

UNIT-2

PROGRAMMING IN C

C Operators

An operator is a symbol used to perform arithmetic and logical operations in a program. That means an operator is a special symbol that tells the compiler to perform mathematical or logical operations. C programming language supports a rich set of operators that are classified as follows.

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

Arithmetic Operators (+, -, *, /, %)

The arithmetic operators are the symbols that are used to perform basic mathematical operations like addition, subtraction, multiplication, division and percentage modulo. The following table provides information about arithmetic operators.

Operator	Meaning	Example
+	Addition	$10 + 5 = 15$
-	Subtraction	$10 - 5 = 5$
*	Multiplication	$10 * 5 = 50$
/	Division	$10 / 5 = 2$
%	Remainder of the Division	$5 \% 2 = 1$

Relational Operators (<, >, <=, >=, ==, !=)

The relational operators are the symbols that are used to compare two values. That means the relational operators are used to check the relationship between two values. Every relational operator has two results TRUE or FALSE. In simple words, the relational operators are used to define conditions in a program. The following table provides information about relational operators.

Operator	Meaning	Example
<	Returns TRUE if the first value is smaller than second value otherwise returns FALSE	10 < 5 is FALSE
>	Returns TRUE if the first value is larger than second value otherwise returns FALSE	10 > 5 is TRUE
<=	Returns TRUE if the first value is smaller than or equal to second value otherwise returns FALSE	10 <= 5 is FALSE
>=	Returns TRUE if the first value is larger than or equal to second value otherwise returns FALSE	10 >= 5 is TRUE
==	Returns TRUE if both values are equal otherwise returns FALSE	10 == 5 is FALSE
!=	Returns TRUE if both values are not equal otherwise returns FALSE	10 != 5 is TRUE

Logical Operators (&&, ||, !)

The logical operators are the symbols that are used to combine multiple conditions into one condition. The following table provides information about logical operators.

Operator	Meaning	Example
&&	Logical AND - Returns TRUE if all conditions are TRUE otherwise returns FALSE	10 < 5 && 12 > 10 is FALSE
	Logical OR - Returns FALSE if all conditions are FALSE otherwise returns TRUE	10 < 5 12 > 10 is TRUE
!	Logical NOT - Returns TRUE if condition is FALSE and returns FALSE if it is TRUE	!(10 < 5 && 12 > 10) is TRUE

⇒ **Logical AND** - Returns TRUE only if all conditions are TRUE, if any of the conditions is FALSE then complete condition becomes FALSE.

⇒ **Logical OR** - Returns FALSE only if all conditions are FALSE, if any of the conditions is TRUE then complete condition becomes TRUE.

Increment & Decrement Operators (++ & --)

The increment and decrement operators are called unary operators because both need only one operand. The increment operators adds one to the existing value of the operand and the decrement operator subtracts one from the existing value of the operand. The following table provides information about increment and decrement operators.

Operator	Meaning	Example
++	Increment - Adds one to existing value	int a = 5; a++; $\Rightarrow$ a = 6
--	Decrement - Subtracts one from existing value	int a = 5; a--; $\Rightarrow$ a = 4

The increment and decrement operators are used in front of the operand (++a) or after the operand (a++). If it is used in front of the operand, we call it as **pre-increment** or **pre-decrement** and if it is used after the operand, we call it as **post-increment** or **post-decrement**.

Assignment Operators (=, +=, -=, *=, /=, %=)

The assignment operators are used to assign right-hand side value (Rvalue) to the left-hand side variable (Lvalue). The assignment operator is used in different variants along with arithmetic operators. The next table describes all the assignment operators in the C programming language.

Operator	Meaning	Example
=	Assign the right-hand side value to left-hand side variable	$A = 15$
+=	Add both left and right-hand side values and store the result into left-hand side variable	$A += 10$ $\Rightarrow A = A + 10$
-=	Subtract right-hand side value from left-hand side variable value and store the result into left-hand side variable	$A -= B$ $\Rightarrow A = A - B$
*=	Multiply right-hand side value with left-hand side variable value and store the result into left-hand side variable	$A *= B$ $\Rightarrow A = A * B$
/=	Divide left-hand side variable value with right-hand side variable value and store the result into the left-hand side variable	$A /= B$ $\Rightarrow A = A / B$
%=	Divide left-hand side variable value with right-hand side variable value and store the remainder into the left-hand side variable	$A \% = B$ $\Rightarrow A = A \% B$

Bitwise Operators (&, |, ^, ~, >>, <<)

The bitwise operators are used to perform bit-level operations in the C programming language. When we use the bitwise operators, the operations are performed based on the binary values. The following table describes all the bitwise operators in the C programming language. Let us consider two variables A and B as A = 25 (11001) and B = 20 (10100).

Operator	Meaning	Example
&	the result of Bitwise AND is 1 if all the bits are 1 otherwise it is 0	A & B ⇒ 16 (10000)
	the result of Bitwise OR is 0 if all the bits are 0 otherwise it is 1	A B ⇒ 29 (11101)
^	the result of Bitwise XOR is 0 if all the bits are same otherwise it is 1	A ^ B ⇒ 13 (01101)
~	the result of Bitwise one complement is negation of the bit (Flipping)	~A ⇒ 6 (00110)
<<	the Bitwise left shift operator shifts all the bits to the left by the specified number of positions	A << 2 ⇒ 100 (1100100)
>>	the Bitwise right shift operator shifts all the bits to the right by the specified number of positions	A >> 2 ⇒ 6 (00110)

Conditional Operator (?:)

The conditional operator is also called a **ternary operator** because it requires three operands. This operator is used for decision making. In this operator, first we verify a condition, then we perform one operation out of the two operations based on the condition result. If the condition is TRUE the first option is performed, if the condition is FALSE the second option is performed. The conditional operator is used with the following syntax.

Condition ? TRUE Part : FALSE Part;

Example

A = (10<15)?100:200; ⇒ A value is 100

Special Operators (sizeof, pointer, comma, dot, etc.)

The following are the special operators in c programming language.

sizeof operator

This operator is used to find the size of the memory (in bytes) allocated for a variable. This operator is used with the following syntax.

sizeof(variableName);

Example

sizeof(A); ⇒ the result is 2 if A is an integer

Pointer operator (*)

This operator is used to define pointer variables in c programming language.

Comma operator (,)

This operator is used to separate variables while they are declaring, separate the expressions in function calls, etc.

Dot operator (.)

This operator is used to access members of structure or union.

Operator Precedence and Associativity in C

The concept of **operator precedence and associativity in C** helps in determining which operators will be given priority when there are multiple operators in the expression. It is very common to have multiple operators in C language and the compiler first evaluates the operator with higher precedence. It helps to maintain the ambiguity of the expression and helps us in avoiding unnecessary use of parenthesis.

Precedence: Precedence refers to the priority or order in which operators are evaluated in an expression. Operators with higher precedence are evaluated before operators with lower precedence. When an expression contains multiple operators, their precedence dictates the order in which the operations are performed. For example, in the expression $a + b * c$, the $*$ operator has higher precedence than $+$, so $b * c$ is evaluated first, and then the result is added to a .

Here are some common C operators and their precedence levels (from highest to lowest):

- $()$ (parentheses) - Highest precedence
- $*$ and $/$ (multiplication and division)
- $+$ and $-$ (addition and subtraction)
- $=$ (assignment) - Lowest precedence

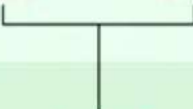
You can also use parentheses to explicitly specify the order of evaluation. For example, $(a + b) * c$ ensures that $a + b$ is evaluated before the result is multiplied by c .

Associativity: Associativity defines the order in which operators of the same precedence are evaluated. In C, operators can be left-associative or right-associative.

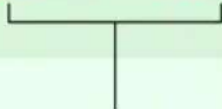
- Left-Associative:** For left-associative operators, the evaluation proceeds from left to right. For example, in the expression $a - b - c$, the $-$ operator is left-associative, so $a - b$ is evaluated first, and then the result is subtracted from c .

- Right-Associative:** Right-associative operators are evaluated from right to left. However, there are very few right-associative operators in C, and the most common one is the assignment operator $=$. For example, in the expression $a = b = c$, the assignment is right-associative, so c is assigned to b first, and then the result of that assignment is assigned to a .

Operator Precedence

$$10 + 20 * 30$$


Multiply will happen first as * has *higher* precedence

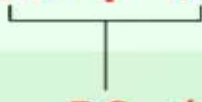
$$10 + 600$$


Now addition will happen as + has *lower* precedence

$$610$$

We have got the *solution* for the equation

Operator Associativity

$$100 / 10 * 10$$


"/" is evaluated **first** as it is leftmost

$$10 * 10$$


Now "*" is evaluated.

$$100$$

We have got the solution for the equation

/ & * both have the same precedence but *left to right (LTR) associativity*

Operator Precedence and Associativity

$$100 + 200 / 10 - 3 * 10$$

Divide will happen first as / has **Higher Precedence** than + & -. It has the same precedence as * but **Higher Associativity**.

$$100 + 20 - 3 * 10$$

Multiply will happen second as * has **Higher Precedence** than + & -. It has the same precedence as / but **Lower Associativity**.

$$100 + 20 - 30$$

Addition will happen third as + has **Lower Precedence** than / & -. It has the same precedence as - but **Higher Associativity**.

$$120 - 30$$

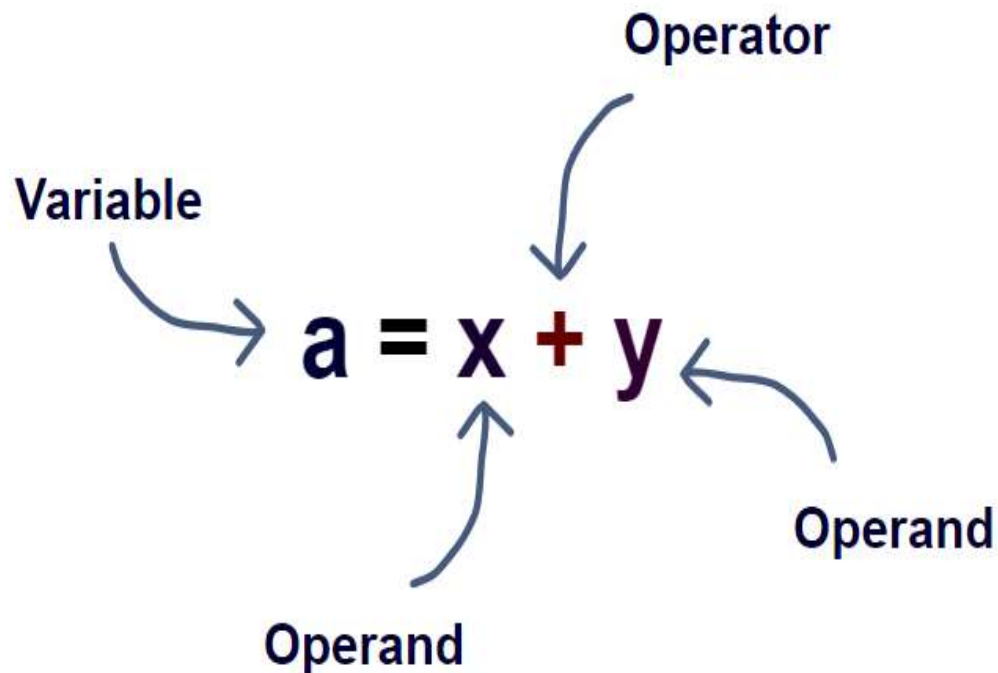
Subtract will happen last as - has **Lower Precedence** than / & *. It has the same precedence as + but **Lower Associativity**.

$$90$$

We have got the **solution** for the equation

Expression in C

An **expression** in C is a combination of *operands* and *operators* – it computes a single value stored in a *variable*. The **operator** denotes the action or operation to be performed. The **operands** are the items to which we apply the operation.



Statement and its types in C

In C programming, a statement is a fundamental unit of code that performs a specific action. Statements are executed sequentially in a C program, and they can vary in complexity. There are several types of statements in C:

1.Expression Statement:

- An expression statement is the simplest type of statement.
- It consists of an expression followed by a semicolon (;).
- When executed, it evaluates the expression and executes the result.

Example:

```
c
```

```
x = 5; // Assignment statement  
y = x + 3; // Expression statement
```

Compound Statement (Block):

- A compound statement, also known as a block, is a group of statements enclosed within curly braces {}.
- It allows multiple statements to be treated as a single unit.
- Typically used for defining functions, loops, and conditional statements.

```
c
{
    int a = 10;
    int b = 20;
    int sum = a + b;
}
```

Selection Statements:

- Selection statements allow you to make decisions in your program based on conditions.
- The common selection statements in C are:
 - **if statement:** Executes a block of code if a condition is true.
 - **else statement:** Provides an alternative block of code to execute when the if condition is false.
 - **switch statement:** Allows you to select one of several code blocks to execute based on the value of an expression.

```
if (x > 0) {  
    // Code to execute if x is greater than 0  
} else {  
    // Code to execute if x is not greater than 0  
}
```

Iteration Statements (Loops):

- Iteration statements, or loops, allow you to repeatedly execute a block of code.
- The common iteration statements in C are:
 - **for loop:** Executes a block of code a specified number of times.
 - **while loop:** Repeatedly executes a block of code as long as a condition is true.
 - **do-while loop:** Similar to the while loop but ensures that the block of code is executed at least once.

```
for (int i = 0; i < 5; i++) {  
    // Code to execute repeatedly  
}
```

Jump Statements:

- Jump statements allow you to control the flow of execution in your program.
- The common jump statements in C are:
 - **break:** Used to exit a loop or switch statement prematurely.
 - **continue:** Used to skip the current iteration of a loop and continue with the next iteration.
 - **return:** Used to exit a function and optionally return a value.
 - **goto (less commonly used):** Allows you to transfer control to a labeled statement within the same function.

```
for (int i = 0; i < 10; i++) {  
    if (i == 5) {  
        break; // Exit the loop when i is 5  
    }  
    if (i % 2 == 0) {  
        continue; // Skip even numbers  
    }  
    // Code to execute for odd numbers  
}
```

C Input Functions

C programming language provides built-in functions to perform input operations. The input operations are used to read user values (input) from the keyboard. The c programming language provides the following built-in input functions.

1. **scanf()**
2. **getchar()**
3. **getch()**
4. **gets()**
5. **fscanf()**

scanf() function

The scanf() function is used to read multiple data values of different data types from the keyboard. The scanf() function is built-in function defined in a header file called "stdio.h". When we want to use scanf() function in our program, we need to include the respective header file (stdio.h) using #include statement. The scanf() function has the following syntax...

Syntax:

```
scanf("format strings",&variableNames);
```

Example:

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

void main(){
    int i;
    printf("\nEnter any integer value: ");
    scanf("%d",&i);
    printf("\nYou have entered %d number",i);
}
```

getchar() function

The getchar() function is used to read a character from the keyboard and return it to the program. This function is used to read a single character. To read multiple characters we need to write multiple times or use a looping statement.

```
#include<stdio.h>
#include<conio.h>
void main(){
    char ch;
    printf("\nEnter any character : ");
    ch = getchar();
    printf("\nYou have entered : %c\n",ch);
}
```

getch() function

The getch() function is similar to getchar function. The getch() function is used to read a character from the keyboard and return it to the program. This function is used to read a single character.

```
void main(){
    char ch;
    printf("\nEnter any character : ");
    ch = getch();
    printf("\nYou have entered : %c",ch);
}
```

gets() function

The gets() function is used to read a line of string and stores it into a character array. The gets() function reads a line of string or sequence of characters till a newline symbol enters.

```
void main(){
    char name[30];
    printf("\nEnter your favourite website: ");
    gets(name);
    printf("%s",name);
}
```

fscanf() function

The fscanf() function is used with the concept of files. The fscanf() function is used to read data values from a file. When you want to use fscanf() function the file must be opened in reading mode.

C Output Functions

C programming language provides built-in functions to perform output operation. The output operations are used to display data on user screen (output screen) or printer or any file. The c programming language provides the following built-in output functions...

1.printf()

2.putchar()

3.puts()

4.fprintf()

printf() function

The printf() function is used to print string or data values or a combination of string and data values on the output screen (User screen). The printf() function is built-in function defined in a header file called "stdio.h". When we want to use printf() function in our program we need to include the respective header file (stdio.h) using the #include statement. The printf() function has the following syntax...

Syntax:

```
printf("message to be display!!!");
```

Example:

```
void main(){  
  
    printf("Hello! Welcome to Our Department!!!");  
  
}
```

putchar() function

The putchar() function is used to display a single character on the output screen. The putchar() functions prints the character which is passed as a parameter to it and returns the same character as a return value. This function is used to print only a single character. To print multiple characters we need to write multiple times or use a looping statement.

```
void main(){  
  
    char ch = 'A';  
    putchar(ch);  
  
}
```

puts() function

The puts() function is used to display a string on the output screen. The puts() functions prints a string or sequence of characters till the newline. Consider the next example program...

```
void main(){  
    char name[30];  
    printf("\nEnter your favourite website: ");  
    gets(name);  
    puts(name);  
}
```

fprintf() function

The fprintf() function is used with the concept of files. The fprintf() function is used to print a line into the file. When you want to use fprintf() function the file must be opened in writing mode.

Header Files in C

In **C language**, header files contain a set of predefined standard library functions. The .h is the extension of the header files in C and we request to use a header file in our program by including it with the C preprocessing directive “**#include**”. **C language** has numerous libraries that include predefined functions to make programming easier.

Syntax of Header Files in C/C++

We can include header files in C by using one of the given two syntaxes whether it is a pre-defined or user-defined header file.

```
#include <filename.h>                // for files in system/default directory
```

or

```
#include "filename.h"                // for files in same directory as source file
```

The “#include” preprocessor directs the compiler that the header file needs to be processed before compilation and includes all the necessary data types and function definitions.

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

int main() {
    -----
    -----
}
```

```
// C program to demonstrate the use of header files
// standard input and output stdio.h header file
```

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

```
int main()
{
    printf(
        "Printf() is the function in stdio.h header file");
    return 0;
}
```

Output

Printf() is the function in stdio.h header file

Types of Header Files

There are two types of header files in C and C++:

Standard / Pre-existing header files

Non-Standard / User-defined header files

1. Standard Header Files in C and their Uses

Standard header files contain the libraries defined in the ISO standard of the C programming language. They are stored in the default directory of the compiler and are present in all the C compilers from any vendor. For example: <float.h>, <Stdio.h>, <math.h>, <string.h> etc.

2. Non-Standard Header Files in C and Their Uses

Non-standard header files are not part of the language's ISO standard. They are generally all the header files defined by the programmers for purposes like containing custom library functions etc. They are manually installed by the user or maybe part of the compiler by some specific vendor. For example: <conio.h>

Preprocessor directives in C are special instructions that are processed by the C preprocessor before the actual compilation of your code. They serve to manipulate the source code or customize the compilation process. Two commonly used preprocessor directives are `#include` and `#define`:

1. `#include`:

- The `#include` directive is used to include the contents of another file (typically header files) into your C source code before compilation.
- It allows you to reuse code or declarations from external files, such as standard library headers or user-defined header files.
- There are two common syntaxes for `#include`:
 - To include system header files, you use angle brackets:

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

- To include user-defined header files, you use double quotes:

```
#include "myheader.h"
```

Example:

```
#include <stdio.h>
int main() {
    printf("Hello, world!\n");
    return 0;
}
```

2. #define:

- The #define directive is used to create macros or symbolic constants in your C code.
- Macros are preprocessor substitutions. When the preprocessor encounters a macro in the code, it replaces the macro with its defined value before compilation.
- Macros are defined using the following syntax:

```
#define MACRO_NAME value
```

Example:

```
#define PI 3.14159265
```

- Macros can also be used for function-like operations using arguments:

```
#define SQUARE(x) ((x) * (x))
```

Example:

```
int main() {  
    int result = SQUARE(5); // This gets replaced as: ((5) * (5))  
    return 0;  
}
```

- Macros are often written in uppercase letters to distinguish them from regular C code.

These preprocessor directives are essential for code organization, reusability, and customization in C programming. They allow you to manage dependencies by including external files and create reusable constants and macros to improve code readability and maintainability.

UNIT-2

COMPLETE