Chapter 1

Introduction to C

Software is a collection of programs and a program is a collection of instructions given to the computer. Development of software is a stepwise process. Before developing a software, number of processes are done. The first step is to understand the user requirements. Problem analysis arises during the requirement phase of software development. Problem analysis is done for obtaining the user requirements and to determine the input and output of the program.

For solving the problem, an "algorithm" is implemented. Algorithm is a sequence of steps that gives method of solving a problem. This "algorithm" creates the logic of program. On the basis of this "algorithm", program code is written. The steps before writing program code are as-

$$\begin{tikzcd}
\text{User requirements} \arrow[d] \\
\text{Problem analysis} \arrow[d] \\
\text{Input and Output} \arrow[d] \\
\text{Designing algorithm} \arrow[d] \\
\text{Program coding}
\end{tikzcd}$$

Process of program development

1.1 Design Methods

Designing is the first step for obtaining solution of a given problem. The purpose of designing is to represent the solution for the system. It is really difficult to design a large system because the complexity of system cannot be represented easily. So various methods have been evolved for designing.

1.1.1 Top-Down Design

Every system has several hierarchies of components. The top-level component represents the whole system. Top-Down design method starts from top-level component to lowest level (bottom) component. In this design method, the system is divided into some major components.

Then each major component is divided into lower level components. Similarly other components are divided till the lowest level component.

1.1.2 Bottom-Up Design

Bottom-Up design method is the reverse of Top-Down approach. It starts from the lowest level component to the highest-level component. It first designs the basic components and from these basic components the higher-level components are designed.

1.1.3 Modular Approach

It is better to divide a large system into modules. In terms of programming, module is logically a well-defined part of program. Each module is a separate part of the program. It is easy to modify a program written with modular approach because changes in one module don't affect other modules of program. It is also easy to check bugs in the program in module level programming.

1.2 Programming Languages

Before learning any language, it is important to know about the various types of languages and their features. It is interesting to know what were the basic requirements of the programmers and what difficulties they faced with the existing languages. The programming languages can be classified into two types-

1. Low level languages
2. High level languages

1.2.1 Low Level Languages

The languages in this category are the Machine level language and Assembly language.

1.2.1.1 Machine Level Language

Computers can understand only digital signals, which are in binary digits i.e. 0 and 1. So the instructions given to the computer can be only in binary codes. The machine language consists of instructions that are in binary 0 or 1. Computers can understand only machine level language.

Writing a program in machine level language is a difficult task because it is not easy for programmers to write instructions in binary code. A machine level language program is error-prone and its maintenance is very difficult. Furthermore machine language programs are not portable. Every computer has its own machine instructions, so the programs written for one computer are not valid for other computers.

1.2.1.2 Assembly Language

The difficulties faced in machine level language were reduced to some extent by using a modified form of machine level language called assembly language. In assembly language instructions are given in English like words, such as MOV, ADD, SUB etc. So it is easier to write and understand assembly programs. Since a computer can understand only machine level language, hence assembly language program must be translated into machine language. The translator that is used for translating is called "assembler"

Although writing programs in assembly language is a bit easier, but still the programmer has to know all the low level details related with the hardware of a computer. In assembly language, data is stored in computer registers and each computer has different set of registers. Hence the assembly language program is also not portable. Since the low level languages are related with the hardware, hence the execution of a low-level program is faster.

1.2.2 High-Level Languages

High-level languages are designed keeping in mind the features of portability i.e. these languages are machine independent. These are English like languages, so it is easy to write and understand the programs of high-level language. While programming in a high level language, the programmer is not concerned with the low level details, and so the whole attention can be paid to the logic of the problem being solved. For translating a high-level language program into machine language, compiler or interpreter is used. Every language has its own compiler or interpreter. Some languages in this category are- FORTRAN, COBOL, BASIC, Pascal etc.

1.3 Translators

We know that computers can understand only machine level language, which is in binary 1 or 0. It is difficult to write and maintain programs in machine level language. So the need arises for converting the code of high-level and low-level languages into machine level language and translators are used for this purpose. These translators are just computer programs, which accept a program written in high level or low-level language and produce an equivalent machine language program as output. The three types of translators used are-

- Assembler
- Compiler
- Interpreter

Assembler is used for converting the code of low-level language (assembly language) into machine level language.

Compilers and interpreters are used to convert the code of high-level language into machine language. The high level program is known as source program and the corresponding machine language program is known as object program. Although both compilers and interpreters perform the same task but there is a difference in their working.

A compiler searches all the errors of program and lists them. If the program is error free then it converts the code of program into machine code and then the program can be executed by separate commands. An interpreter checks the errors of program statement by statement. After checking one statement, it converts that statement into machine code and then executes that statement. This process continues until the last statement of program or an erroneous statement occurs.

1.4 History Of C

In earlier days, every language was designed for some specific purpose. For example FORTRAN (Formula Translator) was used for scientific and mathematical applications, COBOL (Common Business Oriented Language) was used for business applications. So need of such a language was felt which could withstand most of the purposes. "Necessity is the mother of invention". From here the first step towards C was put forward by Dennis Ritchie.

The C language was developed in 1970's at Bell laboratories by Dennis Ritchie. Initially it was designed for programming in the operating system called UNIX. After the advent of C, the whole UNIX operating system was rewritten using it. Now almost the entire UNIX operating system and the tools supplied with it including the C compiler itself are written in C.

The C language is derived from the B language, which was written by Ken Thompson at AT&T Bell laboratories. The B language was adopted from a language called BCPL (Basic Combined Programming Language), which was developed by Martin Richards at Cambridge University.

In 1982 a committee was formed by ANSI (American National Standards Institute) to standardize th
C language. Finally in 1989, the standard for C language was introduced known as ANSI C. Generall
most of the modern compilers conform to this standard.

1.5 Characteristics of C

It is a middle level language. It has the simplicity of a high level language as well as the power of
low level language. This aspect of C makes it suitable for writing both application programs and systen
programs. Hence it is an excellent, efficient and general-purpose language for most of the application
such as mathematical, scientific, business and system software applications.

C is small language, consisting of only 32 English words known as keywords (if, else, for, break etc.
The power of C is augmented by the library functions provided with it. Moreover, the language i
extendible since it allows the users to add their own library functions to the library.

C contains control constructs needed to write a structured program hence it is considered a structure
programming language. It includes structures for selection (if...else, switch), repetition (while, fo
do...while) and for loop exit (break).

The programs written in C are portable i.e. programs written for one type of computer or operatin
system can be run on another type of computer or operating system.

1.6 Structure of a C Program

C program is a collection of one or more functions. Every function is a collection of statements an
performs some specific task. The general structure of C program is-

```
Comments
Preprocessor directives
Global variables
main( ) function
{
    local variables
    statements
    ..........
    ..........
}
func1( )
{
    local variables
    statements
    ..........
    ..........
}
func2( )
{
    local variables
    statements
    ..........
    ..........
}
```

Comments can be placed anywhere in a program and are enclosed between the delimiters /* ar
*/. Comments are generally used for documentation purposes.

Preprocessor directives are processed through preprocessor before the C source code passes through compiler. The commonly used preprocessor directives are #include and #define. #include is used for including header files. #define is used to define symbolic constants and macros.

Every C program has one or more functions. If a program has only one function then it must be main(). Execution of every C program starts with main( ) function. It has two parts, declaration of local variables and statements. The scope of the local variable is local to that function only. Statements in the main() function are executed one by one. Other functions are the user-defined functions, which also have local variables and C statements. They can be defined before or after main( ). It may be possible that some variables have to be used in many functions, so it is necessary to declare them globally. These variables are called global variables.

1.7 Environment For C

The steps for the execution of C program are as-

1. Program creation
2. Program compilation
3. Program execution

The C programs are written in mostly two environments, UNIX and MS-DOS.

1.7.1 Unix Environment

Generally a command line C compiler is provided with the UNIX operating system. This compiler is named cc or gcc.

(a) Program creation

In unix environment, file can be created with vi editor as-

```
$ vi filename.c
```

Here $ is the unix prompt. The file can be saved by pressing ESC and SHIFT+zz.

(b) Program compilation

After creation of C program, it can be compiled as-

```
$cc filename.c
```

If the program has mathematical function then it is compiled as-

```
$cc filename.c  -lm
```

After compilation, the executable code is stored in the file a.out .

(c) Program execution

After the compilation of program, it can be executed as-

```
$ a.out
```

1.7.2 MS-DOS Environment

In MS-DOS environment creation, compilation and execution can be done using command line or IDE (Integrated Development Environment).

1.7.2.1 Command Line

In Borland C, the command line compiler is bcc.exe and in Turbo C the command line compiler is tcc.exe.

(a) Program creation

The program file can be created using any editor and should be saved with .c extension

(b) Program compilation

After saving the file, C program can be compiled at DOS prompt by writing-

```
C:\>tcc  filename  (in Turbo C)
C:\>bcc filename  (in Borland C)
```

(c) Program execution

After compilation of C program, the executable file filename.exe is created. It is executed at DOS promp by writing-

```
C:\>filename
```

1.7.2.2 Integrated Development Environment

All these steps can be performed in an IDE using menu options or shortcut keys. In Borland C th program bc.exe is the IDE and in Turbo C the program tc.exe is the IDE. So we can open the IDI by typing bc or tc at the command prompt.

(a) Program creation

A new file can be created from menu option New. The file can be saved by menu option Save. If th file is unnamed then it is saved by menu option Save as. An existing file can be opened from the men option Open.

(b) Program compilation

The file compiled by the menu option Compile. (Alt+F9)

(c) Program execution

The file can be executed by the menu option Run. (Ctrl+F9). The output appears in the output windo that can be seen using the keys Alt+F5.

We have given you just a preliminary knowledge of how to execute your programs. There are severa other options that you can explore while working and it is best to check the manual of your compile to know about these options.

Chapter 2

Elements of C

Every language has some basic elements and grammatical rules. Before understanding programming, it is must to know the basic elements of C language. These basic elements are character set, variables, datatypes, constants, keywords (reserved words), variable declaration, expressions, statements etc. All of these are used to construct a C program.

2.1 C Character Set

The characters that are used in C programs are given below-

2.1.1 Alphabets

A, B, C ....Z

a, b, c ....z

2.1.2 Digits

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

2.1.3 Special characters

Character	Meaning	Character	Meaning
+	plus sign	-	minus sign(hyphen)
*	asterisk	%	percent sign
\	Backward slash	/	forward slash
<	less than sign	=	equal to sign
>	greater than sign	_	underscore
(	left parenthesis	)	right parenthesis
{	left braces	}	right braces
[	left bracket	]	right bracket
,	comma	.	period
'	single quotes	"	double quotes
:	colon	;	Semicolon
?	Question mark	!	Exclamation sign
&	ampersand	|	vertical bar
@	at the rate	^	caret sign
$	dollar sign	#	hash sign
~	tilde sign	`	back quotation mark

2.2 Execution Characters/Escape Sequences

Characters are printed on the screen through the keyboard but some characters such as newline, tab, backspace cannot be printed like other normal characters. C supports the combination of backslash (\) and some characters from the C character set to print these characters.

These character combinations are known as escape sequences and are represented by two characters. The first character is "\" and second character is from the C character set. Some escape sequences are given below-

Escape Sequence	Meaning	ASCII Value	Purpose
\b	backspace	008	Moves the cursor to the previous position of the current line
\a	bell(alert)	007	Produces a beep sound for alert
\r	carriage return	013	Moves the cursor to beginning of the current line.
\n	newline	010	Moves the cursor to the beginning of the next line
\f	form feed	012	Moves the cursor to the initial position of the next logical page.
\0	null	000	Null
\v	vertical tab	011	Moves the cursor to next vertical tab position
\t	Horizontal tab	009	Moves the cursor to the next horizontal tab position.
\\	backslash	092	Presents a character with backslash (\ )

Blank, horizontal tab, vertical tab, newline, carriage return, form feed are known as whitespace in C language.

2.3 Trigraph Characters

There is a possibility that the keyboard doesn't print some characters. C supports the facility of "trigraph sequence" to print these characters. These trigraph sequences have three characters. First two are '??' and third character is any character from C character set. Some trigraph sequences are as given below-

Trigraph Sequence	Symbol
??<	{ left brace
??>	} right brace
??(	[left bracket
??)	] right bracket
??!	| vertical bar
??/	\ backslash
??=	# hash sign
??-	~ tilde
??'	^ caret

2.4 Delimiters

Delimiters are used for syntactic meaning in C. These are as given below-

:	colon	used for label
;	semicolon	end of statement
( )	parentheses	used in expression
[ ]	square brackets	used for array
{ }	curly braces	used for block of statements
#	hash	preprocessor directive
,	comma	variable delimiter

2.5 Reserved Words / Keywords

There are certain words that are reserved for doing specific tasks. These words are known as keywords and they have standard, predefined meaning in C. They are always written in lowercase. There are only 32 keywords available in C which are given below-

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

2.6 Identifiers

All the words that we'll use in our C programs will be either keywords or identifiers. Keywords are predefined and can't be changed by the user, while identifiers are user defined words and are used to give names to entities like variables, arrays, functions, structures etc. Rules for naming identifiers are given below-

(1) The name should consist of only alphabets (both upper and lower case), digits and underscore sign(_).

(2) First character should be an alphabet or underscore.

(3) The name should not be a keyword.

(4) Since C is case sensitive, the uppercase and lowercase letters are considered different. For example code, Code and CODE are three different identifiers.

(5) An identifier name may be arbitrarily long. Some implementations of C recognize only the first eight characters, though most implementations recognize 31 characters. ANSI standard compilers recognize 31 characters.

The identifiers are generally given meaningful names. Some examples of valid identifier names-

Value   a   net_pay   rec1   _data   MARKS

Some examples of invalid identifier names are-

5bc	First character should be an alphabet or underscore
int	int is a keyword
rec#	# is a special character
avg no	blank space is not permitted

2.7 Data Types

C supports different types of data. Storage representation of these data types is different in memory. There are four fundamental datatypes in C, which are int, char, float and double.

'char' is used to store any single character , 'int' is used to store integer value, 'float' is used for storing single precision floating point number and 'double' is used for storing double precision floating point number. We can use type qualifiers with these basic types to get some more types.

There are two types of type qualifiers-

1. Size qualifiers - short, long
2. Sign qualifiers - signed, unsigned

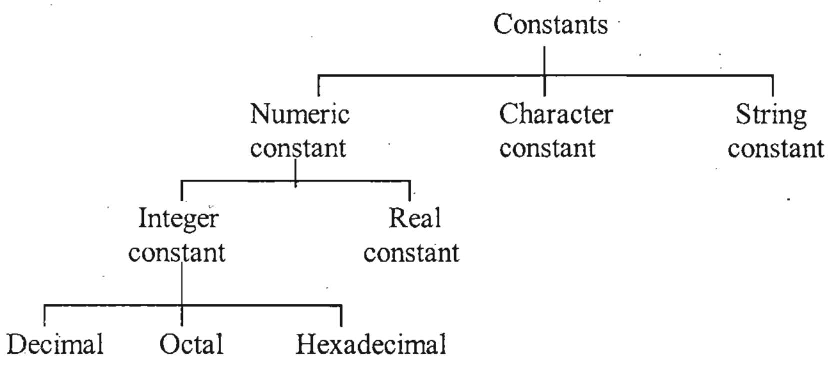
When the qualifier unsigned is used the number is always positive, and when signed is used number may be positive or negative. If the sign qualifier is not mentioned, then by default signed qualifier is assumed. The range of values for signed data types is less than that of unsigned type. This is because in signed type, the leftmost bit is used to represent the sign, while in unsigned type this bit is also used to represent the value.

The size and range of different data types on a 16-bit machine is given in the following table. The size and range may vary on machines with different word sizes.

Basic data types	Data types with type qualifiers	Size(bytes)	Range
char	char or signed char	1	-128 to 127
	unsigned char	1	0 to 255
int	int or signed int	2	-32768 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	-2147483648 to 2147483647
	unsigned long int	4	0 to 4294967295
float	float	4	3.4E-38 to 3.4E+38
double	double	8	1.7E-308 to 1.7E+308
	long double	10	3.4E-4932 to 1.1E+4932

2.8 Constants

Constant is a value that cannot be changed during execution of the program. There are three types of constants-



2.8.1 Numeric Constants

Numeric constants consist of numeric digits, they may or may not have decimal point(.). These are the rules for defining numeric constants-

1. Numeric constant should have at least one digit.
2. No comma or space is allowed within the numeric constant.
3. Numeric constants can either be positive or negative but default sign is always positive.

There are two types of numeric constants-

2.8.1.1 Integer constant

Integer constants are whole numbers which have no decimal point (.). There are three types of integer constants based on different number systems. The permissible characters that can be used in these constants are-

Decimal constants -	0,1,2,3,4,5,6,7,8,9	(base 10)
Octal constants -	0,1,2,3,4,5,6,7	(base 8)
Hex decimal constants -	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F,a,b,c,d,e,f	(base 16)

Some valid decimal integer constants are-

0
123
3705
23759

Some invalid decimal integer constants are-

Invalid	Remark
2.5	illegal character (.)
3#5	illegal character (#)
98 5	No blank space allowed
0925	First digit can not be zero
8,354	Comma is not allowed

In octal integer constants, first digit must be 0. For example-

0
05

077
0324

In hexadecimal integer constants, first two characters should be 0x or 0X. Some examples are as-

0x
0X23
0x515
0XA15B
0xFFF
0xac

By default the type of an integer constant is int. But if the value of integer constant exceeds the range of values represented by int type, the type is taken to be unsigned int or long int. We can also explicitly mention the type of the constant by suffixing it with l or L(for long), u or U (for unsigned), ul or UL (for unsigned long). For example-

6453			Integer constant of type int
45238722UL	or	45238722ul	Integer constant of type unsigned long int
6655U	or	6655u	Integer constant of type unsigned int

2.8.1.2 Real (floating point) Constants

Floating point constants are numeric constants that contain decimal point. Some valid floating point constants are-

0.5
5.3
4000.0
0.0073
5597.
39.0807

For expressing very large or very small real constants, exponential (scientific) form is used. Here the number is written in the mantissa and exponent form, which are separated ~~by~~ 'e' or 'E'. The mantissa can be an integer or a real number, while the exponent can be only an integer (positive or negative). For example the number 1800000 can be written as 1.8e6, here 1.8 is mantissa and 6 is the exponent. Some more examples are as-

Number			Exponential form
2500000000	→	$2.5*10^9$	2.5e9
0.0000076	→	$7.6*10^{-6}$	7.6e-6
-670000	→	$-6.7*10^5$	-6.7E5

By default the type of a floating point constant is double. We can explicitly mention the type of constant by suffixing it with a f or F (for float type), l or L (for long double). For example-

2.3e5	floating point constant of type double
2.4e-9l or 2.4e-9L	floating point constant of type long double
3.52f or 3.52F	floating point constant of type float

2.8.2 Character Constants

A character constant is a single character that is enclosed within single quotes. Some valid character constants are-

'9' 'D' '$' ' ' '#'

Some invalid character constants are-

Invalid	Remark
'four'	There should be only one character within quotes
"d"	Double quotes are not allowed
''	No character between single quotes
y	Single quotes missing

Every character constant has a unique integer value associated with it. This integer is the numeric value of the character in the machine's character code. If the machine is using ASCII (American Standard Code for Information Interchange), then the character 'G' represents integer value 71 and the character '5' represents value 53. Some ASCII values are-

A - Z	ASCII value (65 - 90)
a - z	ASCII value (97 - 122)
0 - 9	ASCII value (48 - 57)
;	ASCII value (59)

ASCII values for all characters are given in Appendix A.

2.8.3 String Constants

A string constant has zero, one or more than one character. A string constant is enclosed within double quotes (" "). At the end of string, \0 is automatically placed by the compiler.

Some examples of string constants are-

"Kumar"
"593"
"8"
" "
"A"

Note that "A" and 'A' are different, the first one is a string constant which consists of character A and \0 while the second one is a character constant which represents integer value 65.

2.8.4 Symbolic Constants

If we want to use a constant several times then we can provide it a name. For example if we have to use the constant 3.14159265 at many places in our program, then we can give it a name PI and use this name instead of writing the constant value everywhere. These types of constants are called symbolic constants or named constants.

A symbolic constant is a name that substitutes for a sequence of characters. The characters may represent a numeric constant, a character constant or a string constant.

These constants are generally defined at the beginning of the program as-

```
#define name value
```

Here 'name' is the symbolic name for the constant, and is generally written in uppercase letters. 'value' can be numeric, character or string constant.

Some examples of symbolic constants are as-

```
#define MAX 100
#define PI 3.14159625
#define CH 'y'
#define NAME "Suresh"
```

In the program, these names will be replaced by the corresponding values. These symbolic constants improve the readability and modifiability of the program.

2.9 Variables

Variable is a name that can be used to store values. Variables can take different values but one at a time. These values can be changed during execution of the program. A data type is associated with each variable. The data type of the variable decides what values it can take. The rules for naming variables are same as that for naming identifiers.

2.9.1 Declaration of Variables

It is must to declare a variable before it is used in the program. Declaration of a variable specifies its name and datatype. The type and range of values that a variable can store depends upon its datatype. The syntax of declaration of a variable is-

```
datatype variablename;
```

Here datatype may be int, float, char, double etc. Some examples of declaration of variables are-

```
int x;
float salary;
char grade;
```

Here x is a variable of type int, salary is a variable of type float, and grade is a variable of type char. We can also declare more than one variable in a single declaration. For example-

```
int x, y, z, total;
```

Here x, y, z, total are all variables of type int.

2.9.2 Initialisation of Variables

When a variable is declared it contains undefined value commonly known as garbage value. If we want we can assign some initial value to the variable during the declaration itself, this is called initialisation of the variable. For example-

```
int a = 5;
float x = 8.9, y = 10.5;
char ch = 'y';
double num = 0.15197e-7;
int l, m, n, total = 0;
```

In the last declaration only variable total has been initialised.

2.10 Expressions

An expression is a combination of operators, constants, variables and function calls. The expression can be arithmetic, logical or relational.

Some examples are as-

x+y	- arithmetic operation
a = b+c	- uses two operators (=) and (+)
a > b	- relational expression
a==b	- logical expression
func(a, b)	- function call

We'll study about these operators and expressions in the next chapter.

2.11 Statements

In a C program, instructions are written in the form of statements. A statement is an executable part of the program and causes the computer to carry out some action. Statements can be categorized as-

(i) Expression statements
(ii) Compound statements
(iii) Selection statements (if, if...else, switch)
(iv) Iterative statements (for, while, do...while)
(v) Jump statements (goto, continue, break, return)
(vi) Label statements (case, default, label statement used in goto)

2.11.1 Expression Statement

Expression statement consists of an expression followed by a semicolon. For example-

```
x = 5;
x = y - z;
func( a , b );
```

A statement that has only a semicolon is also known as null statement. For example-

```
; /* null statement */
```

2.11.2 Compound Statement

A compound statement consists of several statements enclosed within a pair of curly braces { }. Compound statement is also known as block of statements. Note that there is no semicolon after the closing brace. For example-

```
{
    int  l=4,b=2,h=3;
    int  area,volume;
    area=2*(l*b+b*h+h*l);
    volume=l*b*h;
}
```

If any variable is to be declared inside the block then it can be declared only at the beginning of the block. The variables that are declared inside a block can be used only inside that block. All other categories of statements are discussed in further chapters.

2.12 Comments

Comments are used for increasing readability of the program. They explain the purpose of the program and are helpful in understanding the program. Comments are written inside /* and */. There can be single line or multiple line comments. We can write comments anywhere in a program except inside a string constant or a character constant.

Some examples of comments are-

```
/*Variable b represents basic salary*/          (single line comment)
/*This is a C program to calculate              (multiple line comment)
     simple interest  */
```

Comments can't be nested i.e. we can't write a comment inside another comment.