**ASSIGNMENT – 22**

**PROBLEM STATEMENT**

Write a program in C to implement evaluation of postfix expressions.

**ALGORITHMS**

Algorithm **Evaluate**

**Input:** The input array holding the postfix expression to be evaluated

**Output:** Result of the input postfix expression.

**Remarks:** It is assumed that the input expression is valid

**Steps;**

1. i=0,top=0
2. **While** end of input expression is not reached, **repeat steps 3 to 13**
3. **If** the element at i is an operand **then**
4. **While** a blank space is not encountered **repeat steps 5 to 6**
5. **Push**(stack,operand,top)
6. i=i+1
7. **Else If** the element at i is an operator **then**
8. **While** a blank space is not encountered **repeat steps 9 to 13**
9. y=**Pop**(stack,top) //pop second operand
10. x=**Pop**(stack,top) //pop first operand
11. res=**solve**(x,operator,y)//solve present operator and operands
12. **Push**(stack,top,res) //push result back into the stack
13. i=i+1 //increment i by one
14. res=**Pop**(stack,top) //pop final result from stack
15. **Return** res
16. **Stop**

Algorithm **Push**

**Input:** The stack in which elements are to be pushed, the value of top indicator and the element ‘item’ to be inserted

**Output:** The element ‘item’ inserted into the stack at appropriate position.

**Remarks:** The value of top must be passed as pointer

**Steps:**

1. top=top+1 //increment value of top by one
2. stack[top]=item //insert item at top of stack
3. **Return**
4. **Stop**

Algorithm **Pop**

**Input:** The stack from which elements are to be popped and it’s top indicator.

**Output:** The element at top removed from the stack and returned.

**Remarks:** top must be passed in as a pointer

**Steps:**

1. ele=stack[top] //stop element at top in ele
2. top=top-1 //decrement top by one
3. **Return** ele
4. **Stop**

Algorithm **Solve**

**Input:** Two operands x and y along with the operator opr

**Output:** The operands evaluated as per (x “operator” y)

**Remarks:** The supported operations are addition,subtraction and multiplication

**Steps:**

1. **If**(operator= \* ) **then** //for multiplication
2. res=x\*y
3. **Else If**(operator= / ) **then** //for division
4. **If**(y=0) **then**
5. Print “Divide by zero error”
6. **Exit**
7. **EndIf**
8. **Else If**(operator= + ) **then** //for addition
9. res=x+y
10. **Else** //for subtraction
11. res=x-y
12. **EndIf**
13. **Return** res
14. **Stop**

**3.SOURCE CODE**

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

//function to solve for present operator and operands

int solve(int x,char opr,int y)

{

    int res;

    if(opr=='\*')

        res = x\*y;

    else if(opr=='/')

    {

        if(y==0)

        {

            printf("Divide by zero error...please retry");

            exit(1);

        }

        res = x/y;

    }

    else if(opr=='+')

        res = x+y;

    else

        res = x-y;

    return res;

}

//function to push elements into stack

void push(int \*stack,int \*top,int opr)

{

    (\*top)++;

    stack[\*top]=opr;

}

//function to pop elements from stack

int pop(int \*stack,int \*top)

{

    int ele;

    if(\*top==-1)

        printf("Empty Stack");

    else

    {

        ele=stack[\*top];

        (\*top)--;

    }

    return ele;

}

//function to evaluate a given postfix expression

int evaluate(char \*expr)

{

    int i=0,count=0,opr=0,x,y,res,stack[100],top=-1;

    while(expr[i]!='\0') //while the end of string is not reached

    {

        if(expr[i]>47 && expr[i]<58) //if element at i is an operand

        {

            while(expr[i]!=' ') //while a space is not encountered

            {

                opr=(opr\*pow(10,count))+(expr[i]-48); //calculate operand

                count++;i++;

            }

            push(stack,&top,opr); //push operand into stack

            opr=0;count=0;//reset opr and count to zero for next iteration

        }

        else //an operator is encountered

        {

            while(expr[i]!=' ' && expr[i]!='\0')

            {

                if(expr[i]>41 && expr[i]<48) //if element at i is operator

                {

                    y=pop(stack,&top); //pop second operand

                    x=pop(stack,&top); //pop first operand

                    res=solve(x,expr[i],y); //solve

                    push(stack,&top,res); //push result into stack

                    i++;

                }

            }

        }

        if(expr[i]=='\0') //if end of the string is reached

            break;

        else

            i++;

    }

    res=pop(stack,&top); //pop final result from the stack

    return res;

}

int main(void)

{

    char expr[100];

    int res;

    printf("Enter the postfix expression\nPlease separate adjacent numbers and operators using spaces:\n");

    gets(expr);

    printf("Entered expression: ");

    puts(expr);

    res=evaluate(expr);

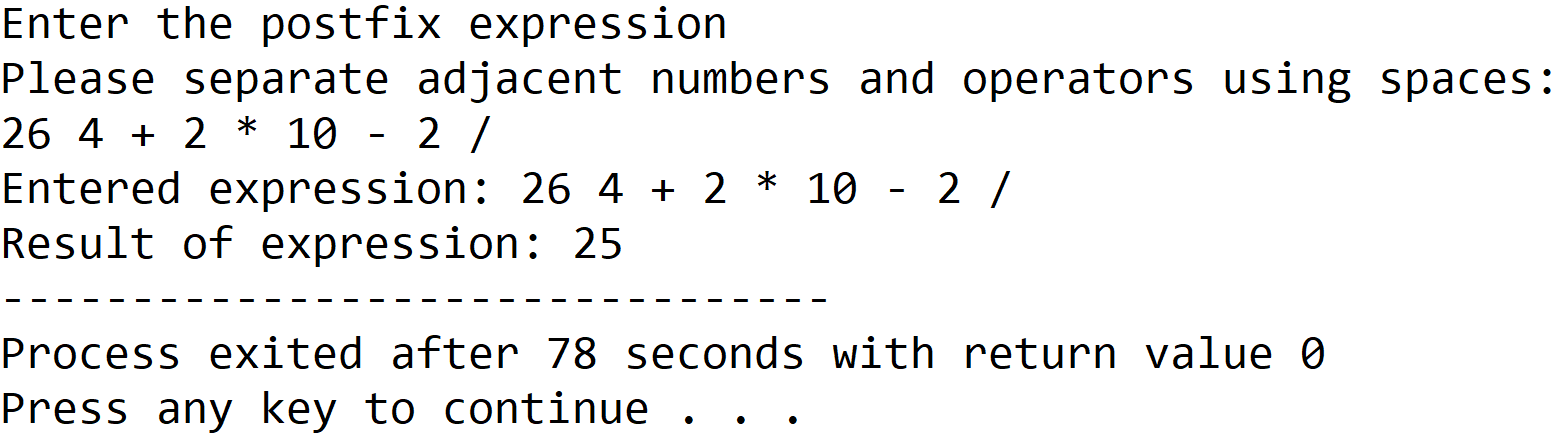
    printf("Result of expression: %d",res);

    return 0;

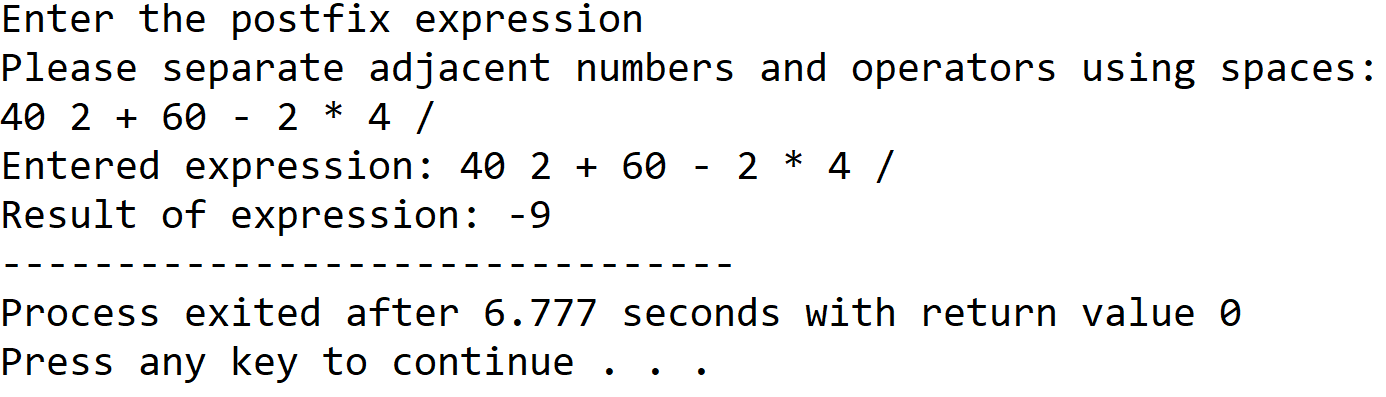
}

**4.OUTPUT**

**SET 1:** Positive integer as result

****

**SET 2:** Negative integer as result



**5.DISCUSSIONS**

**Variable Description**

* **expr:** string to hold the user input postfix expression.
* **res:** To store result of a calculation.
* **Opr:** To calculate value of operand.
* **x,y:** To hold values of operands.
* **Stack:** An integer array representing a stack.
* **Top:** pointer to the topmost element in the stack.
* **i,count:** loop counter.

**Limitations**

* It is assumed that the user enters a valid postfix expression and the if the input postfix expression is invalid, undesired output will be received.
* Arrays have been used to hold the postfix expression and also to represent the stack, since they are static data structures, their size cannot be manipulated once they are constructed in the memory.

**Uses**

* The program can be used to evaluate a postfix expression with integer operands and the above mentioned operations, the program can find implementation in a calculator application.

**Future Scope**

* The data structures can be replaced from arrays to linked lists making the program more memory effiecient and terminating the dependency on contiguous memory locations.