

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:- RADHOD ROHIT

ENROLLMENT NUMBER:- 216400307118

NAME:- BHAVSAR DAKSH

Index

SR NO.	TOPIC NAME	PAGE NO.
1	General	3
2	Infix to post fix convert	4
3	Advantage of postfix expression	5
4	Algorithm to convert infix to postfix	6
5	Infix to post fix convert	7
6	Code screen shot	8-9
7	Output screen shot	10

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 to post fix convert

❑ Infix Expression

- It follows the scheme of $\langle \text{operand} \rangle \langle \text{operator} \rangle \langle \text{operand} \rangle$ i.e. an $\langle \text{operator} \rangle$ is preceded and succeeded by an $\langle \text{operand} \rangle$. Such an expression is termed infix expression. E.g., $A+B$

❑ Postfix Expression

- It follows the scheme of $\langle \text{operand} \rangle \langle \text{operand} \rangle \langle \text{operator} \rangle$ i.e. an $\langle \text{operator} \rangle$ is succeeded by both the $\langle \text{operand} \rangle$. 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]

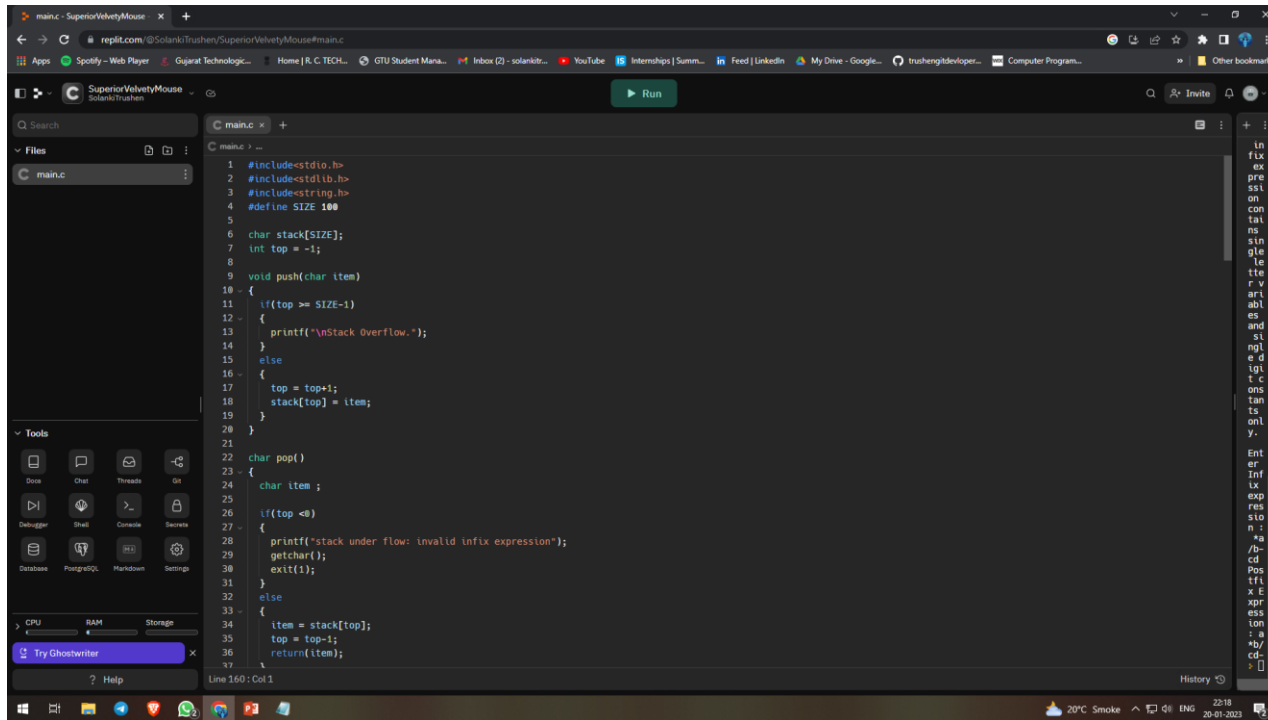
Infix to post fix convert

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

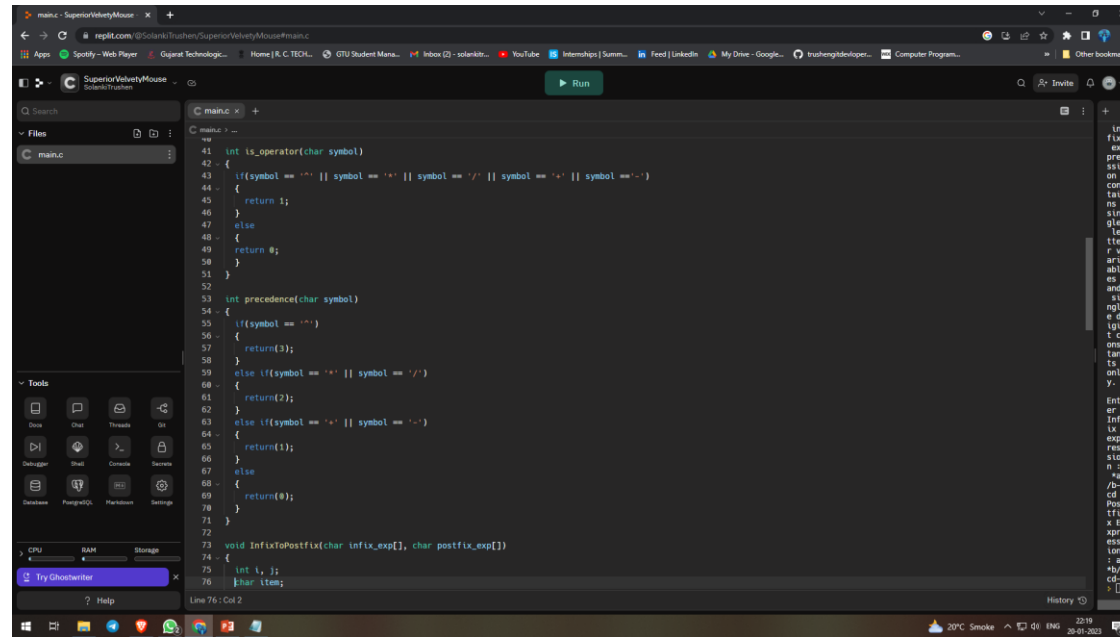
Resultant Postfix Expression: ABC*DEF^/G*-H*+

Code screen shot

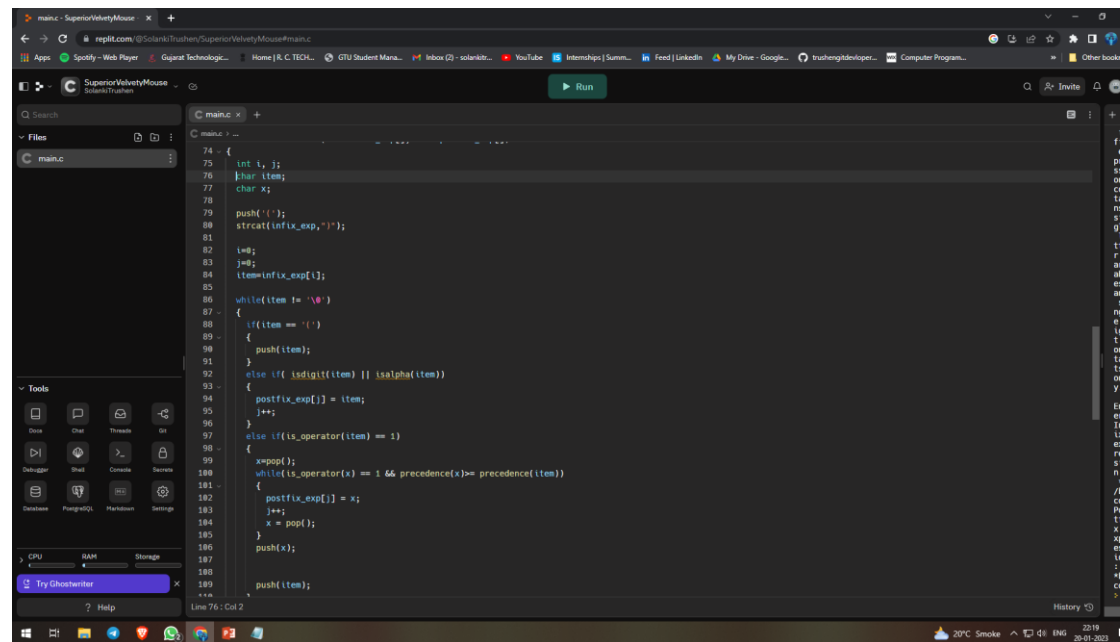


```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<string.h>
4 #define SIZE 100
5
6 char stack[SIZE];
7 int top = -1;
8
9 void push(char item)
10 {
11     if(top >= SIZE-1)
12     {
13         printf("\nStack Overflow.");
14     }
15     else
16     {
17         top = top+1;
18         stack[top] = item;
19     }
20 }
21
22 char pop()
23 {
24     char item ;
25
26     if(top <= 0)
27     {
28         printf("stack under flow: invalid infix expression");
29         getchar();
30         exit(1);
31     }
32     else
33     {
34         item = stack[top];
35         top = top-1;
36         return(item);
37     }
38 }
```

Line 160 : Col 1

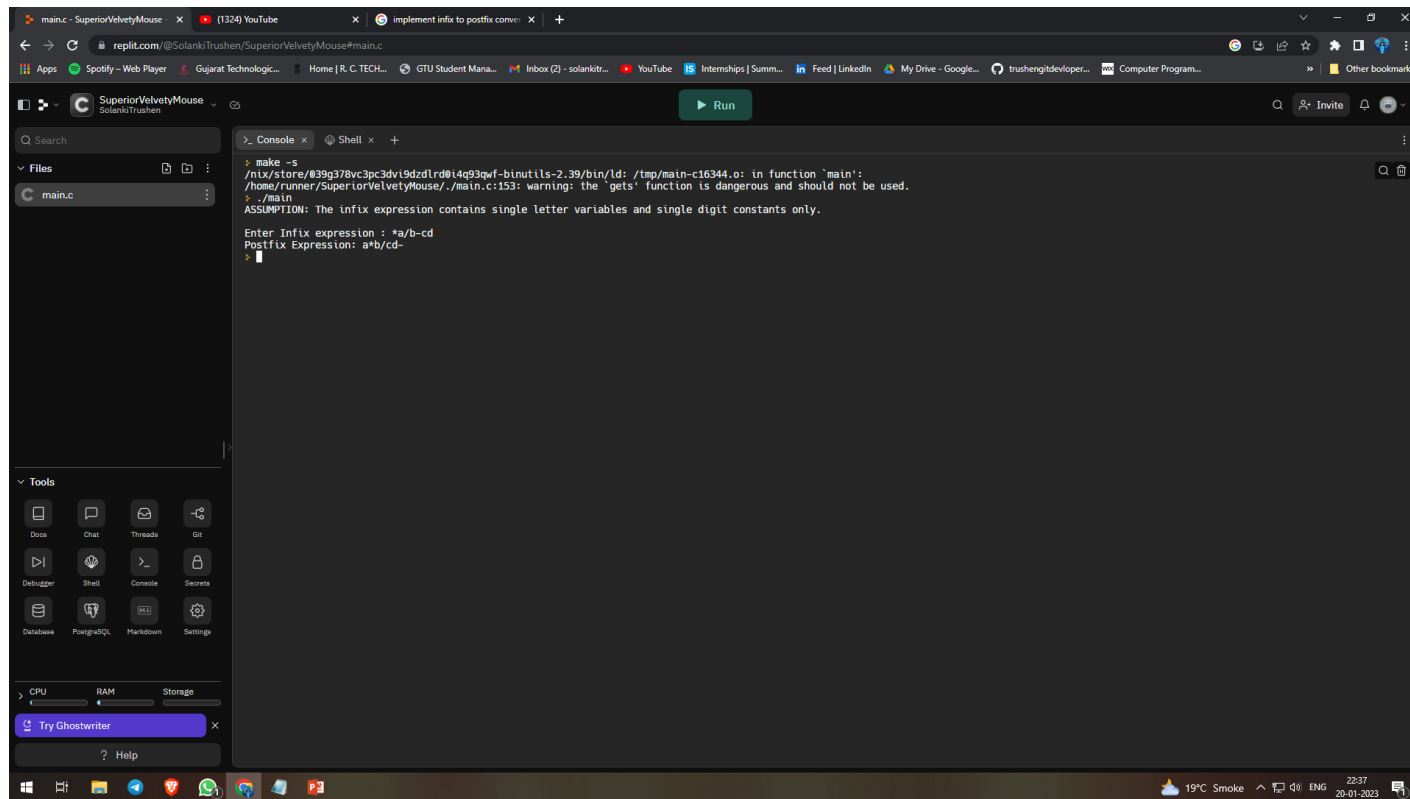


```
41 int is_operator(char symbol)
42 {
43     if(symbol == '^' || symbol == '*' || symbol == '/' || symbol == '+' || symbol == '-')
44     {
45         return 1;
46     }
47     else
48     {
49         return 0;
50     }
51 }
52
53 int precedence(char symbol)
54 {
55     if(symbol == '^')
56     {
57         return(3);
58     }
59     else if(symbol == '*' || symbol == '/')
60     {
61         return(2);
62     }
63     else if(symbol == '+' || symbol == '-')
64     {
65         return(1);
66     }
67     else
68     {
69         return(0);
70     }
71 }
72
73 void InfixToPostfix(char infix_exp[], char postfix_exp[])
74 {
75     int i, j;
76     char item;
```



```
74 {
75     int i, j;
76     char item;
77     char x;
78
79     push('^');
80     strcat(infix_exp, "");
81
82     i=0;
83     j=0;
84     item=infix_exp[i];
85
86     while(item != '\0')
87     {
88         if(item == '(')
89         {
90             push(item);
91         }
92         else if (isdigit(item) || isalpha(item))
93         {
94             postfix_exp[j] = item;
95             j++;
96         }
97         else if(is_operator(item) == 1)
98         {
99             x=pop();
100             while(is_operator(x) == 1 && precedence(x)>= precedence(item))
101             {
102                 postfix_exp[j] = x;
103                 j++;
104                 x = pop();
105             }
106             push(x);
107
108             postfix_exp[j] = item;
109             j++;
110         }
```

Output screen shot



The screenshot displays a web-based IDE interface. The top browser window shows the URL `replit.com/@SolankiTrushen/SuperiorVelvetyMouse#main.c`. The IDE's left sidebar contains a 'Files' panel with `main.c` selected and a 'Tools' panel with icons for Docs, Chat, Threads, Git, Debugger, Shell, Console, Secrets, Database, PostgreSQL, Markdown, and Settings. The main area is a terminal window titled 'Console' with the following output:

```
> make -s
/nix/store/03g378vc3pc3dvi9dzdldrd0l4g93qwf-binutils-2.39/bin/ld: /tmp/main-ci6344.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-
```

The bottom status bar of the IDE shows system information: 19°C, Smoke, 22:37, 20-01-2023, and a language setting of ENG.