- Infix to Prefix Conversion
- Postfix to Infix Conversion
- **» Prefix to Infix Conversion**
- Prefix Infix Postfix converter

Queues

Recursion

Searching

Sorting Algorithms

Algorithms

Tree

Online Shopping

Prefix to Infix Conversion

Algorithm of Prefix to Infix

```
This algorithm is a non-tail recursive method.
1.The reversed input string is completely pushed into a stack.
        prefixToInfix(stack)
2.IF stack is not empty
a. Temp -->pop the stack
b. IF temp is a operator
                Write a opening parenthesis to output
                prefixToInfix(stack)
                Write temp to output
                prefixToInfix(stack)
                Write a closing parenthesis to output
c. ELSE IF temp is a space -->prefixToInfix(stack)
d. ELSE
                Write temp to output
                IF stack.top NOT EQUAL to space -->prefixToInfix(stack)
```


- Prefix Infix Postfix converter Tool Online

Prefix to Infix conversion

Example

```
*+a-bc/-de+-fgh
```

Expression	Stack
*+a-bc/-de+-fgh	NuLL
*+a-bc/-de+-fg	"h"
*+a-bc/-de+-f	"g" "h"
*+a-bc/-de+-	"f" "g" "h"
*+a-bc/-de+	"f - g" "h"
*+a-bc/-de	"f-g+h"
*+a-bc/-d	"e" "f-g+h"
*+a-bc/-	"d" "e" "f-g+h"
*+a-bc/	"d - e" "f-g+h"
*+a-bc	"(d-e)/(f-g+h)"
*+a-b	"c" "(d-e)/(f-g+h)"
*+a-	"b" "c" "(d-e)/(f-g+h)"
*+a	"b-c" "(d-e)/(f-g+h)"
*+	"a" "b-c" "(d-e)/(f-g+h)"

*	"a+b-c" "(d-e)/(f-g+h)"
End	"(a+b-c)*(d-e)/(f-g+h)"

Ans = (a+b-c)*(d-e)/(f-g+h)

Postfix to Infix implementation in c

```
#include <string.h>
#include <ctype.h>
#include <conio.h>
char opnds[50][80],oprs[50];
int  topr=-1,topd=-1;
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 main()
{
    char prfx[50],ch,str[50],opnd1[50],opnd2[50],opr[2];
    int i=0,k=0,opndcnt=0;
    clrscr();
    printf("Give an Expression = ");
    gets(prfx);
    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,"(");
                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]);
    getch();
}
```