IMPLEMENT INFIX TO POSTFIX CONVERSION USING STACK IN C

DATA STRUCTURE & ALGORITHM PROJECT

Guided by Mrs. SONAM MODI

MEMBER NAME

ENROLLMENT NUMBER:- 216400307127

NAME:- SOLANKI TRUSHEN

ENROLLMENT NUMBER:- 216400307129

NAME:- PITRODA RUDRA

ENROLLMENT NUMBER:- 216400307125

NAME:- RATHOD ROHIT

ENROLLMENT NUMBER:- 216400307118

NAME:- BHAVSAR DAKSH

INDEX

General

- One of the applications of Stack is in the conversion of arithmetic expressions in high-level programming languages into machine readable form. As our computer system can only understand and work on a binary language, it assumes that an arithmetic operation can take place in two operands only e.g., A+B, C*D,D/A etc. But in our usual form an arithmetic expression may consist of more than one operator and two operands e.g. (A+B)*C(D/(J+D)).
- These complex arithmetic operations can be converted into polish notation using stacks which then can be executed in two operands and an operator form.

Infix & Postfix

❑ Infix Expression

- It follows the scheme of **<operand><operator><operand>** i.e. an <operator> is preceded and succeeded by an <operand>. Such an expression is termed infix expression. E.g., **A+B**

❑ Postfix Expression

- It follows the scheme of **<operand><operand><operator>** i.e. an <operator> is succeeded by both the <operand>. E.g., **AB+**

Advantage of Postfix Expression

Advantage of Postfix Expression over Infix Expression

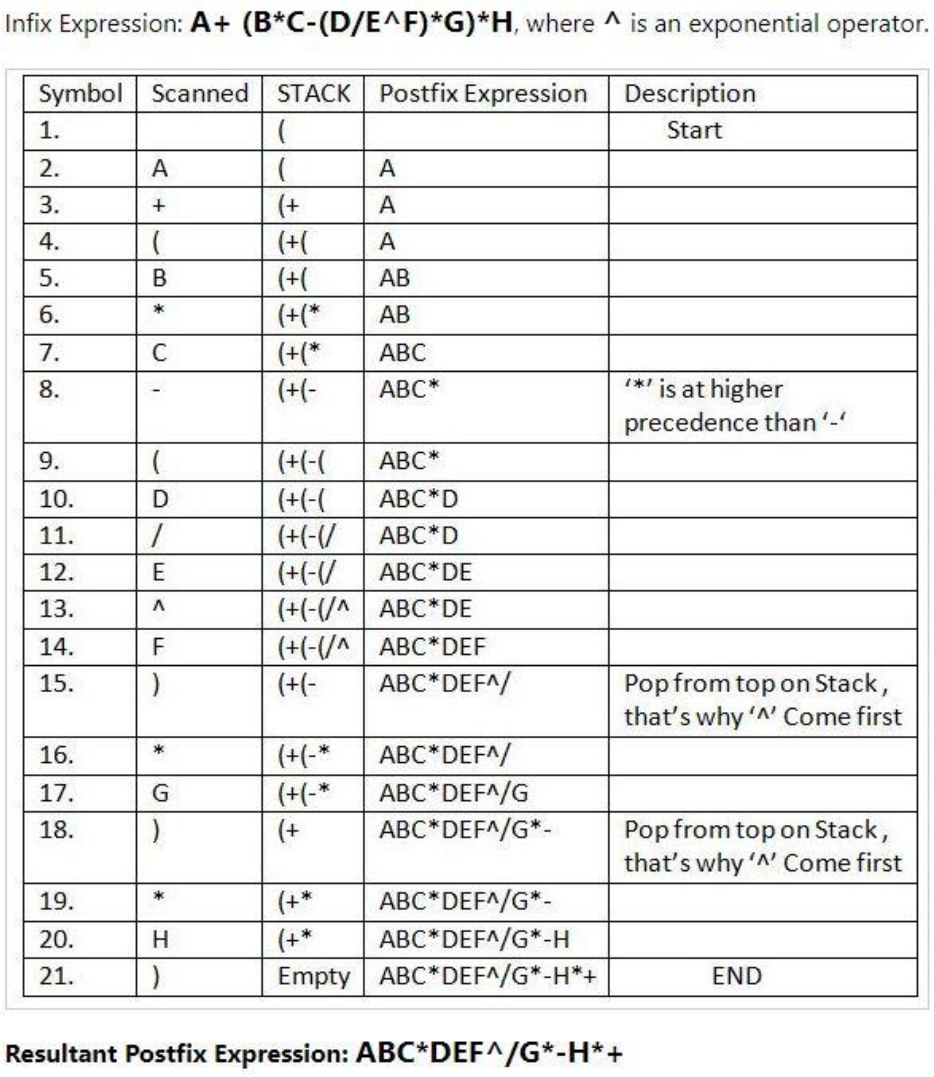
- An infix expression is difficult for the machine to know and keep track of precedence of operators. On the other hand, a postfix expression itself determines the precedence of operators (as the placement of operators in a postfix expression depends upon its precedence).Therefore, for the machine it is easier to carry out a postfix expression than an infix expression.

Algorithm to convert Infix To Postfix

Let, X is an arithmetic expression written in infix notation. This algorithm finds the equivalent postfix expression Y.

1. Push “(“onto Stack, and add “)” to the end of X.
2. Scan X from left to right and repeat Step 3 to 6 for each element of X until the Stack is empty.
3. If an operand is encountered, add it to Y.
4. If a left parenthesis is encountered, push it onto Stack.
5. If an operator is encountered ,then:
    1. Repeatedly pop from Stack and add to Y each operator (on the top of Stack) which has the same precedence as or higher precedence than operator.
    2. Add operator to Stack.
       [End of If]
6. If a right parenthesis is encountered ,then:
    1. Repeatedly pop from Stack and add to Y each operator (on the top of Stack) until a left parenthesis is encountered.
    2. Remove the left Parenthesis.
       [End of If]
       [End of If]

Infix to Postfix conversion

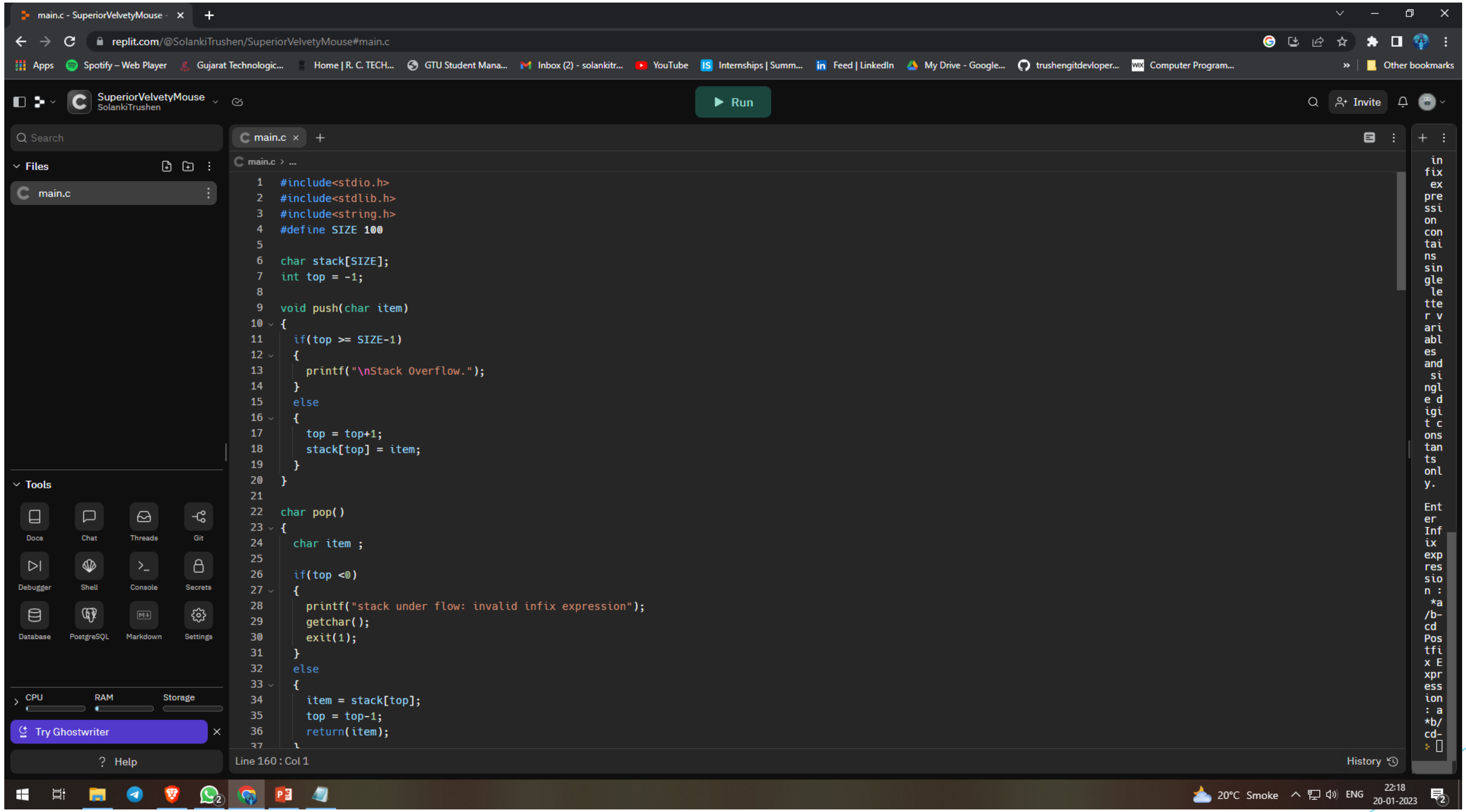


Infix Expression: **A+ (B*C-(D/E^F)*G)*H**, where ^ is an exponential operator.

Symbol	Scanned	STACK	Postfix Expression	Description
1.		(		Start
2.	A	(	A	
3.	+	(+	A	
4.	(	(+(	A	
5.	B	(+(	AB	
6.	*	(+(*	AB	
7.	C	(+(*	ABC	
8.	-	(+(-	ABC*	'*' is at higher precedence than '-'
9.	(	(+(-(	ABC*	
10.	D	(+(-(	ABC*D	
11.	/	(+(-(/	ABC*D	
12.	E	(+(-(/	ABC*DE	
13.	^	(+(-(/^	ABC*DE	
14.	F	(+(-(/^	ABC*DEF	
15.	)	(+(-	ABC*DEF^/	Pop from top on Stack , that's why '^' Come first
16.	*	(+(-*	ABC*DEF^/	
17.	G	(+(-*	ABC*DEF^/G	
18.	)	(+	ABC*DEF^/G*-	Pop from top on Stack , that's why '^' Come first
19.	*	(+*	ABC*DEF^/G*-	
20.	H	(+*	ABC*DEF^/G*-H	
21.	)	Empty	ABC*DEF^/G*-H*+	END

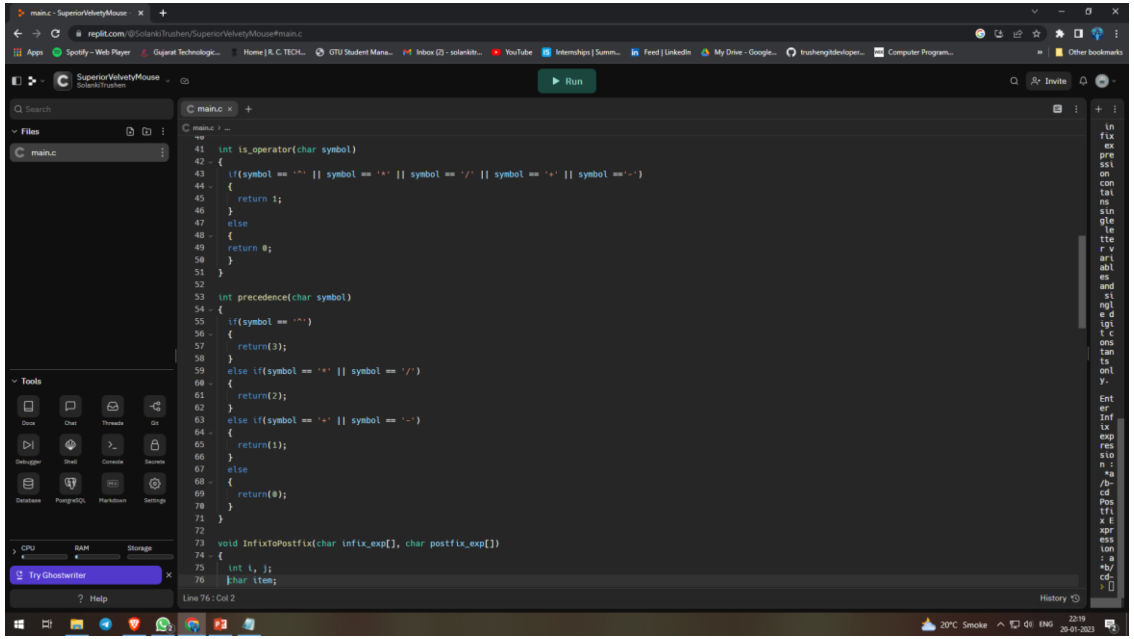
Resultant Postfix Expression: ABC*DEF^/G*-H*+

Code screen shot



```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define SIZE 100

char stack[SIZE];
int top = -1;

void push(char item)
{
  if(top >= SIZE-1)
  {
    printf("\nStack Overflow.");
  }
  else
  {
    top = top+1;
    stack[top] = item;
  }
}

char pop()
{
  char item ;

  if(top <0)
  {
    printf("stack under flow: invalid infix expression");
    getchar();
    exit(1);
  }
  else
  {
    item = stack[top];
    top = top-1;
    return(item);
  }
```

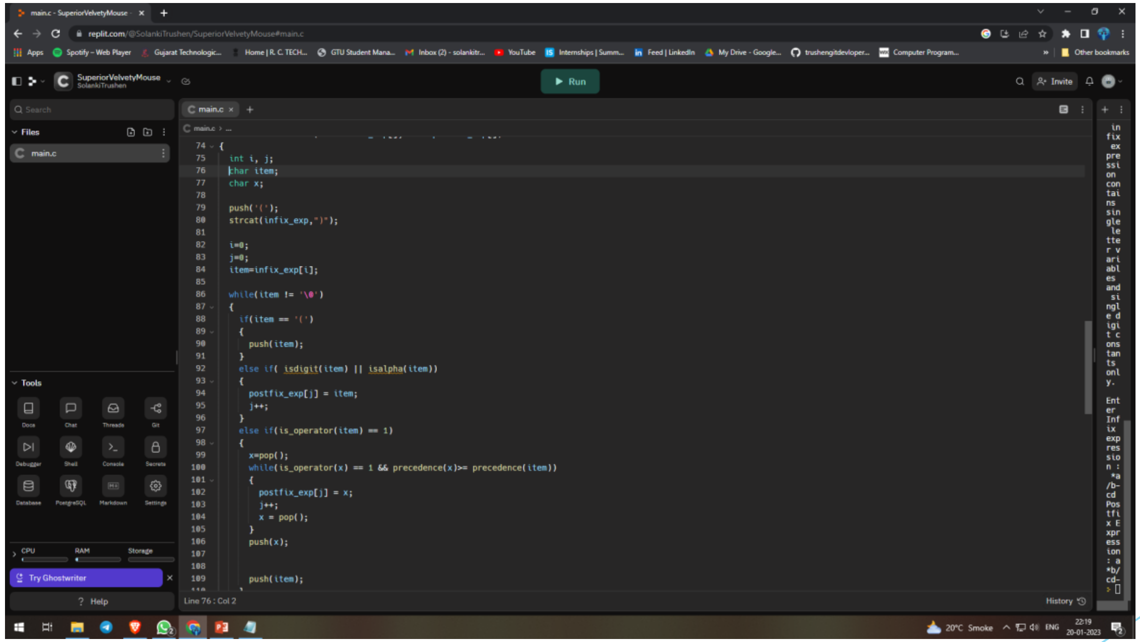
```
 in
fix
 ex
pre
ssi
on
con
tai
ns
sin
gle
 le
tte
r v
ari
abl
es
and
 si
ngl
e d
igi
t c
ons
tan
ts
onl
y.

Ent
er
Inf
ix
exp
res
sio
n :
 *a
/b-
cd
Pos
tfi
x E
xpr
ess
ion
: a
*b/
cd-
> 
```



```c
int is_operator(char symbol)
{
  if(symbol == '^' || symbol == '*' || symbol == '/' || symbol == '+' || symbol =='-')
  {
    return 1;
  }
  else
  {
  return 0;
  }
}

int precedence(char symbol)
{
  if(symbol == '^')
  {
    return(3);
  }
  else if(symbol == '*' || symbol == '/')
  {
    return(2);
  }
  else if(symbol == '+' || symbol == '-')
  {
    return(1);
  }
  else
  {
    return(0);
  }
}

void InfixToPostfix(char infix_exp[], char postfix_exp[])
{
  int i, j;
  char item;
```

```
infix expression contains single letter variables and single digit constants only.

Enter Infix expression : *a/b-cd
Postfix Expression : a*b/cd-
> 
```

Line 76 : Col 2

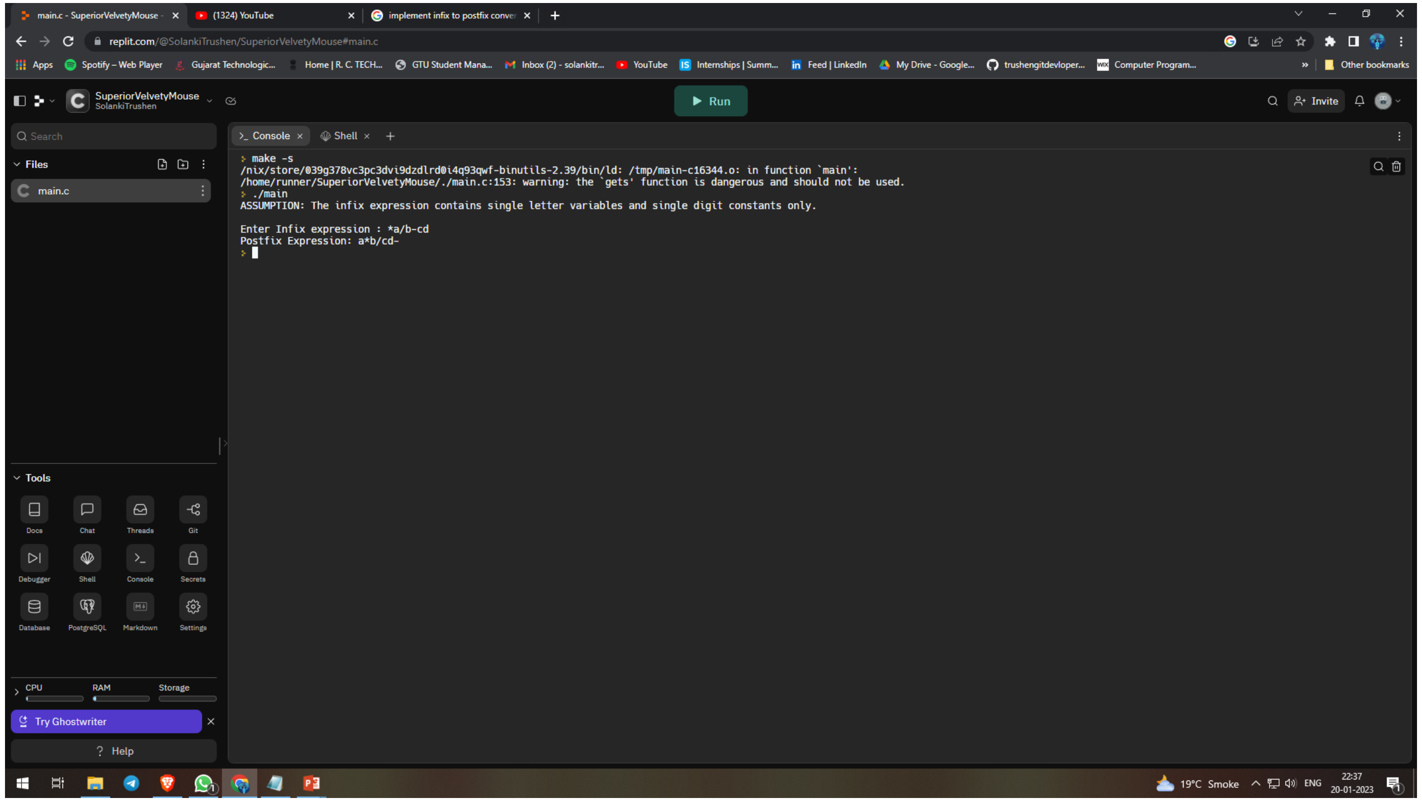


```c
{
  int i, j;
  char item;
  char x;

  push('(');
  strcat(infix_exp,")");

  i=0;
  j=0;
  item=infix_exp[i];

  while(item != '\0')
  {
    if(item == '(')
    {
      push(item);
    }
    else if( isdigit(item) || isalpha(item))
    {
      postfix_exp[j] = item;
      j++;
    }
    else if(is_operator(item) == 1)
    {
      x=pop();
      while(is_operator(x) == 1 && precedence(x)>= precedence(item))
      {
        postfix_exp[j] = x;
        j++;
        x = pop();
      }
      push(x);


      push(item);
```

```
infix expression contains single letter variables and single digit constants only.

Enter Infix expression : *a/b-cd
Postfix Expression : a*b/cd-
> 
```

Line 76 : Col 2

Output screen shot



```
> make -s
/nix/store/039g378vc3pc3dvi9dzdlrd0i4q93qwf-binutils-2.39/bin/ld: /tmp/main-c16344.o: in function `main':
/home/runner/SuperiorVelvetyMouse/./main.c:153: warning: the `gets' function is dangerous and should not be used.
> ./main
ASSUMPTION: The infix expression contains single letter variables and single digit constants only.

Enter Infix expression : *a/b-cd
Postfix Expression: a*b/cd-
>
```