## 1.1 Algorithm and Flowchart

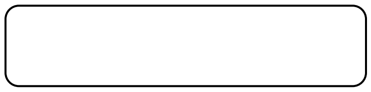
### Algorithm:

- ❖ The step by step procedure for solving a problem is known as algorithm.
- ❖ A sequence of precise and unambiguous instructions for solving a problem is known as algorithm.

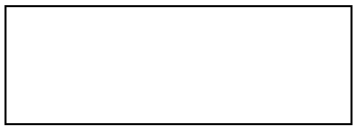
### Flowchart:

- ❖ A flowchart is a diagrammatic representation that illustrates the sequence of operations to be performed to arrive at the solution.
- ❖ A pictorial representation, which uses predefined symbols, to describe either the logic of a computer program (program flowchart), or the data flow and processing step of a system.
- ❖ A graphical representation of an algorithm is known as flowchart.

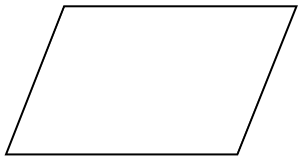
In drawing flowchart, certain convention has come into use.


START or END

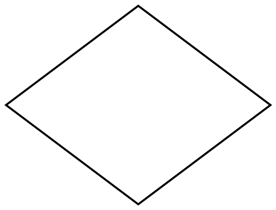
Indicates START or END of the program


PROCESSING

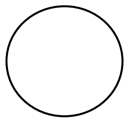
Indicates Computational steps (arithmetic operations)


INPUT or OUTPUT

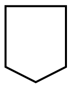
Indicates an operation of reading or writing


DECISION BOX

Indicates a Decision Point. A test is performed and the program flow continues on each outgoing path conditional to the answer to the test


ON PAGE CONNECTOR

Connector or joining of two parts of program. On Page Connector is used to connect two parts of the flowchart on the same page.



OFF PAGE CONNECTOR

Off Page Connector is used to connect two parts of the flowchart on the different page.

$\longrightarrow$ FLOW LINES

Indicates flow of the data

## Advantages and Disadvantages of Flowchart:

**Advantages:**
**There are following advantages of flowchart:**

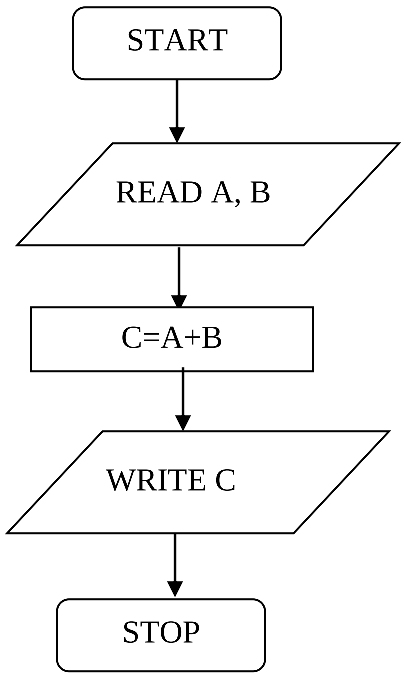
- **Communication**
  Flowchart are better way of communicating the logic of the system
- **Effective analysis**
  With the help of flowchart, a problem can be analyzed in a more effective way.
- **Efficient Coding**
  The flowchart act as a guide or blue print during the system analysis and program development phase.
- **Proper debugging**
  The flowchart helps in debugging process. Debugging means finding and correcting problems or mistakes or errors.
- **Proper documentation**
  A flowchart serves as a good program documentation which is needed for future purpose.
- **Efficient program maintenance**
  Using the flowchart, we can easily manage the program or a system,

**Disadvantages:**
**There are following disadvantages of flowchart:**

- **Complex logic**
  Sometimes the program logic is complicated in that case flowchart becomes more complex.
- **Alteration and modification**
  If changes are required then the flowchart may require to redraw completely.
- **Reproduction**
  As the flowchart symbols are not to be typed, reproduction of flowchart may require to draw again.

## Difference between Algorithm and Flowchart

| Algorithm | Flowchart |
|---|---|
| Algorithm is step by step procedure for solving a problem. | Flowchart is graphical representation of an algorithm. |
| In algorithm, there are steps and statements. | In flowchart, there are predefined symbols. |
| Algorithm is difficult to understand. | Flowchart is easy to understand. |
| Eg: | Eg. |

| | |
|---|---|
| Step 1: START<br>Step 2: READ A and B<br>Step 3: CALCULATE C=A+B<br>Step 4: WRITE C<br>Step 5: STOP |  |

## 1.2 Structured Programming

C is called a structured programming language because to solve a large problem, C programming language divides the problem into smaller modules called functions or procedures each of which handles a particular responsibility. The program which solves the entire problem is a collection of such functions.
Structured programming (sometimes known as modular programming) is a subset of procedural programming that enforces a logical structure on the program being written to make it more efficient and easier to understand and modify.

Here is an example of Matrix addition program, which is divided into these sub procedures - input matrix, display matrix, add matrix, save result matrix to file. Here is a pictorial structural view of the program.

Another good example is calculate student's grade. Program is divided into these sub modules - input student marks, get student record, update student record, display student record, calculate grade. Here is a structural view of the program.

One major drawback of C language is that similar functions cannot be grouped inside a module or class. Also functions cannot be associated to a type or structure. Thus data and functions cannot be bound together. C++ language overcomes these problems by introducing object oriented functionality in its programming capabilities.

**Advantages**

- C structured programming is simple and easy to understand and implement.
- It is well sited for small size implementation. However this is not restricted. A good design can extend it to large size implementation.
- Programmers do not require to know complex design concepts to start a new program.

**Disadvantages**

- Data and methods and not be bind together in a module.
- Polymorphism and inheritance are not available.
- Complex design and full object oriented design cannot be implemented.
- Programmers generally prefer object oriented programming language over structured programming language when implementing a complex gaming applications or front end business applications.

## 1.3 Concepts of Compiler, Interpreter, Editor, Debugging & Testing

### Compiler:

**A compiler** checks the entire user-written program (known as the *source program*) and, if error-free, produces a complete program in machine language (known as *object program*).

### Interpreter:

**An interpreter** does a similar job but in a different style. The interpreter translates one statement at a time and, if error-free, executes. This continues till the last statement. Thus an interpreter translates and executes the first instruction before it goes to the second, while a compiler translates the whole program before execution.

### Difference between Compiler and Interpreter:

| COMPILER | INTERPRETER |
|---|---|
| Scans the entire program and translates it as a whole into machine code. | Translates program one statement at a time. |
| It takes large amount of time to analyze the source code but the overall execution time is comparatively faster. | It takes less amount of time to analyze the source code but the overall execution time is slower. |
| Generates intermediate object code which further requires linking, hence requires more memory. | No intermediate object code is generated, hence are memory efficient. |
| It generates the error message only after scanning the whole program. Hence debugging is comparatively hard. | Continues translating the program until the first error is met, in which case it stops. Hence debugging is easy. |
| Programming language like C, C++ use compilers. | Programming language like Python, Ruby use interpreters. |

### Editor:

A source code editor is a text editor program designed specifically for editing source code of computer programs. It may be a standalone application or it may be built into an integrated development environment (IDE) or web browser.

### Debugging:

Debugging is the process of detecting and correcting the syntax errors in a program. This consists of two stages:

1. Systematic desk checking
2. Translator system checking

1. *Desk Checking*

When a program is completed, a thorough desk check will eliminate many needless errors. This check may include are view by a second person who is an expert in computer programming.

2. *Translator Checking*

'C' system has certain rules which must be followed, such as rules for naming variable and for using punctuations. If rules are not followed or if the instructions are not coded properly, the translator system will detect these errors during the translation process.

## Testing:

Testing is a process of identifying defects, where a defect is any different between actual and expected result.

## Errors:

Any mistake in program is known as Error.

## Types of Error:

There are following categories of errors:
1. Syntax errors
2. Logical errors
3. Data errors

### 1. Syntax Errors

Syntax Errors in computer programs that typically involve incorrect punctuation, incorrect word sequence, undefined terms, or misuse of terms.

These errors are automatically detected, and pointed out by language processor.
For example:

```
int   X(10)
pintf("Syntax Error");
clrser();
int x=y+/2;
```

All have syntax errors. All such errors should be corrected before running the program.

### 2. Logical Errors

Errors like taking a wrong path, failure to consider a particular condition, incorrect order of evaluation of expressions, incorrect order of evaluation of statements, etc., belong to this category. These errors are primarily due to poor understanding of the problem, incorrect translation of algorithm into program and abuse of brackets and operators.
For example:
Writing if A < B ….. in place of IF A > B
Writing if X = Y ….. when X and Y are real numbers.
Are valid but might produce incorrect results. Sometimes such errors might create infinite loops.

**3. Data Errors**

Data errors are the most common errors. This may be due to wrong typing of constants, mismatching of variable list and data order, and failure to assume the range of values a variable might take in the program.
For example:
asking the computer to divide a quantity by zero, or to compute the square root of a negative number are due to failure to anticipate the ranges of data.

## 1.4 Character Set

A set of characters that are allowed to use in computer program is known as Character Set.
The following character sets are available in 'C' programming Language.
(1) Alphabets
(2) Numbers or Digits
(3) Special Characters
(4) White Space Characters

**(1) Alphabets:**
A,B,C,D….Z
A,b,c,d….z

**(2) Numbers or Digits**
0,1,2,3,4….9

**(3) Special Characters**

| | | |
|---|---|---|
| - Minus | + plus | * asterisk |
| / forward slash | \ back slash | ( Left parenthesis |
| ) right parenthesis | , comma | ; semicolon |
| : colon | $ dollar sign | . dot operator |
| > greater than | < less than | = equal to |
| " quotation mark | ? question mark | ! exclamation mark |
| \| vertical bar | ~ tilde | _ underscore |
| ^ caret | [ opening square bracket | ] closing square bracket |
| { opening curly bracket | } closing curly bracket | # has sign |
| @ at symbol | & ampersand | ' apostrophe |
| % percentage sign | | |

**(4) White Space characters**
Blank Space
Horizontal Tab
New Line
Carriage Return
Form Feed

**Trigraph Characters:**

Many non-english keyboards do not support all the characters like( # , [ , ] , { , } , | , \ , ^ , ~ ).

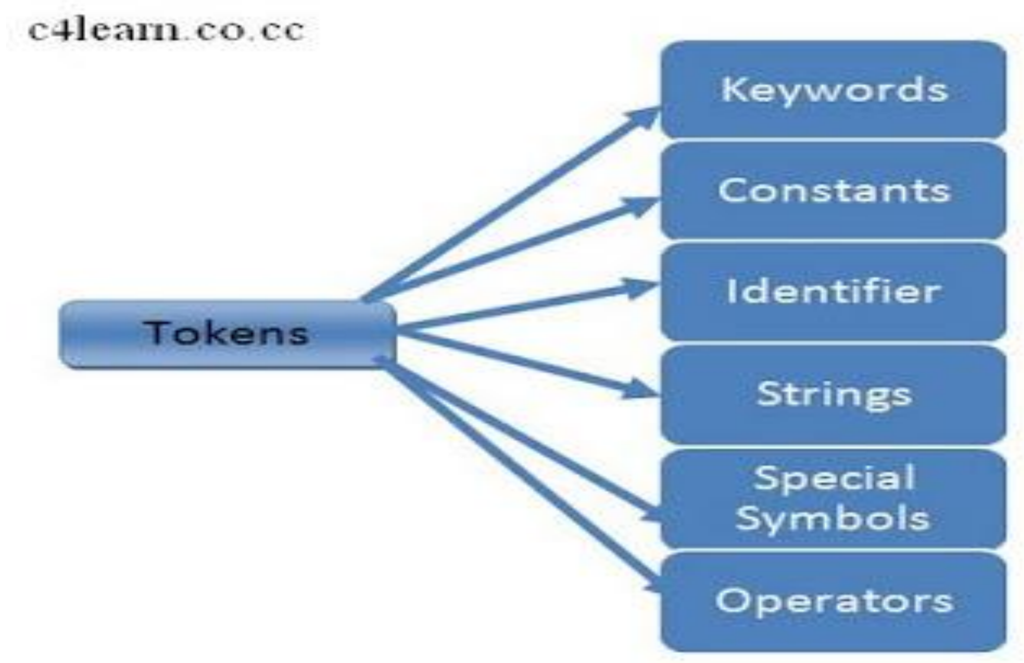
'C' introduces the concept of "trigraph" sequence to provide a way to enter certain characters that are not available.

| Trigraph sequence | Translation |
|---|---|
| ??= | # number sign |
| ??( | [ left bracket |
| ??) | ] right bracket |
| ??< | { left brace |
| ??> | } right brace |
| ??! | \| vertical bar |
| ??/ | \ back slash |
| ??- | ~ tilde |

## 1.5 Identifiers, Key words, Data types

**Tokens:**

Smallest individual element or unit 'C' is known as Token.



1. **Keywords:**

   Keywords are predefined, reserved words used in programming that have special meanings to the compiler.

Keywords in C Language

| | | | |
|---|---|---|---|
| auto | double | int | struct |
| break | else | long | switch |
| case | enum | register | typedef |

| | | | |
|---|---|---|---|
| char | extern | return | union |
| continue | for | signed | void |
| do | if | static | while |
| default | goto | sizeof | volatile |
| const | float | short | unsigned |

2. **Constants:**
   Constants are those whose values will not change during program executions. To declare constant 'const' keyword will be used.
   Eg:
```
#include<stdio.h>
#include<conio.h>
void main()
{
        const int a=10;
        const float pi=3.14;
        clrscr();
        a=20;
        pi=31.4;
        printf("%d",a);
        printf("%f",pi);
        getch();
}
```
3. **Identifiers:**
   Identifiers refers to name of constants, variables, functions, structures.
   **Rules:**
   1) First character must ne alphabets or underscore.
   2) Must be consists letters, digits and underscore.
   3) Only 31 characters are significant.
   4) Cannot use keyword.
   5) Must not contain white space.

**Example**:
stud
tot
stus_name

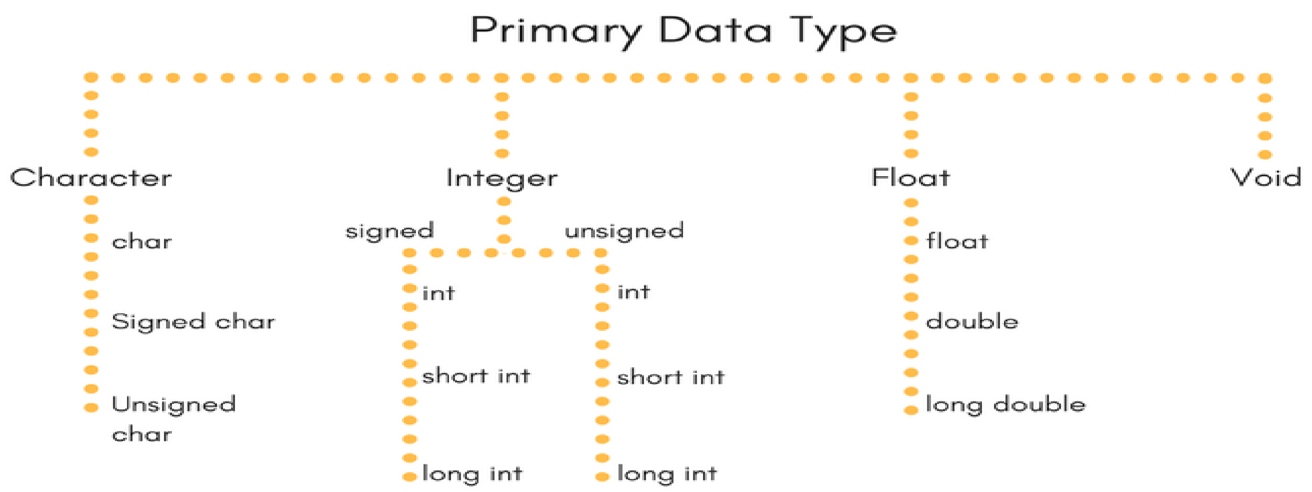
4. **Strings:**
   Strings are set of characters which can be alphabets, digits or special characters.
   Example:
   "Gujarat"
   "MCA- THE Great Class"
   "gujaratuniversity.ac.in"

5. **Special Symbols:**
   ‘C’ is having large number of Special symbols or special characters.
   Eg: $,#,@,!,^,& etc…

6. **Operators:**
   An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators.

**Data types:**

Data types simply refers to the type and size of data associated with variables.



**Integer type**

Integers are used to store whole numbers.

**Size and range of Integer type on 16-bit machine:**

| **Type** | **Size(bytes)** | **Range** |
|---|---|---|
| int or signed int | 2 | -32,768 to 32767 |
| unsigned int | 2 | 0 to 65535 |
| short int or signed short int | 1 | -128 to 127 |
| unsigned short int | 1 | 0 to 255 |
| long int or signed long int | 4 | -2,147,483,648 to 2,147,483,647 |
| unsigned long int | 4 | 0 to 4,294,967,295 |

**Floating point type**

Floating types are used to store real numbers.

**Size and range of Integer type on 16-bit machine**

| **Type** | **Size(bytes)** | **Range** |
|---|---|---|
| Float | 4 | 3.4E-38 to 3.4E+38 |
| double | 8 | 1.7E-308 to 1.7E+308 |
| long double | 10 | 3.4E-4932 to 1.1E+4932 |

**Character type**

Character types are used to store characters' value.

**Size and range of Integer type on 16-bit machine**

| **Type** | **Size(bytes)** | **Range** |
|---|---|---|
| char or signed char | 1 | -128 to 127 |
| unsigned char | 1 | 0 to 255 |

**void type**

void type means no value. This is usually used to specify the type of functions which returns nothing. We will get acquainted to this datatype as we start learning more advanced topics in C language, like functions, pointers etc.

**Other Data Types:**

(a) Pointer
(b) Array
(c) Structure
(d) Union
(e) Enum

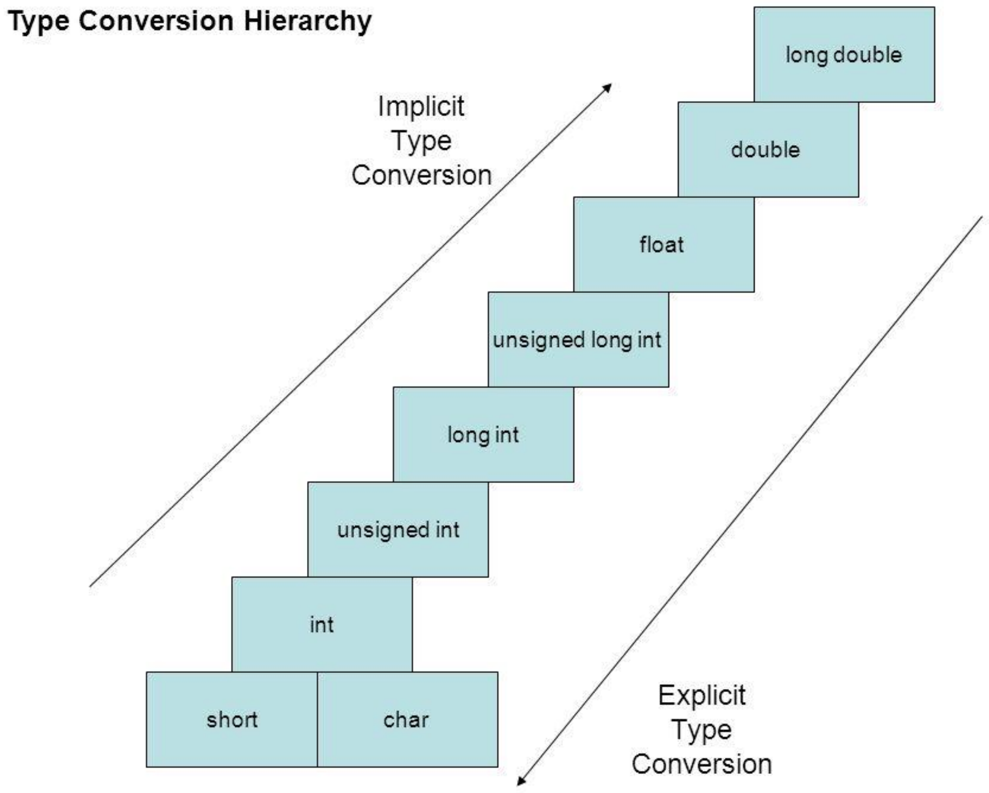
## Type Casting OR Type Conversion:

Type casting is a way to convert a variable from one data type to another data type. For example, if you want to store a long value into a simple integer then you can typecast long to int. You can convert values from one type to another explicitly using the cast operator.

There are two forms of Type Casting:

1. Implicit type casting
2. Explicit type casting

1. Implicit type casting
   It is also known as Automatic Type Casting.
   'C' permits to convert low range data type to high range data types.



```
Eg.
#include<stdio.h>
#include<stdio.h>
void main()
{
        int a =10;
        float b;
        clrscr();
        b=a+100; // Implicit type casting
        printf("%f",b);
        getch();
}
Output:
110.000000
```

2. Explicit type casting

Explicit Type casting means we force 'C' to change the data types.
When we perform explicit type casting, the variable may loss some value.
Syntax:
(type name) expression;
Here type name is any data type like int, float, double…
Eg:

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

```
#include<stdio.h>
void main()
{
    int a;
    clrscr();
    a=(int)7.77;// explicit type casting
    printf("%d",a);
    getch();
}
```

Output:

```
7
```

## Data Overflow and underflow:

Assigning a value which is more than its upper limit is called overflow and less than its lower limit is called underflow.

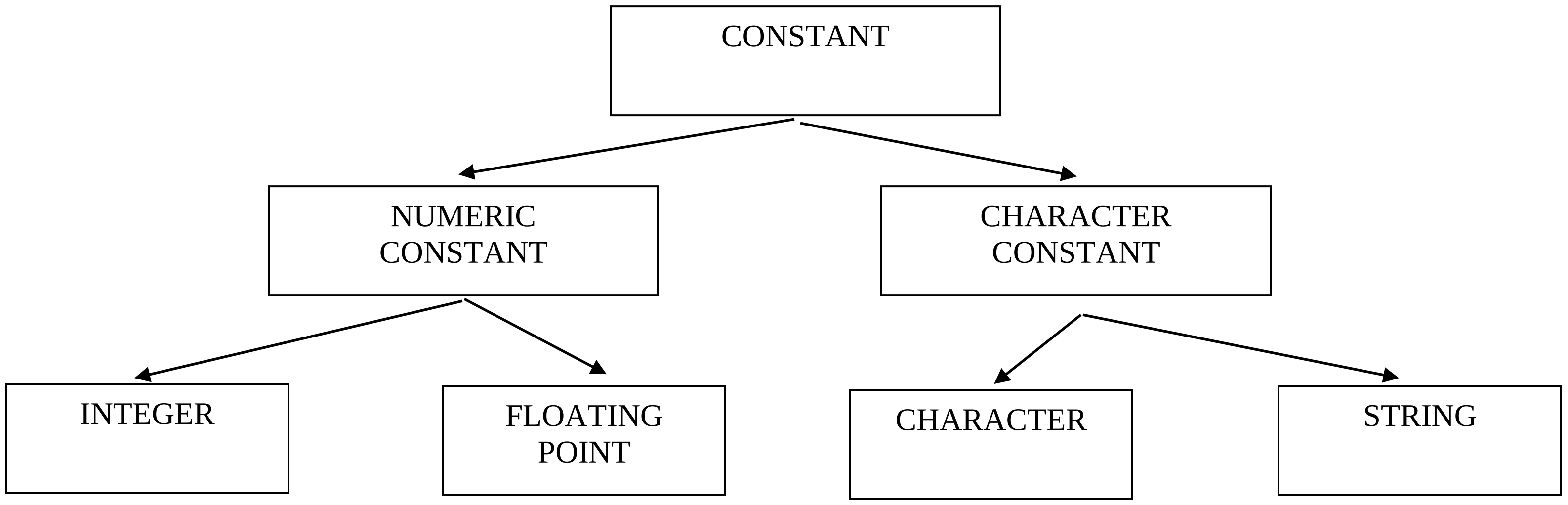
In case of integer types overflow results wrapping towards negative side and underflow results wrapping towards positive.

In case of floating point types overflow results +INF and underflow results -INF

## 1.6 Constants and Variables – Needs & Definition

**Constants:**

Constants are those whose value will not change during the program execution.
To declare constant 'const' keyword will be used.



1) Numeric Constant
   Numeric constant is used to store numeric data.
   The numeric constant can be : 1,20000, 345, 99,-3.14, 7.777 etc
   There are two types of numeric constants:
   a. **Integer**
      An integer constant refers to a sequence of digits.
      There are three types of integer:
      I. Decimal integer
         Decimal integer consists a set of digits from 0 to 9.
         Eg: 123, 987, -88, +98
      II. Octal integer
         An Octal integer consist any number leading with 0.
      III. Hexa-decimal
         An hexa decimal integer consist any number leading with 0x or 0X.
   b. **Floating point**
      Real constants are used to declare floating point values like: price, percentage, temperature, heights, distance etc...
      Examples: 7.7777, -77.77, 0.0005, 1.5e2

**Program:**

```
#include<stdio.h>
#include<stdio.h>
void main()
{
        const int a=10;
        const float pi=3.14;
        clrscr();
        a=100;
        pi=31.4;
        printf("%d",a);
        printf("%f",pi);
        getch();
}
```

2) Character Constant
   There are two category of Character Constants:
   a. Character
   b. String

   a. **Character**:

   Character constant represents a single character within a single quote(' ').
   Example: 'A', 'Z', '*', ' ', '5'

   b. **String**:

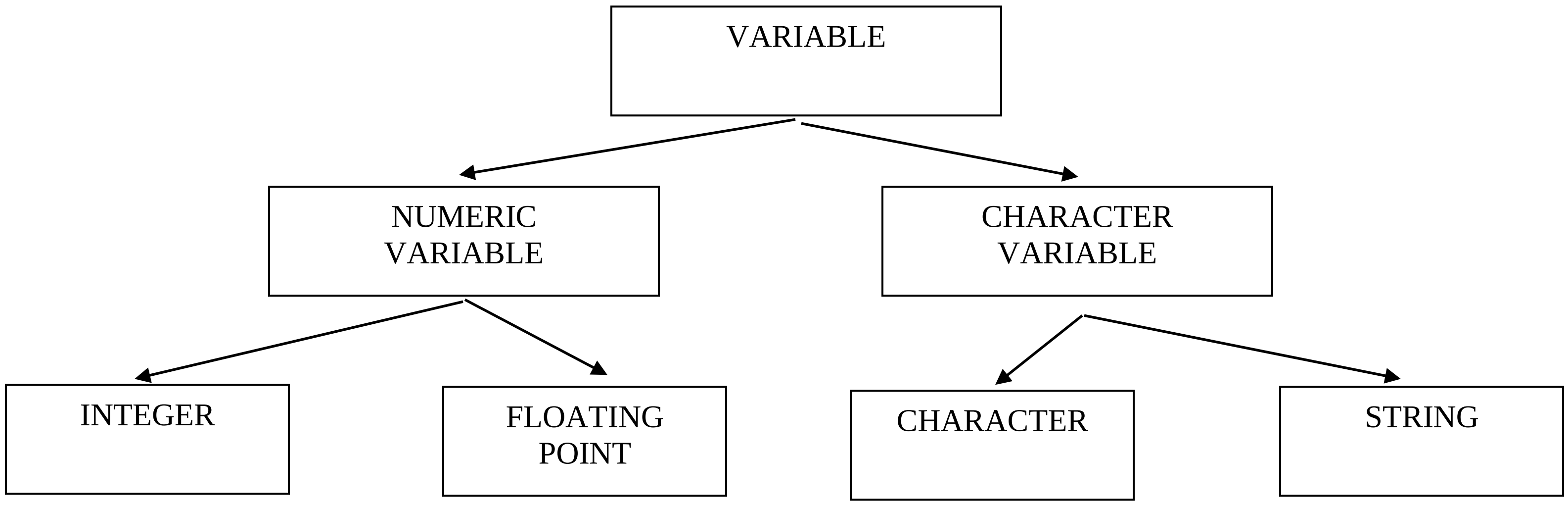
   A string constant is a sequence of character enclosed in " " (double quote).
   Example: "MCA", "GU", "MSC", "5+3".

**Program**:

```
#include<stdio.h>
#include<stdio.h>
void main()
{
        const char a='c';
        clrscr();
        a='z';
        printf("%c",a);
        getch();
}
```

**Variable:**

Variables are those whose value will change during the program execution.



3) Numeric Variable
   Numeric variable is used to store numeric data.
   The numeric variable can be : 1,20000, 345, 99,-3.14, 7.777 etc

There are two types of numeric variable:

a. **Integer**
   An integer variable refers to a sequence of digits.
   There are three types of integer:
   IV. Decimal integer
       Decimal integer consists a set of digits from 0 to 9.
       Eg: 123, 987, -88, +98
   V. Octal integer
       An Octal integer consist any number leading with 0.
   VI. Hexa-decimal
       An hexa decimal integer consist any number leading with 0x or 0X.

b. **Floating point**
   Real variable are used to declare floating point values like: price, percentage, temperature, heights, distance etc…
   Examples: 7.7777, -77.77, 0.0005, 1.5e2

**Program:**

```
#include<stdio.h>
#include<stdio.h>
void main()
{
	int a=10;
	float pi=3.14;
	clrscr();
	a=100;
	pi=31.4;
	printf("%d",a);
	printf("%f",pi);
	getch();
}
```

1) Character variable
   There are two category of Character variable:
   a. Character
   b. String
      a. **Character**:
   Character variable represents a single character within a single quote(‘ ‘).
   Example: ‘A’, ‘Z’, ‘*’, ‘ ‘, ‘5’
      b. **String**:
      A string variable is a sequence of character enclosed in “ “ (double quote).
      Example: “MCA”, ”GU”, “MSC”, “5+3”.

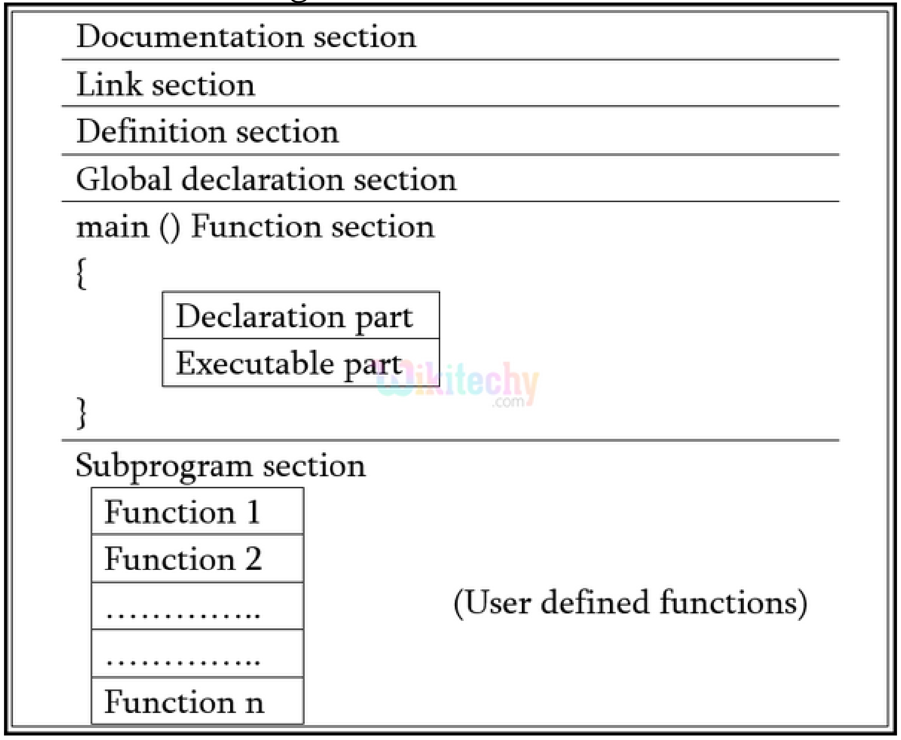
**Program**:

```
#include<stdio.h>
#include<stdio.h>
void main()
{
	char a='c';
	clrscr();
	a='z';
```

```
        printf("%c",a);
        getch();
}
```

## Program Structure:

The ‘C’ program has the following sections.



1. Documentation Section
It consist set of comments given by the author, programmer or users.
This section can have the detail about the program. the details can be program definition, programmers name, date, expected output etc…
Eg.
//write a program to calculate simple interest

2. Link Section
This section provides instructions to the complier to link functions from the system library.
Here, we can link the header files like stdio, conio, string, math etc…
Eg.
#include<stdio.h>

3. Definition Section
It contains all the symbolic constants.
To declare symbolic constant we have to use #define preprocessor directives.
Eg.
#define pi 3.14
4. Global Declaration Section
There are some variables that are used in more than one function, such variable are known as global variables.
We can access global variables in main functions as well as subprograms.

Eg.
int a =10;

5. main() function section
every ‘C’ program must have main() function.
The actual logic will be put over here.
It has two parts:

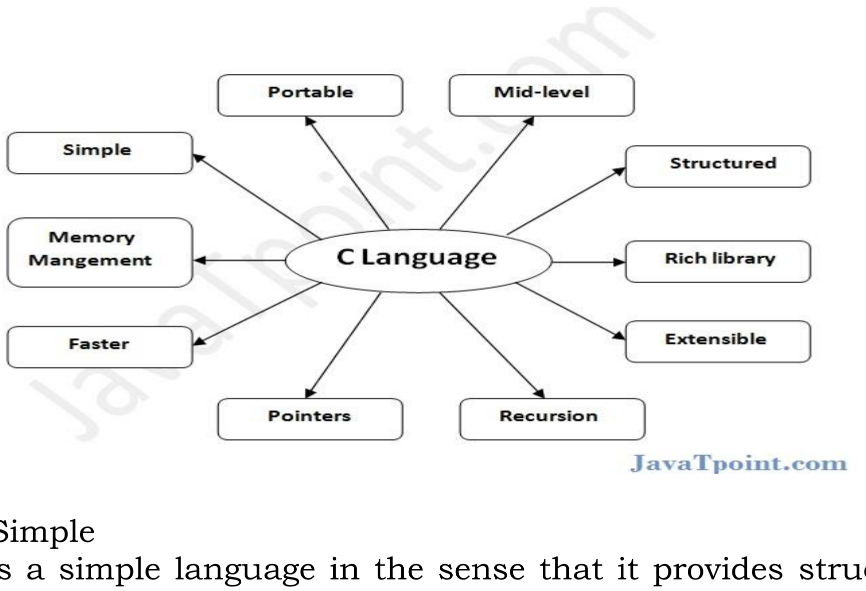
   a. Declaration Part
   Here we can declare the local variables.
   Eg.
   int a =10;
   float b=12.2;
   b. Executable part
   In this part we can perform any logic, calculations etc.
   Eg.
   C=a+b;
   Temp=a;

6. Subprogram section
It contains one or more User Defined Functions(UDF).

## What is ‘C’? History of ‘C’ with its features:

C is a high-level and general-purpose programming language that is ideal for developing firmware or portable applications. Originally intended for writing system software.
C was developed at Bell Labs by Dennis Ritchie for the Unix Operating System in the early 1970s.

## Features:



1) Simple
C is a simple language in the sense that it provides structured approach (to break the problem into parts), rich set of library functions, data types etc.

2) Machine Independent or Portable

Unlike assembly language, c programs can be executed in many machines with little bit or no change. But it is not platform-independent.

3) Mid-level prorgramming language
C is also used to do low level programming. It is used to develop system applications such as kernel, driver etc. It also supports the feature of high level language. That is why it is known as mid-level language.

4) Structured prorgramming language
C is a structured programming language in the sense that we can break the program into parts using functions. So, it is easy to understand and modify.

5) Rich Library
C provides a lot of inbuilt functions that makes the development fast.

6) Memory Management
It supports the feature of dynamic memory allocation. In C language, we can free the allocated memory at any time by calling the free() function.

7) Speed
The compilation and execution time of C language is fast.

8) Pointer
C provides the feature of pointers. We can directly interact with the memory by using the pointers. We can use pointers for memory, structures, functions, array etc.

9) Recursion
In c, we can call the function within the function. It provides code reusability for every function.

10) Extensible
C language is extensible because it can easily adopt new features.

## Comments in ‘C’

Comments are non-executable statements.
There are two types of comment in ‘C’.

1. Single Line Comment
2. Multi Line Comments

1. Single Line Comment

Single line comments are represented by double slash \\. Let's see an example of single line comment in C.

```
#include<stdio.h>
void main()
{
   //printing information
   printf("Hello C");
}
```

2. Multi Line Comments

Multi line comments are represented by slash asterisk * ... *\. It can occupy many lines of code but it can't be nested.

Syntax:

```
/*
code
to be commented
*/
```

```
#include<stdio.h>
void main()
{
   /*printing information
     Multi Line Comment*/
   printf("Hello C");
}
```