

CHAPTER 1: Steps in program development

1.1 Programming process,

1.2 Algorithm

1.3 Flowcharting & different symbols

Study of 'C' as a programming language

1.4 History of 'C'

1.5 Introduction to 'C'

1.6. Basic structure 'C' program, sample 'c' program

1.7 Execution of 'C' program

Constant variables and data types

1.8. Character set

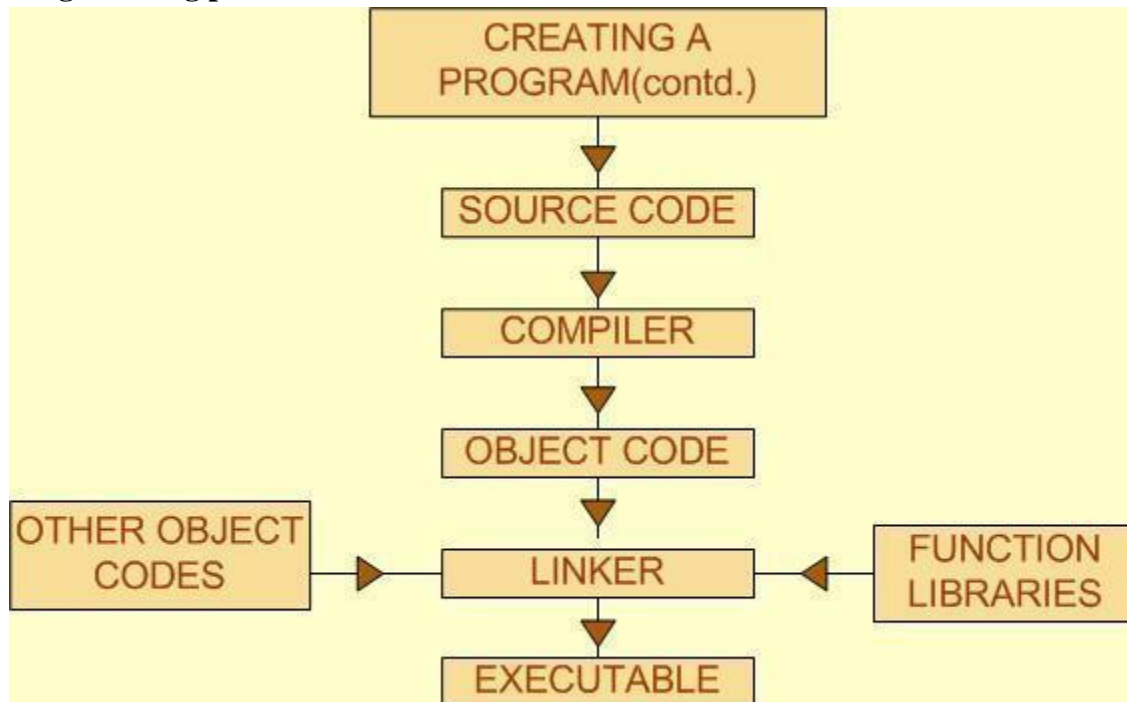
1.9. Key words and identifiers

1.10 Constants

1.11 Data types

1.12 Variables and declaration of variables

1.1 Programming process:



1.2 Algorithm

1] In programming, algorithm are the set of well defined instruction in sequence to solve a program. An algorithm should always have a clear stopping point.

[2] *Qualities of a good algorithm :*

Inputs and outputs should be defined precisely.

Each steps in algorithm should be clear and unambiguous.

Algorithm should be most effective among many different ways to solve a problem.

An algorithm shouldn't have computer code. Instead, the algorithm should be written in such a way that, it can be used in similar programming languages.





An algorithm to add two numbers entered by user.





- Step 1: Start
- Step 2: Declare variables num1, num2 and sum.
- Step 3: Read values num1 and num2.
- Step 4: Add num1 and num2 and assign the result to sum.
 $sum \leftarrow num1 + num2$
- Step 5: Display sum
- Step 6: Stop

An algorithm to find the largest among three different numbers entered by user.

- Step 1: Start
 - Step 2: Declare variables a,b and c.
 - Step 3: Read variables a,b and c.
 - Step 4: If $a > b$
 - If $a > c$
 - Display a is the largest number.
 - Else
 - Display c is the largest number.
 - Else
 - If $b > c$
 - Display b is the largest number.
 - Else
 - Display c is the greatest number.
 - Step 5: Stop
- [Refer manual for algorithm]

1.2 Flowcharting & different symbols

Symbol	Purpose	Description
	Flow line	Used to indicate the flow of logic by connecting symbols.
	Terminal(Stop/Start)	Used to represent start and end of flowchart.
	Input/Output	Used for input and output operation.
	Processing	Used for airthmetic operations and data-manipulations.

Symbol	Purpose	Description
	Decision	Used to represent the operation in which there are two alternatives, true and false.
	On-page Connector	Used to join different flowline
	Off-page Connector	Used to connect flowchart portion on different page.
	Predefined Process/Function	Used to represent a group of statements performing one processing task.

Flowcharts are diagrams that visually present the process of solving problems. They are drawn according to steps described in the algorithms

Rules of Drawing Flowcharts for Algorithms

There are some basic shapes and boxes included in flowcharts that are used in the structure of [explaining steps of algorithms](#). Knowing how to use them while drawing flowcharts is crucial. Here are some rules that should be known:

All boxes of flowcharts are connected with arrows to show the logical connection between them,

Flowcharts will flow from top to bottom,

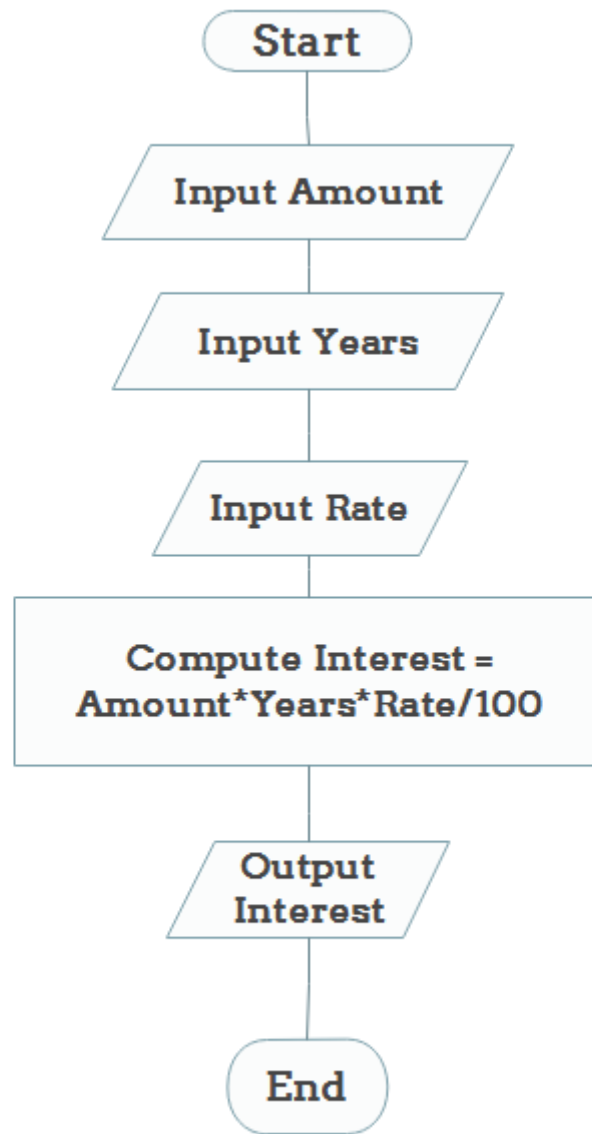
All flowcharts start with a Start Box and end with a Terminal Box

Example 1: Calculate the Interest of a Bank Deposit

Algorithm:

- **Step 1: Read amount,**
- **Step 2: Read years,**
- **Step 3: Read rate,**

- **Step 4:** Calculate the interest with formula " $\text{Interest} = \text{Amount} * \text{Years} * \text{Rate} / 100$ "
- **Step 5:** Print interest,



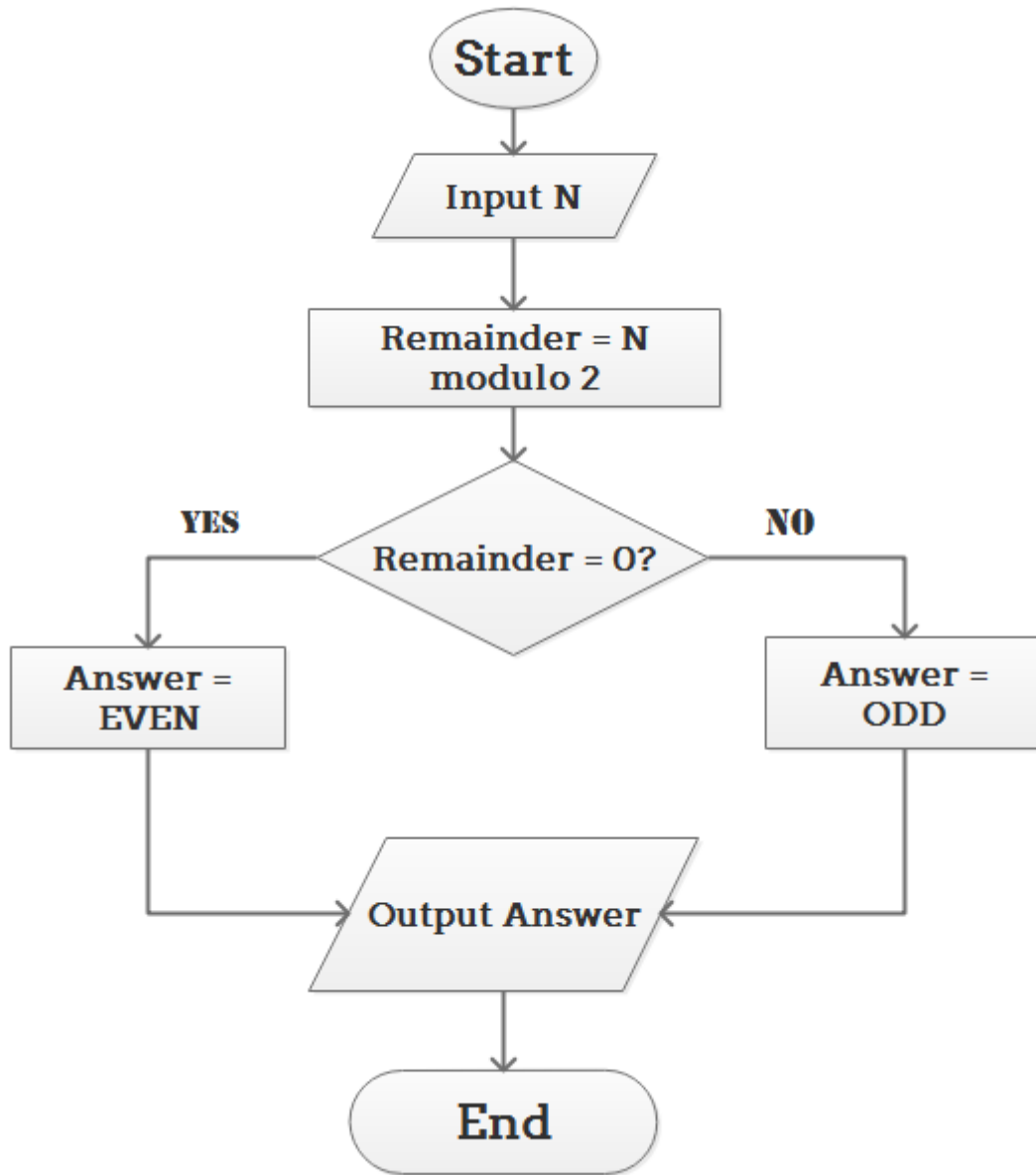
\

Example 2: Determine and Output Whether Number N is Even or Odd

Algorithm:

- Step 1: Read number N,
- Step 2: Set remainder as N modulo 2,

- Step 3: If remainder is equal to 0 then number N is even, else number N is odd,
- Step 4: Print output.

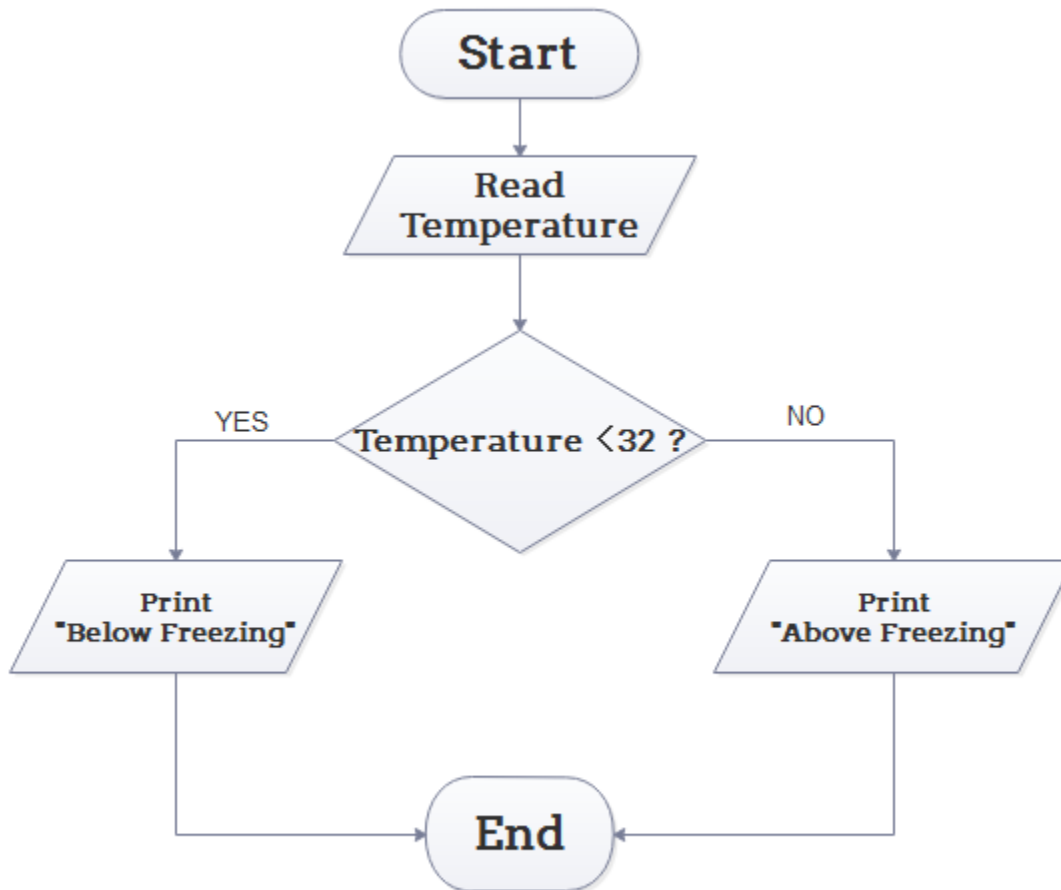


Example 3: Determine Whether a Temperature is Below or Above the Freezing Point

Algorithm:

- Step 1: Input temperature,

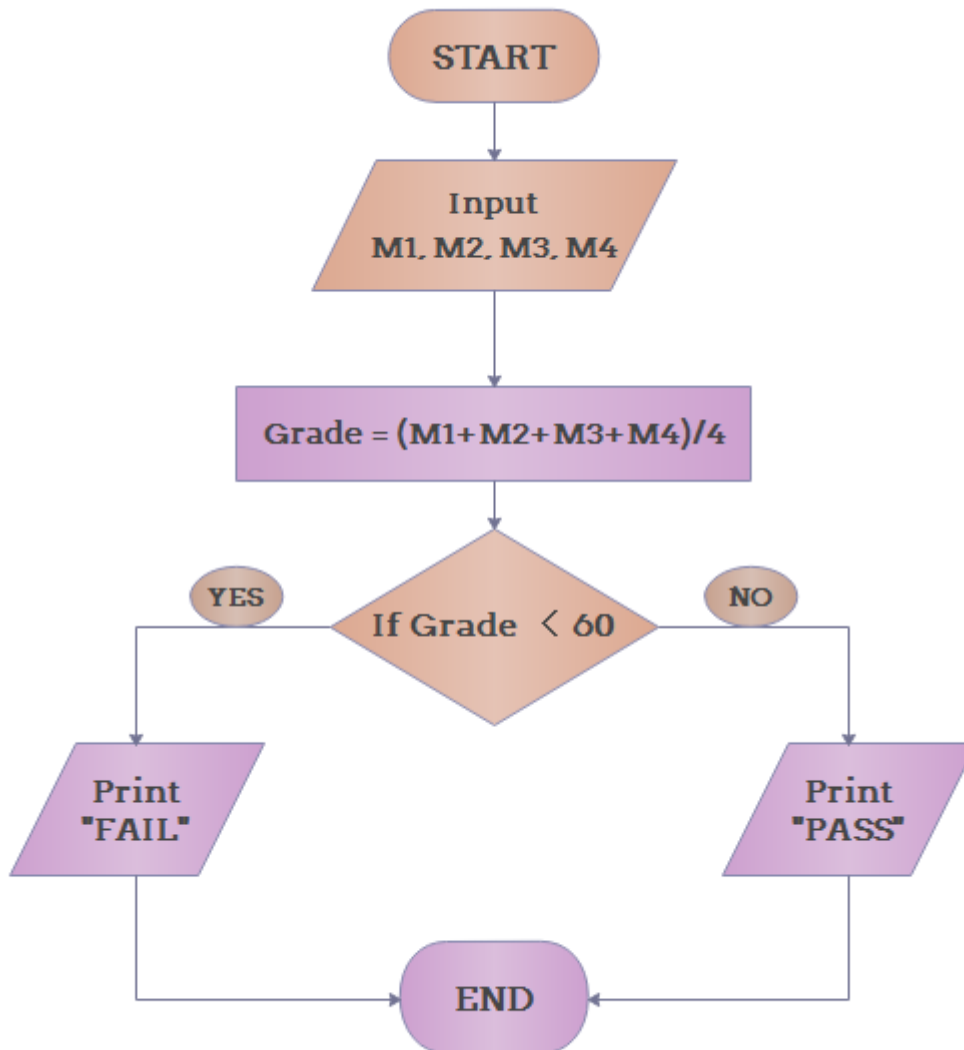
- Step 2: If it is less than 32, then print "below freezing point", otherwise print "above freezing point"



Example 4: Determine Whether A Student Passed the Exam or Not:

Algorithm:

- Step 1: Input grades of 4 courses M1, M2, M3 and M4,
- Step 2: Calculate the average grade with formula " $\text{Grade} = \frac{M1 + M2 + M3 + M4}{4}$ "
- Step 3: If the average grade is less than 60, print "FAIL", else print "PASS".



1.3 History of 'C'

- 2 C is a general-purpose language which has been closely associated with the **UNIX** operating system for which it was developed - since the system and most of the programs that run it are written in C.
- 3 Many of the important ideas of C stem from the language **BCPL**, developed by Martin Richards. The influence of BCPL on C proceeded indirectly through the language **B**, which was written by Ken Thompson in 1970 at Bell Labs, for the first UNIX system on a **DEC PDP-7**. **BCPL** and **B** are "type less" languages whereas C provides a variety of data types.
- 4 In 1972 Dennis Ritchie at Bell Labs writes C and in 1978 the publication of The C Programming Language by Kernighan & Ritchie caused a revolution in the computing world.

- 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.

1.5 Introduction to 'C'

C has been used successfully for every type of programming problem imaginable from operating systems to spreadsheets to expert systems - and efficient **compilers** are available for machines ranging in power from the **Apple** Macintosh to the **Cray** supercomputers. The largest measure of C's success seems to be based on purely practical considerations:

1. the portability of the compiler;
2. the standard library concept;
3. a powerful and varied repertoire of operators;
4. an elegant syntax;
5. ready access to the hardware when needed;
6. and the ease with which applications can be optimised by hand-coding isolated procedures

C is often called a "Middle Level" programming language. This is not a reflection on its lack of programming power but more a reflection on its capability to access the system's low level functions. Most high-level languages (e.g. Fortran) provides everything the programmer might want to do already built into the language. A low level language (e.g. **assembler**) provides nothing other than access to the machines basic instruction set. A middle level language, such as C, probably doesn't supply all the constructs found in high-languages - but it provides you with all the building blocks that you will need to produce the results you want!

Uses of C

C was initially used for system development work, in particular the programs that make-up the operating system. Why use C? Mainly because it produces code that runs nearly as fast as code written in assembly language. Some examples of the use of C might be:

1. Operating Systems
2. Language Compilers
3. Assemblers
4. Text Editors
5. Print Spoolers
6. Network Drivers

- 7. Modern Programs
- 8. Data Bases
- 9. Language Interpreters
- 10. Utilities

1.6. Basic structure ‘C’ program, sample ‘c’ program

BASIC STRUCTURE OF A C PROGRAM:

Structure of C program is defined by set of rules called protocol, to be followed by programmer while writing C program. All C programs are having sections/parts which are mentioned below.

- 1. Documentation section
- 2. Link Section
- 3. Definition Section
- 4. Global declaration section
- 5. Function prototype declaration section
- 6. Main function
- 7. User defined function definition section

C Basic commands	Explanation
#include <stdio.h>	This is a preprocessor command that includes standard input output header file(stdio.h) from the C library before compiling a C program
int main()	This is the main function from where execution of any C program begins.
{	This indicates the beginning of the main function.
/* some comments */	whatever is given inside the command “/* */” in any C program, won’t be considered for compilation and execution.
printf(“Hello World! “);	printf command prints the output onto the screen.
getch();	This command waits for any character input from keyboard.
return 0;	This command terminates C program (main function) and returns 0.

}	This indicates the end of the main function.
---	--

```

#include <stdio.h> /* Link section */
int total = 0; /* Global declaration, definition section */
int sum (int, int); /* Function declaration section */
int main () /* Main function */
{
    printf ("This is a C basic program \n");
    total = sum (1, 1);
    printf ("Sum of two numbers : %d \n", total);
    return 0;
}

int sum (int a, int b) /* User defined function */
{
    return a + b; /* definition section */
}

```

Sections	Description
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.

1.7 Execution of 'C' program

The compilation and execution process of C can be divided in to multiple steps:

Preprocessing - Using a Preprocessor program to convert C source code in expanded source code. "#includes" and "#defines" statements will be processed and replaced actually source codes in this step.

It is the first pass of any C compilation. It processes include-files, conditional compilation instructions and macros

Compilation - Using a Compiler program to convert C expanded source to assembly source code. **Compilation** is the second pass. It takes the output of the preprocessor, and the source code, and generates assembler source code.

Assembly - Using a Assembler program to convert assembly source code to object code.

Assembly is the third stage of compilation. It takes the assembly source code and produces an assembly listing with offsets. The assembler output is stored in an object file.

Linking - Using a Linker program to convert object code to executable code. Multiple units of object codes are linked to together in this step.

Linking is the final stage of compilation. It takes one or more object files or libraries as input and combines them to produce a single (usually executable) file. In doing so, it resolves references to external symbols, assigns final addresses to procedures/functions and variables, and revises code and data to reflect new addresses (a process called relocation).

Loading - Using a Loader program to load the executable code into CPU for execution

Table showing input and output of each step in the compilation and execution process:

Input	Program	Output
source code	> Preprocessor	> expanded source code
expanded source code	> Compiler	> assembly source code
assembly code	> Assembler	> object code
object code	> Linker	> executable code
executable code	> Loader	> execution

1.8.Character set

Letters : C language comprises the following set of letters to form a standard program. They are :
A to Z in Capital letters.
a to z in Small letters.

Digits :C language comprises the following sequence of numbers to associate the letters.0 to 9 digits.

Special Characters: C language contains the following special character in association with the letters and digits

Character	Name	Character	Name
,	Comma	&	Ampersand
.	Period	^	Caret
;	Semicolon	*	Asterisk
:	Colon	-	Minus
?	Question mark	+	Plus
'	Single quote	=	Equal
"	Double quote	<	Less than
!	Exclamation	>	Greater than
	Vertical bar	(Left parenthesis
/	Slash)	Right parenthesis
\	Back slash	[Left square bracket
~	Tilde]	Right square bracket
_	Underscore	{	Left curly bracket
\$	Dollar	}	Right curly bracket
%	Percentage	#	Hash-Number sign

1.9.Key words and identifiers

Keyword is a predefined or reserved word in C library with a fixed meaning and used to perform an internal operation. C Language supports 32 keywords.

Every Keyword exists in lower case latter like auto, break, case, const, continue, int etc.

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	While

Identifiers

Identifiers are the names you can give to entities such as variables, functions, structures etc.

Identifier names must be unique. They are created to give unique name to a C entity to identify it during the execution of a program.

For example:

```
int money;
```

```
double accountBalance;
```

Here, money and accountBalance are identifiers.

Identifier names must be different from keywords. we cannot use int as an identifier because int is a keyword.

Rules for writing an identifier

A valid identifier can have letters (both uppercase and lowercase letters), digits and underscore only.

The first letter of an identifier should be either a letter or an underscore.

In such cases, compiler will complain about it. Some system names that start with underscore are `_fileno`, `_iob`, `_w fopen` etc.

There is no rule on the length of an identifier. However, the first 31 characters of identifiers are discriminated by the compiler. So, the first 31 letters of two identifiers in a program should be different.

In short : Define :

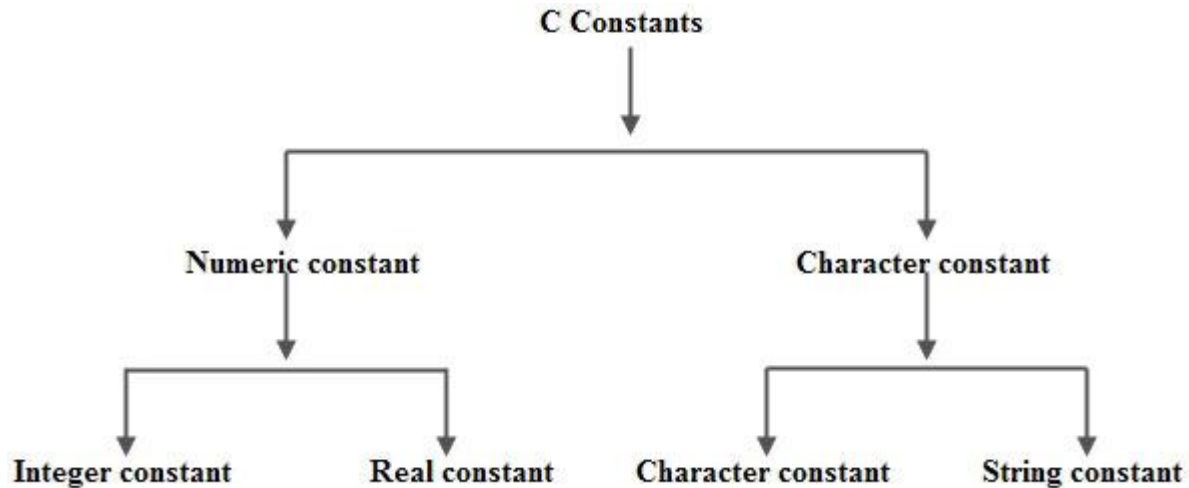
i) Keyword: keywords are reserved words of the language which has specific meaning and cannot be used as variable or constant names.

ii) **Identifier:** Identifier is used for naming variables, functions or labels.

iii) **Variable:** Variable is a user defined element that represents a memory location that can store a value.

iv) **Constant:** Constant is a value that does not change (i.e. fixed value)

1.10 Constants



1.11 Data types

Data types in C Language

Data types specify how we enter data into our programs and what type of data we enter. C language has some predefined set of data types to handle various kinds of data that we use in our program. These datatypes have different storage capacities.

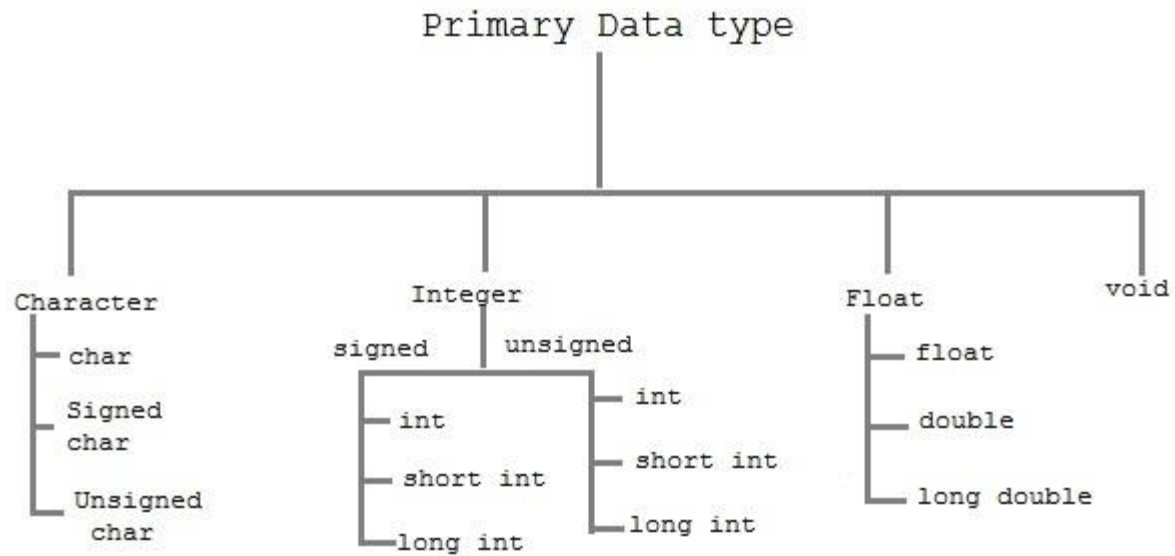
C language supports 2 different type of data types,

Primary data types

These are fundamental data types in C namely integer(**int**), floating(**float**), character(**char**) and **void**.

Derived data types

Derived data types are like array, function, stucture, union and pointer.



1.12 Variables and declaration of variables

```
int i, j, k;
```

```
char c, ch;
```

```
float f, salary;
```

```
double d;extern int d = 3, f = 5; // declaration of d and f.
```

```
int d = 3, f = 5; // definition and initializing d and f.
```

```
byte z = 22; // definition and initializes z.
```

```
char x = 'x'; // the variable x has the value 'x'.
```

AND THAT IS END OF FIRST CHAPTER .=====