KCA102 -Expressions & Operators

- An **expression** is a combination of operands (variables, constants), operators and function call.
  A=6, B=8
  C= A+B // expression
  A and B operands,
  + and = Operator ,
  C identifier/variable // storing output
  C=6+8
  C<-14
- It can be arithmetic, logical and relational
  Example: - int z= x+y // arithmatic expression
  a>b //relational a==b // logical
  sum_func(a, b) // function call
- Expressions consisting entirely of constant values are called constant expressions.
  Const int pi=3.14
  Int C;
  C=4
  C=20
  C+ 2 * pi

- So, the expression 121 + 17 - 110 is a constant expression because each of the terms of the expression is a constant value.
- But if i were declared to be an integer variable, the expression 180 + 2 – j would not represent a constant expression.

Operator

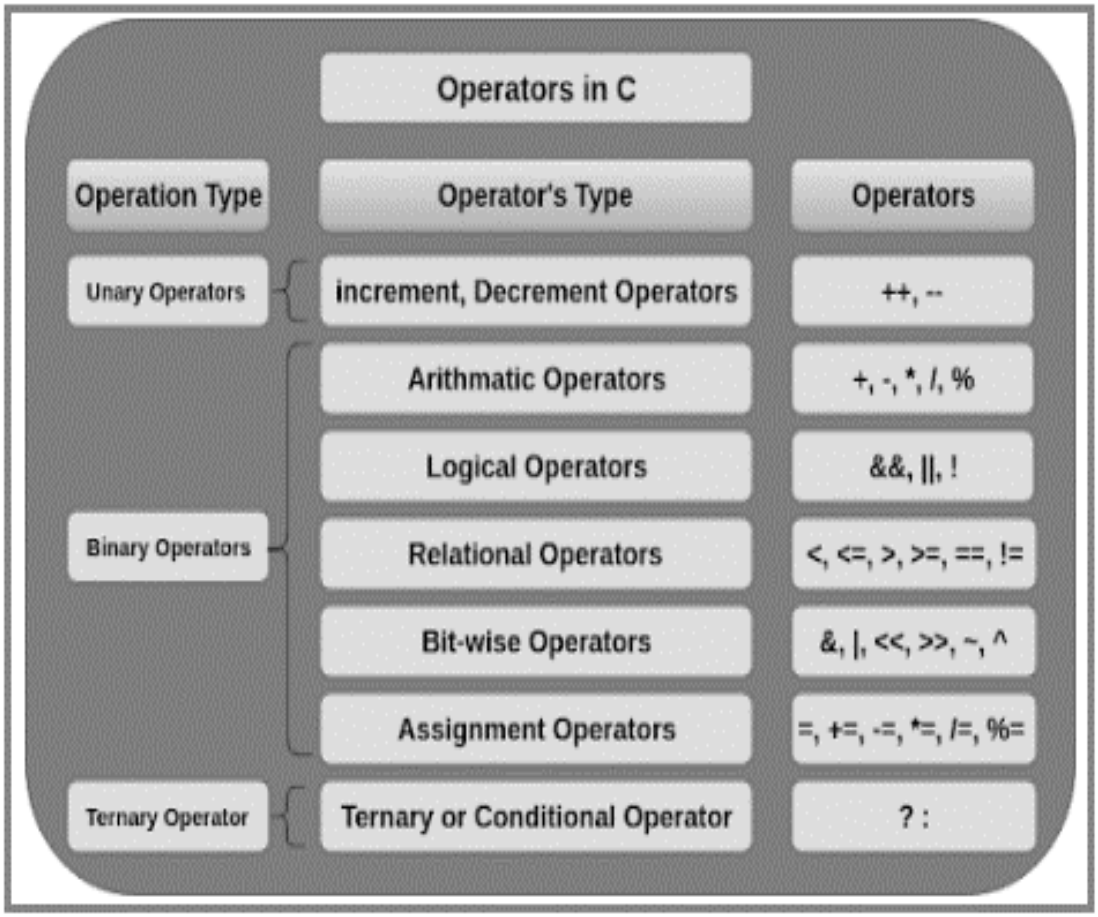
C programming language offers various types of operators having different functioning capabilities.

1. **Unary Operators**
2. **Arithmetic Operators**
3. **Relational Operators**
4. **Logical Operators**
5. **Assignment Operators**
6. **Increment and Decrement Operators**
7. **Conditional Operator**
8. **Bitwise Operators**
9. **Special Operators**

Operator is a symbol used to perform some operation on variables, operands or with the constant. Some operator required 2 operands to perform operation or Some required single operation.

In C programming Language there are several types operators like arithmetic operator, assignment, increment, decrement, logical, conditional, comma, sizeof , bitwise and others.

1. Unary Operators – Operated on one Operand
2. Binary Operators- operated on at least **two operands**
3. Ternary or Conditional Operator Special Type of Operator condition evaulation



1. **Unary Operator:**
   - Unary Operators are special type of Operators, operate on one operand
   - **Following Unary Operators are used in C programming** language.

SNo	Operators	Symbols
2	**Unary minus**	-
3	**Increment operator**	++
4	**Decrement operator**	--
5	**Address of Operator**	&
6.	**Logical Negation**	!
7.	**sizeof() Operator**	sizeof()

   - Unary (+) and Unary (-) is different from addition and subtraction. Can be operated on one Operand
   - Increment Operator ++ increases the value by 1 whereas
   - Decrement Operator -- decreases the value by 1.
     Increment operator and Decrement operator can be used in pre/post order
     - Pre increment(++variable)
     - Post increment(variable++)
     - Pre decrement(–variable)
     - Post decrement(variable–)
   - a++ and ++a are equivalent to a += 1.
   - a-- and --a are equivalent to a -= 1.
   - ++a op b is equivalent to a ++; a op b;
   - a++ op b is equivalent to a op b; a++;

   Example

   Let b = 10

(++b)+b+b = 33
b+(++b)+b = 33
b+b+(++b) = 31
b+b*(++b) = 132

Example: Print them and decrementing each time using postfix mode for x and prefix mode for y

```
#include <stdio.h>
void main()
{
int x=5, y=5;
/*Print them and decrementing each time using postfix mode for a and prefix mode for b.*/
printf("\n%d %d",x--,--y);
printf("\n%d %d",x--,--y);
printf("\n%d %d",x--,--y);
printf("\n%d %d",x--,--y);
printf("\n%d %d",x--,--y);
printf("%d", !a);
}
```

Example 2: Increment and Decrement Operators

```
#include <stdio.h>
int main()
{
   int a = 10, b = 100;
   float c = 10.5, d = 100.5;
   printf("++a = %d \n", ++a);
   printf("--b = %d \n", --b);
   printf("++c = %f \n", ++c);
   printf("--d = %f \n", --d);
   return 0;
}
```

Example : sizeof() Operator

```
#include <stdio.h>
int main()
{
   int a;
   float b;
   double c;
   char d;
   printf("Size of int=%lu bytes\n",sizeof(a));
   printf("Size of float=%lu bytes\n",sizeof(b));
   printf("Size of double=%lu bytes\n",sizeof(c));
   printf("Size of char=%lu byte\n",sizeof(d));
   return 0;
}
```

2. **Arithmetic Operator (+, -, *, /, %)**

When both the operand are integer then it is called integer arithmetic and the result is always integer. When both the operand are floating point then it is called floating arithmetic and

when operand is of integer and floating point then it is called mix type or mixed mode arithmetic . And the result is in float type.

```
void main()
{
   int a = 12, b = 14, c;
   c = a + b;
   printf("a+b = %d \n",c);
   c = a-b;
   printf("a-b = %d \n",c);
   c = a*b;
   printf("a*b = %d \n",c);
   c = a/b;
   printf("a/b = %d \n",c);
   c = a%b;
   printf("Remainder when a divided by b = %d \n",c);
      }
```

3. **C Assignment Operators**
   - An assignment operator is used for assigning a value to a variable. The most common assignment operator is =

Operator	Example	Same as
=	a = b	a = b
+=	a += b	a = a+b
-=	a -= b	a = a-b
*=	a *= b	a = a*b
/=	a /= b	a = a/b
%=	a %= b	a = a%b

Example : Assignment Operators

```
#include <stdio.h>
int main()
{    int a = 7, c;
   c = a;      // c is 7
   printf("c = %d\n", c);
   c += a;
   printf("c = %d\n", c);
   c -= a;
   printf("c = %d\n", c);
   c *= a;
  printf("c = %d\n", c);
   c /= a;
   printf("c = %d\n", c);
   c %= a;
   printf("c = %d\n", c);
}
```

4. **Relational operators:** Relational operators are used to comparing two quantities or values.

Operator	Description
==	Is equal to
!=	Is not equal to
>	Greater than

<	Less than
>=	Greater than or equal to
<=	Less than or equal to
%=	Modulus then assign
<<=	Left shift and assign
>>=	Right shift and assign
&=	Bitwise AND assign
^=	Bitwise exclusive OR and assign
|=	Bitwise inclusive OR and assign

5. **Logical Operators**

There are three logical operators when we test more than one condition to make decisions.

Operator	Description
&& logical AND	And operator. It performs logical conjunction of two expressions. (if both expressions evaluate to True, result is True. If either expression evaluates to False, the result is False)
|| logical OR	Or operator. It performs a logical disjunction on two expressions. (if either or both expressions evaluate to True, the result is True)
! logical NOT	Not operator. It performs logical negation on an expression.

KCA102 Unit 1.5 Bitwise Operators

6. **Bitwise Operators:** C provides a special operator for bit operation between two variables.

There are six different types of **Bitwise Operators in C**. These are:

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) =12, i.e., 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) = 61, i.e., 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, i.e., 0011 0001
~	Binary One's Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = ~(60), i.e,. A=1100 ~1100= 0011
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 = 240 i.e., 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 = 15 i.e., 0000 1111

- **The Bitwise AND (&) in C:** The C compiler recognizes the Bitwise AND with & operator. It takes two operands and performs the AND operation for every bit of the two operand numbers. It is a binary operator. The output of this operator will result in 1 only if both bits are 1.
- For Example : **72 & 184 = 8**

A=01001000 B= 10111000

A&B

01001000 &
10111000 =

00001000

- **The Bitwise OR (|) in C:** The C compiler recognizes the Bitwise OR with | operator. It takes two operands and performs the OR operation for every bit of the two operand numbers. It is also a binary operator. The output of this operator will result in 1 if any one of the two bits is 1.
- **Example** 72 | 184 = 248

1001000 |
10111000 =

11111000

- **The Bitwise XOR (^) in C:** The C compiler recognizes the Bitwise XOR with ^ operator. It takes two operands and performs the XOR operation for every bit of the two operand numbers. It is also a binary operator. The output of this operator will result in 1 if both the bits have different values.
- **Binary One's Complement or Bitwise NOT operator (~) in C:** The C compiler recognizes the Bitwise NOT with ~ operator. It takes only one operand and performs the inversion of all digits of it. It is a unary operator. The output of this operator will invert all the existing bits of that operand.
- **Bitwise Left shift operator (<<) in C:** The C compiler recognizes the **left shift operation** with this <<. It takes only two operands and shifts all the bits of the first operand to the left. The second operand decides how many numbers of places this operator will shift its bits. It is a binary operator.
- **Bitwise Right shift operator (>>) in C:** The C compiler recognizes the left shift operation with this >>. It takes only two operands and shifts all the bits of the first operand to the right. The second operand decides how many numbers of places this operator will shift its bits. It is a binary operator.

TRUTH TABLE FOR BIT WISE OPERATION & BIT WISE OPERATORS:

x	y	x|y	x&y	x^y
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

Example
Consider x=40 and y=80. Binary form of these values are given below.

x= 00101000
y= 01010000
X|Y= 01111000
X&Y 00000000
All bit wise operations for x and y are given below.
X&y = 00000000 (binary) = 0 (decimal)
X|y = 01111000 (binary) = 120 (decimal)

- ~x = **Bit wise NOT :**
  Value of 40 in binary is 00000000000000000000000000000000
  So, all 0's are converted into 1's in bit wise NOT operation.

x=101000
~ 010111
x^y = 01111000 (binary) = 120 (decimal)

- **Bit wise left shift and right shift :** In left shift operation "x << 1 ", 1 means that the bits will be left shifted by one place. If we use it as "x << 2 ", then, it means that the bits will be left shifted by 2 places.
  x=00101000
  x << 1 = 01010000 (binary) = 80 (decimal)
  x >> 1 = 00010100 (binary) = 20 (decimal)

EXAMPLE PROGRAM FOR BIT WISE OPERATORS IN C:

```
#include <stdio.h>
main()
{
  unsigned int a = 60; /* (60)10 = (0011 1100)2 */
  unsigned int b = 13; /* (13)10 = 0000 1101 */
  int c = 0;
  c = a & b;       /* 12 = 0000 1100 */
  printf("Line 1 - Value of c is %d\n", c );
  c = a | b;       /* 61 = 0011 1101 */
  printf("Line 2 - Value of c is %d\n", c );
  c = a ^ b;       /* 49 = 0011 0001 */
  printf("Line 3 - Value of c is %d\n", c );
  c = ~a;          /*-61 = 1100 0011 */
  printf("Line 4 - Value of c is %d\n", c );
  c = a << 2;     /* 240 = 1111 0000 */
  printf("Line 5 - Value of c is %d\n", c );
  c = a >> 2;     /* 15 = 0000 1111 */
  printf("Line 6 - Value of c is %d\n", c );
}

#include <stdio.h>
int main()
{
int x = 40, y = 80,AND_op,OR_op,XOR_op,NOT_op ;
AND_op = (x&y);
OR_op = (x|y);
NOT_op = (~x);
XOR_op = (x^y);
printf("AND_op value = %d\n",AND_op );
```

```
    printf("OR_op value = %d\n",OR_op );
    printf("NOT_op value = %d\n",NOT_op );
    printf("XOR_op value = %d\n",XOR_op );
    printf("left_shift value = %d\n", x << 1);
    printf("right_shift value = %d\n", x >> 1);
    }
```

Example program using all the bitwise operators.

```
#include <stdio.h>
int main()
 {
int  x = 20, y = 21; // x = 20 (00010100), y = 21 (00010101)
        int r = 0;
  r = x & y; /* 20 = 010100 */
  printf(" Result of Bitwise AND is %d \n", r );
 r = x | y; /* 21 = 010101 */
  printf("Result of Bitwise OR is %d \n", r );
  r = x ^ y; /* 1 = 0001 */
  printf(" Rresult of Bitwise XOR is %d \n", r );
  r = ~x;
  printf(" Result of Bitwise NOT is %d \n", r );
  r = x << 1;
  printf(" Result of Bitwise Left Shift is %d \n", r );
  r = x >> 1;
  printf(" Result of Bitwise Right Shift is %d \n", r );
return 0;
}
```

OUTPUT:

```
The result of Bitwise AND is 20
 The result of Bitwise OR is 21
 The result of Bitwise XOR is 1
 The result of Bitwise NOT is -21
 The result of Bitwise Left Shift is 40
 The result of Bitwise Right Shift is 10
```

References

1. Hanly J. R. and Koffman E. B.,"Problem Solving and Program Design in C", Pearson Education.
2. Schildt H., "C- The Complete Reference", McGraw-Hill.
3. Kanetkar Y., "Let Us C", BPB Publications.
4. Gottfried B., "Schaum's Outlines- Programming in C", McGraw-Hill Publications.
5. Kochan S.G., "Programming in C", Addison-Wesley.
6. Dey P. and Ghosh M., "Computer Fundamentals and Programming in C", Oxford University Press.
7. Goyal K. K., Sharma M. K. and Thapliyal M. P. "Concept of Computer and C Programming", University Science Press.
8. Goyal K. K. and Pandey H.M., Trouble Free C", University Science Press