Operators Expressions and statements

- Operators are functions that use a symbolic name.
    - ➔ Perform mathematical or logical functions.
- Operators are predefined in C, just like they are in most operators tent to be combined with the infix style.
- A logical operator (sometimes called a “Boolean operator”) is and operator that returns a Boolean result that’s based on the Boolean result of one or two other expressions.
- An arithmetic operator is a mathematical function that takes two operands and performs a calculation on them.
- Other operators include assignment, relational (<, >, !=), bitwise (<<.>>, ~)

- Statements form the basic program steps of C, and most statements are constructed from expressions.

Expressions

- An expression consists of a combination of operators and operands.
    - ➔ Operands are what an operator operates on.
    - ➔ Operands can be constants , variables , or combinations of the two.
    - ➔ Every expression has a value.

-6 (minus is a unary operator, single argument)
4+21 (the plus operator in between two constants)
a*(b + c/d)/20 (* multiplication , + for addition, / for divison,)
Q = 5*2 (statement)
X = ++q%3 (++ is an incrementation operator) (% is the modulus operator)
q>3 (comparison operator is q greater than 3)

Statements

- Statements are the building blocks of a program (declaration)
    - ➔ A program is a series of statements with special syntax ending with a semicolon (simple statements)
    - ➔ A complete instruction to the computer
- Declaration statement: **int Jason**;
- Assignment Statement : **Jason = 5;**
- Function call statement: **printf(“Jason”);**
- Structure Statement: **while (Jason < 20) Jason = Jason +1;**
- Return Statement: **return 0;**

C considers any expression to be a statement if you append a semicolon (expression statements)

- ➔ So , 8; and 3 - 4 ; are perfectly valid in C

Compound Statements

- Two or more statements grouped together by enclosing them in braces (block)

```
int index = 0;
while (index < 10)
{
      printf(“hello”);
      index = index +1;
}
```