Name : Devansh Manish Mhatre

UID no : 2023300147

Experiment no : 1

Aim

Evaluation of Post-fix expression using Stack.

Theory

A stack is a simple data structure in computer science that follows the Last In, First Out (LIFO) principle. This means that the last element added to the stack will be the first one to be removed.

In postfix expressions, also known as Reverse Polish Notation (RPN), every operator follows all of its operands.

In infix expressions, every operator in present between its operands.

Algorithm

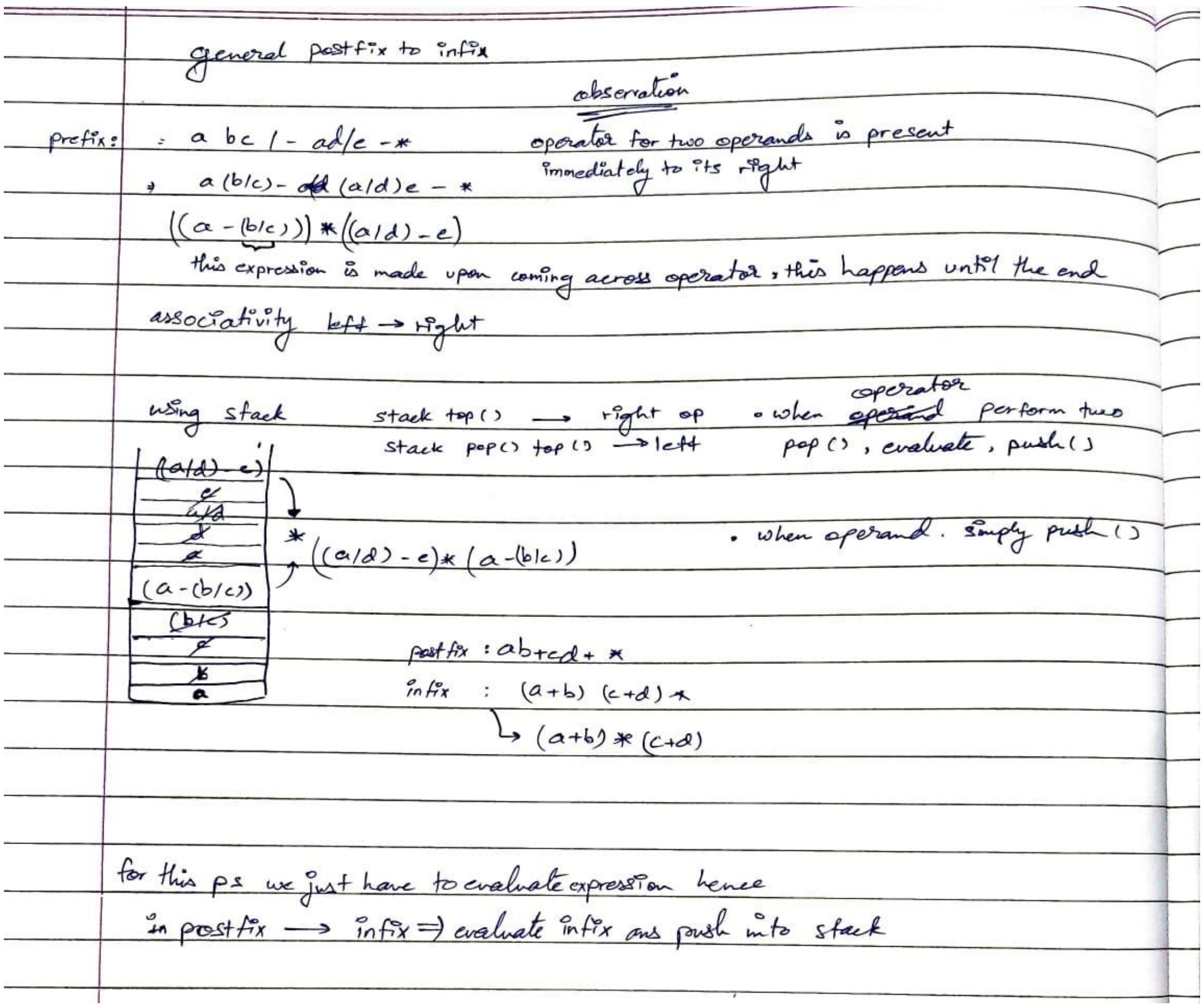
1.Take input as string , initialise a empty stack , iterate through characters of string using a for loop.

2.if the current character is operand , simply push into stack

3.if current character is a operator , pop last to elements of stack , evaluate the expression , push into stack.

4.End of loop iteration stack would only have one element (complete expression / answer in this case). Print stack top()

Problem-solving on the concept (students are required to solve the problem on paper and add its snapshot. The paper should have their name written on it)



general postfix to infix

observation

operator for two operands is present immediately to its right

prefix: : a bc / - ad/e - *

⇒ a (b/c) - ~~ad~~ (a/d) e - *

((a - (b/c))) * ((a/d) - e)

this expression is made upon coming across operator, this happens until the end

associativity left → right

using stack

stack top() → right op

stack pop() top() → left

• when ~~operand~~ operator perform two pop(), evaluate, push()

• when operand. simply push()

~~((a/d) - e)~~
~~e~~
~~a/d~~
~~d~~
~~a~~
(a - (b/c))
~~(b/c)~~
~~c~~
~~b~~
a

* ((a/d) - e) * (a - (b/c))

postfix : ab+cd+ *

infix : (a+b) (c+d) *

↳ (a+b) * (c+d)

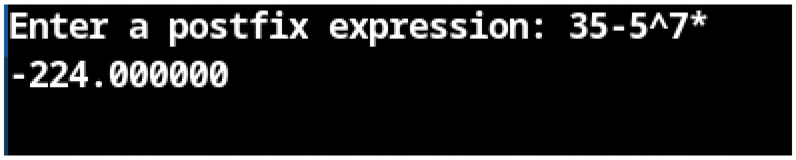
for this ps we just have to evaluate expression hence in postfix → infix ⇒ evaluate infix ans push into stack

Program Code:

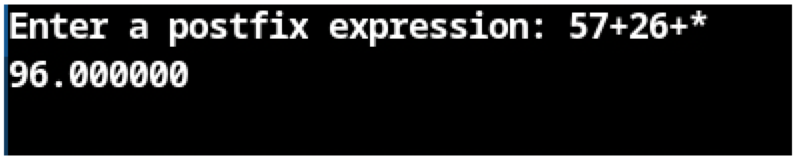
```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

typedef struct{
    int size;
    int top;
    float *Arr;
}Stack;

void stackPush(Stack* self,float data){
    if(self->size<=self->top){
        return;
    }
    self->Arr[++self->top]=data;



}
void stackPop(Stack* self){
    if(self->top<0){
        return ;
    }
    self->Arr[self->top--]=0.00;

}
Stack* constructStack(int size){
    Stack* stack=(Stack*)malloc(sizeof(Stack));
  stack->Arr = (float*) malloc(size*sizeof(char));
  stack->top=-1;
  stack->size=size;
  return stack;
}

int stackSize(Stack* self){
    return (self->top)+1;
}

float stackTop(Stack* self){
    return self->Arr[self->top];
}

void printStack(Stack* self){
    int curSize=(self->top)+1;
     printf("Elements of Stack : \n");
    for(int i=0;i<curSize;i++){
        printf("%f ",self->Arr[i]);
    }
}

int main()
{
    Stack* stack = constructStack(5);
    char* s = "35-5^7*";
    int stringSize =  (sizeof(s)/sizeof(s[0]))-1;
    //printf(" %d ",stringSize);

   for(int i=0;i<stringSize;i++){
       if(s[i]>='0' && s[i]<='9'){
           int floatnum = (float) s[i]-'0';
            stackPush(stack,floatnum);
       }

       else if (s[i]=='*'){
           float rightOp = stackTop(stack);
           stackPop(stack);
           float leftOp = stackTop(stack);
           stackPop(stack);
           float res = leftOp*rightOp;
           stackPush(stack,res);
       }
       else if (s[i]=='+'){
           float rightOp = stackTop(stack);
           stackPop(stack);
           float leftOp = stackTop(stack);
           stackPop(stack);
           float res = leftOp+rightOp;
           stackPush(stack,res);
       }
       else if (s[i]=='-'){
            float rightOp = stackTop(stack);
           stackPop(stack);
           float leftOp = stackTop(stack);
           stackPop(stack);
           float res = leftOp-rightOp;
           stackPush(stack,res);
       }
       else if (s[i]=='/'){
            float rightOp = stackTop(stack);
           stackPop(stack);
           float leftOp = stackTop(stack);
           stackPop(stack);
           float res = leftOp/rightOp;
           stackPush(stack,res) ;
       }
        else if (s[i]=='^'){
            float rightOp = stackTop(stack);
           stackPop(stack);
           float leftOp = stackTop(stack);
           stackPop(stack);
           float res = (float) pow((double)leftOp,(double)rightOp);
           stackPush(stack,res) ;
       }
   }
   printf("%f",stackTop(stack));
```

```
return 0;
}
```

Output screenshot:



```
Enter a postfix expression: 35-5^7*
-224.000000
```



```
Enter a postfix expression: 57+26+*
96.000000
```

Conclusion:

Successfully understood and evaluated a postfix expression using a stack. From this process, and derived the algorithm to convert and evaluate postfix expressions efficiently.