string s:

If c is a digit, then Convert c to an integer (c - '0') and push it onto the stack.

Else if c is an operator ('+', '-', '\*', '/'), then

Pop the top two elements from the stack (x and y).

Perform the operation based on c:

If c is '+', push x + y onto the stack.

If c is '-', push x - y onto the stack.

If c is '\*', push x \* y onto the stack.

If c is '/', push x / y onto the stack.

5. After processing all characters, pop the result from the stack.

6. Display the result.

7. END

**5. Postfix Evaluation**

1. START

2. Initialize an empty stack St

3. Initialize a string s representing a postfix expression (e.g., "363/-63/1-\*")

4. For each character c in the string s (from right to left):

If c is a digit, then Convert c to an integer (c - '0') and push it onto the stack.

Else if c is an operator ('+', '-', '\*', '/'), then

Pop the top two elements from the stack (x and y).

Perform the operation based on c:

If c is '+', push x + y onto the stack.

If c is '-', push x - y onto the stack.

If c is '\*', push x \* y onto the stack.

If c is '/', push x / y onto the stack.

5. After processing all characters, pop the result from the stack.

6. Display the result.

7. END