Algorithm:

- It is any well-defined computational preocedure that takes some value, or set of values, as input and produces some value, or set of values as output.
- **Example:**

  Input: A sequence of n numbers $< a_1, a_2, \ldots, a_n >$

  Output: A permutation (reordering) $<a_1', a_2', \ldots, an'>$ of the input sequence such that $a_1' \leq a_2' \leq \ldots \leq a_n'$
- It is a tool for solving a well-specified computational problem.
- It is a part of the plan for the computer program.
- It is an effective method for solving a problem expressed as a finite sequence of instructions.

Features of an Algorithm:

- **Finiteness:** It must terminate.
- **Definiteness:** The specifications of the actions should be unambiguous.
- **Effectiveness:** The operations used should be basic & should be performed exactly in a fixed duration of time.
- **Input:** It should have one or more inputs.
- **Output:** It should have one or more outputs.

Example 1 (Checking a no is even or odd)

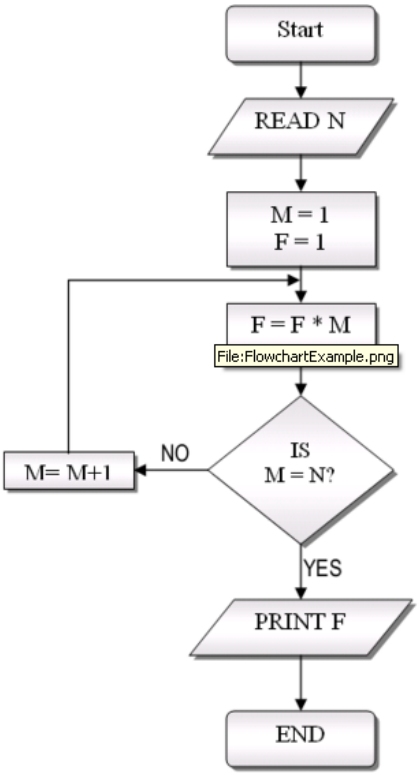
1. START
2. PRINT "ENTER THE NUMBER"
3. INPUT N
4. Q=N/2 (Integer division)
5. R=N-Q*2
6. IF R !=0 THEN
7. PRINT "N IS ODD"
8. ELSE
9. PRINT "N IS EVEN"
10. STOP

Example 2 (Finding largest among three numbers)

1. START
2. PRINT "ENTER THREE NUMBERS"
3. INPUT A, B, C
4. IF A>=B AND B>=C THEN
5. PRINT A
6. IF B>=C AND C>=A THEN
7. PRINT B
8. ELSE
9. PRINT C
10. STOP

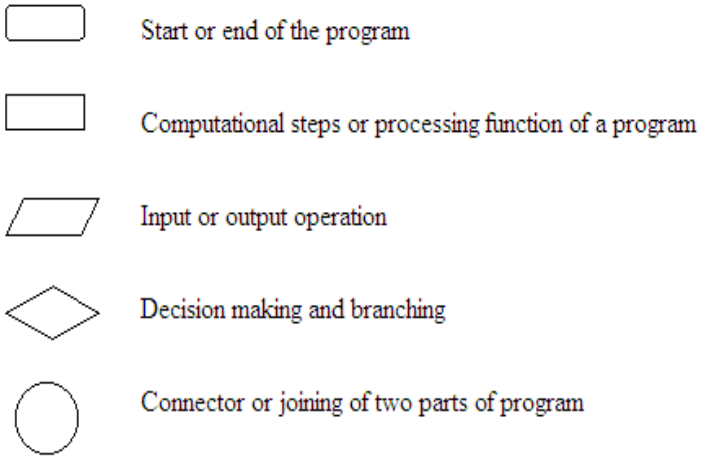
Flowchart:

- It is a graphical representation of an algorithm.
- It is one of the tool used to design an algorithm.
- Flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields.
- **Example:** Finding factorial of a no:-------→



Flowchart Symbols:

- **Start and end symbols:** Represented as circles, ovals or rounded rectangles, usually containing the word "Start" or "End", or another phrase signaling the start or end of a process.
- **Input/Output:** Represented as a parallelogram. Examples: Get X from the user; display X.
- **Processing steps:** Represented as rectangles. Examples: "Add 1 to X"; "replace identified part"; "save changes" or similar.
- **Conditional or decision:** Represented as a diamond (rhombus). These typically contain a Yes/No question or True/False test. This symbol is unique in that it has two arrows coming out of it, usually from the bottom point and right point, one corresponding to Yes or True, and one corresponding to No or False. The arrows should always be labeled. More than two arrows can be used, but this is normally a clear indicator that a complex decision is being taken, in which case it may need to be broken-down further, or replaced with the "pre-defined process" symbol.
- **Arrows:** Showing what's called "flow of control" in computer science. An arrow coming from one symbol and ending at another symbol represents that control passes to the symbol the arrow points to.
- **Connector:** Connects two parts of a flowchart.



Evolution of Programming Languages (Generation of Programming Languages):

- **First–generation programming language/Machine-level programming language/Binary language:**
  - All instructions are in binary (0 or 1) form.
  - Computer understands only this language directly.

- It depends on machine architecture.
- Execution is faster and efficient.
- Difficult for the user to learn and write programs using it.
- Difficult to edit errors if error occurs.

➢ **Second-generation programming language/Assembly-level language**
- Some symbolic instructions are used.
- Needs to be translated to binary language with the help of a translator known as assembler.
- Easier for the user to understand than binary language.
- It is also machine dependant.
- Mainly used for system side programming and in extremely intensive processing such as games, video editing, graphics etc.
- **Example:** MOV AL, 61h
  This instruction means: Move (really a copy) the hexadecimal value '61' into the processor register known as "AL". (The h-suffix means hexadecimal or = 97 in decimal)

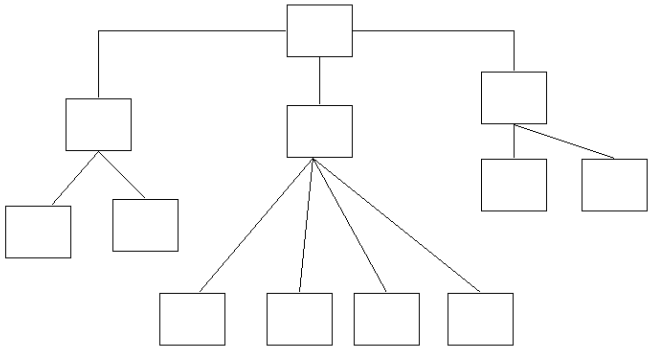
➢ **Third-generation programming language/High-level Language**
- It is a refinement of a second-generation programming language.
- The third generation brought refinements to make the languages more programmer-friendly. It is just like English language.
- Easy to read and write by using this language.
- Most 3GLs support structured programming.
- **Example:** Fortran, ALGOL, and COBOL are early examples.
  C, C++, C#, Java, and Python, are recent examples.

➢ **Fourth-generation programming language**
- A fourth-generation programming language (1970s-1990) is a programming language designed with a specific purpose in mind, such as the development of commercial business software.
- All 4GLs are designed to reduce programming effort, the time it takes to develop software, and the cost of software development.
- They are not always successful in this task, sometimes resulting in inelegant and unmaintainable code.
- Fourth-generation languages are a subset of domain-specific programming languages (DSLs).
- **Example:** FoxPro, Ingres 4GL, SQL, MATLAB

➢ **Fifth-generation programming language**
- A fifth-generation programming language is a programming language based around solving problems using constraints given to the program, rather than using an algorithm written by a programmer.
- Most constraint-based and logic programming languages and some declarative languages are fifth-generation languages.
- While fourth-generation programming languages are designed to build specific programs, fifth-generation languages are designed to make the computer solve a given problem without the programmer.
- Mainly used in artificial intelligence research
- **Example:** Prolog, OPS5, and Mercury.

Structured Programming:

- The code is broken into modules.
- Each module has one and only one function.
- There is one and only one logical entry and exit to each module.
- Modules are loosely coupled.
- There should be no GOTO statement.



Introduction to C

- It is a middle level language (in between 2GL and 3GL) developed by Denis Ritchie in the year 1972 at AT & T's Bell Lab.
- Why it is middle level language because it allows manipulation of bits, bytes and addresses. Mainly it is used for system side programming.
- It is a structured language.

Structure of a C Program

```
<Header files>
<Declaration of global variables>
main( )
{
        <Declaration of local variables>
        [Input part]
        Calculation part
        Display part
}
<sub program – function blocks>
```

Example:

```
/*Addition of two number*/
#include<stdio.h>
main()
{
        int n1,n2,sum;
        printf("Enter two nos:");
        scanf("%d%d", &n1,&n2);
        sum=n1+n2;
        printf("Sum of two nos=%d",sum);
}
```

Explanation:

- **Comment line:** Non-executable lines.
- **Header files:** Header files commonly contain forward declarations of classes, subroutines, variables, and other identifiers.

- **Global variable:** Variables which can be used anywhere in the program. They are declared before main( ) function
- **main():** It is a special function from where the execution starts. Every program must have a main() function. Here main doesn't have any arguments.
- **{**: Beginning of main function.
- **}**: End of main function.
- **Function body:** Lines between the '{'and '}'.
- **Local variable:** Can be used in the same function. n1, n2, sum are global variables.
- **Variable declaration:** All variables should be declared to indicate what type of data that variable could store.
- **Library Function:** printf(), scanf().
- **Input:** Input can be given by some predefined library function like scanf(), getchar() etc.
- **Calculation:** Normally calculation is carried out after input.
- **Output:** Normally output is printed at the end of the function by using some library function like printf().
- **User defined Function:** Defined after defination of main() function.
- **Semicolon(;):** Every statement of C ends with a semicolon.

Compilation Process:

- **Source code:** Program written in any programming language is known as source code. Source code is written in an editor.
- **Compilation:** It is the process of translating source program into computer understandable form (binary form). A system software known as **compiler** does this compilation. In this stage, the compiler first checks the syntax of the program. If everything (syntax and semantics) is all right, then it translates the source code and stores it in another file. If there is some error, it shows the error message. In case of error, we have to do the correction and again compile it.
- **Object code:** After translation of the source code, the file created is known as object code. In Linux Operating System if the source file is "fact.c" then the object code would be "fact.o".
- **Linking:** It is the process of putting together other program files and functions that are required by the program. **Linker** is software, which does this. For example if the program uses a mathematical function "pow()" then the object code of this function should be brought from the math library.
- **Executable code:** The compiled and linked code is known as executable code. In Linux the executable code is store automatically in a file known as "a.out".
- **Loading: Loader** is a program, which accepts the object program, prepares it for execution & initiates the execution. It allocates the space in the main memory for loading the program.
- **Execution:** To execute the executable code we have to just type the name of the executable file "a.out". During this stage it asks for the input. We provide the input and it gives us the output.

Steps in Learning English:

- Alphabets------→Words-------→Statements------→Paragraph

Steps in Learning C:

- Character Set (Alphabets, Digits, Symbols)------→Constants, Variables, Keywords-----→Instruction------→Program.