

**Rizvi College of Engineering**

*Department of Computer Engineering*

Mini Project Report

On

**CONVERSION OF STRINGS**

Second Year of Engineering

By

ADNAN SHAIKH(38)

FAISAL SAYED(36)

ASIF LOHAR(24)

AMEEN M. ALI(03)

Guide:

Prof. Khemani Bharti



University of Mumbai (2019-20)

**ABSTRACT**

C is a general-purpose programming language that is extremely popular, simple and flexible. It is machine-independent, structured programming language which is used extensively in various applications.

C was the basics language to write everything from operating systems (Windows and many others) to complex programs like the Oracle database, Git, Python interpreter and more.

Infix, Postfix and Prefix notations are three different but equivalent ways of writing expressions. It is easiest to demonstrate the differences by looking at examples of operators that take two operands.

Infix notation: X + Y

Operators are written in-between their operands. This is the usual way we write expressions. An expression such as A \* ( B + C ) / D is usually taken to mean something like: "First add B and C together, then multiply the result by A, then divide by D to give the final answer.”

Postfix notation (also known as "Reverse Polish notation"): X Y +

Operators are written after their operands. The infix expression given above is equivalent to A B C + . \* D /   
The order of evaluation of operators is always left-to-right, and brackets cannot be used to change this order. Because the "+" is to the left of the "\*" in the example above, the addition must be performed before the multiplication

Prefix notation (also known as "Polish notation"): + X Y

Operators are written before their operands. The expressions given above are equivalent to / \* A + B C D   
As for Postfix, operators are evaluated left-to-right and brackets are superfluous. Operators act on the two nearest values on the right. I have again added (totally unnecessary) brackets to make this clear:   
(/ (\* A (+ B C) ) D)



**Rizvi College of Engineering,**

Department of Computer Engineering

Off Carter Road, Bandra (W), Mumbai - 400050.

**Certificate**

This is to certify that the project report entitled “**CONVERSION OF STRINGS”** has been submitted by **Adnan Shaikh, Faisal Sayed, Asif Lohar** and **Ameen M. Ali** under the guidance of **Prof. Khemani Bharti** in partial fulfilment of the requirement for the award of the second year of Engineering in **Computer Engineering** from **University of Mumbai.**

Certified By

Prof. Khemani Bharti

Project Guide

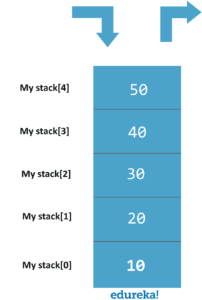
**Index**

|  |  |  |
| --- | --- | --- |
| **Sr. No** | **Title Page No.** |  |
| **1**. | **Introduction** …………........................................ 5 |
| **2**. | **Requirements**………..……………..………….. 6  **2.1**. **Hardware Requirements**…..……………... 6  **2.1**. **Software Requirements**……….…………... 6 |
| **3.** | **Proposed Methodology & Project Status**……. 7  **3.1 Flowchart**…..…………………………..…… 7  **3.2 Algorithm**…................................................... 8  **3.3 Program and Output**….…………………… 9  **3.4 Conclusion**………………………………….. 15  **3.5 Future Scope**………………………………..16  **References**………………………………………. 17 |
|  |  |

**Chapter 1: Introduction**

[**Data Structures in C**](https://www.edureka.co/blog/introduction-to-c-programming-algorithms/)are used to store data in an organised and efficient manner. The C Programming language has many data structures like an *array, stack, queue, linked list, tree, etc.* A programmer selects an appropriate data structure and uses it according to their convenience.

A stack is a linear data structure. It follows the**last in first out** approach. A new item is added at the top of a stack. Both insert and deletion operation is performed from one end of the stack.



There are two functions associated with stacks. Push function to add elements to the stack and pop function to remove elements from the stack.

**Chapter 2: Requirements**

**2.1 Hardware Requirements:**

2.1.1 PC (Monitor/Laptop)

2.1.2 RAM

2.1.3 Mouse

2.1.4 Keyboard

**2.2 Software Requirements:**

2.2.1 Turbo C++

2.2.2 Code Blocks

**Chapter 3: Proposed Methodology & Status of Work**

**3.1Flowchart:**

Print”Enter the type of String”

Print”Enter the string”

Print ”Enter your choice of Conversion

1. Infix
2. Postfix
3. prefix

Switchcase

case

Choice 3. Prefix

Choice 2. Postfix

22

Choice 1. Infix

Would you like to cont.?

**Yes NO**

**3.2 Algorithm:**

1. Step 1: start

2. Step 2: “Enter the type of String” “Enter the string”

3. Step 3: enter your choice

Convert \n 1. Infix \n 2. Postfix \n 3. prefix \n

4. Step 4: Convert the input string to desired option

5. Step 5: “The Desired string is (Choice)”

6. Step 6: “Would you like to Continue?”

7. Step 7: If YES then return to Step 3

8. Step 8: If NO, then STOP

**3.4 Program and Output:**

#include<stdio.h>

#include<stdlib.h> /\* for exit() \*/

#include<ctype.h> /\* for isdigit(char ) \*/

#include<string.h>

#define MAX 50

#define SIZE 100

/\* declared here as global variable because stack[]

\* is used by more than one fucntions \*/

char stack[SIZE];

int top = -1;

char opnds[50][80],oprs[50];

int topr=-1,topd=-1;

/\* define push operation \*/

void push(char item)

{

if(top >= SIZE-1)

{

printf("\nStack Overflow.");

}

else

{

top = top+1;

stack[top] = item;

}

}

/\* define pop operation \*/

char pop()

{

char item ;

if(top <0)

{

printf("stack under flow: invalid infix expression");

getchar();

/\* underflow may occur for invalid expression \*/

/\* where ( and ) are not matched \*/

exit(1);

}

else

{

item = stack[top];

top = top-1;

return(item);

}

}

/\* define function that is used to determine whether any symbol is operator or not

(that is symbol is operand)

\* this fucntion returns 1 if symbol is opreator else return 0 \*/

int is\_operator(char symbol)

{

if(symbol == '^' || symbol == '\*' || symbol == '/' || symbol == '+' || symbol =='-')

{

return 1;

}

else

{

return 0;

}

}

/\* define fucntion that is used to assign precendence to operator.

\* Here ^ denotes exponent operator.

\* In this fucntion we assume that higher integer value

\* means higher precendence \*/

int precedence(char symbol)

{

if(symbol == '^')/\* exponent operator, highest precedence\*/

{

return(3);

}

else if(symbol == '\*' || symbol == '/')

{

return(2);

}

else if(symbol == '+' || symbol == '-') /\* lowest precedence \*/

{

return(1);

}

else

{

return(0);

}

}

void InfixToPostfix(char infix\_exp[], char postfix\_exp[])

{

int i, j;

char item;

char x;

push('('); /\* push '(' onto stack \*/

strcat(infix\_exp,")"); /\* add ')' to infix expression \*/

i=0;

j=0;

item=infix\_exp[i]; /\* initialize before loop\*/

while(item != '\0') /\* run loop till end of infix expression \*/

{

if(item == '(')

{

push(item);

}

else if( isdigit(item) || isalpha(item))

{

postfix\_exp[j] = item; /\* add operand symbol to postfix expr \*/

j++;

}

else if(is\_operator(item) == 1) /\* means symbol is operator \*/

{

x=pop();

while(is\_operator(x) == 1 && precedence(x)>= precedence(item))

{

postfix\_exp[j] = x; /\* so pop all higher precendence operator and \*/

j++;

x = pop(); /\* add them to postfix expresion \*/

}

push(x);

/\* because just above while loop will terminate we have

oppped one extra item

for which condition fails and loop terminates, so that one\*/

push(item); /\* push current oprerator symbol onto stack \*/

}

else if(item == ')') /\* if current symbol is ')' then \*/

{

x = pop(); /\* pop and keep popping until \*/

while(x != '(') /\* '(' encounterd \*/

{

postfix\_exp[j] = x;

j++;

x = pop();

}

}

else

{ /\* if current symbol is neither operand not '(' nor ')' and nor

operator \*/

printf("\nInvalid infix Expression.\n"); /\* the it is illegeal symbol \*/

getchar();

exit(1);

}

i++;

item = infix\_exp[i]; /\* go to next symbol of infix expression \*/

} /\* while loop ends here \*/

if(top>0)

{

printf("\nInvalid infix Expression.\n"); /\* the it is illegeal symbol \*/

getchar();

exit(1);

}

if(top>0)

{

printf("\nInvalid infix Expression.\n"); /\* the it is illegeal symbol \*/

getchar();

exit(1);

}

postfix\_exp[j] = '\0'; /\* add sentinel else puts() fucntion \*/

/\* will print entire postfix[] array upto SIZE \*/

}

/\* main function begins \*/

int InfixToPostmain(char input[])

{

char infix[SIZE], postfix[SIZE]; /\* declare infix string and postfix string \*/

/\* why we asked the user to enter infix expression

\* in parentheses ( )

\* What changes are required in porgram to

\* get rid of this restriction since it is not

\* in algorithm

\* \*/

strcpy(infix,input);

//printf("ASSUMPTION: The infix expression contains single letter variables and single digit constants only.\n");

//printf("\nEnter Infix expression : ");

//scanf("%s",&infix);

InfixToPostfix(infix,postfix); /\* call to convert \*/

printf("Postfix Expression: ");

puts(postfix); /\* print postfix expression \*/

return 0;

}

//---------------------------------------------------------------------------------------------------------------------------------

int pr(char elem)

{ /\* Function for precedence \*/

switch(elem)

{

case '#': return 0;

case ')': return 1;

case '+':

case '-': return 2;

case '\*':

case '/': return 3;

}

}

void InfixToPremain(char input[])

{ /\* Main Program \*/

char infx[50],prfx[50],ch,elem;

int i=0,k=0;

//printf("\n\nEnter the infix Expression ");

//scanf("%s",infx);

strcpy(infx,input);

push('#');

strrev(infx);

while( (ch=infx[i++]) != '\0')

{

if( ch == ')') push(ch);

else

if(isalnum(ch)) prfx[k++]=ch;

else

if( ch == '(')

{

while( stack[top] != ')')

prfx[k++]=pop();

elem=pop(); /\* Remove ) \*/

}

else

{ /\* Operator \*/

while( pr(stack[top]) >= pr(ch) )

prfx[k++]=pop();

push(ch);

}

}

while( stack[top] != '#') /\* Pop from stack till empty \*/

prfx[k++]=pop();

prfx[k]='\0'; /\* Make prfx as valid string \*/

strrev(prfx);

strrev(infx);

printf("\n\nGiven Infix Expn: %s Prefix Expn: %s\n",infx,prfx);

}

//------------------------------------------------------------------------------------------------------------------------------------

pushd(char \*opnd)

{

strcpy(opnds[++topd],opnd);

}

char \*popd()

{

return(opnds[topd--]);

}

pushr(char opr)

{

oprs[++topr]=opr;

}

char popr()

{

return(oprs[topr--]);

}

int empty(int t)

{

if( t == 0) return(1);

return(0);

}

void PreToInmain(char input[])

{

char prfx[50],ch,str[50],opnd1[50],opnd2[50],opr[2];

int i=0,k=0,opndcnt=0;

//printf("Enter prefix expression:\t");

//scanf("%s",&prfx);

strcpy(prfx,input);

while( (ch=prfx[i++]) != '\0')

{

if(isalnum(ch))

{

str[0]=ch; str[1]='\0';

pushd(str); opndcnt++;

if(opndcnt >= 2)

{

strcpy(opnd2,popd());

strcpy(opnd1,popd());

strcpy(str,"(");

strcat(str,opnd1);

ch=popr();

opr[0]=ch;opr[1]='\0';

strcat(str,opr);

strcat(str,opnd2);

strcat(str,")");

pushd(str);

opndcnt-=1;

}

}

else

{

pushr(ch);

if(opndcnt==1)opndcnt=0; /\* operator followed by single operand\*/

}

}

if(!empty(topd))

{

strcpy(opnd2,popd());

strcpy(opnd1,popd());

strcpy(str,"(");

strcat(str,opnd1);

ch=popr();

opr[0]=ch;opr[1]='\0';

strcat(str,opr);

strcat(str,opnd2);

strcat(str,")");

pushd(str);

}

printf(" Infix Expression: ");

puts(opnds[topd]);

}

//------------------------------------------------------------------------------------------------------------------------------------

void PreToPostmain(char input[])

{

char prfx[50],ch,str[50],opnd1[50],opnd2[50],opr[2];

int i=0,k=0,opndcnt=0;

gets(prfx);

strcpy(prfx,input);

//printf(" Given Prefix Expression : %s\n",prfx);

while( (ch=prfx[i++]) != '\0')

{

if(isalnum(ch))

{

str[0]=ch; str[1]='\0';

pushd(str); opndcnt++;

if(opndcnt >= 2)

{

strcpy(opnd2,popd());

strcpy(opnd1,popd());

strcpy(str,opnd1);

strcat(str,opnd2);

ch=popr();

opr[0]=ch;opr[1]='\0';

strcat(str,opr);

pushd(str);

opndcnt-=1;

}

}

else

{

pushr(ch);

if(opndcnt==1)opndcnt=0; /\* operator followed by single operand\*/

}

}

if(!empty(topd))

{

strcpy(opnd2,popd());

strcpy(opnd1,popd());

strcpy(str,opnd1);

strcat(str,opnd2);

ch=popr();

opr[0]=ch;opr[1]='\0';

strcat(str,opr);

pushd(str);

}

printf(" Postfix Expression: ");

puts(opnds[topd]);

}

//------------------------------------------------------------------------------------------------------------------------------------

void main()

{

int ch,ch1,ch2,ch3;

char infixIn[MAX],PrefixIn[MAX],PostfixIn[MAX];

//clrscr();

L1: printf ("\nSelect the form of your primary expression\n1. POSTFIX\n2. PREFIX\n3. INFIX\n4. EXIT\n");

scanf("%d",&ch);

do{

switch (ch)

{case 1: printf("Enter the Postfix Expression\n");

scanf("%s",&PostfixIn);

L2: printf("Select the form in which you want it to be converted\n1. INFIX\n2. PREFIX\n3. EXIT");

scanf("%d",&ch1);

do{

switch(ch1)

{

case 1: PostToInmain(PostfixIn);

printf("\n");

goto L2;

break;

case 2: PostToPremain(PostfixIn);

printf("\n");

goto L2;

break;

case 3: printf("Thank You\n");

printf("\n");

goto L1;

break;

default: printf("Enter proper choice\t");

ch1=100;

}

}while(ch1==100);

break;

case 2: printf(" Enter the Prefix expression");

scanf("%s",&PrefixIn);

L3: printf("Select the form in which you want it to be converted\n1. INFIX\n2. POSTFIX\n3. EXIT");

scanf("%d",&ch2);

do{

switch(ch2)

{

case 1: PreToInmain(PrefixIn);

goto L3;

break;

case 2: PreToPostmain(PrefixIn);

goto L3;

break;

case 3: printf("Thank You\n");

goto L1;

break;

default: printf("Enter proper choice\t");

ch2=100;

}

}while(ch2==100);

break;

case 3: printf(" Enter the Infix expression");

scanf("%s",&infixIn);

L4: printf("Select the form in which you want it to be converted\n1. PREFIX\n2. POSTFIX\n3. EXIT");

scanf("%d",&ch3);

do{

switch (ch3)

{

case 1: InfixToPremain(infixIn);

goto L4;

break;

case 2: InfixToPostmain(infixIn);

goto L4;

break;

case 3: printf("Thank You\n");

goto L1;

break;

default: printf("Enter proper choice\t");

scanf("%d",&ch3);

}

}while(ch3<1 || ch>3);

break;

case 4: printf("Thank You\n");

break;

default: printf("Enter proper choice");

scanf("%d",&ch3);

}

}while(ch!=4);

}

//------------------------------------------------------------------------------------------------------------------------------------

struct postfixp2p

{ char stack[MAX][MAX], target[MAX] ;

char temp1[2], temp2[2] ;

char str1[MAX], str2[MAX], str3[MAX] ;

int ip2p, topp2p ;

};

void initpostfixp2p ( struct postfixp2p \* ) ;

void setexprp2p ( struct postfixp2p \*, char \* ) ;

void pushp2p ( struct postfixp2p \*, char \* ) ;

void popp2p ( struct postfixp2p \*, char \* ) ;

void convert ( struct postfixp2p \* ) ;

void show ( struct postfixp2p ) ;

void PostToPremain(char input[])

{

struct postfixp2p q ;

char expr[MAX] ;

initpostfix ( &q ) ;

//printf ( "\nEnter an expression in postfix form: " ) ;

//scanf("%s", &expr ) ;

strcpy(expr,input);

setexpr ( &q, expr ) ;

convert ( &q ) ;

printf ( "\nThe Prefix expression is: " ) ;

show ( q ) ;

printf("\n");

}

/\* initializes the elements of the structure \*/

void initpostfix ( struct postfixp2p \*p )

{

p -> ip2p = 0 ;

p -> topp2p = -1 ;

strcpy ( p -> target, "" ) ;

}

/\* copies given expr. to target string \*/

void setexpr ( struct postfixp2p \*p, char \*c )

{

strcpy ( p -> target, c ) ;

}

/\* adds an operator to the stack \*/

void pushp2p ( struct postfixp2p \*p, char \*str )

{

if ( p -> topp2p == MAX - 1 )

printf ( "\nStack is full." ) ;

else

{

p -> topp2p++ ;

strcpy ( p -> stack[p -> topp2p], str ) ;

}

}

/\* pops an element from the stack \*/

void popp2p ( struct postfixp2p \*p, char \*a )

{

if ( p -> topp2p == -1 )

printf ( "\nStack is empty." ) ;

else

{

strcpy ( a, p -> stack[p -> topp2p] ) ;

p -> topp2p-- ;

}

}

/\* converts given expr. to prefix form \*/

void convert ( struct postfixp2p \*p )

{

while ( p -> target[p -> ip2p] != '\0' )

{

/\* skip whitespace, if any \*/

if ( p -> target[p -> ip2p] == ' ')

p -> ip2p++ ;

if( p -> target[p -> ip2p] == '%' || p -> target[p -> ip2p] == '\*' ||

p -> target[p -> ip2p] == '-' || p -> target[p -> ip2p] == '+' ||

p -> target[p -> ip2p] == '/' || p -> target[p -> ip2p] == '$' )

{

popp2p ( p, p -> str2 ) ;

popp2p ( p, p -> str3 ) ;

p -> temp1[0] = p -> target[ p -> ip2p] ;

p -> temp1[1] = '\0' ;

strcpy ( p -> str1, p -> temp1 ) ;

strcat ( p -> str1, p -> str3 ) ;

strcat ( p -> str1, p -> str2 ) ;

pushp2p ( p, p -> str1 ) ;

}

else

{

p -> temp1[0] = p -> target[p -> ip2p] ;

p -> temp1[1] = '\0' ;

strcpy ( p -> temp2, p -> temp1 ) ;

pushp2p ( p, p -> temp2 ) ;

}

p -> ip2p++ ;

}

}

/\* displays the prefix form of expr. \*/

void show ( struct postfixp2p p )

{

char \*temp = p.stack[0] ;

while ( \*temp )

{

printf ( "%c ", \*temp ) ;

temp++ ;

}

}

//----------------------------------------------------------------------------------------------------------------------

int topp2i = 10;

struct node

{

char chp2i;

struct node \*next;

struct node \*prev;

} \*stackp2i[11];

typedef struct node node;

void pushp2i(node \*str)

{

if (topp2i <= 0)

printf("Stack is Full ");

else

{

stackp2i[topp2i] = str;

topp2i--;

}

}

node \*popp2i()

{

node \*exp;

if (topp2i >= 10)

printf("Stack is Empty ");

else

exp = stackp2i[++topp2i];

return exp;

}

void convertp2i(char exp[])

{

node \*op1, \*op2;

node \*temp;

int i;

for (i=0;exp[i]!='\0';i++)

if (exp[i] >= 'a'&& exp[i] <= 'z'|| exp[i] >= 'A' && exp[i] <= 'Z')

{

temp = (node\*)malloc(sizeof(node));

temp->chp2i = exp[i];

temp->next = NULL;

temp->prev = NULL;

pushp2i(temp);

}

else if (exp[i] == '+' || exp[i] == '-' || exp[i] == '\*' || exp[i] == '/' ||

exp[i] == '^')

{

op1 = popp2i();

op2 = popp2i();

temp = (node\*)malloc(sizeof(node));

temp->chp2i = exp[i];

temp->next = op1;

temp->prev = op2;

pushp2i(temp);

}

}

void display(node \*temp)

{

if (temp != NULL)

{

display(temp->prev);

printf("%c", temp->chp2i);

display(temp->next);

}

}

void PostToInmain(char input[])

{

char exp[50];

//printf("Enter the postfix expression :");

//scanf("%s", exp);

strcpy(exp,input);

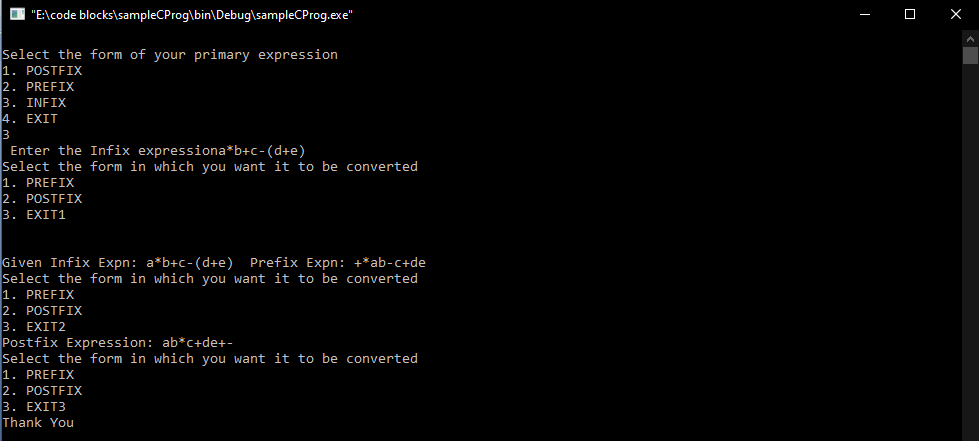
convertp2i(exp);

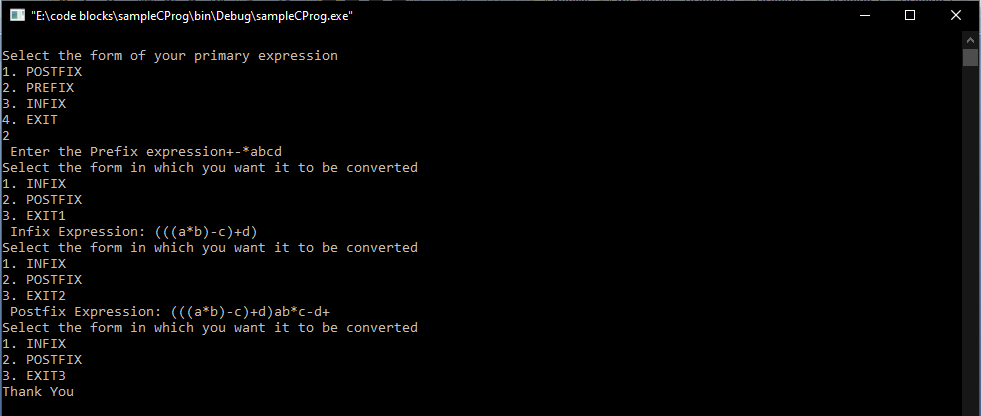
printf("\nThe Equivalant Infix expression is:");

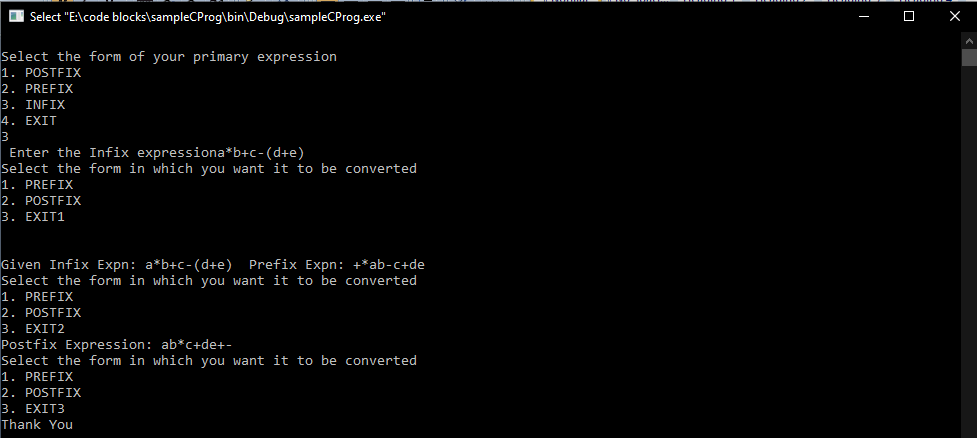
display(popp2i());

}

**OUTPUT:**

****

****

****

**3.5 Conclusion:**

So we have Successfully performed conversion of strings (Expression) into infix, postfix and prefix using Stack.

**3.5 Future Scope:**

There are numerous benefits from learning C; however, the most important benefit is that the C programming language is recognized worldwide and used in a multitude of applications, including advanced scientific systems and operating systems. In today’s world, a computer programmer needs to be able to communicate with colleagues in different countries. Therefore it’s important that even if they don’t speak the same verbal language, at least the Although numerous computer languages are used for writing computer applications, the computer programming language, C, is the most popular language worldwide. Everything from microcontrollers to operating systems is written in C since it’s very flexible and versatile, allowing maximum control with minimal commands. If you are interested in a career in computer programming, it would be wise to start by learning the C programming language.

computer language is understandable to all.

**References:**

[1] BOOKS: Data Structures using C and C++ by Rajesh K. Shukla

### [2] BOOKS: **Data Structures in C by Reema Thareja**

[3] https://www.geeksforgeeks.org