Jaskirat Singh

2020CSC1008

STACK USING ARRAY

```
//program to implement the stack using array
#include <iostream>
using namespace std;

//a class for the stack using array
class stack
{
private:
   int top;
   char *a;
   int size;

public:
   stack(int s);

   bool isEmpty();
   bool isFull();

   void numElements();
   char peek();

   bool push(char x);
   int pop();

   int prec(char c);
```

```
    void infixToPostfix(char s[], int len);
};

//parameterized constructor for stack class
stack::stack(int s)
{
    top = -1;
    size = s;
    a = new char[s];
}

//method to check if the stack is empty or not
bool stack::isEmpty()
{
    return (top < 0);
}

//method to check if the stack is full or not
bool stack::isFull()
{
    return (top >= (size - 1));
}

//method to return the number of elements in the stack
void stack::numElements()
{
    cout << "Number of Elements in the stack : " << (top + 1);
}

//method to push an element in the stack
bool stack::push(char x)
```

```
{
   if (isFull())
   {
      //stack is full
      cout << "\nStack Overflow\n";
      return false;
   }
   else
   {
      a[++top] = x;
      return true;
   }
}

//method to pop an element from the stack
int stack::pop()
{
   if (isEmpty())
   {
      //stack is empty
      cout << "\nStack Underflow\n";
      return false;
   }
   else
   {
      char x = a[top];
      top--;
      return x;
   }
}
```

```
//method to return a reference to the last element of the stack without removing it
char stack::peek()
{
   if (isEmpty())
   {
      cout << "\nStack is Empty\n";
      return 0;
   }
   else
   {
      return a[top];
   }
}

//method to return precedence of an operator
int stack::prec(char c) //Function to return precedence of operators
{
   if (c == '^')
      return 3;
   else if (c == '*' || c == '/')
      return 2;
   else if (c == '+' || c == '-')
      return 1;
   else
      return -1;
}

// method to change the infix expression into postfix
void stack::infixToPostfix(char s[], int len)
{
   stack stc(len);
```

```
    char ns[len];

    int j = 0;

    for (int i = 0; i < len; i++)
    {
        if (isalpha(s[i]) || isdigit(s[i]))
        {
            ns[j++] = s[i];
        }
        else if (s[i] == '(')
        {
            stc.push('(');
        }
        // If the scanned character is an ‘)’, pop and to output string from the stack until an ‘(‘ is encountered.
        else if (s[i] == ')')
        {
            while (!stc.isEmpty() && stc.peek() != '(')
            {
                char c = stc.pop();
                ns[j++] = c;
            }
            if (stc.peek() == '(')
            {
                stc.pop();
            }
        }
        else
        {
            //If an operator is scanned
```

```
            while (!stc.isEmpty() && stc.prec(s[i]) <= stc.prec(stc.peek()))
            {
                char c = stc.pop();
                ns[j++] = c;
            }
            stc.push(s[i]);
        }
    }
    while (!stc.isEmpty())
    {
        char c = stc.pop();
        ns[j++] = c;
    }

    //printing the postfix expression
    cout << "\nPrinting the Postfix Expression" << endl;
    for (int i = 0; i < len; i++)
    {
        cout << ns[i];
    }
    cout << endl;
}

//driver code
int main()
{



    int size;
    cout << "\nEnter the size of stack you want to make: ";
    cin >> size;
```

```
stack st(size);
cout << "\n Stack of size " << size << " is created successfully" << endl;

//pushing elements
cout << "\nPushing 6 elements into the stack continously: \n";
char ch = '+';
st.push(ch);
cout <<   ch << " is pushed into the stack <<|>>\n";
ch = 'c';
st.push(ch);
cout <<   ch << " is pushed into the stack <<|>>\n";
ch = 'b';
st.push(ch);
cout <<  ch << " is pushed into the stack <<|>>\n";
ch = '8';
st.push(ch);
cout << ch << " is pushed into the stack <<|>>\n";
ch = '2';
st.push(ch);
cout << ch << " is pushed into the stack <<|>>\n";
ch = 'z';
st.push(ch);
cout <<  ch << " is pushed into the stack <<|>>\n";

// popping elements
cout << "\nPopping 3 elements from the stack continously: \n";
char x = st.pop();
cout <<  x << " is popped out of the stack\n";
x = st.pop();
cout <<  x << " is popped out of the stack\n";
```

```
x = st.pop();
cout << x << " is popped out of the stack\n";

//checking of the stack is empty of not
cout << "\nChecking if the Stack is Empty or not\n";
if (st.isEmpty())
{
   cout << "\nStack is Empty" << endl;
}
else
{
   cout << "\nStack is not Empty" << endl;
}

//checking of the stack is full of not
cout << "\nChecking if the Stack is Full or not\n";
if (st.isFull())
{
   cout << "\n Stack is Full" << endl;
}
else
{
   cout << "\n Stack is not Full" << endl;
}

//returning the Last element of the Stack
char top = st.peek();
cout << "\n| Element on the Top of the Stack: " << top << endl;
cout << endl;

//returning the number of elements in the Stack
```

```
    st.numElements();
    cout << endl;

    //taking size of the expression form the user
    int len;
    cout << "\nEnter the length of your infix expression: ";
    cin >> len;
    char c[len];

    cout << "\nEnter the char of expression one by one: " << endl;
    for (int i = 0; i < len; i++)
    {
        cin >> c[i];
    }

    st.infixToPostfix(c, len);



    return 0;
}
```

```
Enter the size of stack you want to make: 7

 Stack of size 7 is created successfully

Pushing 6 elements into the stack continously:
+ is pushed into the stack <<|>>
c is pushed into the stack <<|>>
b is pushed into the stack <<|>>
8 is pushed into the stack <<|>>
2 is pushed into the stack <<|>>
z is pushed into the stack <<|>>

Popping 3 elements from the stack continously:
z is popped out of the stack
2 is popped out of the stack
8 is popped out of the stack

Checking if the Stack is Empty or not

Stack is not Empty

Checking if the Stack is Full or not

 Stack is not Full

| Element on the Top of the Stack: b

Number of Elements in the stack : 3

Enter the length of your infix expression: 7

Enter the char of expression one by one:
A
*
C
+
B
 is pushed into the stack <<|>>
 is pushed into the stack <<|>>
 is pushed into the stack <<|>>
 is pushed into the stack <<|>>

opping 3 elements from the stack continously:
 is popped out of the stack
 is popped out of the stack
 is popped out of the stack

hecking if the Stack is Empty or not

tack is not Empty

hecking if the Stack is Full or not

Stack is not Full

 Element on the Top of the Stack: b

umber of Elements in the stack : 3

nter the length of your infix expression: 7

nter the char of expression one by one:
A
*
C
+
B

rinting the Postfix Expression
AC*BD*+


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