

LAB – 4

SOLVED EXERCISE:

1) Program for evaluation of postfix expression in C.

headerfile : postfixfunc.h

Code :

```
#define MAX 20
typedef struct stack
{
    int data[MAX];
    int top;
} stack;

void init(stack *);
int empty(stack *);
int full(stack *);
int pop(stack *);
void push(stack *,int);
int evaluate(char x,int op1,int op2);
int evaluate(char x,int op1,int op2)
{
    if(x=='+')
        return(op1+op2);
    if(x=='-')
        return(op1-op2);
    if(x=='*')
        return(op1*op2);
    if(x=='/')
        return(op1/op2);
    if(x=='%')
        return(op1%op2);
}
void init(stack *s)
{
    s->top=-1;
}
int empty(stack *s)
{
    if(s->top==-1)return(1);
    return(0);
}
int full(stack *s)
{
    if(s->top==MAX-1)return(1);
    return(0);
}
void push(stack *s,int x)
{
    s->top=s->top+1;
```

```

    s->data[s->top]=x;
}
int pop(stack *s)
{
    int x;
    x=s->data[s->top];
    s->top=s->top-1;
    return(x);
}

```

Main function : postfixeval.c

Code :

```

#include <stdio.h>
#include <ctype.h>
#include "postfixfunc.h"

int main()
{
    stack s;
    char x;
    int op1,op2,val;
    init(&s);
    printf("Enter the expression(eg: 59+3*)\nsingle digit operand and operators only:");
    while((x=getchar())!='\n')
    {
        if(isdigit(x))
            push(&s,x-'0');    /*x-'0'for removing the effect of ascii */
        else
        {
            op2=pop(&s);
            op1=pop(&s);
            val=evaluate(x,op1,op2);
            push(&s,val);
        }
    }
    val=pop(&s);
    printf("\nvalue of expression=%d\n",val);
    return 0;
}

```

Output :

```

Student@project-lab:~/Desktop/200905130/DSAlab4/programs$ gcc postfixfunc.h
Student@project-lab:~/Desktop/200905130/DSAlab4/programs$ gcc postfixeval.c
Student@project-lab:~/Desktop/200905130/DSAlab4/programs$ ./a.out
Enter the expression(eg: 59+3*)
single digit operand and operators only:24+3*
value of expression=18

```

Questions for Lab4

Write a C program to :

1) Evaluate a given prefix expression using stack.

Code :

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define STACK_SIZE 50
#define EXPR_SIZE 50

typedef enum {lparan,rparan,plus,minus,times,divide,mod,eos,operand} PRECEDENCE;
int stack[STACK_SIZE];
char expr[EXPR_SIZE];
void push(int *top,int item)
{
    stack[++(*top)]=item;
}

int pop(int *top)
{
    return stack[(--*top)];
}

PRECEDENCE get_token(char *symbol,int *n)
{
    *symbol = expr[(--*n)];
    switch(*symbol)
    {
        case '+':
            return plus;
        case '-':
            return minus;
        case '*':
            return times;
        case '/':
            return divide;
        case '%':
            return mod;
        case '(':
            return lparan;
        case ')':
            return rparan;
        case '\0':
            return eos;
        default :
            return operand;
    }
}

int eval()
```

```

{
    PRECEDENCE token;
    char symbol;
    int n=0,c;
    int op1,op2;
    int top=-1;
    token = get_token(&symbol,&n);
    while(token!=eos)
    {
        if(token==operand)
        {
            c = symbol - '0';
            push(&top,c);
        }
        else
        {
            op2 = pop(&top);
            op1 = pop(&top);
            if(token==plus)
            {
                push(&top,op2+op1);
            }
            else if(token==minus)
            {
                push(&top,op2-op1);
            }
            else if(token==times)
            {
                push(&top,op2*op1);
            }
            else if(token==divide)
            {
                push(&top,op2/op1);
            }
            else if(token==mod)
            {
                push(&top,op2%op1);
            }
        }
        token = get_token(&symbol,&n);
    }

    //printf("%d\n",stack[top]);
    }
    return pop(&top);
}

int main()
{
    char ex[50];
    int i,j=0;
    printf("Enter a prefix expression : \n");
    scanf("%s",ex);

```

```

for(i=strlen(ex)-1; i>=0; i--)
{
    expr[j]=ex[i];
    j++;
}
expr[j]='\0';
printf("Reverse is %s\n",expr);
printf("\nAnswer: %d\n",eval());
return 0;
}

```

Output :

```

Student@project-lab:~/Desktop/200905130/DSAlab4/programs$ gcc 4_1.c
Student@project-lab:~/Desktop/200905130/DSAlab4/programs$ ./a.out
Name : Manoj M Mallya
Reg.no : 200905130
Batch : C2

Enter a prefix expression :
+4*26
Reverse is 62*4+

Answer: 16

```

2)Convert an infix expression to prefix.

Code :

```

#define size 50
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct
{
    char s[size];
    int top;
} STACK;

void push(STACK *s1,char elem)
{
    s1->top++;
    s1->s[s1->top]=elem;
}

char pop(STACK *s1)
{
    return s1->s[s1->top--];
}

```

```

int pre(char elem)
{
    switch(elem)
    {
        case '#':
            return 0;
        case '(':
            return 1;
        case '+':
        case '-':
            return 2;
        case '*':
        case '/':
            return 3;
        case '$':
            return 4;
    }
}

```

```

int main()
{
    printf("Name : Manoj M Mallya\nReg.no : 200905130\nBatch : C2\n\n");
    STACK s1;
    s1.top=-1;

    char infix[50], prefix[50], revinf[50], revpref[50], elem, temp, ch;
    int i=0,k=0;

    printf("Enter infix expression\n");
    scanf("%s",infix);
    int len = strlen(infix);
    push(&s1,'#');
    for(i=len-1; i>=0; i--)
    {
        revinf[k]=infix[i];
        k++;
    }

    revinf[k]='\0';
    i=0,k=0;

    while((ch=revinf[i++])!='\0')
    {
        switch(ch)
        {
            case ')':
                push(&s1,ch);
                printf("PUSH %c\n",ch);
                break;
            case '(':
                while(s1.s[s1.top]!='')
                {

```

```

        temp=pop(&s1);
        prefix[k++]=temp;
    }
    elem=pop(&s1);
    break;
case '+':
case '-':
case '*':
case '/':
    while(pre(s1.s[s1.top])>pre(ch))
    {
        temp=pop(&s1);
        prefix[k++]=temp;
        printf("POP and APPEND %c\n",temp);
    }

    push(&s1,ch);
    printf("PUSH %c\n", ch);
    break;

case '$':
    if(s1.s[s1.top]=='$')
    {
        temp=pop(&s1);
        prefix[k++]=temp;
        printf("POP and APPEND %c\n",temp);
    }
    push(&s1,'$');
    break;
default:
    prefix[k++]=ch;
}
}
int m=0;
while(s1.s[s1.top]!='#')
{
    temp=pop(&s1);
    prefix[k++]=temp;
}

prefix[k]='\0';
for(i=len-1; i>=0; i--)
{
    revpref[m]=prefix[i];
    m++;
}

revpref[m]='\0';
printf("\n\nGiven Infix Expn: %s Prefix Expn: %s\n",infix,revpref);
return 0;
}

```

Output :

```
Student@project-lab:~/Desktop/200905130/DSAlab4/programs$ gcc 4_2.c
Student@project-lab:~/Desktop/200905130/DSAlab4/programs$ ./a.out
Name : Manoj M Mallya
Reg.no : 200905130
Batch : C2

Enter infix expression
3+8*7
PUSH *
POP and APPEND *
PUSH +

Given Infix Expn: 3+8*7 Prefix Expn: +3*87
```

3)Implement two stacks in an array.

Code :

```
#include <stdio.h>
#define SIZE 10

int ar[SIZE];
int top1 = -1;
int top2 = SIZE;

//Functions to push data
void push_stack1 (int data)
{
    if (top1 < top2 - 1)
    {
        ar[++top1] = data;
    }
    else
    {
        printf ("Stack Full! Cannot Push\n");
    }
}

void push_stack2 (int data)
{
    if (top1 < top2 - 1)
    {
        ar[--top2] = data;
    }
    else
    {
        printf ("Stack Full! Cannot Push\n");
    }
}
```



```
}  
}
```

//Functions to pop data

```
void pop_stack1 ()  
{  
    if (top1 >= 0)  
    {  
        int popped_value = ar[top1--];  
        printf ("%d is being popped from Stack 1\n", popped_value);  
    }  
    else  
    {  
        printf ("Stack Empty! Cannot Pop\n");  
    }  
}
```

```
void pop_stack2 ()  
{  
    if (top2 < SIZE)  
    {  
        int popped_value = ar[top2++];  
        printf ("%d is being popped from Stack 2\n", popped_value);  
    }  
    else  
    {  
        printf ("Stack Empty! Cannot Pop\n");  
    }  
}
```

//Functions to Print Stack 1 and Stack 2

```
void print_stack1 ()  
{  
    int i;  
    printf("Stack 1 :- ");  
    for (i = top1; i >= 0; --i)  
    {  
        printf ("%d ", ar[i]);  
    }  
    printf ("\n");  
}
```

```
void print_stack2 ()  
{  
    int i;  
    printf("Stack 2 :- ");  
    for (i = top2; i < SIZE; ++i)  
    {  
        printf ("%d ", ar[i]);  
    }  
    printf ("\n");  
}
```

```

int main()
{
    printf("Name : Manoj M Mallya\nReg.no : 200905130\nBatch : C2\n\n");

    int ar[SIZE];
    int i;
    int num_of_ele;

    printf ("We can push a total of 10 values\n");

    //Number of elements pushed in stack 1 is 6

    //Number of elements pushed in stack 2 is 4

    for (i = 1; i <= 6; ++i)
    {
        push_stack1 (i);
        printf ("Value Pushed in Stack 1 is %d\n", i);
    }

    for (i = 1; i <= 4; ++i)
    {
        push_stack2 (i);
        printf ("Value Pushed in Stack 2 is %d\n", i);
    }

    //Print Both Stacks
    print_stack1 ();

    print_stack2 ();

    //Pushing on Stack Full
    printf ("Pushing Value in Stack 1 is %d\n", 11);
    push_stack1 (11);

    //Popping All Elements From Stack 1
    num_of_ele = top1 + 1;
    while (num_of_ele)
    {
        pop_stack1 ();
        --num_of_ele;
    }

    //Trying to Pop From Empty Stack
    pop_stack1 ();

    return 0;
}

```

Output :

```
Student@project-lab:~/Desktop/200905130/DSAlab4/programs$ gcc 4_3.c
Student@project-lab:~/Desktop/200905130/DSAlab4/programs$ ./a.out
Name : Manoj M Mallya
Reg.no : 200905130
Batch : C2

We can push a total of 10 values
Value Pushed in Stack 1 is 1
Value Pushed in Stack 1 is 2
Value Pushed in Stack 1 is 3
Value Pushed in Stack 1 is 4
Value Pushed in Stack 1 is 5
Value Pushed in Stack 1 is 6
Value Pushed in Stack 2 is 1
Value Pushed in Stack 2 is 2
Value Pushed in Stack 2 is 3
Value Pushed in Stack 2 is 4
Stack 1 :- 6 5 4 3 2 1
Stack 2 :- 4 3 2 1
Pushing Value in Stack 1 is 11
Stack Full! Cannot Push
6 is being popped from Stack 1
5 is being popped from Stack 1
4 is being popped from Stack 1
3 is being popped from Stack 1
2 is being popped from Stack 1
1 is being popped from Stack 1
Stack Empty! Cannot Pop
```
