Name: KAUSHAL VASHISTH   Roll Number: 18ETCS002147

Laboratory 5

1. Questions

    1. Implement the PUSH, POP and PRINT operations on stack
    2. Write a C program to convert infix to postfix notation using stack
    3. Write a C program to evaluate a postfix expression.

2. Algorithms:-

Q1:- Implement the PUSH, POP and PRINT operations on stack:-

Push operation:-

- **Step 1** – Checks if the stack is full.
- **Step 2** – If the stack is full, prints "stack is full"
- **Step 3** – If the stack is not full, increments **top** to point next empty space.
- **Step 4** – Adds data element to the stack location, where top is pointing.

Pop operation:-

- **Step 1** − Checks if the stack is empty.
- **Step 2** − If the stack is empty, prints "stack is empty"
- **Step 3** − If the stack is not empty, accesses the data element at which **top** is pointing.
- **Step 4** − Decreases the value of top by 1.

Print operation:-

- **Step 1** − Checks if the stack is empty.
- **Step 2** − If the stack is empty, prints "stack is empty"
- **Step 3** − If the stack is not empty, print the elements of stack by running 'for loop' from top to -1.

Main function :-

- **Step 1** – run do while loop if choice not eual to 4.
- **Step 2** – print “1: push 2:pop 3:print 4:exit ” and input int variable choice
- **Step 3** – make use of switch(choice) statement to perform step 2 cases and hence call the respective functions.

Q2:- C program to convert infix to postfix notation using stack:-

Algorithm:-

1. Scan the infix expression from left to right.
2. If the scanned character is an operand, output it.
3. Else,
…..3.1 If the precedence of the scanned operator is greater than the precedence of the operator in the stack(or the stack is empty or the stack contains a ‘(‘), push it.

…..3.2 Else, Pop all the operators from the stack which are greater than or equal to in precedence than that of the scanned operator. After doing that Push the scanned operator to the stack. (If you encounter parenthesis while popping then stop there and push the scanned operator in the stack.)

4. If the scanned character is an ‘(‘, push it to the stack.
5. If the scanned character is an ‘)’, pop the stack and and output it until a ‘(‘ is encountered, and discard both the parenthesis.
6. Repeat steps 2-6 until infix expression is scanned.
7. Print the output
8. Pop and output from the stack until it is not empty.

Q3:- C program to evaluate a postfix expression:-

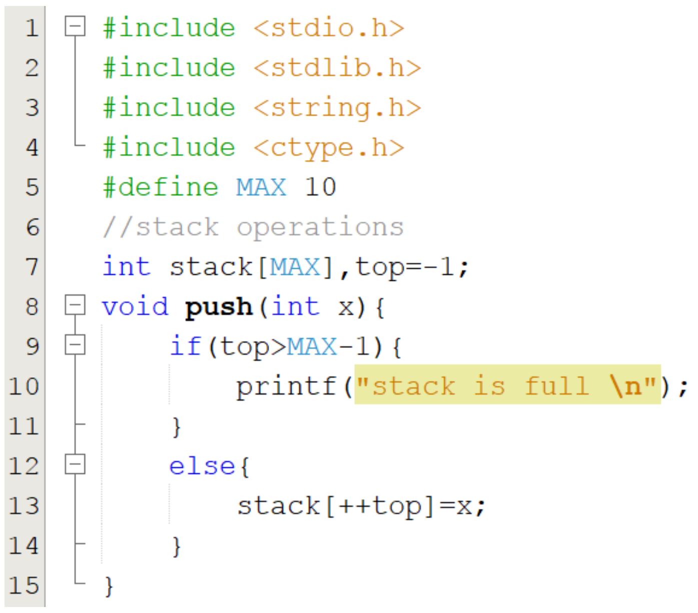
Algorithm:-

1) Create a stack to store operands (or values).
2) Scan the given expression and do following for every scanned element.
…..a) If the element is a number, push it into the stack
…..b) If the element is a operator, pop operands for the operator from stack.
Evaluate the operator and push the result back to the stack
3) When the expression is ended, the number in the stack is the final answer

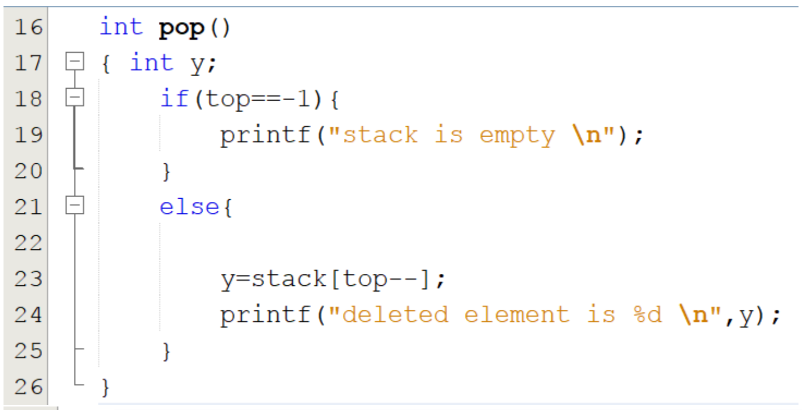
3. Program:

Q1: [Implement the PUSH, POP and PRINT operations on stack]:-

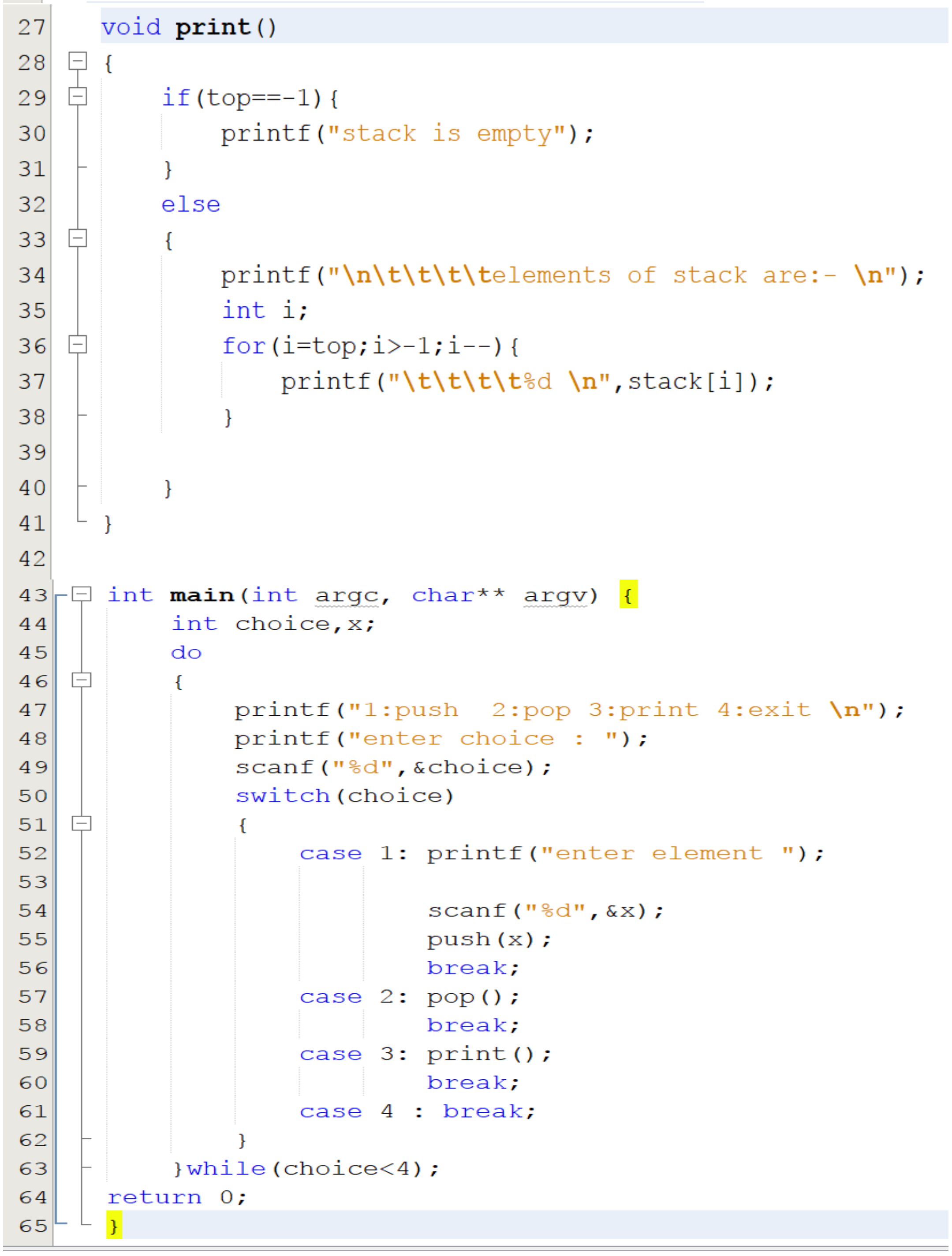
Name: KAUSHAL VASHISTH Roll Number: 18ETCS002147



```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#define MAX 10
//stack operations
int stack[MAX],top=-1;
void push(int x){
    if(top>MAX-1){
        printf("stack is full \n");
    }
    else{
        stack[++top]=x;
    }
}
```

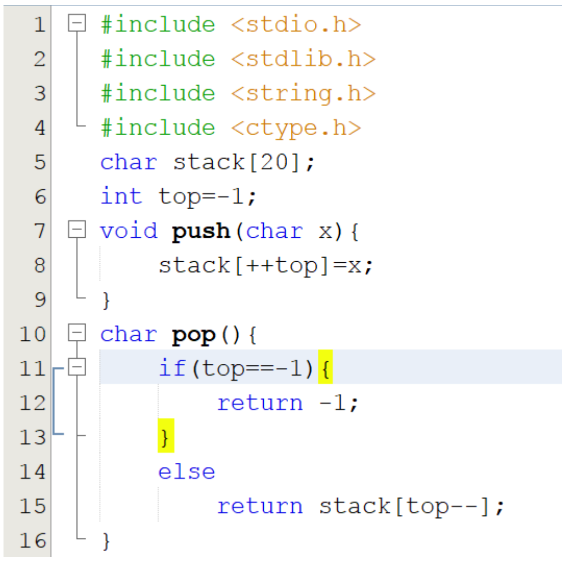


```c
int pop()
{ int y;
    if(top==-1){
        printf("stack is empty \n");
    }
    else{

        y=stack[top--];
        printf("deleted element is %d \n",y);
    }
}
```

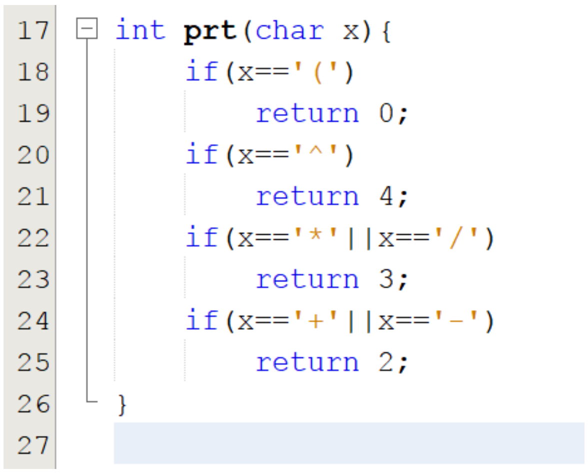


```c
void print()
{
    if(top==-1){
        printf("stack is empty");
    }
    else
    {
        printf("\n\t\t\t\telements of stack are:- \n");
        int i;
        for(i=top;i>-1;i--){
            printf("\t\t\t\t%d \n",stack[i]);
        }

    }
}

int main(int argc, char** argv) {
    int choice,x;
    do
    {
        printf("1:push  2:pop 3:print 4:exit \n");
        printf("enter choice : ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1: printf("enter element ");

                    scanf("%d",&x);
                    push(x);
                    break;
            case 2: pop();
                    break;
            case 3: print();
                    break;
            case 4 : break;
        }
    }while(choice<4);
return 0;
}
```

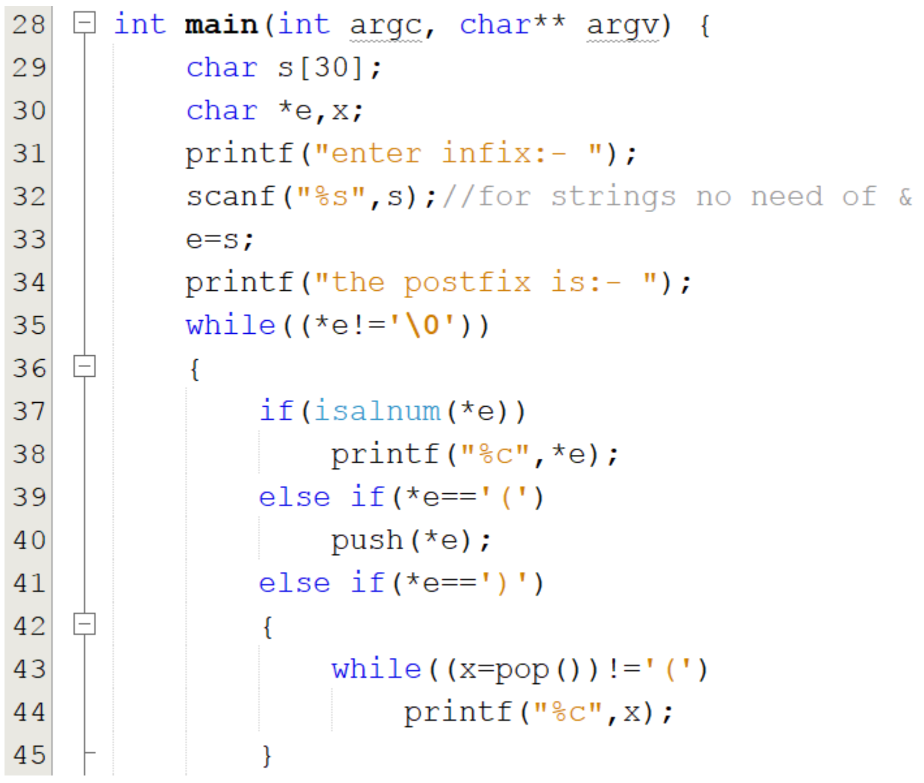
Q2: Write a C program to convert infix to postfix notation using stack:-



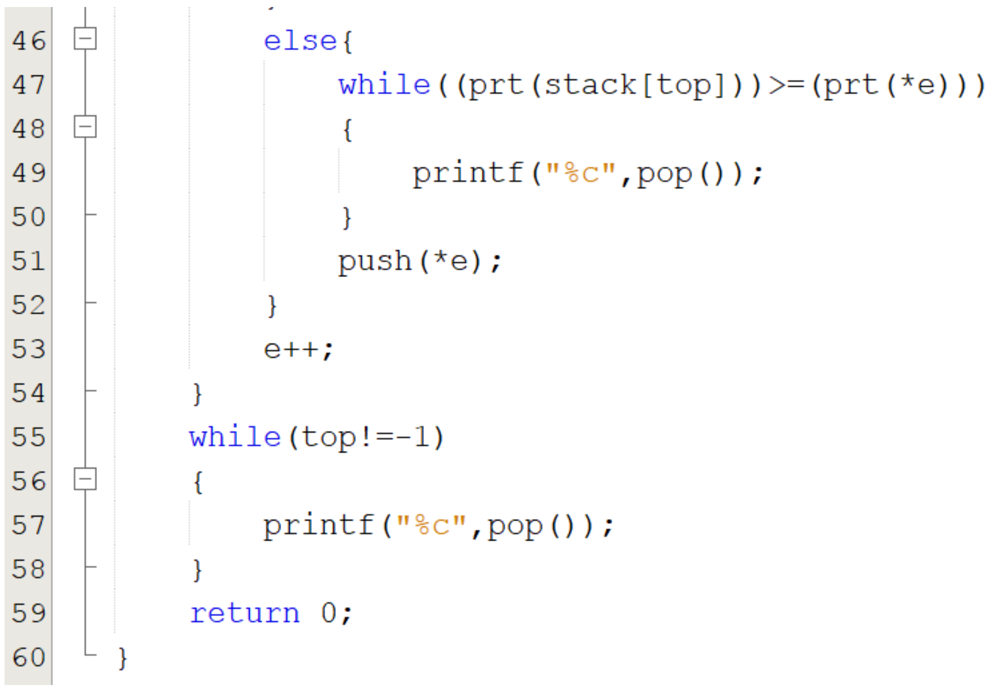
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
char stack[20];
int top=-1;
void push(char x){
    stack[++top]=x;
}
char pop(){
    if(top==-1){
        return -1;
    }
    else
        return stack[top--];
}
```



```c
int prt(char x){
    if(x=='(')
        return 0;
    if(x=='^')
        return 4;
    if(x=='*'||x=='/')
        return 3;
    if(x=='+'||x=='-')
        return 2;
}
```

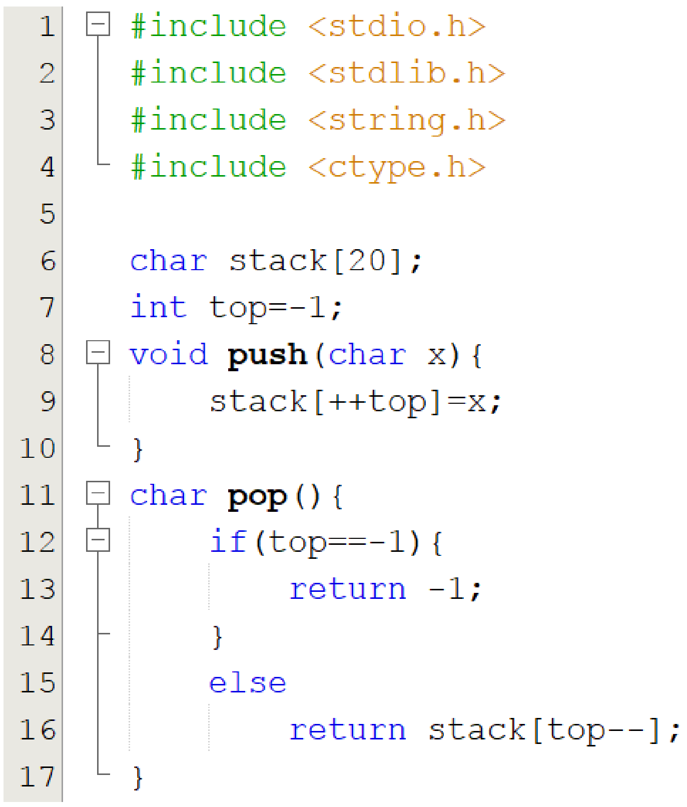


```c
int main(int argc, char** argv) {
    char s[30];
    char *e,x;
    printf("enter infix:- ");
    scanf("%s",s);//for strings no need of &
    e=s;
    printf("the postfix is:- ");
    while((*e!='\0'))
    {
        if(isalnum(*e))
            printf("%c",*e);
        else if(*e=='(')
            push(*e);
        else if(*e==')')
        {
            while((x=pop())!='(')
                printf("%c",x);
        }
```

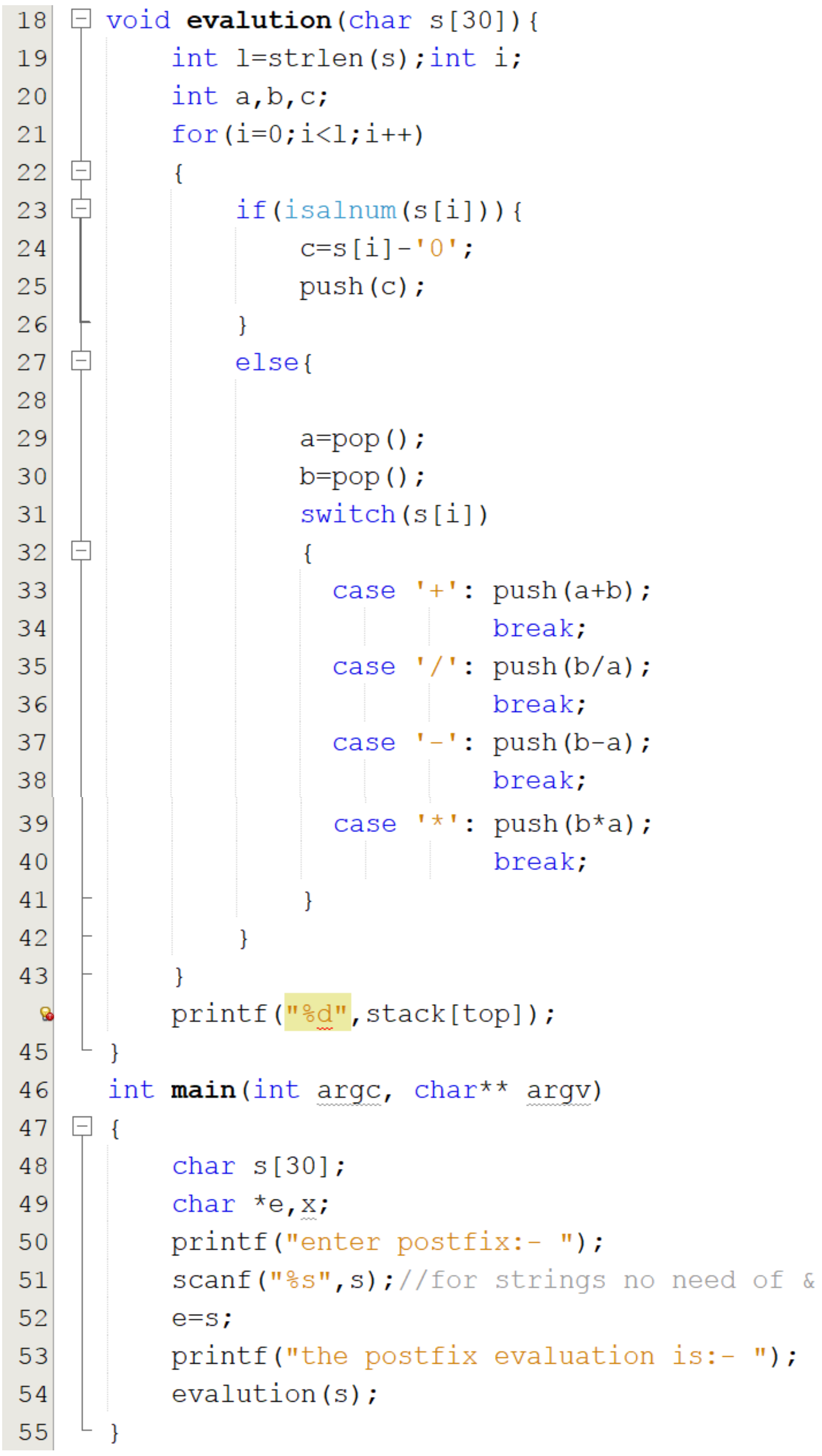


```c
        else{
            while((prt(stack[top]))>=(prt(*e)))
            {
                printf("%c",pop());
            }
            push(*e);
        }
        e++;
    }
    while(top!=-1)
    {
        printf("%c",pop());
    }
    return 0;
}
```

Q3: [to evaluate a postfix expression]:-



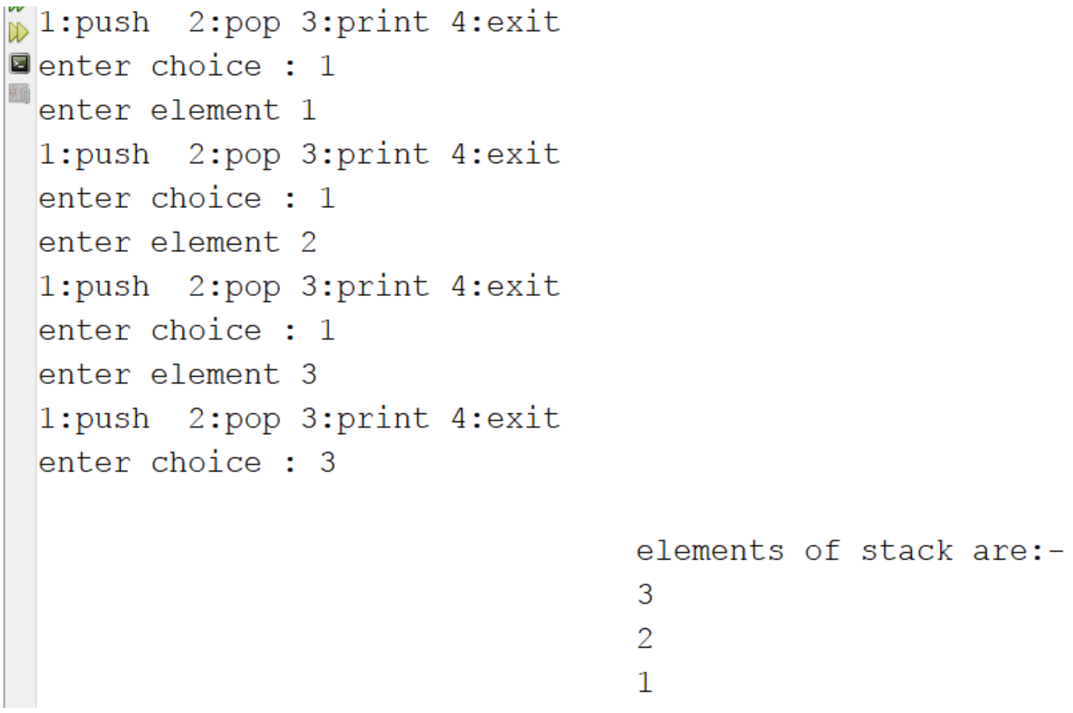
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

char stack[20];
int top=-1;
void push(char x){
    stack[++top]=x;
}
char pop(){
    if(top==-1){
        return -1;
    }
    else
        return stack[top--];
}
```



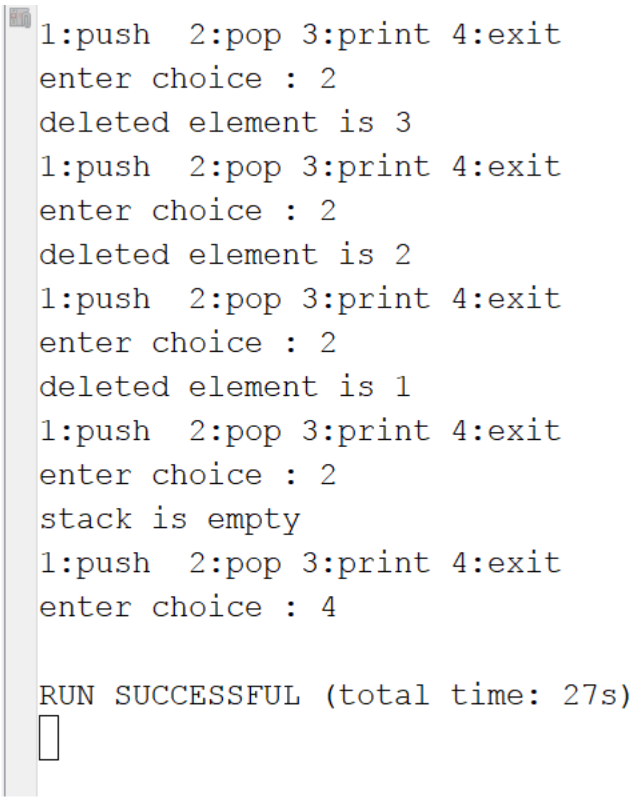
```c
void evalution(char s[30]){
    int l=strlen(s);int i;
    int a,b,c;
    for(i=0;i<l;i++)
    {
        if(isalnum(s[i])){
            c=s[i]-'0';
            push(c);
        }
        else{

            a=pop();
            b=pop();
            switch(s[i])
            {
              case '+': push(a+b);
                        break;
              case '/': push(b/a);
                        break;
              case '-': push(b-a);
                        break;
              case '*': push(b*a);
                        break;
            }
        }
    }
    printf("%d",stack[top]);
}
int main(int argc, char** argv)
{
    char s[30];
    char *e,x;
    printf("enter postfix:- ");
    scanf("%s",s);//for strings no need of &
    e=s;
    printf("the postfix evaluation is:- ");
    evalution(s);
}
```

4. Presentation of Results

Q1: [Implement the PUSH, POP and PRINT operations on stack]:-

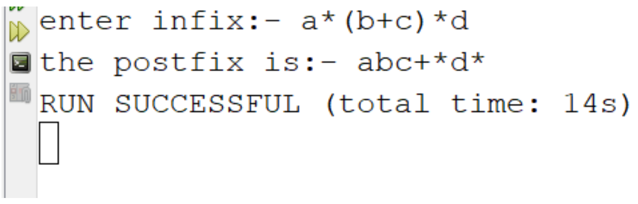


```
1:push  2:pop 3:print 4:exit
enter choice : 1
enter element 1
1:push  2:pop 3:print 4:exit
enter choice : 1
enter element 2
1:push  2:pop 3:print 4:exit
enter choice : 1
enter element 3
1:push  2:pop 3:print 4:exit
enter choice : 3

                                elements of stack are:-
                                3
                                2
                                1
```



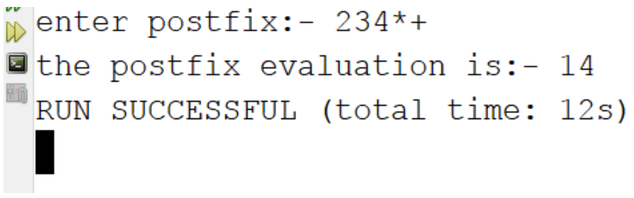
```
1:push  2:pop 3:print 4:exit
enter choice : 2
deleted element is 3
1:push  2:pop 3:print 4:exit
enter choice : 2
deleted element is 2
1:push  2:pop 3:print 4:exit
enter choice : 2
deleted element is 1
1:push  2:pop 3:print 4:exit
enter choice : 2
stack is empty
1:push  2:pop 3:print 4:exit
enter choice : 4

RUN SUCCESSFUL (total time: 27s)
```

Q2: Write a C program to convert infix to postfix notation using stack:-



```
enter infix:- a*(b+c)*d
the postfix is:- abc+*d*
RUN SUCCESSFUL (total time: 14s)
```

Q3: [to evaluate a postfix expression]:-



```
enter postfix:- 234*+
the postfix evaluation is:- 14
RUN SUCCESSFUL (total time: 12s)
```

5. Conclusions :

   Stack operations, infix to postfix and postfix evaluation have been done successfully

   And the results are as expected.