C PROGRAMMING

Bitwise Operators

In the arithmetic-logic unit (which is within the CPU), mathematical operations like: addition, subtraction, multiplication and division are done in bit-level. To perform bit-level operations in C programming, bitwise operators are used.

Operators	Meaning of operators
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
>>	Shift right
<<	Shift left

Other Operators

Comma Operator

Comma operators are used to link related expressions together.

sizeof Operator

The sizeof is a unary operator that returns the size of data (constants, variables, array, structure, etc).

ternary operator ?: - will be discussed with if else.

reference operator & - will be discussed with pointer.

dereference operator * - will be discussed with pointer.

member selection operator . - will be discussed with union

operators precedence and associativity

1. “/” and “*” are with same precedence

2. “+” and “-” are with same precedence

operators precedence and associativity

10 + 20 * 30

Multiplication will be calculated first, it has higher precedence than +.	10+600
Now addition will be performed as + has lower precedence	610

operators precedence and associativity

100 / 10 * 10

Operators Associativity is used when two operators of same precedence appear in an expression. Associativity can be either Left to Right or Right to Left.

For example: ‘*’ and ‘/’ have same precedence and their associativity is Left to Right, so the expression “100 / 10 * 10” is treated as “(100 / 10) * 10”.

operators precedence and associativity

100 + 200 / 10 - 3 * 10

Division will be performed first it has higher precedence than+ and -. It has same precedence as * but high associativity.

Multiplication will be performed second it has higher precedence than + and -. It has same precedence as / but low associativity.

Addition will be performed third it has lower precedence than / and *. It has same precedence as – but higher associativity.

Subtraction will be performed last it has lower precedence than / and *. It has same precedence as + but lower associativity.

```
int x = 0;
int f1()
{
    x = 5;
    return x;
}
int f2()
{
    x = 10;
    return x;
}
void main()
{
    int p = f1() + f2();
    printf("%d ", x);
}
```

Output = ?

Output = 10

Associativity is only used when there are two or more operators of same precedence.
The point to note is associativity doesn’t define the order in which operands of a single operator are evaluated. For example, consider the following program, associativity of the + operator is left to right, but it doesn’t mean f1() is always called before f2(). The output of the following program is in-fact compiler dependent.

All operators with the same precedence have same associativity
This is necessary, otherwise, there won’t be any way for the compiler to decide evaluation order of expressions which have two operators of same precedence and different associativity. For example + and – have the same associativity.

```
void main()
{
    int a;
    a = 1, 2, 3;
    printf("%d", a);
}

// Evaluated as (a = 1), 2, 3
```

Comma has the least precedence among all operators and should be used carefully For example consider the following program, the output is 1.

Associativity

Operator	Associativity
++ – – Postfix	left-to-right
++ – – Prefix	right-to-left
* / %	left-to-right
+ –	left-to-right
< <=	left-to-right
&& || !	left-to-right
=	right-to-left
+= -=	right-to-left
*= /=	right-to-left
%= &=	right-to-left
,	left-to-right

Postfix and Prefix

```
x = 5;
y = x++;
printf("%d, %d",
x, y);
```

Output = 6, 5

```
x = 5;
y = ++x;
printf("%d, %d",
x, y);
```

Output = 6, 6