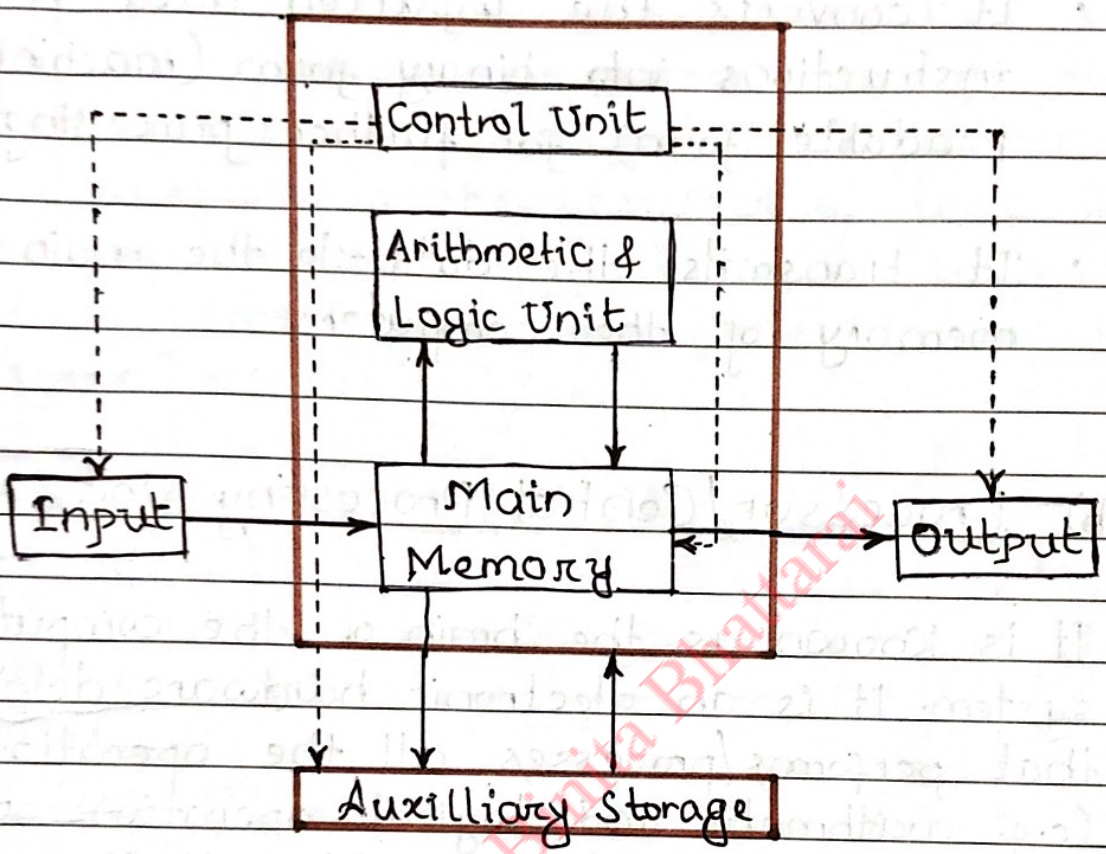
Date |
Page |

CHAPTER-1

Block Diagram of Computer



Solid arrow (⟶): data/instruction flow
Dashed line (- - -): signal flow

A computer is integration of software & hardware. It is a fully functionable system.
The various components of a computer are given below:

a) **Input Unit**: The input unit consists of input devices such as a mouse, keyboard, scanner, joystick etc. These devices are used to provide information or instruction into the computer system.

Date
Page

The input unit performs the following major functions:

- It converts the inputted data or instructions into binary form (machine readable form) for further processing.
- It transmits the data to the main memory of the computer.

b) Processor / Central Processing Unit

It is known as the brain of the computer system. It is an electronic hardware device that performs/processes all the operations (e.g. arithmetic and logical operations) of the computer. In other words, all the major calculations, operations or comparisons are performed inside the CPU. It is also responsible for handling the operations of several other units.

In the block diagram, the **control unit (CU)** and **Arithmetic & Logic Unit (ALU)** are jointly called the Central Processing Unit (CPU).

★ Control Unit:

As the name suggests, the control unit of a CPU controls all the activities and operations of the computer. It is also responsible for controlling input/output, memory, and other

devices connected to the CPU.
The CU acts like the supervisor which determines the sequence in which computer programs & instructions are executed. It helps to retrieves instructions from memory, decodes the instructions, interprets the instructions & understands the sequence of tasks to be performed accordingly. It further transmits the instructions to the other parts of the computer system to execute them.

★ Arithmetic Logic Unit (ALU)

As the name suggests, it carries out (performs) arithmetic & logical operation. It performs basic arithmetic calculation such as addition, subtraction, multiplication etc, & performs logical operation such as ~~addition~~, comparing greater than, less than, equal to, ~~etc~~ AND, OR, etc.

c) Memory Unit:

It is an essential part of the computer system which is used to store data & instructions before and after processing.
There are two types of memory units:
* Primary (Main) Memory
* Secondary Memory.

Date
Page

★ Main Memory: It is a volatile (temporary) memory of the computer. The stored data will be lost if they are disconnected from the power supply. As soon as a computer starts, it stores all running application and operating system. A program/application opened in primary memory interact with the system processor to perform all application specific tasks. It also stores the input data & immediate calculation results. This memory is (unlike secondary) directly accessible to the processor.
Random Access Memory (RAM) is the example of main memory. It can handle instruction & data at high speed. Its storage capacity is smaller than that of secondary memory.

★ Secondary Memory:
This is the memory for storing data permanently for future use. It is also known as auxilliary memory. Hard disk is usually considered a secondary memory. Its storage capacity is high & is less expensive. It has slower access rate.

V BAD FASE MA

Date
Page

Primary Memory v/s Secondary Memory

	Basis for Comparison	Primary Memory	Secondary Memory
1)	Basic	It is directly accessible by CPU	not directly accessible by CPU.
2)	Altered name	Main memory	Auxilliary memory
3)	Data	Instructions/data to be currently executed are copied to main mem.	Data to be permanently stored are kept in secondary memory.
4)	Volatility	It is volatile.	non-volatile
5)	Formation	made of semiconductor	made of magnetic & optical materials.
6)	Access Speed	Accessing data from this is faster.	slower
7)	Access	It is accessed by the data bus	indirectly accessed by input-output ~~channels~~ operations.
8)	Size	Computer has a small primary memory.	has a larger secondary memory.
9)	Expense	Costlier than secondary memory.	Cheaper than Primary memory.
10)	Memory	is an internal memory.	is an external memory.

Date
Page

Software:

It is a set of instruction data, programs, documentation (guide about installation & use) used to operate computer and execute specific tasks. Software forms the heart of the computer system. It is opposite of hardware, which describes the physical aspects of a computer. Software is a generic term used to refer to applications, scripts & programs that run on a device.

There are two types of computer software. They are:

A) System Software:

It is a computer software designed to operate the computer hardware & provide a platform for running application software. It helps in running the computer hardware & computer system, system services, windowing system & utilities.

Some common type of system software are:

1) Operating system: It is the most important program that runs on a computer. It manages all the other computer programs. The functions of OS:

- recognize input from keyboard
- send output to display screen
- keep track of files & directories on the desk containing peripheral devices such as disk drive & printers

- provide security and back up
- provide interface between software & hardware
- management of memory etc.

Examples: Windows 2000, UNIX, DOS, Windows XP etc.

2) Utility Software: Utility programs help to manage, maintain & control computer resources. These programs help to keep the system running at peak performance.
Example: Virus scanning software, backup software, scandisk etc.

3) Language Processor: It is a special type of computer system software that can be used to translate the program written in one language to another language. Examples are compiler, interpreter, assembler.

B) Application Software:

It enables the uses to accomplish certain specific tasks. Business software, database & educational software are some examples. An application can be self-contained (single) or it can be a group of programs.
(More examples: office suites, graphics software, web browsers, word processors, software development tools, image editors etc.

Date
Page

It can be further divided into two parts:

a) Package Software: It is mainly designed by software companies to generalize the tasks. They are general purpose software. Examples: word processing software (like Ms-Word), database software, spreadsheet (Ms-Excel) etc.

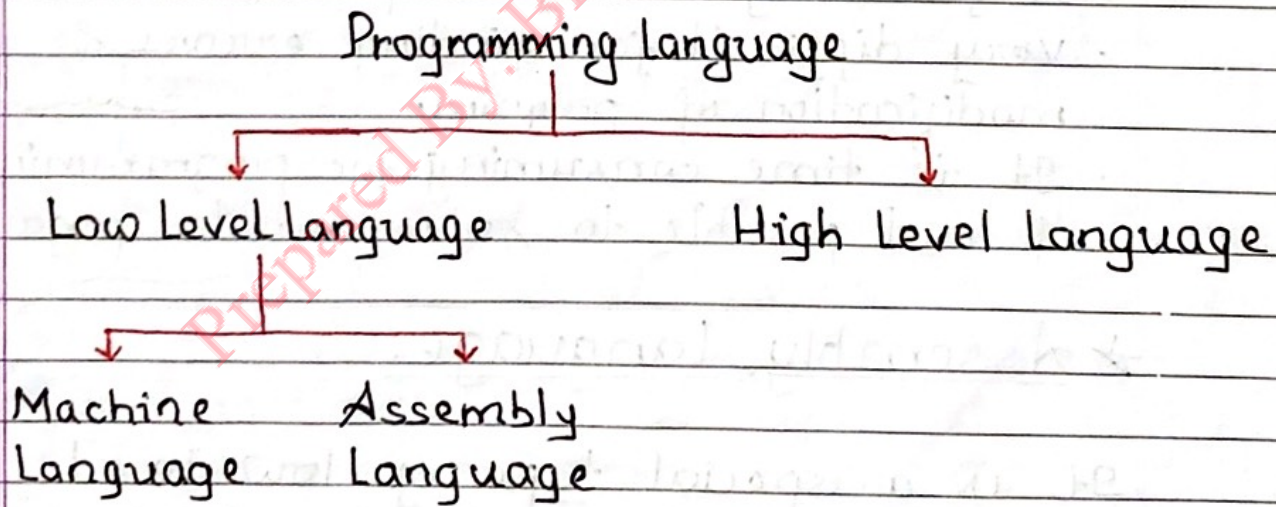
Custom → b) Tailored Software: It is specific purpose software. It is also called small type of software. They are written in high level language such as JAVA, C, C++, COBOL (Common Business Oriented Language) etc.
Examples: Banking software, hospital software, hotel reservation software etc.

Date |
Page |

Programming Language: [PL]

Programming language is defined as a set of rules that provide a platform for instructing computer to perform specific tasks. The process of writh writing a program is known as PROGRAMMING & the person who writes program codes is known as PROGRAMMER. Hence, the language needed by programmers to communicate with computers is called Programming language. PL is mainly used to develop desktop applications, websites, & mobile applications. Examples: Assembly, C, C++, JAVA, Python etc.

PL is classified as below:



Date
Page

★ Machine Language:

It is the first language of computer system, a language till date. It is the language of CPU which consists of set of instruction composed of 0s and 1s (binary digital numbers).

Advantage:
- It doesn't require translation process because it is already the language of CPU, so doesn't require translator.
- Execution time of machine language is extremely faster.

Disadvantage:
- Programming is very complex & tedious.
- very difficult for finding errors & modification of program.
- It is time consuming for programming & not possible to solve complex program.

★ Assembly Language:

It is a special type of low level language which consists of set of alphanumeric intruction called mnemonics. Meaningful & easily memorable symbols are selected for this purpose. For example: ADD for addition, SUB for subtraction, MUL for multiplication etc.

Date |
Page |

Advantage:
- easier to understand than machine language.
- more standard form of language compared to machine language.
- consumes very less amount of memory, so it executes faster.
- easier to find error because translator itself identifies error.

Disadvantage:
- Program written in this language is very lengthy & complex.
- Programmer should have depth knowledge about computer hardware.
- Program execution is slower & less efficient than machine language.

★ High Level Language: ✓

It is a special type of language which is closer to natural (human) language.

Advantage:
- closer to natural language.
- easy to use
- machine independent, easier to maintain & debug
- has readability.

Disadvantage:
- slower compared to low level language
- Translator is required i.e. compiler & interpreter.

Difference between Compiler and Interpreter:

Compiler	Interpreter
1.) It translates the entire source program in a single attempt & then only object program is executed.	1. It translate one statement at a time, execute it and continues another statement.
2. It is faster than interpreter (5-25 times)	2. slower compared to compiler.
3. It is complex program i.e. larger than interpreter.	3.) simple program compared to compiler.
4. It occupies more memory	4.) occupies less memory
5. It is bit difficult & slower to detect & correct error.	5.) Easier to detect & correct error.
6. The compiler program is permanently saved to hard disk for future use.	6.) Interpreter program is not saved.
7. Examples: C, C++, Visual basic etc.	7. BASIC, LISP, PERL etc.

Structured Programming:

Structured Programming Approach, as the word suggests, can be defined as a programming approach in which the program is made as a single structure. It means that the code will execute instruction by instruction one after the another. It doesn't support the possibility of jumping from one instruction to some other with the help of any statement like GOTO etc. Therefore, instructions in this approach will be executed in a serial ~~manner~~ & structured manner. The languages that support structured programming approach are:

- C
- C++
- Java
- C# (C-sharp) etc.

Structured programming is sometimes also known as modular programming which facilitate the creation of programs with readable code & reusable components. It encourages dividing an application program into a hierarchy of modules or autonomous (आपस में independent भाग चल सकते) elements, which may, in turn, contain other such elements. Within each element, code may be further structured using blocks of related logic designed to improve readability & maintainability.

Advantages:

- complexity can be reduced using the concepts of divide & conquer.
- Increase in productivity by allowing multiple programmers to work on different parts of the project independently at the same time.
- Modules can be reused many times, hence, it saves time, reduces complexity & increases reliability.
- Easier to update/fix the program by replacing individual modules rather than large amount of code.
- Ability to either eliminate or at least reduce the necessity of employing GOTO statement.

Disadvantages:

- Since GOTO statement is not used, the structure of the program needs to be planned with great attention.
- Lack of encapsulation
  (direct access गराउनमा restriction लगाउने तरिकालाई encapsulation भनिन्छ)
  ↳ Lack of information hiding
- Reduction in execution efficiency
- Greater memory usage
- Usually the development in this approach takes longer time as it is language dependent. Whereas in case of assembly language, the development takes lesser time as it is fixed for the machine.

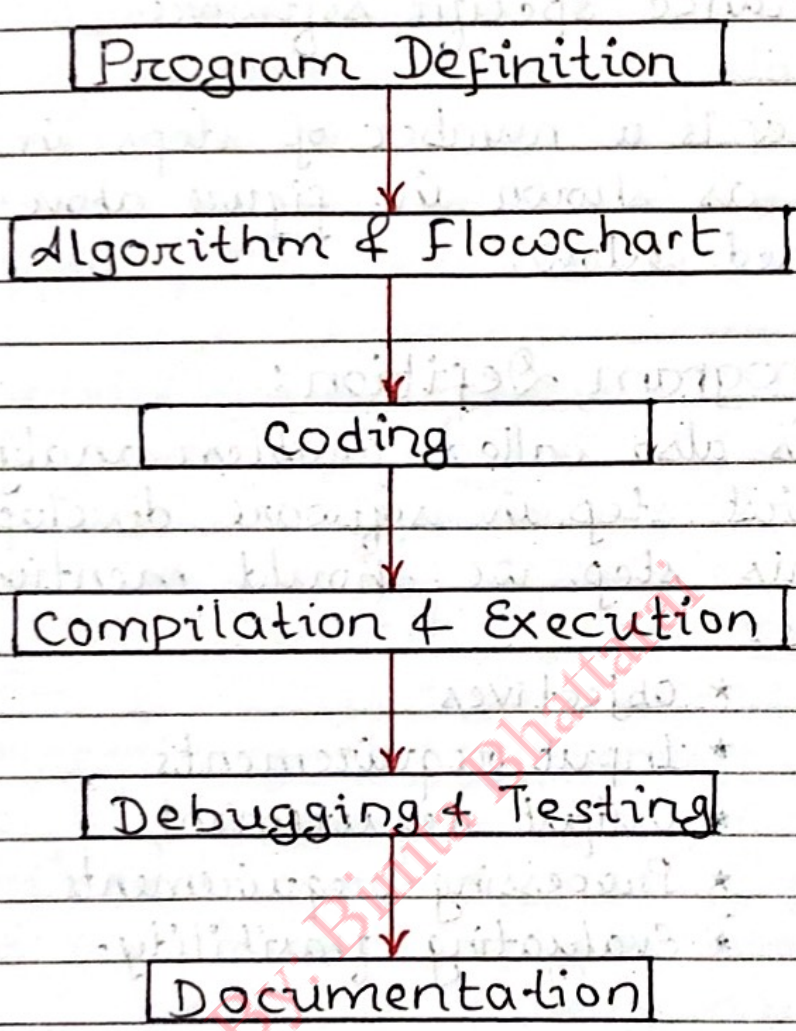
★ Why is C called structured programming language?

⇒ It is called so, because, to solve a large program, C programming language divides the problem into smaller modules called functions or procedures each of which handles a particular responsibility. The problem which solves the entire problem is a collection of such functions.
Here is an example of Matrix addition program which is divided into few sub procedures - input matrix, display matrix, add matrix, save result matrix in to file.

Here is the pictorial structural view of the program:

- main() matrix addition
  - AddMatrix
  - DisplayMatrix
  - InputMatrix
  - SaveMatrixToFile

Software Development Life Cycle (SDLC)



SDLC is the process used by the software industry to design, develop and test high quality softwares. The SDLC aims to produce a high-quality software that meets or exceeds customer expectations, reaches completion within time and cost estimates.

It is also called Software Development Process.

SDLC is a framework defining tasks performed as each step in the software development process.

SDLC consists of a detailed plan describing how to develop, maintain, replace & alter or enhance specific software.

There are is a number of steps involved in SDLC as shown in figure above. They are explained below:

a) Program Defition:

It is also called problem analysis. It is the first step in software development. In this step, we should mention following things:

* Objectives
* Input requirements
* Output requirements
* Processing requirements
* Evaluating feasibility.

{ feasibility → time frame, resource availability, cost, complexity etc.

b) Algorithm & Flowchart:

Algorithm is the sequence of instructions designed in such a way that if the instruction are executed in the specified sequence, the desired result is obtained.

Algorithm must terminate so it must not repeat one or more statement infinitely. Algorithm is a step-by-step procedure for solving a problem. Programming language are essentially a way for expressing algorithm.

Guidelines for writing algorithm:

- Use plain Language
- Do not use any language-specific syntax.
- Do not make any assumption.
- Make sure that an algorithm has single entry & exit point.

Features/Properties of Algorithm:

1.) Finiteness/Termination: An algorithm should terminate after finite number of steps. It must not repeat one or more statement infinitely.

2.) Input: An algorithm can have zero or more inputs. Inputs can be given initially or as program runs.

3) Output: An algorithm can give one or more output results. After an algorithm terminates, algorithm must provide desired results.

4.) Definiteness: Each & every step of an algorithm must be precisely defined so that it is well understood.

5.) Effectiveness: An algorithm must be more effective among different ways of problem solving. An algorithm must be simple & should be done infinite time to produce effective result.

An algorithm to add two number:

1. Start
2. Declare variables a, b & sum.
3. Input a & b.
4. calculate sum = a+b
5. Display/print sum.
6. Stop.

To calculate area of rectangle:

1. Start
2. Declare variables l, b, & area.
3. Input l & b.
4. Calculate area = l * b
5. Print area
6. Stop.

To calculate simple interest:

1. Start
2. Declare variables p, t, r & int.
3. Input p, t, r
4. Calculate int = (p * t * r)/100
5. Print int
6. Stop.

To calculate area of a triangle when three sides are given:

1. Start
2. Declare variables a, b, c, s & area
3. Input a, b, c
4. Calculate $s = (a+b+c)/2$
5. Calculate $area = \sqrt{s*(s-a)*(s-b)*(s-c)}$
6. Print area
7. Stop.

To find whether given number is positive or negative:

1. Start
2. Declare variable x
3. Input x
4. If x > 0
   Print number is positive
   Else
   Print number is negative

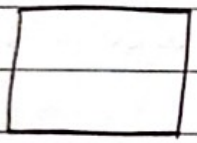
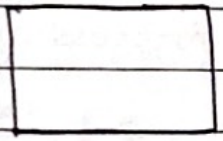
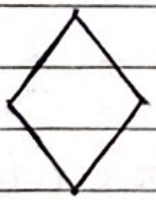
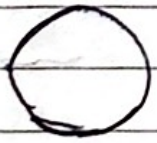
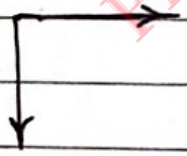
To find whether given number is odd or even:

1. Start
2. Declare variable num
3. Input num
4. if (num % 2 == 0)
   Print : number is even
   else
   Print : number is odd.

Flowchart:

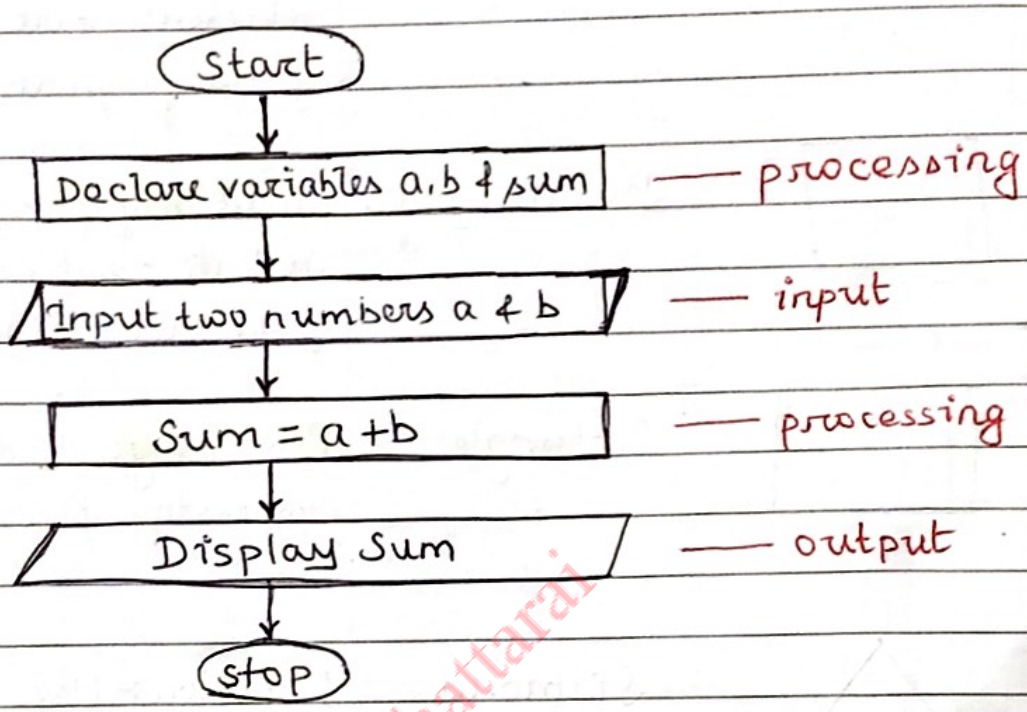
It is a diagrammatic representation of an algorithm. It is the pictorial representation of a program. It is drawn by using basic block where each block have their usual meaning.

Date
Page

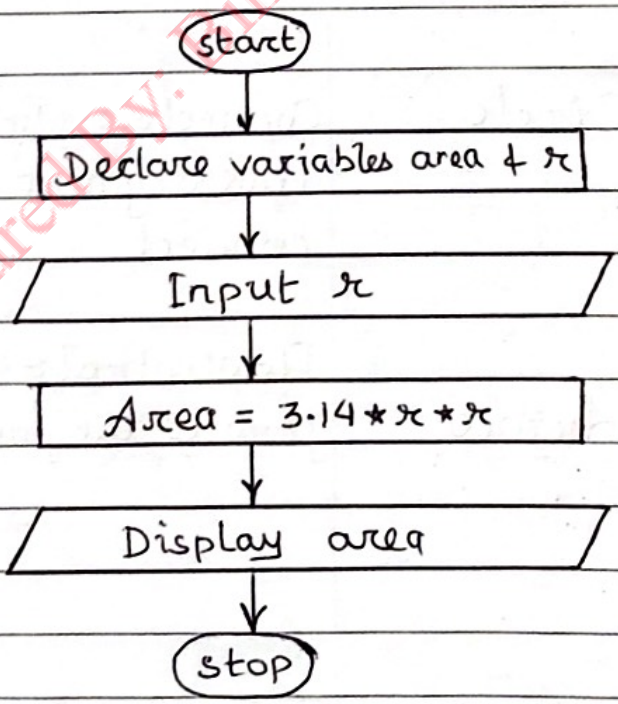
Symbol	Name	Meaning
	Oval	**Start/Stop** :- To represent start & stop of the program.
	Parallelogram	**Input/Output** :- Used to indicate input & output of the data used for display
	Rectangle	**Processing** :- Used for processing of data.
	Diamond	**Decision** :- Used when decision or branching is to be done.
	Circle	**Connector** :- Used to link segment & connect.
	Arrow	**FlowControl** :- indicate flow of the program.

Date
Page

Draw a flowchart to find sum of two numbers.

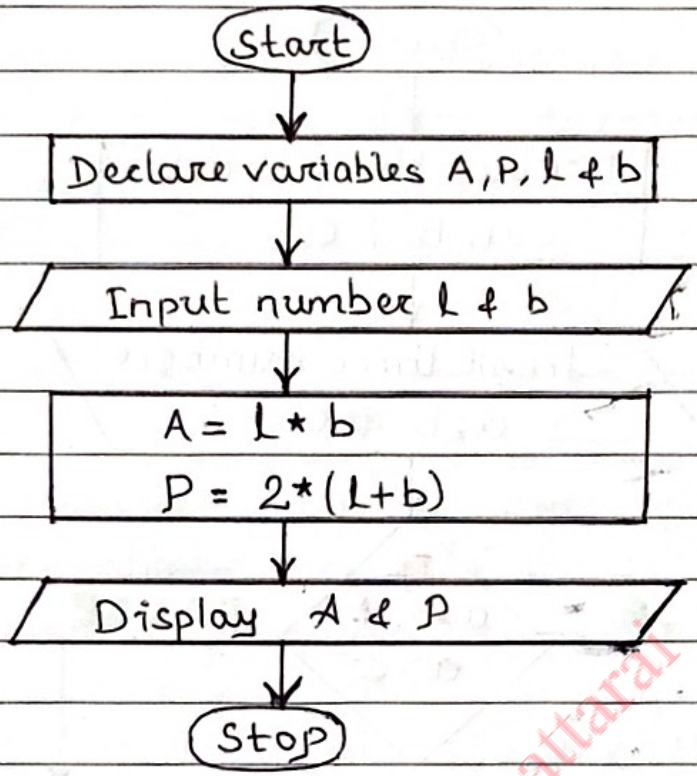


Draw a flowchart to find area of circle

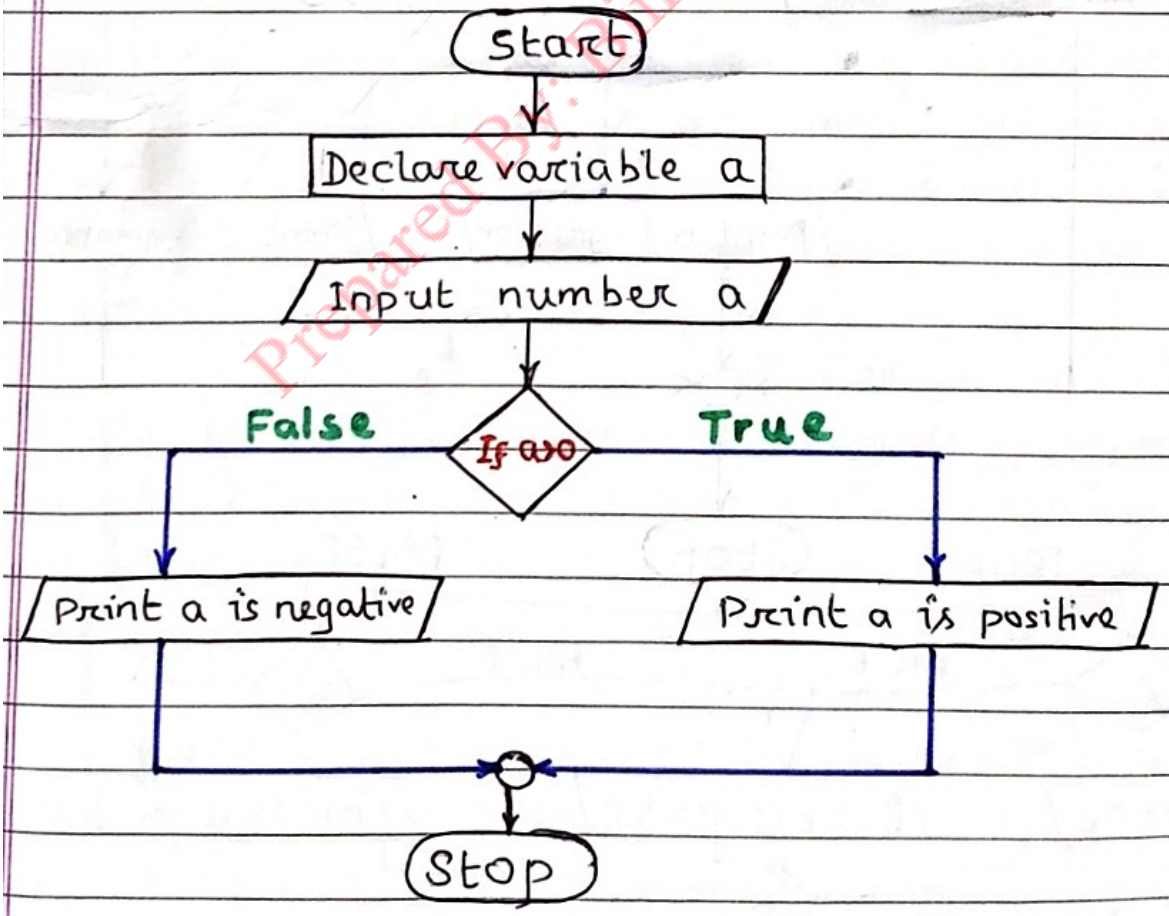


Date
Page

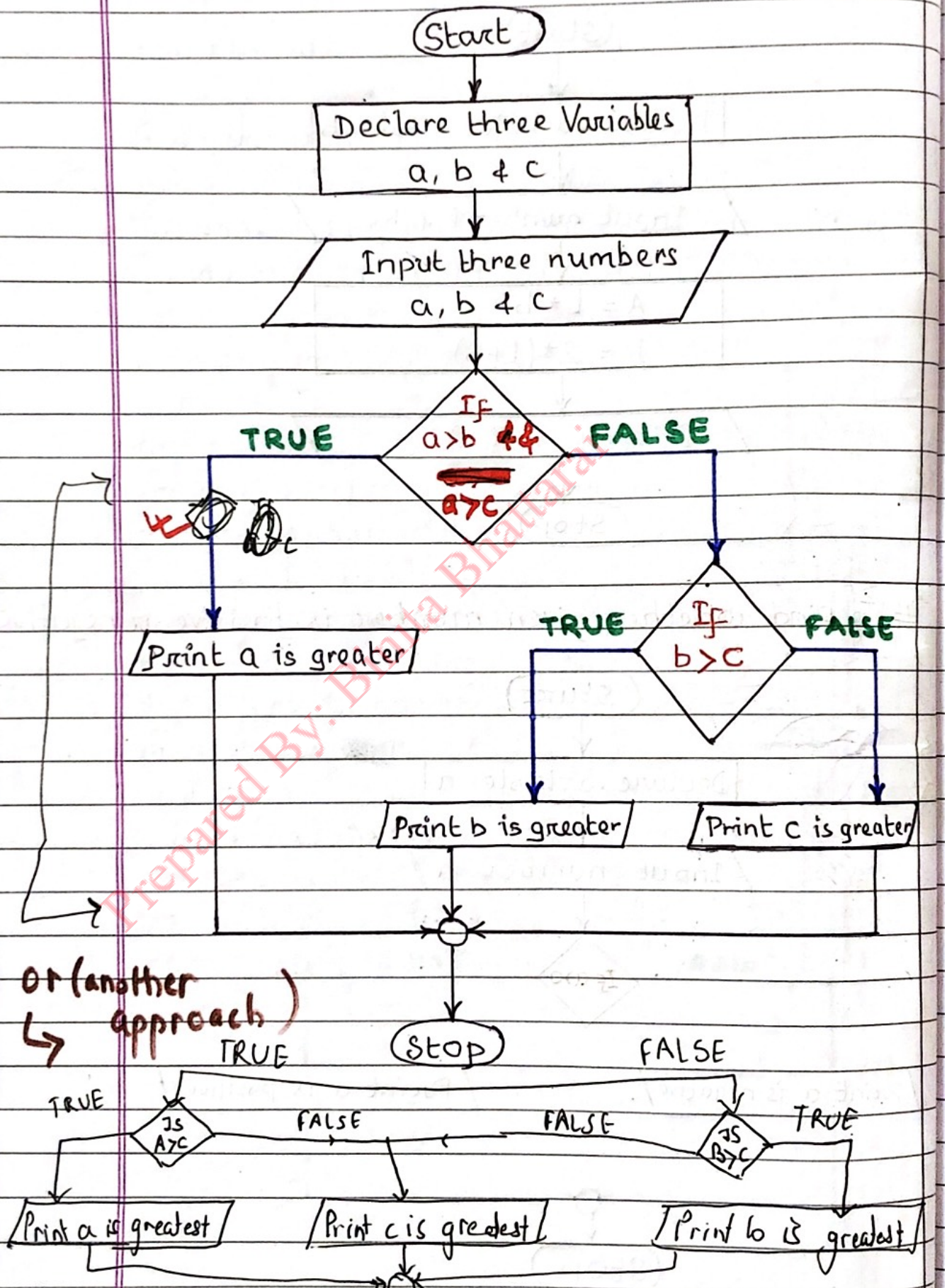
To calculate area & perimeter of a rectangle



To find whether given number is positive or negative



To find maximum of 3 numbers



Date |
Page |

3.) Coding :

((Algorithms & flowcharts are designed for humans. Computer can't understand them. So, it must be written in programming language. Coding can be done in high- as well as low-level Language. The coding must be done in reliable, readable & efficient way. Each programming language has its own syntax. (Syntax is the grammer, structure (ढाँचा) or order of the elements in a language statement). So, programmer should properly follow rules & syntax while coding. For C programming, coding we use text editor of C++ compiler. The statements should be arranged & comments should be added wherever necessary.))

In SDLC, this phase is the third step which comes after Designing phase. Hence, we consider the output of designing phase as the input for this Coding phase. The basic role of this phase is to convert design (algorithms & flowcharts) into code using programming language decided in the designing phase. A well-developed code in this phase can help to reduce the efforts required in testing and maintenance. ((It means, even if a silly (not a big one) mistake is made in coding, it may lead us to put much extra efforts in testing and maintenance.))

In business perspective, the cost for testing efforts and maintenance (सरसर्ती काम गर्न तेति गारो पैन तर पछि टेस्टमा कहा mistake छ या कहाँ कमी कमजोरि छ भनि पत्ता लगाउन धेरै गारो छ) is much higher than coding. Hence, it makes sense to spend sufficient time

on coding phase to produce efficient code. Hence, this is one of the longest phases in SDLC.

Few points we need to take care in this phase:

* Before begining the actual coding, we must spend time on selecting development tool, which will be suitable for debugging, coding, modification and designing needs.

* Before actual coding, some standard (~~कोडिंग~~ ~~Coding~~) should be defined, as multiple developers are going to use the same file for coding.

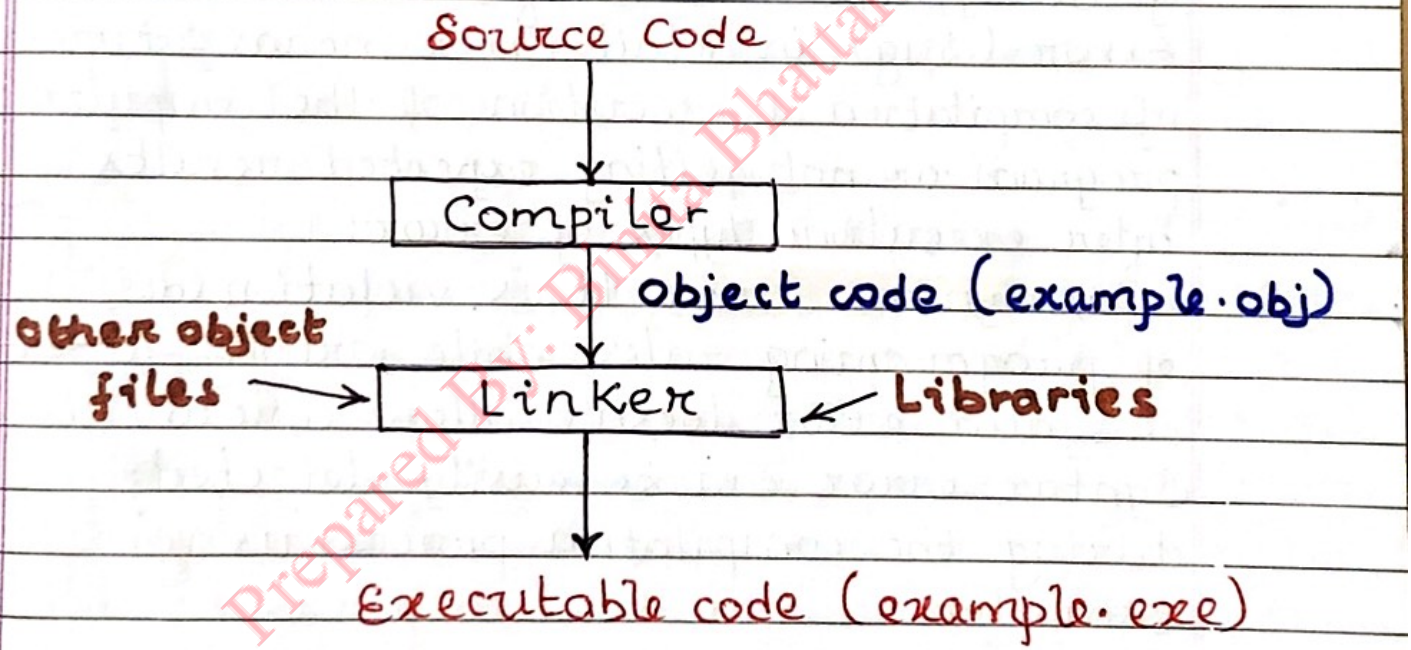
* During development, developer should write appropriate comments so that other developers will come to know the logic behind the code.

* There should be a regular review meeting need to be conducted in this stage. It helps to identify defects in an early stage. It helps to improve product & coding quality.

Date |
Page |

4.) Compilation & Execution

Program written in high level language is not directly understood by the computer. Hence, it must be converted into low level language that is understood by computer. So, compiler does job of translating high level language into low level language. Compiler first checks the syntax error in the program. Logical errors are not checked by compilers, they are shown during run time, i.e. execution. The compilation process is shown below:



Date
Page

5.) Debugging & Testing :

(Debugging is a necessary process in almost every software or hardware development process.) It is a process of finding & reducing the defects in a computer program or electronic hardware. Debugger is a debugging tool helps in identifying coding errors at various software development phases.)

To understand the meaning & significance of debugging, we must first know about Error (bug) in detail. Error means failure of compilation & execution of the computer program or not getting expected results after execution. Types of error:

a) **Syntax Error:** It is violation (दोष) of programming rules while writing it. A syntax error doesn't allow code to run. Syntax error can be easily detected during the compilation process using compiler.

b) **Logical Error:** It occurs when a programmer has applied incorrect logic for solving problem left out a programming procedure. When logic error occurs, program executes but fails to produce a correct result.

Date
Page

c) **Run time Error**: It occurs during execution of program. Stack overflow, divide by zero, floating point error etc are examples of runtime error.

Hence, debugging is the process of fixing (solving) a bug (error) in the software. It can be defined as the identifying, analyzing & removing errors. This activity **begins** after the software fails to execute properly and **concludes (ends)** by solving the problem & successfully testing the software.
It is considered to be extremely complex & tedious task because errors need to be resolved at all stages of debugging.

Testing is the process of verifying & validating that a software or application is bug free, meets the technical requirements as guided by its design and development, & meets the user requirements effectively & efficiently. Testing is generally performed by testing team which repetitively executes program with intent to find error. After testing, list of errors & related information is sent to program developer or development team.

Implementation:
It includes user notification, user training, installation of hardware, installation of software onto computers, & integration of the system into daily work progresses. This phase continues

until the system is operating in production in accordance with the defined user requirements.

Maintenance:

It occurs after the production is in full operation. It can include software upgrades (जस्तै जस्तो mobile app लाई upgrade गर्नु भनेर आउँछ, त्यहि नै हो) repairs & fixes of the software if it breaks.
Software often has design faults. Two major forms of maintenance activities are:

- Corrective maintenance
- Adaptive maintenance

For large systems, removing all the faults before delivery is often extremely difficult & the faults will be discovered long after the system in installed. As these faults are detected, they have to be removed. Maintenance activities related to such fixing of errors fall under Corrective maintenance.

As time passes, there comes change in environment or requirements of the system. This change in environment often changes what is desired from the system. Furthermore, as users work with the

Date
Page

software, sometime later, requirements that were not identified during requirement analysis phase will be uncovered. (देखा पर्नु) Similarly, there might be also changes in the input data (सुरुमा integer numbers मात्र deal गर्थ्यो, अब float or character पनि थप्न पर्ग्यो), the system environments & output formats. All these require modification in software. The maintenance activity related to such modification fall under Adaptive Maintenance.

6.) Documentation:

Every step so far (अहिले सम्म) in the project is documented for future reference & for the improvement of the software in SDLC. For a programmer, reliable documentation is always a must. It helps to keep track of all aspects of an application & it improves the quality of a software product. It main focus are development, maintenance & knowledge transfer to other developers. Successful documentation will make information easily accessible, helps new users learn quickly, simplify the product & helps in reducing the cost used in support. It is usually focused on:

- Server environments
- business rules
- databases/files
- troubleshooting
- application installation
- code deployment.

Date |
Page |

New Chapter

Variables and Datatypes

"Some Useful terminologies":

★ C character set:

The set of characters that are used to form words, numbers & expression in 'C' is called C character set.

Characters in C are:

1. Letters or alphabets
2. Digits
3. Special characters
4. White space

1.) **Letters or alphabets:**

★ Lowercase alphabets: a, b, c, d ..... z

★ Uppercase alphabets: A, B, C, D ..... Z

2.) **Digits:**

All decimal numbers - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

3.) **Special Characters:**

, , &, *, :, !, #, %, $

4) **White Space:**

Blank space, horrizontal tab, new line etc'

★ Constants & Variables:

A constant can be defined as a fixed value, which is used in algebraic expressions & equations. A constant doesn't change over time & has a fixed value.

(For example, the size of a shoe or cloth, mass of a particular object will not change at any point.)

In an algebraic equation, $x+y=8$, 8 is a constant value, & it can't be changed. In a program a constant is actually stored in at any location of memory which holds a single fixed value throughout the execution of that program.

There are various types of constants in 'C'. But it has two major categories:

Primary and Secondary Constants.

* Integer constants
* Real Constants
* Character & string constants

} Primary Constants

* structure
* Array
* Pointer
* union, etc

} Secondary Constants

* **Integer Constants:**

These are the whole numbers without fractional parts. It can be an octal integer, decimal or even a hexadecimal integer. We specify a decimal integer value as a direct integer value, while we specify prefix (उपसर्ग) the octal integer values with '0'. We also prefix the hexadecimal integer values with '0x'.

Date
Page

Examples:

55 → Decimal Integer value
0x5B → Hexadecimal " "
023 → Octal " "

There can also be integer constant of types unsigned or long.
For unsigned, we suffix with 'u'.
For Long, we suffix with 'l'.
For unsigned long, we suffix with 'ul'.

Examples:

50l → Long integer constant
30u → unsigned " "
75ul → unsigned long " "

* Real Constants / Floating Point:

It contains both the parts - decimal as well as integers. Sometimes, it may also contain the exponential part. If so, it is of 2 types:

a) Fractional form : e.g. 2.53, -2.8, 653.25
b) Exponential form: e.g:

2.389 e2 ⇒ 2.389×10^2 ⇒ 238.9
0.5e2
$6.8e^5$ etc.

* Character Constants:

These are symbols that are enclosed in one single quotation (' '). The maximum length of a character quotation is of only one character.

Examples: 'B', '5', '+'

Some character constant are pre-defined in C, known as escape sequences. Each escape sequence consists of a special functionality of its own, & each of these gets prefixed with a '\' symbol. We use these escape sequences in output functions known as 'printf()'.

Rules of Constructing Constants in C

Integer Constants:

- It must have atleast one digit.
- There must be no decimal point.
- Doesn't allow any blanks or commas
- can be both +ve & -ve.
- Allowable range for integer constants is from -32768 to 32767.

Real Constants:

- must consist of more than one digit at least.
- should contain decimal point.
- no blanks & commas
- both +ve & -ve.

Date
Page

String or character constant:

- can be a single digit, a single alphabet, or even a single special symbol that stays enclosed within single quotes.
- string constants get enclosed within double quotes.
-

Backslash Character Constants

Meaning of character	Backslash character
Backspace	\b
New line	\n
Form feed	\f
Horrizontal tab	\t
Carriage return	\r
Single quote	\'
Double quote	\"
Vertical tab	\v
Backslash	\\
Question mark	\?
Alert or Bell	\a

Note: Till now, we complete the "Constant" part of the topic "Constants & Variables". Next we go for "Variable" part of the same topic.

Date
Page

Variables:

A variable is nothing but a name given to a storage area that our programs can manipulate. Each variable in C has a specific type, which determines –

- the size & layout of variable's memory
- the range of values that can be stored within that memory, and
- the set of operations that can be applied to the variable.

In this example of instruction,

$x = 5$

↳ this 'x' is variable name specified to a memory space which now contains data value '5'.

Defining a variable tells the compiler where & how much storage to create for that variable.

Rules for naming Variable:

* A variable name can be any combination of alphabets, digits or underscore.
* First character in a variable can either be an alphabet or underscore (_).
* No comma or blank space is allowed within variable name.
* No other symbol other than underscore is allowed.

Date
Page

* Variable name must not be a keyword.

Keywords: Keywords are the reserved words used in C programming. Their meaning has been already explained by C compiler. Hence, keywords cannot be used as variable name by the programmer.

Examples: int, else, if, float, double, void, while, struct, char, break, etc.

* Variable name is case sensitive, hence, b & B are treated differently.
* Normally, the maximum length of a variable name is 32 characters.
* Variable name must be declared before using it.
* Variable name should be meaningful. Although it is not a hard & fast rule, it is a good practice to use in such a way.
(e.g. cp (for cost price),
vol (for volume).

ASCII: It stands for American Standard Code for International Interchange. ASCII codes are the equivalent numeric form of each character.

A	to	Z	=	65 to 90
a	to	z	=	97 to 122
0	to	9	=	48 to 57

Data Types :

In C programming, data types are declarations for variables. This determines the type and size of data associated with variables. It is a classification that specifies which type of value a variable can have & what type of mathematical, relational or logical operations can be applied to it without causing an error.

Broadly, there are 5 different categories of data types in C, they are:

	Type	Example
1.	Basic	character, integer, floating-point, double
2.	Derived	Array, structure, union etc
3.	Enumerated	enums
4.	Bool type	true or false
5.	void	Empty value

Date
Page

Basic/Primary Data Type:

There are integer-based & floating point based. The memory size of basic data types may change according to 32-bit or 64-bit operating system.

Type	Size (bytes)	Range	Format Specifier
int	atleast 2, usually 4	-32,768 to 32767	%d, %i
char	1	-128 to 127	%c
float	4	3.4E-38 to 3.4E+38	%f
double	8	1.7E-308 to 1.7E+308	%Lf
~~short int~~	~~2 usually~~	~~-32,768 to 32~~	~~%hd~~
short int	2 usually	-32,768 to 32767	%hd
unsigned int	at least 2, usually 4	0 to 65,535	%u
long int	atleast 4, usually 8	-2,147,483,648 to 2,147,483,647	%ld, %li
unsigned long int	atleast 4	0 to 4,294,967,295	%lu
signed char	1	-128 to +127	%c
Unsigned char	1	0 to 255	%c
long double	10	3.4E-4932 to 1.1E+4932	%Lf

* Use of short & long:

If we need to use a large number, we can use a type specifier **long**. e.g.

```
long a;
long b;
Long double c;
```

Here, variables a & b can store integer values. And, c can store a floating-point number.

If we are sure, only a small integer (-32,768 to 32,767)

will be used, then we can (should) use **short** so as to save the memory space.

★ Signed & Unsigned:

There are type modifiers.

- signed - allows for storage of both +ve & -ve numbers.
- unsigned - allows for storage of only +ve numbers.

2. Derived Data Types:

These are also known as user-defined types. This data type is derived out of the primary (basic) data types, thus known as derived data types. But these are capable of storing a set of various values instead of storing one single value. Unions, structures, arrays & enum are some of the most common ones in the C language.

void data type:

It has no values. This is usually used to specify a type of function when it doesn't return any value to calling function.

Date :
Page :

Variable Declaration & Variable Initialization

Naming a proper variable name along with its proper data type for programming use is known as variable declaration. We cannot use any variable in the program without declaring it before.
Declaration does two things:

- It tells the compiler what the variable name is
- It specifies what type of data the variable will hold.

Syntax:

```
int a;
```

We should follow proper naming rules for declaring a variable. In above example, int a means 'a' is a variable of type integer which can only hold the proper integer value.

Variable Initialization:

Giving value to the variable with sign "=" is known as variable initialization. It is of two types:

Static Initialization	Dynamic Initialization
-Variable is assigned a value in advance. This variable then acts as a constant.	- also known as run time initialization. Value is assigned to a variable at run time. Value of this variable can be altered every time the program is being run. User provides value.

```
void main ()
{
   int a;
   a=5 ;
   printf ("a=%d", a);
   getch ();
}
```

```
void main ( )
{
  int a;
  printf ("enter a number");
  scanf ( "%d", &a);
  printf ("value of a=%d", a);
}
```

Syntax of scanf :

scanf ("format specifier", list of address of variable);

Operator :

An operator is a symbol that operates on certain data type or data item. It is used for arithmetic & Logical manipulation. The data items that operators act upon are called Operands.

e.g:

```
c = a + b;
```

Here, '+' is the operator – addition operator and
'a' & 'b' are operands.

Types of Operators :

It can be classified as

A) On the basis of number of operands
B) On the basis of work/utility.

Date :
Page :

	Operators	Type
Unary Operator ←	++ , --	Unary Operator
Binary Operator ←	+, -, *, /, %	Arithmetic Operator
	<, <=, >, >=, ==, !=	Relational Operator
	&, |, <<, >>, ~, ^	Bitwise Operator
	&&, ||, !	Logical Operator
	=, +=, -=, *=, %=	Assignment Operator
Ternary Operator ←	?:	Ternary or Conditional Operator

Here, Unary, Binary & Ternary operators are types of operator defined on the basis of no. of operands.

a) Unary Operator : which operates on only one operand.
b) Binary Operator : " " " two operands.
c) Ternary Operator: " " " three operands.

Ternary operator takes 3 arguments:
* The first argument is the comparison argument.
* The second argument is the result if the condition is true.
* The third argument is the result if the condition is false.

e.g.,

```
(a>b)? a:b
```

first argument ← (a>b)? | second argument ↓ a:b | third argument → b

Date
Page

B) On the basis of work/utility

1.) Arithmatic Operators :

These operators are used to perform arithmetic/mathematical operations on operands.
E.g. +, -, *, /, %, ++, --.

+	plus	Addition
-	minus	Subtraction
*	star	Multiplication
/	slash	Division
%	percentage	Remainder
++	double plus	Increment by 1
--	double minus	Decrement by 1.

Let's understand about ++ & -- in little detail. ++ & -- can be used as **prefix** as well as **postfix**.

- If we use the ++ operator as a prefix like: **++var**, the value of 'var' is incremented by 1 first, & then only it returns the (incremented) value.

- If we use the ++ as a postfix like: **var++**, the original value of 'var' (before increment) is returned first & then 'var' is incremented by 1.

Date
Page

Points to be noted:

- Arithmetic operation between int & int gives int.
- Arithmetic operation between int & float gives float.
- " " " float & float " float.

2.) Relational Operators:

These are used for the comparison of the values of two operands. For example, checking if one operand is equal to the other operand or not, whether an operand is greater than the other operand or not, etc. Some examples are ==, >, <, <= etc.

3.) Logical Operators:

These are used to combine two or more conditions/ constraints or to complement (opposite) the evaluation of the original condition. The result of the operation is a Boolean value, either true or false.
For example, the logical AND represented as '&&' operator in C returns true when both the conditions under consideration are satisfied. Otherwise, it returns false. ∴ (a && b) returns true when both the conditions 'a' & 'b' are true.
e.g.

```
(4!=5)&&(4<5);
```

This returns true

Date
Page

4.) Bitwise Operators:

Note: To understand this, we first must have knowledge about binary number system.

These operators are used to perform bit-level operations on the operands. The operators are first converted to bit-level & then the calculation is performed on the operands.
Mathematical operations such as addition, subtraction, multiplication, etc can be performed at the bit level for faster processing. For example, the bitwise AND operator represented as '&' in C takes two numbers as operands and does AND on every corresponding bits of two numbers. The result of AND is 1 only if both bits are 1 (true).

&	⟶	AND
|	⟶	OR
^	⟶	XOR
<<, >>	⟶	Left, Right Shift

Date |
Page |

5.) Assignment Operators:

These are used to assign value to a variable. The left side operand of this operator is a variable & the right side operand of the assignment operator is a value. The value on the right side must be of the same data type as the variable on the left side otherwise the compiler will raise an error.

Different types of assignment operators are:

a) " = "

This is the simplest assignment operator used to assign the value on the right to the variable on the left. e.g.,

```
a = 10;
b = 20;
ch = 'y';
```

b) " += "

It is the combination of the '+' and '=' operators. This operator first adds the current value of the variable on the left to the value on the right & then assigns the result to the variable on the left. e.g.

(a += b) can be written as (a = a+b).

c) " -= " Same as above but first action is subtraction

(a -= b) can be written as (a=a-b).

Date
Page

d) "*="

- combination of '*' (multiply - first action) & '=' (second action).

(a*= b) can be written as (a=a*b)

e) "/="

e.g.

(a /= b) can be written as (a=a/b)

6.) Other (Special) Operators:

a) sizeof operator:

- It is a compile-time unary operator which can be used to compute the size of its operand.
- Result of 'sizeof' is of the unsigned integer type.

Example:

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

void main()
{
  int a;
  char b;
  double d;
  float c;
  long double e;
```

```
printf("Size of integer is %d", sizeof(a));
printf("Size of float is %d", sizeof(c));
printf("Size of char is %d", sizeof(b));
printf("Size of double is %d", sizeof(d));
printf("Size of long double is %d", sizeof(e));

getch();
} // closing of 'main' function.
```

OUTPUT:

```
Size of integer is 4
Size of float is 4
Size of char is 1
Size of double is 8
Size of long double is 10
```

b) comma operator:

In C, comma can be used as **operator** as well as **separator**.

1) Comma as an operator:

It is a binary operator that evaluates its first operand & discards (त्याग) the result, it then evaluates the second operand & returns this value. e.g.,

```
int i = (5, 10);
```

here, 5 is discarded, & 10 is assigned to i.

```
int j = (f1(), f2());
```

here, function f1() is called (evaluated) first & then f2().
The returned value of f2() is assigned to j.

c) Conditional Operator

Already explained!

Expression1 ? Expression2 : Expression3

Example code:

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

void main ()
{
  int a, b;
  printf ("Enter two numbers");
  scanf ("%d %d", &a, &b);

  (a>b)? printf ("%d is greater", a) : printf ("%d is greater", b);

  getch();
}
```

d) dot (.) and arrow (→) Operators

These are used to reference individual members of classes, structures, & unions. The dot operator is applied to the actual object.

The arrow operator is used with a pointer to an object.

Date
Page

e) Cast operator

- They convert one data type to another.

Example: (int) 2.2 would return 2.

- A cast is a special operator that forces one data type to be converted into another.
- Format of code:

(type) expression

f) &, * Operators

The pointer operator "&" returns the address of a variable. e.g.

&a will give the actual address of the variable.

The pointer operator "*" is a pointer to a variable. e.g.

*var will point to a variable var.

Lets see some examples of ++ & --.

int a;
a = 5;
a++;
then a is 6.
	int a;
	a = 5;
	++a;
	then a is 6
		int a, b;
		a = 5;
		b = a++;
		then, b is 5 & a is 6

int a;
a = 5;
a--;
then a is 4
	int a;
	a = 5;
	--a;
	then a is 4
		int a, b;
		a = 5;
		b = ++a;
		then, b is 6 & a is 6.

Let's understand clearly (step by step):

Case of post increment:

```
int a = 5;
b = a++;
```

→ This line of code means multiple steps: ~~1st~~

- 1st step is - putting the value of 'a' in 'b' [not increment in a]

  i.e. $b = a; \Rightarrow \underline{b = 5}$

- 2nd step is - increment in a by 1

  i.e. $a = a + 1;$

  $a = 5 + 1;$

  $\underline{a = 6;}$

∴ It finally results with

b = 5

a = 6

Case of pre-increment:

```
int a = 5;
b = ++a;
```

- 1st step is - increment in a by 1.

  i.e, $a = a + 1$

  $a = 5 + 1$

  $\underline{a = 6}$

- 2nd step is - putting 'a' in 'b'.

  i.e, $b = a$

  $\underline{b = 6}$

∴ final result :-

a = 6

b = 6

Date |
Page |

More examples:

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

void main()
{
 int a = 10;
 printf("a = %d", a);
 printf("a = %d", ++a);
 printf("a = %d", a++);
 printf("a = %d", a);

 getch();

}
```

→ simply printing value of a which is 10. (a = 10)

→ ++a makes a ← 11, & then it is printed. (a = 11)

→ previous value of a is 11 which gets printed here. After displaying a = 11, then value of a becomes 12. (a = 12)

→ simply prints the final value of a, i.e a = 12.

OUTPUT:

a = 10
a = 11
a = 11
a = 12