Page:
Date: / /

Assignment 10

Class - SE IV Roll NO - 21430
Batch - F4
DOS - 4/12/2020

Problem statement

Implement C++ program for expression conversion as infix to postfix and its evaluation using stack based on given conditions
1. Operands and operator both must be single character.
2. Input postfix expression must be in desired format.
3. only +, -, * and '/' operands are expected

Objective :-

1. Understand the concept of how to convert infix to postfix.
2. Understand how to evaluate the expression using stack.

Outcomes :-

1. Will be able to understand concept of how to convert infix to postfix expression
2. Will be able to understand how to evaluate the expression using stack.

Theory :-

stack is abstract datatype and linear data structure. A stack is DS in which addn of new elem

of exiting element always takes place at a same end. This end is known as the the top of the stack. That means that it is possible to remove elements from a stack in a reverse order from the inseration of elements into the stack.
One other way to describing the stack is as last in first (LIFO) abstract data type and linear data structure.

Operations in Stack-:

1. Push.

Push operation refers to inserting an element in the stack. Since there is only one position at which the new element can be inserted. Top of the stack. The new element is inserted at top of the stack.

2. Pop.

Pop operation refers to the removal of an element. Again since we only have access to the element at the top of the stack, there is only one element that we can remove, we just remove the top of the stack.
We can also choose to return the value of popped element back but its the choice of programmer.

iii. Empty :-
It returns whether stack is empty or not
If top points to -1 it returns True value
otherwise it returns false value.

iv. Full :-
It returns whether stack is full or not
If top points to max-1 it returns True value otherwise it returns false value.

* Pseudocode

* ADT for a class Stack

```
class stack
    char data[Max]  // stores value.
    int top         // point to top element.

    initialize()  // initialize top value
    push()        // Push element in stack
    pop()         // remove element from stack
    full()        // check stack is full or not
    Empty()       // check stack is empty or not.
```

* Psuedocode for initialize

```
void initialize()
{
    top = -1;
}
```

* Pseudocode for push

```
void push()
{   ch = element to be inserted
    if (stack is not full)
    {   top = top+1;
        data[top] = ch;
    }
    END if
    else
        print "stack is full"

END.
}
```

* Pseudocode for pop

```
if (stack is not empty)
    ch = top element
    top = top-1
    return ch
END if
else
    print "stack is empty"

END
```

* Pseudocode to convert infix_to_postfix()

```
Input infix expression
While ( i is not infix.length)
      ch = infix[i]
      if (ch<='g' and ch>='0')
            append ch in postfix
      else
          if (stack is empty())
              s.push(ch)
          else
              while (stack is not empty)
                  sch <- s.pop
                  k <- priority (sch.ch);
                  if(k is 1)
                      append sch in postfix.
                  else
                      s.push (sch)
                      break.
                  END if
              END while
              s.push (ch)
          i <- i+1
END While.
While ( stack is not empty)
     append s.pop in postfix.
END while

END.
```

* Pseudocode for Evaluate

```
i = 0
j = 0
for i from 1 to postfix.length()
        ch ← postfix[i]
        if (ch >= '0' and ch <= '9')
                s.push(ch - 48)
        else
                if (ch is '+')
                        n1 ← s.pop
                        n2 ← s.pop
                        s.push(n2 + n1)
                else if (ch is '*')
                        n1 ← s.pop()
                        n2 ← s.pop()
                        s.push(n1 * n2)
                else if (ch is '-')
                        n1 ← s.pop()
                        n2 ← s.pop()
                        s.push(n2 - n1)
                else if (ch is '/')
                        n1 ← s1.pop()
                        n2 ← s.pop()
                        s.push(n2/n1
                END if
                i = i + 1
        END While
        return s.pop
END.
```

Test Cases

NO.	Description	Input	Expected o/p	Actual o/p	Result
1.	Infix Expression	2*3-1	2,3,*,1,- Ans-5	2,3,*,1,- Ans-5	Pass
2.	Infix Expression	5+2*6-10	5,2,6,*,+,10 - Ans-1	5,2,6,*,+, 10,- Ans-1	Pass

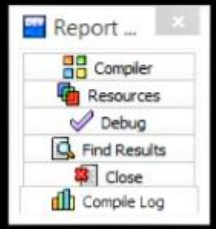
* Complexity.

Function	Time Complexity	Space Complexity
Initialize	O(1)	O(1)
push()	O(1)	O(1)
pop()	O(1)	O(1)
empty()	O(1)	O(1)
Full()	O(1)	O(1)
main()	O(1)	O(1)
convert()	$O(n^2)$	O(n)
evaluate()	$O(n^2)$	O(n)
priority()	O(1)	O(1)

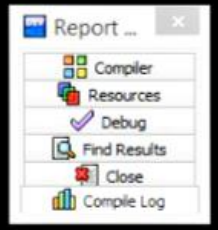
Conclusion.

We learnt to implement Stack Data structure in C++.

```cpp
#include<iostream>
#include<stdlib.h>
using namespace std;
#define MAX 30
class Stack
{
    int top;
    int data[MAX];
    public:
        Stack()
        {
            top=-1;
        }
        void push(int x)
        {
            if(!full())
            {
                top++;
                data[top] = x;
            }
            else
            {
                cout<<"\nStack is Full\n";
            }
        }
        int pop()
        {
            int x;
            if(!empty())
            {
                x=data[top];
                top--;
```

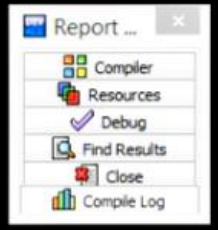


```
        }
        else
        {
            cout<<"\nStack is empty\n";
            return 0;
        }
    }
    int full()
    {
        if(top==MAX-1)
        {
            return 1;
        }
        return 0;
    }
    int empty()
    {
        if(top==-1)
        {
            return 1;
        }
        return 0;
    }
    int priority(int op)
    {
        if(op=='*'||op=='/')
        {
            return 2;
        }
        else if(op=='+'||op=='-')
        {
            return 1;
```

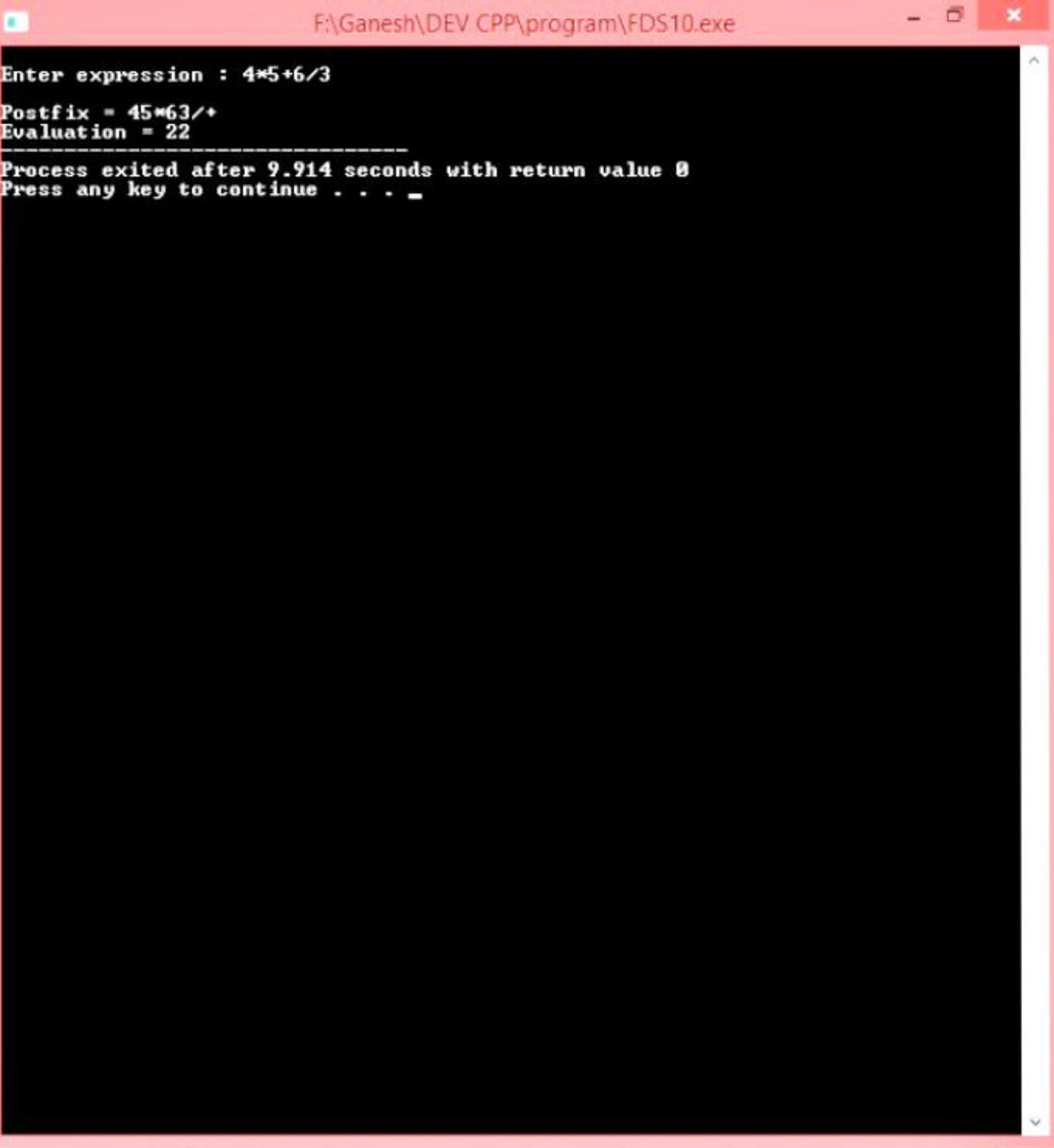


```
66              }
67          }
68  };
69
70  void convert(char infix[30],char postfix[30])
71  {
72      Stack st;
73      char expr,st_op;
74      int i,j;
75      i=0;
76      j=0;
77      while(infix[i]!='\0')
78      {
79          expr=infix[i];
80          if(expr=='*'||expr=='/'||expr=='+'||expr=='-')
81          {
82              if(st.empty())
83              {
84                  st.push(expr);
85              }
86              else
87              {
88                  while(!st.empty())
89                  {
90                      st_op=st.pop();
91                      if(st.priority(st_op)>=st.priority(expr))
92                      {
93                          postfix[j]=st_op;
94                          j++;
95                      }
96                      else
97                      {
```

```
                    break;
                }
            }
            st.push(expr);
        }
    }
    else
    {
        postfix[j]=expr;
        j++;
    }
    i++;
}
while(!st.empty())
{
    postfix[j]=st.pop();
    j++;
}
postfix[j]='\0';
}
int evaluate(char postfix[30])
{
    Stack st;
    char num[2];
    num[1]='\0';
    char st_op;
    int value,i;
    i=0;
    while(postfix[i]!='\0')
    {
        st_op=postfix[i];
        if(st_op=='*'||st_op=='/'||st_op=='+'||st_op=='-')
```



```
            int ans;
            x1=st.pop();
            x2=st.pop();
            if(st_op=='*')
            {
                ans = x2*x1;
                st.push(ans);
            }
            else if(st_op=='/')
            {
                ans = x2/x1;
                st.push(ans);
            }
            else if(st_op=='+')
            {
                ans = x2+x1;
                st.push(ans);
            }
            else if(st_op=='-')
            {
                ans = x2-x1;
                st.push(ans);
            }
        }
        else
        {
            value=st_op-48;
            st.push(value);
        }
        i++;
    }
    return st.pop();
```


F:\Ganesh\DEV CPP\program\FDS10.exe
Enter expression : 4*5+6/3
Postfix = 45*63/+
Evaluation = 22

Process exited after 9.914 seconds with return value 0
Press any key to continue . . .