Bitwise Operators

Bitwise operator works on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ is as follows –

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Assume A = 60 and B = 13 in binary format, they will be as follows –

A = 0011 1100

B = 0000 1101

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A = 1100 0011

The following table lists the bitwise operators supported by C. Assume variable 'A' holds 60 and variable 'B' holds 13, then –

Show Examples

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 Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = -61, i.e,. 1100 0011 in 2's complement form.
<<	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

Pre-processor directive:

Preprocessor extends the power of C programming language. Line that begin with # are called preprocessing directives.

Use of #include

Let us consider very common preprocessing directive as below:

```
#include <stdio.h>
```

Here, "stdio.h" is a header file and the preprocessor replace the above line with the contents of header file.

Use of #define

Preprocessing directive #define has two forms. The first form is:

```
#define identifier token_string
```

token_string part is optional but, are used almost every time in program.

Example of #define

```
#define c 299792458 /*speed of light in m/s */
```

The token string in above line 2299792458 is replaced in every occurance of symbolic constant c.

Macro with argument

Preprocessing directive #define can be used to write macro definitions with parameters as well in the form below:

```
#define identifier(identifier 1,.....identifier n)
token_string
```

Again, the token string is optional but, are used in almost every case. Let us consider an example of macro definition with argument.

```
#define area(r) (3.1415*(r)*(r))
```

Here, the argument passed is r. Every time the program encountersarea(argument), it will be replace by (3.1415*(argument)*(argument)). Suppose, we passed (r1+5) as argument then, it expands as below:

```
area(r1+5) expands to (3.1415*(r1+5)*(r1+5))
```

C Program to find area of a circle, passing arguments to macros. [Area of circle= πr^2]

```
#include <stdio.h>

#define PI 3.1415

#define area(r) (PI*(r)*(r))

int main(){

    int radius;

    float area;

    printf("Enter the radius: ");

    scanf("%d",&radius);

    area=area(radius);
```

```
    printf("Area=%.2f",area);

    return 0;

}
```

Enumerated Type

enumeration is a user-defined data type consists of integral constants and each integral constant is give a name. Keyword enum is used to defined enumerated data type.

```
enum type_name{ value1, value2,...,valueN };
```

Here, type_name is the name of enumerated data type or tag.
And value1,value2,....,valueN are values of type type_name.

By default, value1 will be equal to 0, value2 will be 1 and so on but, the programmer can change the default value.

```
// Changing the default value of enum elements

enum suit{

    club=0;

    diamonds=10;

    hearts=20;

    spades=3;

};
```

Declaration of enumerated variable

Above code defines the type of the data but, no any variable is created.
Variable of type enum can be created as:

```
enum boolean{

    false;

    true;

};

enum boolean check;
```

Here, a variable check is declared which is of type enum boolean.

Example of enumerated type

```
#include <stdio.h>

enum week{ sunday, monday, tuesday, wednesday, thursday,
friday, saturday};

int main(){

    enum week today;

    today=wednesday;

    printf("%d day",today+1);

    return 0;

   }
```

Output

```
4 day
```

You can write any program in C language without the help of enumerations but, enumerations helps in writing clear codes and simplify programming.

Conditional Compilation / Conditional Pre-processing

#if (macro within a condition)

if block

#else

else block

#endif

```
#define max 100
void main()
{
#if max==100
        printf("Yes");
#else
        printf("Noo");
#endif
}
```

execution : gcc –E file.c

[-E - Preprocess only , do not compile or link or assemble]

```
void main()
{
printf("Yes");
}
```

Now the above code will be sent for Compilation

nested if else

```
#if(condition )
```

if block

#elif(condition)

else if block

#else

else block

#endif