



## Module 1: INTRODUCTION TO C LANGUAGE

*Pseudo code solution to problem, Basic concepts in a C program, Declaration, Assignment & Print statements, Data Types, operators and expressions etc, Programming examples and exercise.*

**PSEUDOCODE:** Is one of the tools that can be used to write a preliminary plan that can be developed into a computer program. Pseudocode is a generic way of describing an algorithm without use of any specific programming language syntax.

### Key points:

- No syntax rule – Independent from any programming language
- Write in an ordinary language
- Uses a structure resembling computer structure
- No connector between pages

*Example: Write a pseudocode to add two numbers.*

```
Begin                -Start
  Read A, B           - Input
  Calculate C = A*B   - Action
  Display C           - Output
  Stop                - Terminal
```

*Example: Write a pseudocode to calculate area and perimeter of rectangle*

```
Begin
  Input length, breadth
  Calculate area = length * breadth
  Calculate perimeter = 2*(length+breadth)
  Print area, perimeter
End
```

*Example: Write a pseudocode to find sum and average of given two numbers.*

```
Begin
  WRITE "Please enter two numbers to add"
  READ num1
  READ num2
  Sum = num1+num2
  Avg = Sum/2
  WRITE Sum, Avg
End
```

*Example: Write pseudocode that will take a number as input and tells whether a number is positive, negative or zero.*

```
Begin
  WRITE "Enter a number"
  READ num
  IF num > 0
    THEN
      WRITE "The number is positive"
  ELSE IF num = 0
    THEN
      WRITE "The number is zero"
  ELSE
      WRITE "The number is negative"
  ENDIF
ENDIF
End
```



Example: Write pseudocode that will calculate a sum n natural numbers.

```
Begin
  Read n
  Set sum to 0
  Set i ← 1
  While(i<=n)
    Calculate sum ← sum+i
  Print sum
End
```

### Advantages and Disadvantages of Pseudocode:

Pseudocode Disadvantages

- It's not visual
- There is no accepted standard, so it varies widely from company to company

Pseudocode Advantages

- Can be done easily on a word processor
- Easily modified
- Implements structured concepts well

**ALGORITHM:** Step by step procedure to solve a given problem, and which (like a map or flowchart) will lead to the correct result if followed correctly. Algorithms have a definite beginning and a definite end, and a finite number of steps.

### Characteristics of an Algorithm:

- *Well-ordered:* the steps are in a clear order.
- *Unambiguous:* the operations described are understood by a computing agent without further simplification.
- *Effectively computable:* the computing agent can actually carry out the operation.
- *Finiteness:* algorithms must have a definite beginning and a definite end.

Example: Write an algorithm to find the area of a Circle

```
Step1: Start
Step2: Read Radius r of the Circle // input to problem
Step3: Area PI*r*r // calculation of area
Step4: Print Area //output of the problem
Step5: Stop
```

Example: Write an algorithm to read two numbers and find their sum

```
Step1: Start
Step2: Input the first number as num1.
Step3: Input the second number as num2.
Step4: Sum ← num1+num2 // calculation of sum
Step5: Output Sum
Step6: End
```

Example: Write an algorithm to Convert Temperature from Fahrenheit (°F) to Celsius (°C)

```
Step1: Start
Step 2: Read Temperature in Fahrenheit as F
Step 3: C= 5/9*(F-32)
Step 4: Print Temperature in Celsius: C
Step5: End
```

Example: write algorithm to find the greater number between two numbers

```
Step1: Start
Step2: Read A and B
Step3: If A greater than B then C=A Goto step5
```



Step4: If B greater than A then C=B  
 Step5: Print C  
 Step6: End

Example: write an algorithm to find the largest value of any three numbers.

Step1: Start  
 Step2: Read A,B and C  
 Step3: If (A>=B) and (A>=C) then Max=A Goto Step6  
 Step4: If (B>=A) and (B>=C) then Max=B Goto Step6  
 Step5: If (C>=A) and (C>=B) then Max=C  
 Step6: Print Max  
 Step7: End

Example: Write an algorithm to find the factorial of a number entered by user.




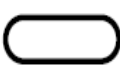


Step 1: Start  
 Step 2: Read value of n  
 Step 3: Initialize variables factorial←1, i←1  
 Step 4: factorial ← factorial\*i  
 Step 5: i←i+1  
 Step 6: Repeat the *steps4* to 5 until i<=n  
 Step 7: Display factorial  
 Step 8: Stop

Example: Write an algorithm to find the Fibonacci series till term≤1000.

Step 1: Start  
 Step 2: Initialize variables first\_term←0 ,second\_term←1  
 Step 3: Display first\_term and second\_term  
 Step 4: Repeat the *steps5* through 8 until second\_term≤1000  
 Step 5: temp←second\_term  
 Step 6: second\_term←second\_term+first term  
 Step 7: first\_term←temp  
 Step 8: Display second\_term  
 Step 6: Stop

**FLOWCHARTS:** A graphical or pictorial representation of an algorithm. Flowcharts use simple geometric symbols and arrows to define relationships. In programming, for instance, the beginning or end of a program is represented by an oval. A process is represented by a rectangle, a decision is represented by a diamond and an I/O process is represented by a parallelogram.

**The most commonly used symbols:**

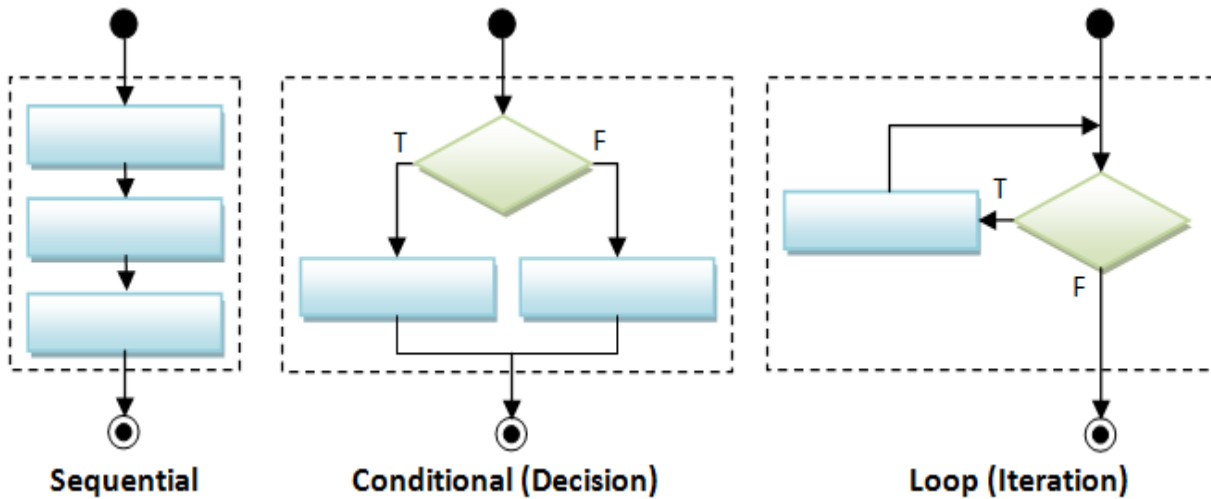
Symbol	Name/Meaning	Symbol	Meaning
	<u>Process</u> – Any type of internal operation: data transformation, data movement, logic operation, etc.		<u>Connector</u> – connects sections of the flowchart, so that the diagram can maintain a smooth, linear flow
	<u>Input/Output</u> – input or output of data		<u>Terminal</u> – indicates start or end of the program or algorithm
	<u>Decision</u> – evaluates a condition or statement and branches depending on whether the evaluation is true or false		<u>Flow lines</u> – <i>arrows</i> that indicate the direction of the progression of the program



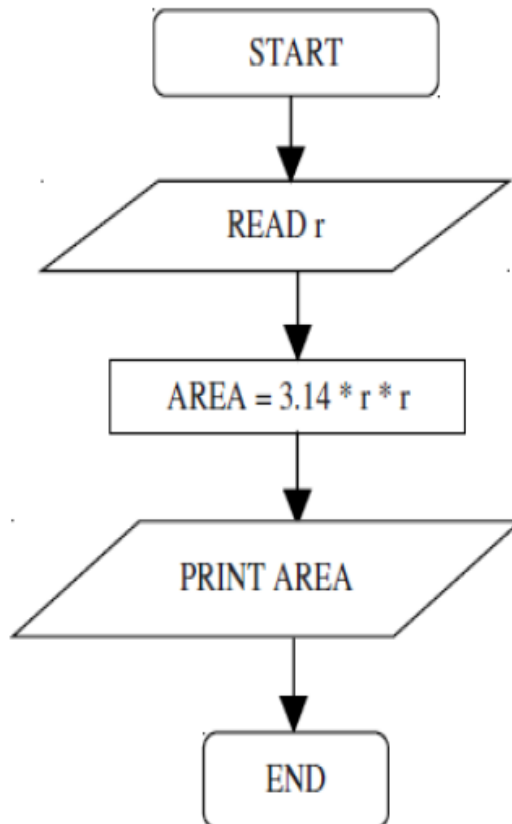
**General rules for flowcharts:**

1. All symbols of the flowchart are connected by flow lines (note *arrows*, not lines)
2. Flowlines enter the top of the symbol and exit out the bottom, except for the Decision symbol, which can have flow lines exiting from the bottom or the sides
3. Flowcharts are drawn so flow generally goes from top to bottom
4. The beginning and the end of the flowchart is indicated using the Terminal symbol

**Flowchart Constructs:**

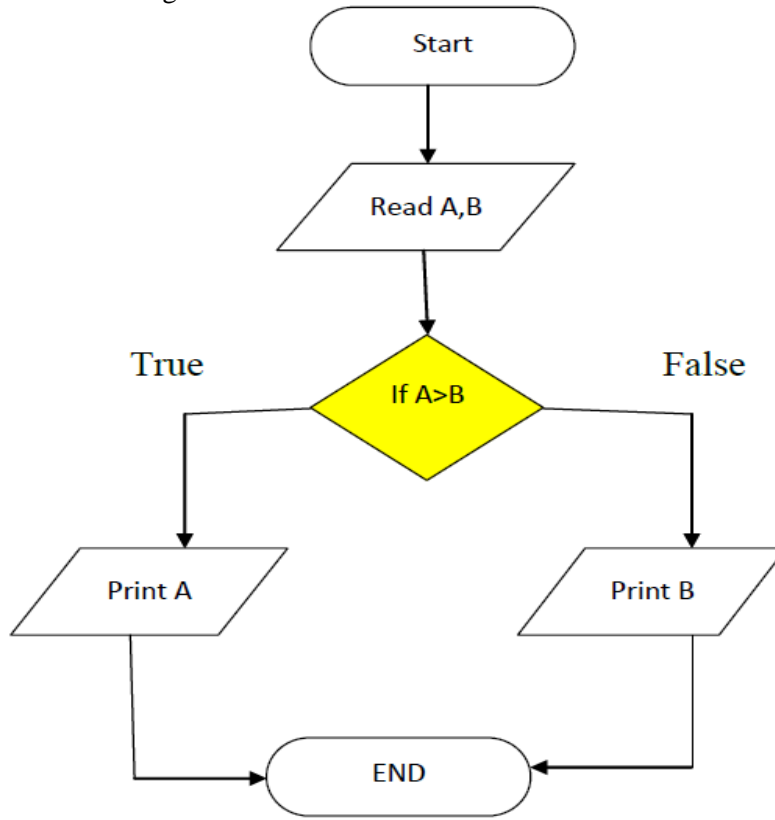


Example: Draw a flowchart for finding the area of a circle.

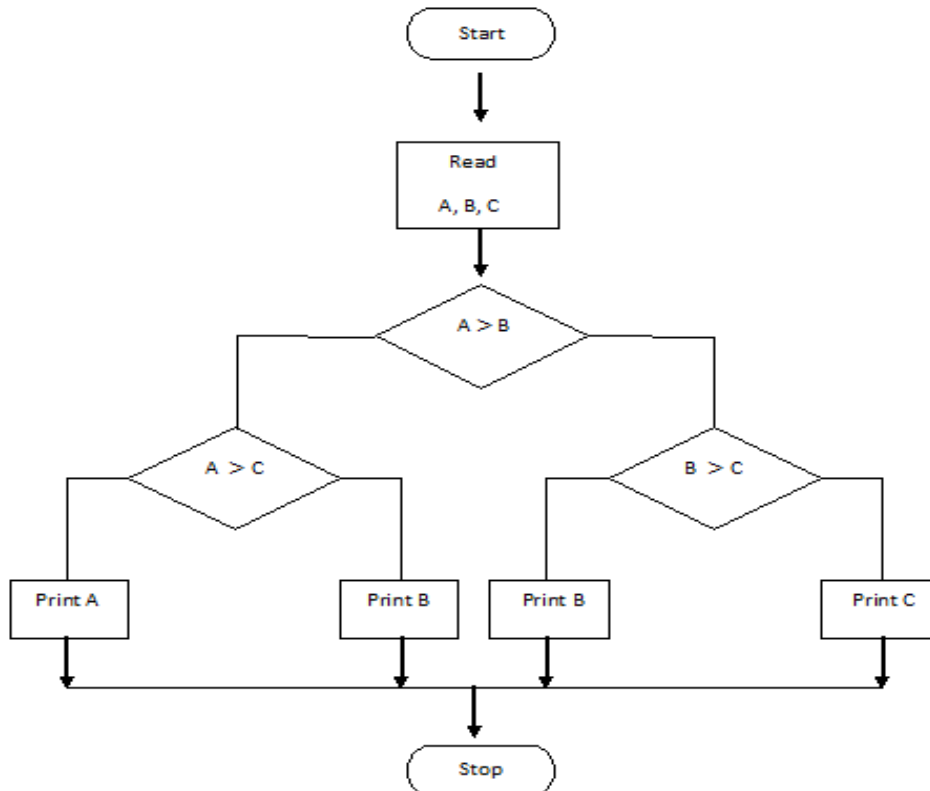




Example: Flowchart for find the greater number between two numbers.

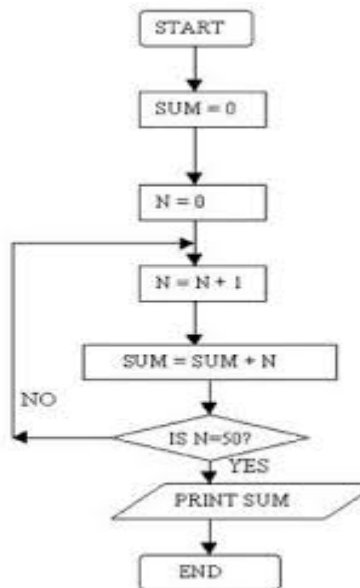


Example: Flowchart for biggest among 3 numbers.





Example: Flowchart for sum of first 50 natural numbers.



## **BASIC CONCEPTS IN A C PROGRAM:**

### **History of C:**

- 1.The C programming language is a structure oriented programming language, developed at Bell Laboratories in 1972 by Dennis Ritchie
- 2.C programming language features were derived from an earlier language called “B” (Basic Combined Programming Language – BCPL)
- 3.C language was invented for implementing UNIX operating system
- 4.In 1978, Dennis Ritchie and Brian Kernighan published the first edition “The C Programming Language” and commonly known as K&R C
- 5.In 1983, the American National Standards Institute (ANSI) established a committee to provide a modern, comprehensive definition of C. The resulting definition, the ANSI standard, or “ANSI C”, was completed late 1988.

### **Features of c programming language:**

C language is one of the powerful languages. Below are some of the features of C language.

- Reliability
- Portability
- Flexibility
- Interactivity
- Modularity
- Efficiency and Effectiveness

### **Which level is c language belonging to?**

There are 3 levels of programming languages. They are,

#### **1.Middle Level languages:**

Middle level languages don't provide all the built-in functions found in high level languages, but provide all building blocks that we need to produce the result we want. Examples: C, C++

#### **2.High Level languages:**

High level languages provide almost everything that the programmer might need to do as already built into the language. Example: Java, Python



### 3.Low Level languages:

Low level languages provide nothing other than access to the machines basic instruction set.

Example: Assembler

### Structure of C program and Example:

#### BASIC STRUCTURE OF A 'C' PROGRAM:

Documentation section [Used for Comments]
Link section
Definition section
Global declaration section [Variable used in more than one function]
main() { Declaration part Executable part }
Subprogram section [User-defined Function] Function1 Function 2 : : Function n

#### Example:

```

//Sample Prog Created by: Bsource
#include<stdio.h>
#include<conio.h>

void fun();

int a=10;

void main()
{
clrscr();
printf("a value inside main(): %d",a);
fun();
}

void fun()
{
printf("na value inside fun(): %d",a);
}

```

**Documentation section:** We can give comments about the program, creation or modified date, author name etc in this section. The characters or words or anything which are given between “/\*” and “\*/”, won’t be considered by C compiler for compilation process. These will be ignored by C compiler during compilation. Example: /\* comment line1 comment line2 comment 3 \*/

**Link Section:** Header files that are required to execute a C program are included in this section.

**Definition Section:** In this section, variables are defined and values are set to these variables.

**Global declaration section:** Global variables are defined in this section. When a variable is to be used throughout the program, can be defined in this section.

**Function prototype declaration section:** Function prototype gives many information about a function like return type, parameter names used inside the function.

**Main function:** Every C program is started from main function and this function contains two major sections called declaration section and executable section.

**User defined function section:** User can define their own functions in this section which perform particular task as per the user requirement.

#### Key points to remember in c programming basics:

1. C programming is a case sensitive programming language.
2. Each C programming statement is ended with semicolon (;) which are referred as statement terminator.
3. printf() command is used to print the output onto the screen.
4. C programs are compiled using C compilers and displays output when executed.



**C LANGUAGE TOKENS:** The smallest individual units in a C program are known as tokens. In a C source program, the basic element recognized by the compiler is the "token." A token is source-program text that the compiler does not break down into component elements.

**TOKEN TYPES IN 'C'**

**C has 6 different types of tokens viz.**

1. **Keywords** [e.g. float, int, while]
2. **Identifiers** [e.g. main, amount]
3. **Constants** [e.g. -25.6, 100]
4. **Strings** [e.g. "SMIT", "year"]
5. **Special Symbols** [e.g. {, }, [, ]]
6. **Operators** [e.g. +, -, \*]

**C programs are written using these tokens.**

**1. THE KEYWORDS:** "Keywords" are words that have special meaning to the C compiler. Their meaning cannot be changed at any instance. Serve as basic building blocks for program statements. All keywords are written in only lowercase. C language supports 32 keywords which are given below.

**Examples of C Keywords:** *auto, double, int, struct, const, float, short, unsigned, break, else, long, switch, continue, for, signed, void, case, enum, register, typedef, default, goto, sizeof, volatile, char, extern, return, union, do, if, static, while*

**2. THE IDENTIFIERS:** Identifiers are names for entities in a C program, such as variables, arrays, functions, structures, unions and labels.

**Rules for constructing identifier name in c:**

1. First character should be an alphabet or underscore ( \_ ).
2. Succeeding characters might be digits or letter.
3. Punctuation and special characters aren't allowed except underscore ( \_ ).
4. Identifiers should not be keywords.

**Examples for Valid and Invalid identifiers:**

- a. **record1** => Valid
- b. **1record** => Invalid Identifier name must start with alphabets
- c. **file\_3** => Valid
- d. **return** => Invalid Keywords cannot be used as identifiers
- e. **#tax** => Invalid Identifier should not contain any special symbols except underscore ( \_ ) symbol.
- f. **name** => Valid
- g. **goto** => Invalid It is a keyword and cannot be used as identifier.
- h. **name and address** => Invalid No white spaces are permitted in name of identifier.
- i. **name-and-address** => Invalid Identifier should not contain any special symbols except underscore ( \_ ) symbol.
- j. **123-45-6789** => Invalid Identifier should not contain any special symbols except underscore ( \_ ) symbol. And also identifier name must start with alphabet.
- k. **void** => Invalid It is a keyword and cannot be used as identifier.
- l. **name\_address** => Valid
- m. **NAME** => Valid

**Differentiate between Keywords words and identifiers:**

Identifier	Keyword
Predefined-word	User-defined word
Must be written in lowercase only	Can written in lowercase and uppercase
Has fixed meaning	Must be meaningful in the program
Whose meaning has already been explained to the C compiler	Whose meaning not explained to the C compiler
Used only for it intended purpose	Used for required purpose





**3. CONSTANTS:** Constants in C are the fixed values that do not change during the execution of a program.

**Constants are classified in to:**

1. Numeric constants
2. Character constants:
3. String constants:
4. Backslash character constants:

**1. Numeric constants:** Numeric constants can be classified as,

- i. Integer constants and
- ii. Real constants.

**i). Integer constants:** An integer constant refers to a sequence of digits. These are further classified into three types depending on the number systems they belong to they are

- a) Decimal integer constants
- b) Octal integer constants
- c) Hexadecimal constants

**Decimal integer constant:** - A decimal integer constant is characterized by the following properties.

- It is a sequence of one (or) more digit ( $\{0\dots9\}$ , the symbols of decimal numbers system).
- It may have an optional + or - sign. In the absence of sign, the constant is assumed to be positive.
- It should not have a period ( . ) as a part of it.
- Commas and blank spaces are not permitted.

Valid decimal integer constant:- 345 , -987

Invalid decimal integer constant:- 3.45 decimal point is not permitted,  
3, 34 commas are not permitted.

**Octal integer constant:-**An octal integer characterized by the following properties.

- It is a sequence of one or more digit ( $\{0\dots7\}$ , symbols octal number system).
- It may have an optional + or - sign. In the absence of sign, the constant is assumed to be positive.
- It should start with digit 0.
- It should not have a period ( . ) as a part of it.
- Commas and blanks are not permitted.

Valid: - 0345

Invalid:- 03.34 decimal point is not permissible

**Hexadecimal integer constant:** -A hexadecimal integer constant is characterized by the following properties.

- It is a sequence of once or more symbols ( $\{0\dots9\}[a\dots z]$ , the symbols of hexadecimal number system).
- It may have an optional + or - sign. In the absence of sign, the constant is assumed to be positive.
- It should start with symbols 0X .

Valid:-0X345

Invalid:-0X3.45 decimal is not permissible

**ii) Real constants:**

- A real constant must have at least one digit
- It must have a decimal point
- It could be either positive or negative
- If no sign precedes a real constant, it is assumed to be positive.
- No commas or blanks are allowed within a real constant.

Valid real constants: 0.0083, -0.78, +67.89 etc.

**2. Character constants:** Single character constant is a constant enclosed within a pair of a single quote marks.

**Example:** 'a', '5', etc

Character constants have integer values that are determined by the computers particular character set .



Most computers make use of the ASCII character set, in which each individual character is numerically encoded.

**Example:** 'A'=65 Maximum of one character is taken within a character constant

**3. String constants:** A string constant contains sequence of zero or more characters (i.e., alphabets, digits, special symbols and blank spaces) enclosed in double quotes.

**Example:** "" ( empty string), "Hello!", "1987", "?...!"

**4. Backslash character constants:**

- There are some characters which have special meaning in C language.
- They should be preceded by backslash symbol to make use of special function of them.
- Given below is the list of special characters and their purpose.

<b>Backslash_character</b>	<b>Meaning</b>
<code>\b</code>	Backspace
<code>\f</code>	Form feed
<code>\n</code>	New line
<code>\r</code>	Carriage return
<code>\t</code>	Horizontal tab
<code>\"</code>	Double quote
<code>\'</code>	Single quote
<code>\\</code>	Backslash
<code>\v</code>	Vertical tab
<code>\a</code>	Alert or bell
<code>\?</code>	Question mark

**How to use constants in a c program?**

We can define constants in a C program in the following ways.

1. By "const" keyword
2. By "#define" preprocessor directive

**VARIABLES IN C:** C variable is a named location in a memory where a program can manipulate the data. This location is used to hold the value of the variable.

**Rules for writing the variable names are given below.**

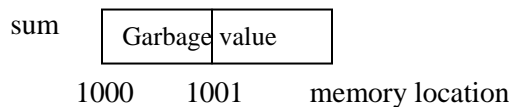
1. Variable names consist of letters, digits and underscore.
2. Variable names must begin with an alphabet or underscore.
3. Variable names could have length up to 31 characters.
4. Variable names are case sensitive i.e., "SUM" and "sum" are not equivalent.
5. Keywords should not be used as, a variable names.
6. Blank spaces, commas and special symbols are not allowed within variable names.

**Variable declaration:** A variable stores data value of different types and to avoid the confusion to compiler, variables are declared with a data type once we declare any variable compiler reserve memory for that variable and stores some garbage (unwanted) value into it. It can store only the data values of the type associated with it.

**Syntax:** data type variable\_name;

Examples: int sum;

float avg, val; or  $\left. \begin{matrix} \text{float avg;} \\ \text{float val;} \end{matrix} \right\}$   
 char ch;





**Variable initialization:** Storing some data into variables or named memory locations is called variable initialization.

**Syntax:** variable\_name = value;

Where value may be any constant, result of any expression or another variable.

Example1: `int avg;` what compiler will do after execution of this line?  
 It will reserve 2 bytes of memory for `avg` and some garbage value will be stored in it.

<code>avg</code>	.....Garbage value.....
	4000                      4001

`avg=456;` //variable initialization  
 After this line value will be stored in already reserved memory by `avg`. (Binary equivalent of 456)

<code>avg</code>	11001000	00000001
	4000	4001

Example2: `float sum=100.34;` //declaration and initialization are at same time.  
`char ch='A'`

Example3: `int x;`  
`x=20+12;` //result of expression is initialized to `x`

Example4: `int a=100,b;`  
`b=a;` //value of `a` is initialized to `b`

Example5: `int a,b,c;`  
`a=b=c=30` //30 is initialized or assigned to `a`, `b` and `c` (multiple variables are initialized in same line)

**DATA TYPES:** C data types are defined as the data storage format that a variable can store a data to perform a specific operation.

- Data types are used to define a variable before to use in a program.
- Size of variable, constant and array are determined by data types.

**There are four data types in C language. They are,**

Types	Data Types
Basic data types	int, char, float, double
Enumeration data type	enum
Derived data type	pointer, array, structure, union
Void data type	void

## **BASIC DATA TYPES IN C LANGUAGE:**

### **1. INTEGER DATA TYPE (int):**

- Integer data type allows a variable to store numeric values.
- “**int**” keyword is used to refer integer data type.
- The storage size of `int` data type is 2 or 4 or 8 byte.
- It varies depend upon the processor in the CPU that we use. If we are using 16 bit processor, 2 byte (16 bit) of memory will be allocated for `int` data type.
- Like wise, 4 byte (32 bit) of memory for 32 bit processor and 8 byte (64 bit) of memory for 64 bit processor is allocated for `int` datatype.
- `int` (2 byte) can store values from -32,768 to +32,767
- If you want to use the integer value that crosses the above limit, you can go for “**long int**” for which the limits are very high.
- Format Specifier for `int` is `%d`



**Note:**

- We can't store decimal values using **int** data type.
- If we use **int** data type to store decimal values, decimal values will be truncated and we will get only whole number. Example: 0, -5, 10

Example: `int id;` // Here, *id* is a variable of type integer. In *id* you can store any integer value within -32,768 to +32,767 if it is 2 byte.

**2. CHARACTER DATA TYPE:**

- Character data type allows a variable to store only one character.
- Storage size of character data type is **1**byte. We can store only one character using character data type.
- “**char**” keyword is used to refer character data type.
- For example, ‘A’ can be stored using char datatype. You can't store more than one character using **char** data type.
- **char** (1 byte) can store values from -128 to 127
- **unsigned char**(1byte) can store values from 0 to 255
- Format Specifier for **char** is **%c**

Example: `char test = 'h';` // Here, *test* is a character variable. The value of *test* is 'h'.

**3. FLOATING POINT DATA TYPE:**

Floating point data type consists of 2 types. They are,

- float**
- double**

**i). FLOAT:**

- Float data type allows a variable to store decimal values.
- Storage size of float data type is **4** bytes. This also varies depend upon the processor in the CPU as “int” data type.
- We can use up-to 6 digits after decimal using **float** data type.
- For example, 10.456789 can be stored in a variable using float data type.
- Range of **float** is -3.4E+38 to +3.4E+38
- Format Specifier for **float** is **%f**

Example: `float x=1.000000` // Here, *x* is a float variable. The value of *x* is 1.000000.

**ii). DOUBLE:**

- Storage size of **double** data type is 8 bytes.
- Double data type is also same as float data type which allows up-to 16 digits after decimal.
- The range for **double** datatype is from -1.7E+308 to +1.7E+308.
- Format Specifier for **double** is **%lf**

Example: `double y;`

**C QUALIFIERS:** Qualifiers alters the meaning of base data types( int, float, double, char) to yield a new data type.

**Size qualifiers:**

- Size qualifiers alter the size of a basic type. There are two size qualifiers, *long* and *short*. For Example: **long double i;**
- The size of double is 8 bytes. However, when long keyword is used, that variable becomes 10 bytes.
- There is another keyword *short* which can be used if you previously know the value of a variable will always be a small number.



**Sign qualifiers:** Integers and floating point variables can hold both negative and positive values. However, if a variable needs to hold positive value only, *unsigned* data types are used.

Example: **unsigned int sum;**

There is another qualifier *signed* which can hold both negative and positive only. However, it is not necessary to define variable signed since a *variable is signed by default*.

**NOTE:** It is important to note that, sign qualifiers can be applied to *int* and *char* types only.

**Constant qualifiers:** An identifier can be declared as a constant. To do so *const* keyword is used.

Example: **const int cost = 20;** // The value of *cost* cannot be changed in the program.

**Volatile qualifiers:** A variable should be declared volatile whenever its value can be changed by some external sources outside the program. Keyword *volatile* is used for creating volatile variables.

Examples: **volatile int foo;**  
**int volatile foo;**

both of these declarations will declare *foo* to be a volatile integer:

The following table provides the details of standard data types with their storage sizes, value ranges and format specifier.

C Data types / storage Size in bytes(16 bit machine)	Range	Format Specifier
char / 1	-127 to 127	%c
int / 2	-32,767 to 32,767	%d
float / 4	1E-37 to 1E+37 with six digits of precision	%f
double / 8	1E-37 to 1E+37 with 16 digits of precision	%lf
long double / 10	1E-37 to 1E+37 with 16 digits of precision	%Lf
long int / 4	-2,147,483,647 to 2,147,483,647	%ld
short int / 2	-32,767 to 32,767	%hd
unsigned short int / 2	0 to 65,535	%hu
signed short int / 2	-32,767 to 32,767	%hd
long long int / 8	-(2 <sup>63</sup> ) to (2 <sup>63</sup> )-1	%lld
signed long int / 4	-2,147,483,647 to 2,147,483,647	%ld
unsigned long int / 4	0 to 4,294,967,295	%lu
unsigned long long int / 8	(2 <sup>64</sup> ) - 1	%llu

### ENUMERATION DATA TYPE:

- Enumeration data type consists of named integer constants as a list.
- It start with 0 (zero) by default and value is incremented by 1 for the sequential identifiers in the list.

Syntax in C: **enum identifier [optional{ enumerator-list }];**

Example: **1. enum month { Jan, Feb, Mar };** or */\* Jan, Feb and Mar variables will be assigned to 0, 1 and 2 respectively by default \*/*

**2. enum month { Jan = 1, Feb, Mar };** */\* Feb and Mar variables will be assigned to 2 and 3 respectively by default \*/*

**3. enum month { Jan = 20, Feb, Mar };** */\* Jan is assigned to 20. Feb and Mar variables will be assigned to 21 and 22 respectively by default \*/*



**DERIVED DATA TYPE:** Derived data types are created from the basic integers, characters and floating data types. Array, pointer, structure and union are called derived data type in C language.

**VOID DATA TYPE:** Void is an empty data type that has no value. This can be used in functions and pointers.

*Simple Example addition of two numbers:*

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a,b,sum;           //variable declaration
    a=10;                 //variable definition or initialization
    b=20;                 //variable definition or initialization
    sum=a+b;
    printf("Sum is %d",sum);
    getch();
}
```

**INPUT/OUTPUT (I/O) STATEMENTS:** There are some library functions which are available for transferring the information between the computer and the standard input and output devices.

**Formatted and Unformatted Input/Output functions of C:**

**Unformatted Input/Output** is the most basic form of input/output. Unformatted input/output transfers the internal binary representation of the data directly between memory and the file.

**Formatted output** converts the internal binary representation of the data to ASCII characters which are written to the output file. **Formatted input** reads characters from the input file and converts them to internal form.

**Formatted Input & Output using printf() and scanf():**

**printf():**This function is used to print text as well as value of the variables on the standard output device (monitor), printf is very basic library function in c language that is declared in stdio.h header file.

**Syntax:**

1. `printf("message");`
2. `printf("message + format-specifier",variable_list);`

First printf() style printf the simple text on the monitor, while second printf() prints the message with values of the variable list

**How to print value of the variables?**

To print values of the variables, you need to understand about format specifiers are the special characters followed by % sign, which are used to print values of the variable s from variable list.

**Format specifiers**

Here are the list some of the format specifiers, use them in printf() & scanf() to format & print values of the variables:

Character	(char)	%c
Integer	(int)	%d
Unsigned integer	(unsigned int)	%u
Long	(long)	%ld



Unsigned long	(unsigned long)	%lu
Float	(float)	%f
Double	(double)	%lf
Octal Value	(octal value)	%o
Hexadecimal Value	(hex value)	%x
String	(char[])	%s

**\*\*NOTE\*\*** Use 'u' for unsigned type modifier, 'l' for long.

Consider the following examples:

```
#include <stdio.h>

int main()
{
    int    num=100;
    float  val=1.23f;
    char   sex='M';
    //print values using different printf
    printf("Output1:");
    printf("%d",num);
    printf("%f",val);
    printf("%c",sex);
    //print values using single printf
    printf("\nOutput2:"); // \n: for new line in c
    printf("%d,%f,%c",num,val,sex);
    return 0;
}
```

**scanf():**This function is used to get (input) value from the keyboard. We pass format specifiers, in which format we want to take input.

**Syntax:**

```
scanf("format-specifier", &var_name);
scanf("format-specifier-list", &var_name_list);
```

First type of scanf() takes the single value for the variable and second type of scanf() will take the multiple values for the variable list. **#include <stdio.h>**

**Example:**

```
int main()
{
    int a;
    float b;
    char c;
    printf("Enter an integer number (value of a)?:");
    scanf("%d",&a); // single value
    printf("Enter a float number (value of b)?:");
    scanf("%f",&b); //single value
    printf("\nEnter value of a,b,c (an integer, a float, a character):");
    scanf("%d%f%c",&a,&b,&c); //multiple variable list
    return 0;
}
```



## Unformatted input/output (I/O) Functions:

**getchar():** function will read a single character from the standard input. The return value of getchar() is the first character in the standard input. The input is read until the Enter key is pressed, but only the first character in the input will be returned.

**putchar():** function will print a single character on standard output. The character to be printed is passed to putchar() function as an argument. The return value of putchar() is the character which was written to the output.

Note: getchar() and putchar() functions are part of the standard C library header stdio.h

Example:

```
#include <stdio.h>
void main()
{
    char ch;
    printf("Input some character and finish by pressing the Enter key.\n");
    ch = getchar();
    printf("The input character is ");
    putchar(ch);
}
```

## **getch(), getche() and putch():**

These functions are similar to getchar() and putchar(), but they come from the library header conio.h. The header file conio.h is not a standard C library header and it is not supported by compilers that target Unix. However it is supported in DOS like environments.

**getch():** reads a single character from the standard input. The input character is not displayed (echoed) on the screen. Unlike the getchar() function, getch() returns when the first character is entered and does not wait for the Enter key to be pressed.

**getche():** is same as getch() except that it echoes the input character.

**putch():** writes a character that is passed as an argument to the console.

Example:

```
#include <conio.h>
#include <stdio.h>
void main()
{
    char ch;
    printf("Press any key\n");
    ch = getch();
    printf("The key pressed is: ");
    putch(ch);
}
```

**gets():** This function is used for accepting any string until enter key is pressed

**Syntax:** char str[length of string in number];  
gets(str);

Example: char xyz[20];  
gets(xyz);





**puts():**This function prints the string or character array. It is opposite to gets()

**Syntax:** char str[length of string in number];

gets(str);

puts(str);

Example: char abc[20];

gets(abc);

puts(abc);

## OPERATORS AND EXPRESSIONS:

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators and provides the following types of operators:

1. Arithmetic operators
2. Assignment operators
3. Relational operators
4. Logical operators
5. Bit wise operators
6. Conditional operators (ternary operators)
7. Increment/decrement operators
8. Special operators

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

Arithmetic operators are of two types:

**a). Unary Operators:** Operators that operates or works with a single operand are unary operators.

For example: (++ , -,etc)

**b). Binary Operators:** Operators that operates or works with two operands are binary operators.

For example: All arithmetic operators (+ , - , \* , /,%)

**Syntax:** *operand1* Operator *operand2*;

Operator	Description	Examples
+	This operator adds two operands.	A+B, 2+A, 20+10.5, 20+100, etc
-	Subtracts second operand from the first.	A-B, B-4, 100-20, 10.23-C, etc
*	Multiplies both operands.	A*B, 10*C, 10.43*2, A*9.0, etc
/	Operator divides the first operand by the second and returns quotient	A/B,100/D, X/20, 100/20.9, etc
%	Operator returns the remainder when first operand is divided by the second <i>Note: Only integer operation (Ex: 20%10), No floating point operation(Ex: 10.5%5.6 is invalid).</i>	A%B, 20% 10, 10%X, Y%5, etc (A,B,X,Y must be integer values)

**2. Assignment Operators:** Assignment operators are used to assign value to a variable. The left side operand of the assignment operator is a variable and right side operand of the assignment operator is a value.

**Syntax:** *Variable* Assignment Operator *Value(constant) or result of any expression or value of other variable*



Operator	Description	Examples
=	This is the simplest assignment operator. This operator is used to assign the value on the right to the variable on the left.	A=B, X=12, C=2+3, D=A-21, S=4/6, B=10.5, Ch='A', etc
+=	This operator is combination of '+' and '=' operators. This operator first adds the current value of the variable on left to the value on right and then assigns the result to the variable on the left.	(a+=b) can be written as (a= a+b), (c+=2) can be written as (c=c+2), Etc.
-=	This operator is combination of '-' and '=' operators. This operator first subtracts the current value of the variable on left from the value on right and then assigns the result to the variable on the left.	(a-=b) can be written as (a= a-b), (c-=2) can be written as (c=c-2), etc.
*=	This operator is combination of '*' and '=' operators. This operator first multiplies the current value of the variable on left to the value on right and then assigns the result to the variable on the left.	(a*=b) can be written as (a= a*b), (c*=2) can be written as (c=c*2), etc.
/=	This operator is combination of '/' and '=' operators. This operator first divides the current value of the variable on left by the value on right and then assigns the result to the variable on the left.	(a/=b) can be written as (a= a/b), (c/=2) can be written as (c=c/2), etc.

**3. Relational operators:** Relational operators are used for comparison of the values of two operands. For example: checking if one operand is equal to the other operand or not, an operand is greater than the other operand or not etc. The result of all relational operators is either true(1) or false(0).

**Syntax:** *operand1 relational operator operand2;*

Operator	Description	Examples
==	Checks if the values of two operands are equal. If yes, then the condition becomes true (1) otherwise condition fails(0).	X==Y, 10==10, 2+2==1+3, 'C'=='C', 10.23==10.23, 10==9(fail), etc
!=	Checks if the values of two operands are not equal. If the values are not equal, then the condition becomes true (1) otherwise condition fails (0).	X!=Y, 10!=10(fail), 10-2!=10-3, 'A'!='B', 2.3!=1.5, a+c!=d*2, etc
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true(1) otherwise condition fails(0).	X>Y, 10>9, 20-2>10, 100.5>200(fail), 'B'>'A', etc
<	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true(1) otherwise condition fails(0).	D<C, 9<10, 25.3-2<11, 'C'<'A'(fail), 2*2<3+5, etc
>=	Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true(1) otherwise condition fails(0).	X>=Y, 10>=10, 20>=30(fail), 10*2>=10+5, etc
<=	Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true(1) otherwise condition fails(0).	E<=W, 10<=20, 10+20<=10*2(fail), etc



**4. Logical Operators:** C provides three logical operators when we test more than one condition to make decisions. The result of all logical operators is either true (1) or false (0).

**Syntax: *operand1* logical Operator *operand2*;**

Operands may be any expression or any variable or any value

Operator	Description	Examples
<b>&amp;&amp;</b>	Logical AND, operator returns true when both the operands in consideration are satisfied. Otherwise it returns false.	If c = 5 and d = 2 then, expression ((c == 5) && (d > 5)) equals to 0.
<b>  </b>	Logical OR, operator returns true when one (or both) of the operands in consideration is satisfied. Otherwise it returns false.	If c = 5 and d = 2 then, expression ((c == 5)    (d > 5)) equals to 1.
<b>!</b>	Logical NOT, It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false. This is unary operator.	If c = 5 then, expression !(c == 5) equals to 0.

**5. Bit wise operators:** The Bitwise operators are used to perform bit-level operations on the operands. The operands are first converted to bit-level and then calculation is performed on the operands. The mathematical operations such as addition, subtraction, multiplication etc. can be performed at bit-level for faster processing.

**Syntax: *operand1* bitwise operator *operand2*;**

Truth table for bitwise &, | and ^

Input A	Input B	Output of A&B	Output of A B	Output of A^B	Output of ~A
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

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

**A = 0011 1100**

**B = 0000 1101**

Operator	Description	Examples
<b>&amp;</b>	Bitwise AND, Takes two numbers as operands and does AND on every bit of two numbers. The result of AND is 1 only if both bits are 1.	$\begin{array}{r} A = 0011\ 1100 \\ B = 0000\ 1101 \\ \hline A\&B = 0000\ 1100 \end{array}$ i.e. 12 in decimal
<b> </b>	Bitwise OR, Takes two numbers as operands and does OR on every bit of two numbers. The result of OR is 1 if any of the two bits is 1.	$\begin{array}{r} A = 0011\ 1100 \\ B = 0000\ 1101 \\ \hline A B = 0011\ 1101 \end{array}$ i.e. 61 in decimal
<b>^</b>	Bitwise XOR, Takes two numbers as operands and does XOR on every bit of two numbers. The result of XOR is 1 if the two bits are different.	$\begin{array}{r} A = 0011\ 1100 \\ B = 0000\ 1101 \\ \hline A\^B = 0011\ 0001 \end{array}$ i.e. 49 in decimal
<b>~</b>	Bitwise NOT, Takes one number as operand and inverts all bits of it. It is unary operator.	$\begin{array}{r} B = 00001101 \\ \hline \sim B = 11110010 \end{array}$ i.e. -14 in decimal



Operator	Description	Examples
<<	Bitwise Left Shift. Takes two numbers, left shifts the bits of the first operand, the second operand decides the number of places to shift.	$A = 0011\ 1100$ $A \ll 2 = 1111\ 0000$ i.e. 240 in decimal
>>	Bitwise Right Shift. Takes two numbers, right shifts the bits of the first operand, the second operand decides the number of places to shift.	$A = 0011\ 1100$ $A \gg 2 = 0000\ 1111$ i.e. 15 in decimal

**Note:** Bit wise left shift and right shift: In left shift operation “ $x \ll 1$ “, 1 means that the bits will be left shifted by one place. If we use it as “ $x \ll 2$ “, then, it means that the bits will be left shifted by 2 places.

**6. Conditional operators (ternary operators):** Conditional operators return one value if condition is true and returns another value if condition is false. This operator is also called as ternary operator.

**Syntax :** (Condition)? true\_value : false\_value;

- Example:1  $y = (A > 100) ? 0 : 1$ ; depending on A the value of y is either 0 or 1;  
 2.  $X = (10 > 20) ? 10 : 20$ ;  
 X value is 20;

**7. Increment/decrement operators:** The ones falling into the category of unary arithmetic operators. These operators are used to either increase or decrease the value of the variable by one.

**Syntax:**

- Increment operator:  $++var\_name$ ;                      pre-increment  
     $var\_name++$ ;                      post-increment  
 Decrement operator:  $--var\_name$ ;                      Pre-decrement  
     $var\_name--$ ;                      post-decrement

Operator	Description	Examples
++	This operator is used to increment the value of an integer by one.	
	When placed before the variable name (also called <b>pre-increment</b> operator), its value is incremented instantly.	$X=3$ ; $Y=++X$ ; Value of Y is 4 and value of X is 4;
	When it is placed after the variable name (also called <b>post-increment</b> operator), its value is used first and it gets updated before the execution of the next statement.	$X=3$ ; $Y=X++$ ; Value of Y is 3 and value of X is 4;
--	This operator is used to decrement the value of an integer by one.	
	When placed before the variable name (also called <b>pre-decrement</b> operator), its value is decremented instantly.	$X=3$ ; $Y=--X$ ; Value of Y is 2 and value of X is 2;
	When it is placed after the variable name (also called <b>post-decrement</b> operator), its value is preserved temporarily until the execution of this statement and it gets updated before the execution of the next statement.	$X=3$ ; $Y=X--$ ; Value of Y is 3 and value is X is 2;



**8. Special operators:** Apart from the above operators there are some other operators available in C used to perform some specific task.

Operator	Description	Examples
sizeof()	Returns the size of a variable.	sizeof(a), where a is integer, will return 4.
,	The comma operator (represented by the token ,) is a binary operator that evaluates its first operand and discards the result, it then evaluates the second operand and returns this value (and type).	int i = (5, 0); 10 is assigned to i
&	Returns the address of a variable.	&a; returns the actual address of the variable.
*	Pointer to a variable.	*a;

**Precedence of operators:** If more than one operators are involved in an expression, C language has a predefined rule of priority for the operators. This rule of priority of operators is called operator precedence.

*Example1:* (1 > 2 + 3 && 4)  
This expression is equivalent to:  
((1 > (2 + 3)) && 4)  
i.e, (2 + 3) executes first resulting into 5  
then, first part of the expression (1 > 5) executes resulting into 0 (false)  
then, (0 && 4) executes resulting into 0 (false)  
result is 0

*Example2:* 10 + 20 \* 30 is calculated as 10 + (20 \* 30) and not as (10 + 20) \* 30.

**Associativity of 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. Associativity of operators indicates the order in which they execute.

*Example1:* 1 == 2 != 3  
Here, operators == and != have same precedence. The associativity of both == and != is left to right, i.e, the expression on the left is executed first and moves towards the right.  
Thus, the expression above is equivalent to :  
((1 == 2) != 3) i.e, (1 == 2) executes first resulting into 0 (false)  
then, (0 != 3) executes resulting into 1 (true)  
result is 1(true)

*Example2:* 100 / 10 \* 10 is treated as (100 / 10) \* 10.

**Note:** 1. Associativity is only used when there are two or more operators of same precedence.

2. All operators with same precedence have same associativity.

3. Precedence and associativity of postfix ++ and prefix ++ are different



**Operators with the highest precedence appear at the top of the table:**

Category	Operator	Associativity
Postfix	() [] -> . ++ --	Left to right
Unary	+ - ! ~ ++ -- (type) * & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR		Left to right
Logical AND	&&	Left to right
Logical OR		Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^=  =	Right to left
Comma	,	Left to right

Example1: Solve the given expression.

$$x = 9 - 12 / 3 + 3 * 2 - 1$$

Solution: **x = 9 - 12 / 3 + 3 \* 2 - 1**

$$x = 9 - 4 + 3 * 2 - 1 \quad [ / ]$$

$$x = 9 - 4 + 6 - 1 \quad [ * ]$$

$$x = 5 + 6 - 1 \quad [ - ]$$

$$x = 11 - 1 \quad [ + ]$$

$$x = 10 \quad [ - ]$$

Example2: Solve the given expression:  $x = 9 - 12 / (3 + 3) * (2 - 1)$

Solution: **x = 9 - 12 / (3 + 3) \* (2 - 1)**

$$x = 9 - 12 / 6 * (2 - 1) \quad [(3+3)]$$

$$x = 9 - 12 / 6 * 1 \quad [(2-1)]$$

$$x = 9 - 2 * 1 \quad [ / ]$$

$$x = 9 - 2 \quad [ * ]$$

$$x = 7 \quad [ - ]$$

Example3: Evaluate the below expression.

$$f = ((i + j) < 10) \&\& (b > (k - j)); \text{ where } i = 5, j = 6, k = 7, b = 0.$$

Solution: **f = ((i + j) < 10) && (b > (k - j));**

$$f = ((5 + 6) < 10) \&\& (0 > (7 - 6));$$

$$f = (11 < 10) \&\& (0 > (7 - 6)); \quad [ + ]$$

$$f = 0 \&\& (0 > (7 - 6)); \quad [ < ]$$

$$f = 0 \&\& (0 > 1); \quad [ - ]$$

$$f = 0 \&\& 0; \quad [ > ]$$

$$f = 0; \quad [ \&\& ]$$

Example4: Evaluate the below expression.

$$a = i++ + j++; \quad \text{if } i = 1, j = 2,$$

Solution: **a = i++ + j++;**

$$a = 1 + j++; \quad [i++, \text{ use } i \text{ value first and update } i. \text{ i.e. } i=i+1=1+1=2]$$

$$a = 1 + 2; \quad [j++, \text{ use } j \text{ value first and update } j. \text{ i.e. } j=j+1=2+1=3]$$

$$a = 3; \quad [ + ]$$



Example5: Evaluate the below expression.

$a = b += c++ - d + --e/-f$ ; With  $a=1, b=2, c=12, d=2, e=5, f=2$

Solution:  $a = b += c++ - d + --e/-f$ ;

$a = b += 12 - d + --e/-f$ ;

$a = b += 12 - d + 4/-f$ ;

$a = b += 12 - d + 4/-2$ ;

$a = b += 12 - d - 2$ ;

$a = b += 10 - 2$ ;

$a = b += 8$ ;

$a = b = 10$ ;

$a = b = 10$ ;

Ans:  $a=10$  and  $b=10$

[ $c++$ , use  $c$  value i.e.12 and update  $c=c+1=12+1=13$ ]

[ $--e$ , decrement  $e$  first i.e.  $e=e-1=5-1=4$  use this value i.e.

$e=4$ ]

[ $-f$ ]

[ $/$ ]

[ $-$ ]

[ $-$ ]

[ $+=$ ]

[ $=$ ]

## Type Conversion/ Type Casting:

Type casting/conversion is a way to convert a variable from one data type to another data type. There are two types of type conversion:

### 1. Implicit Type Conversion: Also known as 'automatic type conversion'.

- Done by the compiler on its own, without any external trigger from the user.
- Generally takes place when in an expression more than one data type is present. In such condition type conversion (type promotion) takes place to avoid loss of data.
- All the data types of the variables are upgraded to the data type of the variable with largest data type.

*bool -> char -> short int -> int -> unsigned int -> long -> unsigned -> long long -> float -> double -> long double*

Example1:  $int\ x=10;$

$float\ y;$

$y=x+1.0;$  //  $x$  is implicitly type converted to float

$y=11.000000$

Example2:  $int\ a=100,b;$

$b=a/13;$  //no type conversion because all the operations are of same type(integer) so result is truncated to only integer part i.e. 7 instead of 7.692307

$b=7;$

Example3:  $int\ p=7, q=3;$

$float\ m;$

$m=p/q;$

$m=2.000000$

In this example right of  $=$  operator the operation is integer division( $p/q$ ), so the result of that operation is only integer (no type conversion) i.e. 2, but this result we are assigning to float variable( $m$ ) that's why type conversion is needed for assignment i.e. instead of integer 2 it will store 2.000000.

### 2. Explicit type Conversion:

The type conversion performed by the programmer by posing the data type of the expression of specific type is known as explicit type conversion.

The explicit type conversion is also known as type casting.

Type casting in c is done in the following form:

**(data\_type)expression;**

where,  $data\_type$  is any valid c data type, and  $expression$  may be constant, variable or expression.

For example,

$x=(int)a+b*d;$

$y=(int)22.3/(int)4.5=22/4=5$



### Programming Examples:

#### 1. C Program to find the simple interest.

```
#include<stdio.h>
void main()
{
    int amount, rate, time, si;
    printf("\nEnter Principal Amount : ");
    scanf("%d", &amount);
    printf("\nEnter Rate of Interest : ");
    scanf("%d", &rate);
    printf("\nEnter Period of Time  : ");
    scanf("%d", &time);
    si = (amount * rate * time) / 100;
    printf("\nSimple Interest : %d", si);
    getch();
}
```

#### 2. Write a c program to Calculate Area of Triangle with given three sides.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
    float a,b,c,s,area;
    clrscr();
    /* reading part */
    printf("Enter Three side of triange -> ");
    scanf("%f%f%f",&a,&b,&c);
    /* processing part */
    s=(a+b+c)/2.0;
    area=sqrt(s*(s-a)*(s-b)*(s-c));
    /*display result */
    printf("Side1 =%f\nSide2 =%f\nSide3 =%f\n",a,b,c);
    printf("\nArea of Triange :-> %f \n",area);
    getch();
}
```

#### 3. C Program to Swap Numbers Using Temporary Variable.

```
#include <stdio.h>
void main()
{
    int firstNumber, secondNumber, temporaryVariable;
    printf("Enter first number: ");
    scanf("%d", &firstNumber);
    printf("Enter second number: ");
    scanf("%d",&secondNumber);
    // Value of firstNumber is assigned to temporaryVariable
    temporaryVariable = firstNumber;
    // Value of secondNumber is assigned to firstNumber
    firstNumber = secondNumber;
    // Value of temporaryVariable (which contains the initial value of firstNumber) is assigned to secondNumber
    secondNumber = temporaryVariable;
    printf("\nAfter swapping, firstNumber = %d\n", firstNumber);
    printf("After swapping, secondNumber = %d", secondNumber);
    getch();
}
```





#### 4. C Program to Swap Number without Using Temporary Variables.

```
#include <stdio.h>
void main()
{
    int firstNumber, secondNumber;
    printf("Enter first number: ");
    scanf("%d", &firstNumber);
    printf("Enter second number: ");
    scanf("%d",&secondNumber);
    // Swapping process
    firstNumber = firstNumber - secondNumber;
    secondNumber = firstNumber + secondNumber;
    firstNumber = secondNumber - firstNumber;
    printf("\nAfter swapping, firstNumber = %d\n", firstNumber);
    printf("After swapping, secondNumber = %d", secondNumber);
    getch();
}
```

#### 5. C Program to find area and circumference of circle.

```
#include<stdio.h>
void main()
{
    int rad;
    float PI = 3.14, area, ci;
    printf("\nEnter radius of circle: ");
    scanf("%d", &rad);
    area = PI * rad * rad;
    printf("\nArea of circle : %f ", area);
    ci = 2 * PI * rad;
    printf("\nCircumference : %f ", ci);
    getch();
}
```

#### 6. C Program to convert temperature from degree centigrade to Fahrenheit.

```
#include<stdio.h>
void main()
{
    float celsius, fahrenheit;
    printf("\nEnter temp in Celsius : ");
    scanf("%f", &celsius);
    fahrenheit = (1.8 * celsius) + 32;
    printf("\nTemperature in Fahrenheit : %f ", fahrenheit);
    getch();
}
```