UNIT II

OPERATORS

Types of Operators	Description
Arithmetic_operators	These are used to perform mathematical calculations like addition, subtraction, multiplication, division and modulus
Assignment_operators	These are used to assign the values for the variables in C programs.
Relational operators	These operators are used to compare the value of two variables.
Logical operators	These operators are used to perform logical operations on the given two variables.
Bit wise operators	These operators are used to perform bit operations on given two variables.
Conditional (ternary) operators	Conditional operators return one value if condition is true and returns another value is condition is false.
Increment/decrement operators	These operators are used to either increase or decrease the value of the variable by one.
Special operators	&, *, sizeof( ) and ternary operator

ARITHMETIC OPERATORS IN C:

C Arithmetic operators are used to perform mathematical calculations like addition, subtraction, multiplication, division and modulus in C programs.

Arithmetic Operators/Operation	Example
+ (Addition)	A+B
– (Subtraction)	A-B
* (multiplication)	A*B
/ (Division)	A/B
% (Modulus)	A%B

EXAMPLE PROGRAM FOR C ARITHMETIC OPERATORS:

```
// Working of arithmetic operators
#include <stdio.h>
int main()
{    int a = 9,b = 4, 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);

    return 0;
}
```

Output

```
a+b = 13
a-b = 5
a*b = 36
a/b = 2
Remainder when a divided by b=1
```

Assignment Operators

The Assignment Operator evaluates an expression on the right of the expression and substitutes it to the value or variable on the left of the expression.

Example
x = a + b
Here the value of a + b is evaluated and substituted to the variable x.
In addition, C has a set of shorthand assignment operators of the form.

Var oper=exp;
Here var is a variable, exp is an expression and oper is a C binary arithmetic operator.
The operator oper = is known as shorthand assignment operator.

Example
x + = 1 is same as x = x + 1

The commonly used shorthand assignment operators are as follows

Shorthand assignment operators

Statement with simple assignment operator	Statement with shorthand operator
a = a + 1	a += 1
a = a – 1	a -= 1
a = a * (n+1)	a *= (n+1)
a = a / (n+1)	a /= (n+1)
a = a % b	a %= b

Relational Operators

Often it is required to compare the relationship between operands and bring out a decision and program accordingly. This is when the relational operator come into picture. C supports the following relational operators.

Operator	Meaning
<	is less than
<=	is less than or equal to
>	is greater than
>=	is greater than or equal to
==	is equal to
!=	is not equal to

It is required to compare the marks of 2 students, salary of 2 persons, we can compare them using relational operators.

A simple relational expression contains only one relational operator and takes the following form.

exp1 relational operator exp2

Where exp1 and exp2 are expressions, which may be simple constants, variables or combination of them. Given below is a list of examples of relational expressions and evaluated values.

6.5 <= 25 TRUE
-65 > 0 FALSE
10 < 7 + 5 TRUE

Relational expressions are used in decision making statements of C language such as if, while and for statements to decide the course of action of a running program.

Logical Operators

C has the following logical operators, they compare or evaluate logical and relational expressions.

Operator	Meaning
&&	Logical AND
||	Logical OR
!	Logical NOT

Logical AND (&&)

This operator is used to evaluate 2 conditions or expressions with relational operators simultaneously. If both the expressions to the left and to the right of the logical operator is true then the whole compound expression is true.

Example

a > b && x = = 10

The expression to the left is a > b and that on the right is x == 10 the whole expression is true only if both expressions are true i.e., if a is greater than b and x is equal to 10.

Logical OR (||)

The logical OR is used to combine 2 expressions or the condition evaluates to true if any one of the 2 expressions is true.

Example

a < m || a < n

The expression evaluates to true if any one of them is true or if both of them are true. It evaluates to true if a is less than either m or n and when a is less than both m and n.

Logical NOT (!)

The logical not operator takes single expression and evaluates to true if the expression is false and evaluates to false if the expression is true. In other words it just reverses the value of the expression.

For example

! (x >= y) the NOT expression evaluates to true only if the value of x is neither greater than or equal to y

Bitwise Operators

C has a distinction of supporting special operators known as bitwise operators for manipulation data at bit level. A bitwise operator operates on each bit of data. Those operators are used for testing, complementing or shifting bits to the right on left. Bitwise operators may not be applied to a float or double.

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise Exclusive
<<	Shift left
>>	Shift right

Conditional or Ternary Operator

The conditional operator consists of 2 symbols the question mark (?) and the colon (:)

The syntax for a ternary operator is as follows
exp1 ? exp2 : exp3

The ternary operator works as follows

exp1 is evaluated first. If the expression is true then exp2 is evaluated & its value becomes the value of the expression. If exp1 is false, exp3 is evaluated and its value becomes the value of the expression. Note that only one of the expression is evaluated.

For example

```
a = 10;
b = 15;
x = (a > b) ? a : b
```

Here x will be assigned to the value of b. The condition follows that the expression is false therefore b is assigned to x.

```
/* Example : To find the maximum value using conditional operator)
#include
void main() {
int i,j,larger;
printf (“Input 2 integers : ”);
scanf(“%d %d”,&i, &j);
larger = i > j ? i : j;
printf(“The largest of two numbers is %d \n”, larger);
}
```

Output

```
Input 2 integers : 34   45
The largest of two numbers is 45
```

Increment and Decrement Operators

The increment and decrement operators are one of the unary operators which are very useful in C language. They are extensively used in for and while loops. The syntax of the operators is given below

1. ++ variable name
2. variable name++
3. – –variable name
4. variable name– –

The increment operator ++ adds the value 1 to the current value of operand and the decrement operator – – subtracts the value 1 from the current value of operand. ++variable name and variable name++ mean the same thing when they form statements independently, they behave differently when they are used in expression on the right hand side of an assignment statement.

Consider the following

m = 5;

y = ++m; (prefix)

In this case the value of y and m would be 6
Suppose if we rewrite the above statement as

m = 5;
y = m++; (post fix)

Then the value of y will be 5 and that of m will be 6. A prefix operator first adds 1 to the operand and then the result is assigned to the variable on the left. On the other hand, a postfix operator first assigns the value to the variable on the left and then increments the operand.

Special Operators

C supports some special operators of interest such as comma operator, size of operator, pointer operators (& and *) and member selection operators (. and ->). The size of and the comma operators are discussed here.

The Comma Operator

The comma operator can be used to link related expressions together. A comma-linked list of expressions are evaluated left to right and value of right most expression is the value of the combined expression.
For example the statement
value = (x = 10, y = 5, x + y);
First assigns 10 to x and 5 to y and finally assigns 15 to value. Since comma has the lowest precedence in operators the parenthesis is necessary. Some examples of comma operator are
In for loops:
for (n=1, m=10, n <=m; n++,m++)
In while loops
While (c=getchar(), c != ‘10’)
Exchanging values
t = x, x = y, y = t;

The size of Operator

The operator size of gives the size of the data type or variable in terms of bytes occupied in the memory. The operand may be a variable, a constant or a data type qualifier.
Example

m = sizeof (sum);
n = sizeof (long int);
k = sizeof (235L);

The size of operator is normally used to determine the lengths of arrays and structures when their sizes are not known to the programmer. It is also used to allocate memory space dynamically to variables during the execution of the

program. Example program that employs different kinds of operators. The results of their evaluation are also shown in comparision

```
main()
{
int a, b, c, d;
a = 15; b = 10; c = ++a-b;
printf (“a = %d, b = %d, c = %d\n”, a,b,c);
d=b++ + a;
printf (“a = %d, b = %d, d = %d\n, a,b,d);
printf (“a / b = %d\n, a / b);
printf (“a %% b = %d\n, a % b);
printf (“a *= b = %d\n, a *= b);
printf (“%d\n, (c > d) ? 1 : 0 );
printf (“%d\n, (c < d) ? 1 : 0 );
}
```

Notice the way the increment operator ++ works when used in an expression. In the statement c = ++a – b; new value a = 16 is used thus giving value 6 to C. That is a is incremented by 1 before using in expression.
However in the statement d = b++ + a; The old value b = 10 is used in the expression. Here b is incremented after it is used in the expression. We can print the character % by placing it immediately after another % character in the control string. This is illustrated by the statement.

printf(“a %% b = %d\n”, a%b);
This program also illustrates that the expression
c > d ? 1 : 0
Assumes the value 0 when c is less than d and 1 when c is greater than d.

C - Input and Output

When we say **Input**, it means to feed some data into a program. An input can be given in the form of a file or from the command line. C programming provides a set of built-in functions to read the given input and feed it to the program as per requirement.

When we say **Output**, it means to display some data on screen, printer, or in any file. C programming provides a set of built-in functions to output the data on the computer screen as well as to save it in text or binary files.

The Standard Files

C programming treats all the devices as files. So devices such as the display are addressed in the same way as files and the following three files are automatically opened when a program executes to provide access to the keyboard and screen.

Standard File	File Pointer	Device
Standard input	stdin	Keyboard
Standard output	stdout	Screen
Standard error	stderr	Your screen

The file pointers are the means to access the file for reading and writing purpose. This section explains how to read values from the screen and how to print the result on the screen.

The getchar() and putchar() Functions

The **int getchar(void)** function reads the next available character from the screen and returns it as an integer. This function reads only single character at a time. You can use this method in the loop in case you want to read more than one character from the screen.

The **int putchar(int c)** function puts the passed character on the screen and returns the same character. This function puts only single character at a time. You can use this method in the loop in case you want to display more than one character on the screen. Check the following example –

```c
#include <stdio.h>
int main( ) {

   int c;

   printf( "Enter a value :");
   c = getchar( );

   printf( "\nYou entered: ");
   putchar( c );

   return 0;
}
```

When the above code is compiled and executed, it waits for you to input some text. When you enter a text and press enter, then the program proceeds and reads only a single character and displays it as follows −

```
$./a.out
Enter a value : this is test
You entered: t
```

The gets() and puts() Functions

The **char *gets(char *s)** function reads a line from **stdin** into the buffer pointed to by **s** until either a terminating newline or EOF (End of File).

The **int puts(const char *s)** function writes the string 's' and 'a' trailing newline to **stdout**.

NOTE: Though it has been deprecated to use gets() function, Instead of using gets, you want to use fgets().

```c
#include <stdio.h>
int main( ) {

   char str[100];

   printf( "Enter a value :");
   gets( str );

   printf( "\nYou entered: ");
   puts( str );

   return 0;
```

```
}
```

When the above code is compiled and executed, it waits for you to input some text. When you enter a text and press enter, then the program proceeds and reads the complete line till end, and displays it as follows −

```
$./a.out
Enter a value : this is test
You entered: this is test
```

The scanf() and printf() Functions

The **int scanf(const char *format, ...)** function reads the input from the standard input stream **stdin** and scans that input according to the **format** provided.

The **int printf(const char *format, ...)** function writes the output to the standard output stream **stdout** and produces the output according to the format provided.

The **format** can be a simple constant string, but you can specify %s, %d, %c, %f, etc., to print or read strings, integer, character or float respectively. There are many other formatting options available which can be used based on requirements. Let us now proceed with a simple example to understand the concepts better −

```
#include <stdio.h>
int main( ) {

   char str[100];
   int i;

   printf( "Enter a value :");
   scanf("%s %d", str, &i);

   printf( "\nYou entered: %s %d ", str, i);

   return 0;
}
```

When the above code is compiled and executed, it waits for you to input some text. When you enter a text and press enter, then program proceeds and reads the input and displays it as follows −

```
$./a.out
Enter a value : seven 7
You entered: seven 7
```

Here, it should be noted that scanf() expects input in the same format as you provided %s and %d, which means you have to provide valid inputs like "string integer". If you provide "string string" or "integer integer", then it will be assumed as wrong input. Secondly, while reading a string, scanf() stops reading as soon as it encounters a space, so "this is test" are three strings for scanf().