Expressions and Operators - II

hitbullseye

Expression

- Expressions in C programing language combine operands, operators, and variables.
- The evaluations of an expression in C happen according to the operator's precedence.
- Once the expressions are processed, the result will be stored in the variable.

Types of Expressions in C

- Arithmetic expressions
- Relational expressions
- Logical expressions
- Conditional expressions

Arithmetic Expressions

- **Arithmetic expressions can be written in 3 different notations - infix, prefix, and postfix.**

- **In the Prefix notation, the operator is written before the operand in an expression.**

- **On the other hand, in the Postfix notation, the operator is written after the operand. The expressions are evaluated using stack.**

Order of Evaluation For Arithmetic Expressions

- To evaluate arithmetic expressions, the compiler has a pre-defined order in which it evaluates any expression. The order of evaluation followed by the compiler is:
- The expressions with parentheses are evaluated first. If two or more parentheses exist in an expression, the parentheses are evaluated from left to right. In the case of nested parentheses, the innermost parentheses are evaluated first, while the outermost one is evaluated last.
- If there are no parentheses, the order of evaluation of an expression is based on the operator precedence and associativity.

Operator Precedence and Associativity:

Precedence	Operator	Associativity
1	Unary plus, Unary minus	Left to Right
1	Built-in functions	Left to Right
2	Multiplication and Division	Left to Right
3	Addition and Subtraction	Left to Right

Examples

$$Z = 2 + 3 - (\underbrace{2 * 4})$$

$$= \underbrace{2 + 3} - (8)$$

$$= 5 - 8$$

$$= -3$$

Polish (or Prefix) Notation

- In the Polish or Prefix notation, the operator is written before the operand in an expression.
- This notation does not need parenthesis because the expression's evaluation is done in a stack. So, we do not need to specify the execution order to evaluate arithmetic expressions.
- The compiler can process the prefix notation faster than the infix notation because it does not need to process any parentheses or follow precedence rules. An expression in the Polish notation looks like this:

Steps to evaluate the value of a prefix expression:

1. Place a variable var at the last element of the expression.
2. If the variable var points to:

- An operand, push that element to the stack.
- An operator X, pop two elements (operands) from the stack and operate on the popped operands using the operator X. Once the operation is performed, push the calculated value back to the stack.

Steps to evaluate the value of a prefix expression:

3. Decrement the value of the pointer by 1.
4. Repeat steps 2 and 3 until all the elements have been traversed.
5. Return the only value present in the stack at the end.

Reverse Polish (or Postfix) Notation

- In the Reverse Polish or Postfix notation, the operator is written after the operand in the expression.
- There is no need for parenthesis in this notation because the expression's order of execution is already defined in the stack.
- An expression in postfix notation looks like this:

X Y +

hitbullseye

Steps to evaluate the value of a postfix expression

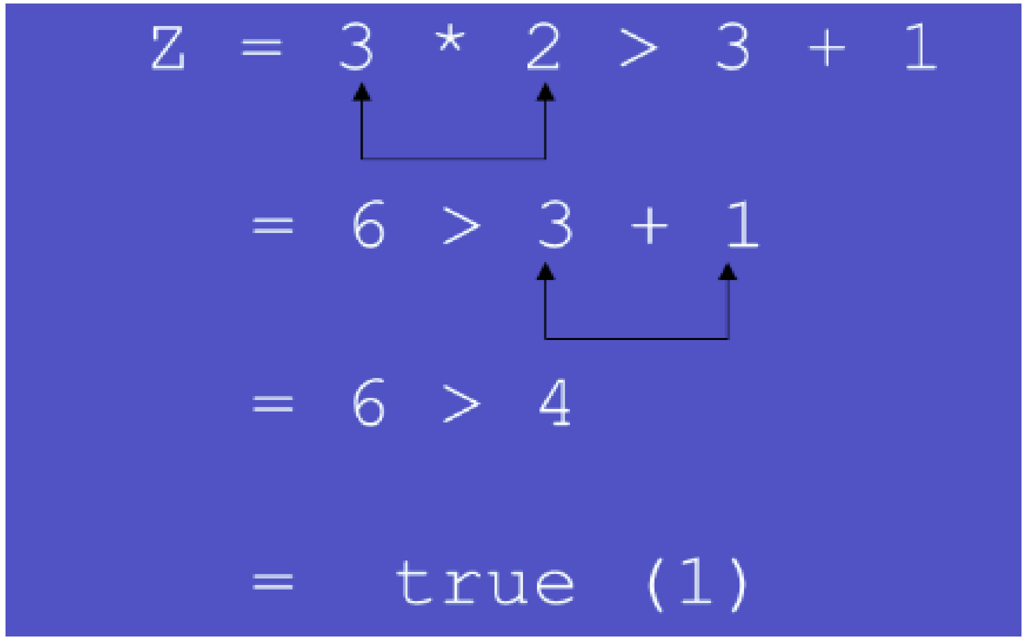
1. Place a pointer at the first element of the string.

- If the pointer points to:
- An operator X, pop the top two elements (operands) from the stack and operate using the operator X.

2. An operand, push that element to the stack.
3. Increase the value of the pointer by 1.
4. Go to step 2 if elements are left to be scanned in the expression.
5. Return the result stored in the stack.

Relational Expressions

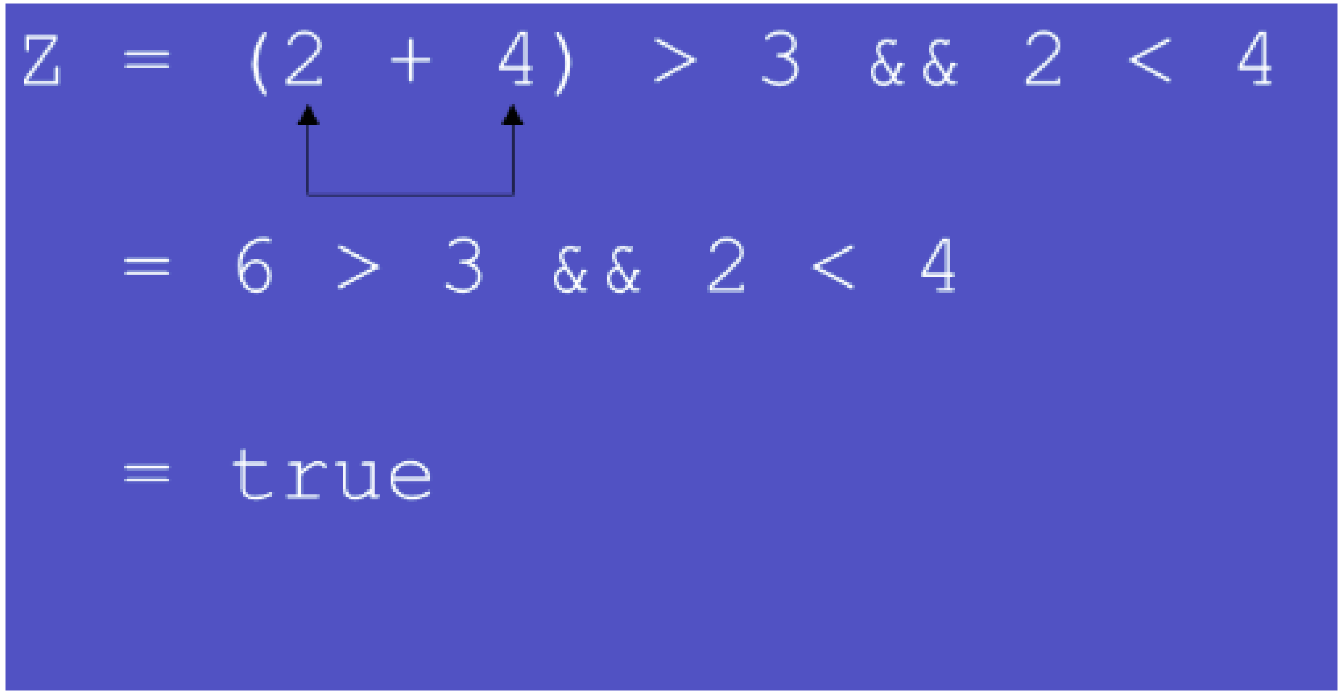
- Relational operators >, <, ==,!= etc are used to compare 2 operands.
- Relational expressions consisting of operands, variables, operators, and the result after evaluation would be either true or false.



$$
\begin{aligned}
Z &= \underbrace{3 * 2} > 3 + 1 \\
&= 6 > \underbrace{3 + 1} \\
&= 6 > 4 \\
&= \text{true } (1)
\end{aligned}
$$

Logical Expressions

- Relational expressions and arithmetic expressions are connected with the logical operators, and the result after an evaluation is stored in the variable, which is either true or false.



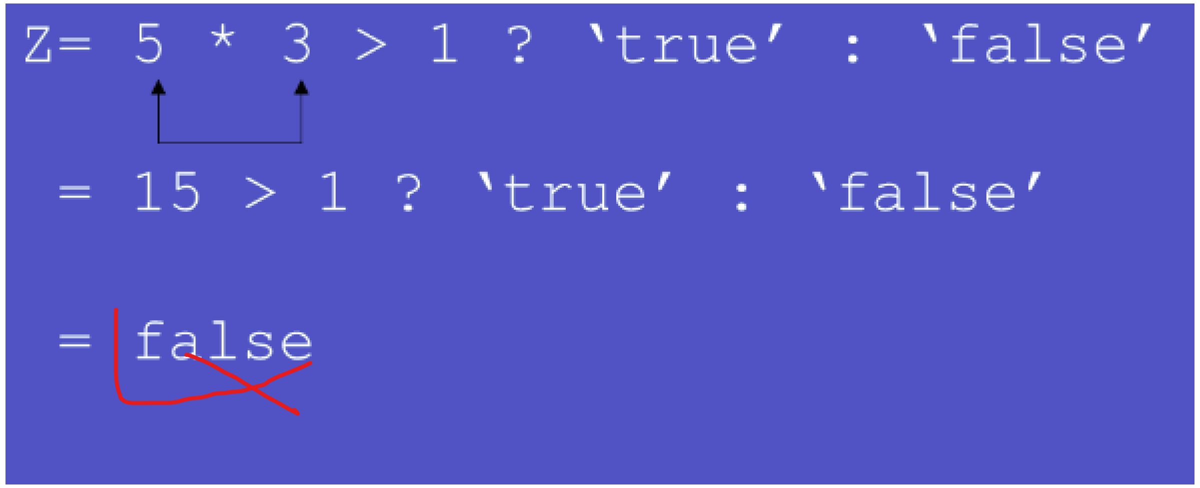
```
Z = (2 + 4) > 3 && 2 < 4

  = 6 > 3 && 2 < 4

  = true
```

Conditional Expressions

- **The general syntax of conditional expression is:**
- **Exp1? Exp2: Exp3**
- **From the given above expressions, the first expression (exp1) is conditional, and if the condition is satisfied, then expression2 will be executed; otherwise, expression3 will be performed.**



```
Z= 5 * 3 > 1 ? 'true' : 'false'

 = 15 > 1 ? 'true' : 'false'

 = false
```

References Link

- https://www.educba.com/expression-in-c/
- https://www.javatpoint.com/c-expressions

THANK YOU

hitbullseye