PARSER PROJECT

TOPIC: C/C++: TERNARY OPERATOR (WITH OR WITHOUT NESTING)

SUBMITTED BY

Gurdeep Singh , Roll no: 20

Rohit , Roll no: 44

Shivam Dixit, Roll no: 52

SUBMITTED TO

Dr. Ankit Rajpal

C/C++: Ternary Operator

Syntax:

condition **?** *value_if_true* **:** *value_if_false*

The statement evaluates to value_if_true if condition is met, and value_if_false otherwise

Nested Ternary operator:

Ternary operator can be nested. A nested ternary Operator can have many forms like :

- a ? b : c
- a?b:c?e:f?g:h
- 11<14 ? Shivam : 13 > 40 ? Rohit: Gurdeep

Priority of the operators used in grammar:

%right '?' ':'
%left OR
%left AND
%left EQ NE
%left LE GE LT GT
%right NOT
%left '(' ')'

Tokens used by the grammar:

NUMBER :	for any numeric value
VARIABLE :	for any variable.
NEWLINE :	for newline character.
LT :	for less than "<"
GT :	for greater than ">”

LE :	for less than equal to "<=".
GE :	for greater than equal to ">=".
EQ :	for equals to "==".
NE :	for not equals "!="
OR :	for logical OR '||'
NOT :	for logical not "!"
AND :	for any numeric value
T :	for Boolean true
F :	for Boolean false

Context Free Grammar:

```
   0 $accept: stmt $end

   1  stmt: stmt S NEWLINE
   2    | /* empty */

   3  S: EXP S1
   4   | '(' S ')'
   5   | '(' S ')' S1

   6  S1: '?' S2

   7  S2: EXP S3
   8   | S S3

   9  S3: ':' S4

  10 S4: EXP
  11  | S

  12   EXP: EXP LT EXP
```

```
13    | EXP GT EXP
14    | EXP LE EXP
15    | EXP GE EXP
16    | EXP EQ EXP
17    | EXP NE EXP
18    | EXP OR EXP
19    | EXP AND EXP
20    | NOT EXP
21    | VAL
22    | '(' EXP ')'

23   VAL: VARIABLE
24    | NUMBER
25    | T
26    | F
```

Assumptions:

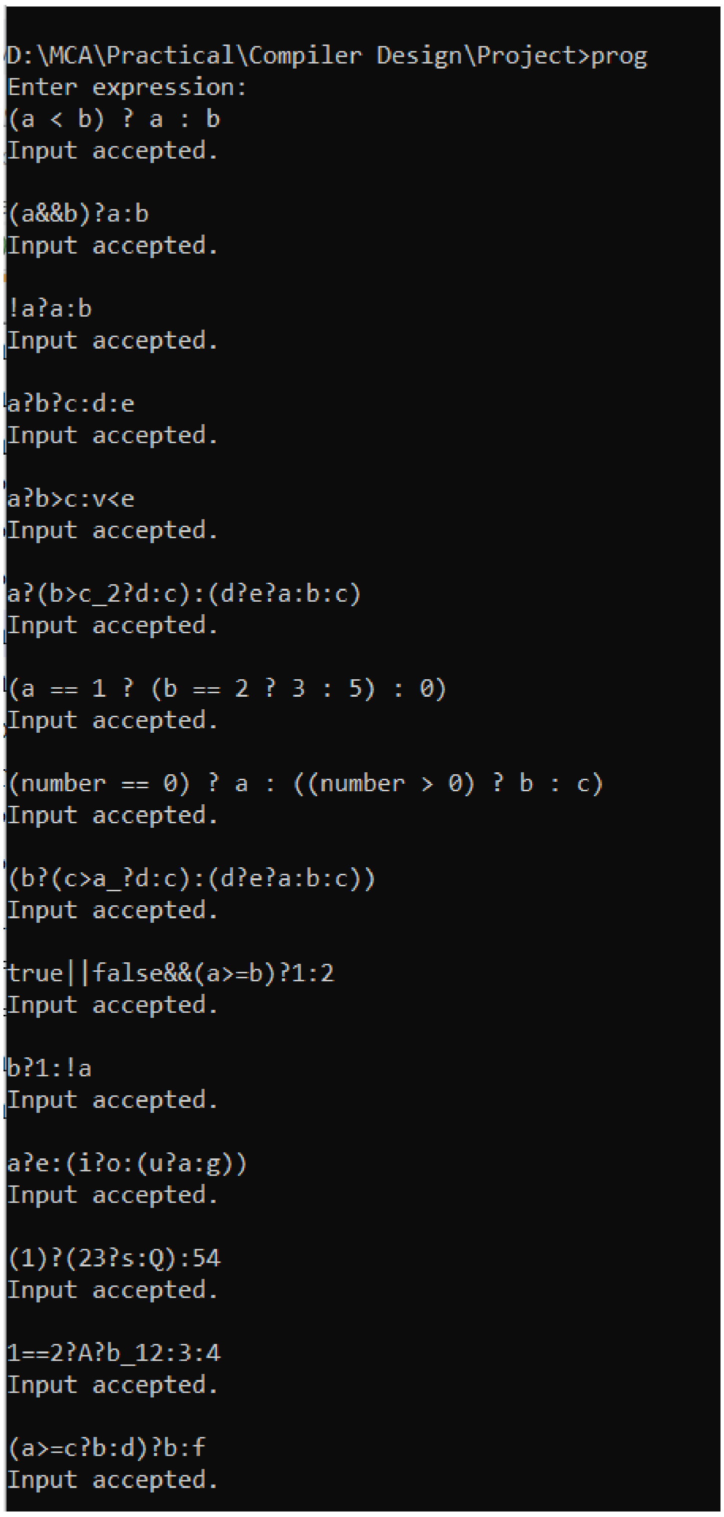
- The parser only accepts boolean, logical and comparison statements only.
- The program will check if the given input is valid or not based on the grammar, rules defined.

Test Cases:

Valid test cases:

1. (a < b) ? a : b
2. (a&&b)?a:b
3. !a?a:b
4. a?b?c:d:e
5. a?b>c:v<e
6. a?(b>c_2?d:c):(d?e?a:b:c)
7. (a == 1 ? (b == 2 ? 3 : 5) : 0)
8. (number == 0) ? a : ((number > 0) ? b : c)

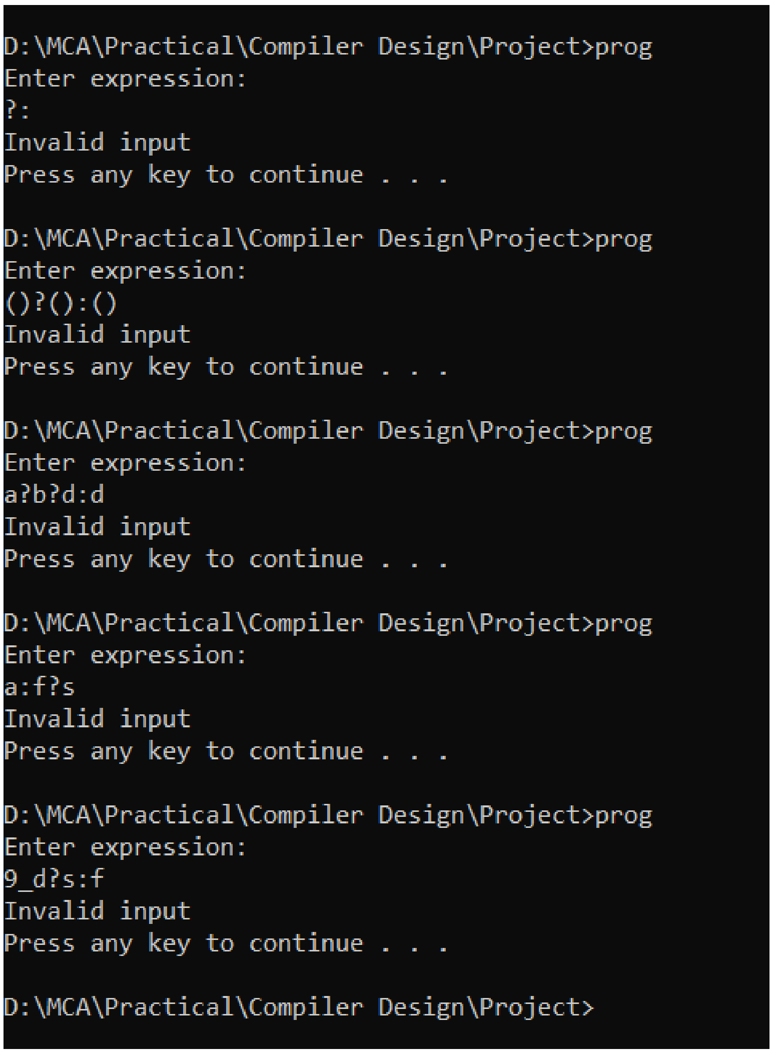
9. (b?(c>a_?d:c):(d?e?a:b:c))
10. true || false&&(a>=b)?1:2
11. b?1:!a
12. a?e:(i?o:(u?a:g))
13. (1)?(1?s:Q):54
14. 1==2?A?b_12:3:4
15. a==b?(c?e:f):d
16. (a?b:s)?c:f
17. (a>=c?b:d)?b:f



```
D:\MCA\Practical\Compiler Design\Project>prog
Enter expression:
(a < b) ? a : b
Input accepted.

(a&&b)?a:b
Input accepted.

!a?a:b
Input accepted.

a?b?c:d:e
Input accepted.

a?b>c:v<e
Input accepted.

a?(b>c_2?d:c):(d?e?a:b:c)
Input accepted.

(a == 1 ? (b == 2 ? 3 : 5) : 0)
Input accepted.

(number == 0) ? a : ((number > 0) ? b : c)
Input accepted.

(b?(c>a_?d:c):(d?e?a:b:c))
Input accepted.

true||false&&(a>=b)?1:2
Input accepted.

b?1:!a
Input accepted.

a?e:(i?o:(u?a:g))
Input accepted.

(1)?(23?s:Q):54
Input accepted.

1==2?A?b_12:3:4
Input accepted.

(a>=c?b:d)?b:f
Input accepted.
```

Invalid test case:

1. ?:
2. ()?():()
3. a?b?d:d
4. a:f?s
5. 9_d?s:f



```
D:\MCA\Practical\Compiler Design\Project>prog
Enter expression:
?:
Invalid input
Press any key to continue . . .

D:\MCA\Practical\Compiler Design\Project>prog
Enter expression:
()?():()
Invalid input
Press any key to continue . . .

D:\MCA\Practical\Compiler Design\Project>prog
Enter expression:
a?b?d:d
Invalid input
Press any key to continue . . .

D:\MCA\Practical\Compiler Design\Project>prog
Enter expression:
a:f?s
Invalid input
Press any key to continue . . .

D:\MCA\Practical\Compiler Design\Project>prog
Enter expression:
9_d?s:f
Invalid input
Press any key to continue . . .

D:\MCA\Practical\Compiler Design\Project>
```