Chapter
1

Introduction to C Language

Students will learn:

- Introduction to computer
- Pseudocode and flowchart
- Basics of C language
- Tokens in C like keywords, identifiers, constants, operators, variables
- Operator precedence and associativity among the operators
- Primitive data types available in C

1.1 INTRODUCTION TO COMPUTER

Computer is an electronic device used for mathematical computation. It accepts the input data, performs a set of operations and produces the desired result. Some of the characteristics of computer are listed below.

- Able to store a huge amount of data in the memory and even accessing of the data is easy.
- Receives the input and set of instructions from the programmer and after processing, displays the resultant output.
- Capable of performing the computational operations at a very high speed.
- The result produced by the computer is accurate and error free.
- Ability to run 24 × 7 without any hassle and provide consistent results.
- Deployed in a variety of domains like education, entertainment, healthcare, games, etc.

The electronic generation of computer started in the year 1946 and is classified into five broad categories:

1. **First generation computers:** This generation of computer spans from 1946–1959 which were built with vaccum tubes. For permanent storage, maganetic tapes were used however, the storage capacity of the primary memory was poor. This generation used machine level languages composed of 0's and 1's. Some of the machines into this generation are ENIAC, EDVAC, EDSAC, and so on.
2. **Second generation computers:** This generation of computers spans from 1957–1964 which used transistors and diodes. Some of the high level languages of second generation are FORTRAN, ALGOL, and Pascal. In terms of speed, memory, and reliability its performance was better than the first generation computer.

Some of the machines which falls into this generation are CDC 1604, HONEYWELL 400, IBM 1401, and so on.

3. **Third generation computers:** This generation of computers span from 1965–1970 which used integrated circuits. The speed improved from millisecond to nanosecond and memory improved by using semiconductor technology. The size of the computer reduced drastically and introduced the concept of operating system. Some of the machines which falls into this generation are CDC 6600, NCR 395, UNIVAC 1108, and so on.
4. **Fourth generation computers:** This generation of computers span from 1970–1990 which used large scale integrated circuits composed of 1000s of transistors in a single chip. The storage capacity was more and speed of operation also increased from nanoseconds to picoseconds. Some of the machines which fall into this generation are IBM 3033, CYBER-205, PDP-11, and so on.
5. **Fifth generation computers:** This generation of computers starts after the year 1980 and they are popularly referred to as supercomputers which embedded millions of transistors into a single chip. This generation gave birth to the concept of artificial intelligence with distributed processing capability. Some of the machines which falls into this generation are PARAM-1000, CRAY, and so on.

1.2 BASIC COMPONENTS OF A COMPUTER

A computer is composed of three main components that is input unit, output unit, and system unit. High level view of the basic components of a computer is given in Fig. 1.1.

Input unit: The input unit is responsible for reading the input and instructions from the user. Some of the commonly used input devices are keyboard, mouse, microphone, scanner, and so on.

Output unit: The output unit is responsible for displaying the output. Some of the output devices are printer, plotter, speaker, disk, and so on.

System unit: It is composed of two subcomponents (i) central processing unit (CPU) and (ii) memory. The CPU is the brain of the computer which does processing of the input data and instructions and composed of two components (i) control unit (CU) and

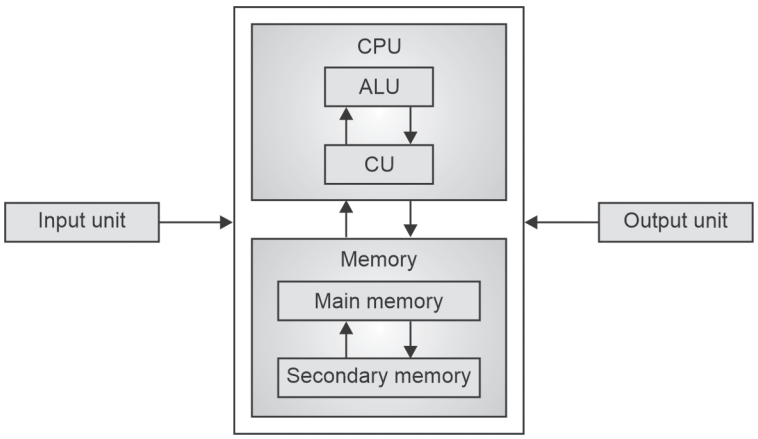

Fig. 1.1: Basic components of a computer

(ii) arithmetic logical unit (ALU). The CU takes the responsibility of coordinating the functionalities of all the components of the system. ALU unit does basic arithemetic and logical operations. The memory component is responsible for storing the data and instructions in the memory. The memory is classified into two types (i) main memory and (ii) secondary memory. Primary memory is the temporary memory which stores data and instructions but once the power goes the data is lost. The secondary memory is the permanent memory which stores the data for longer period of time and even the data does not get lost even when the power goes off.

1.3 PSEUDOCODE AND FLOWCHART

Pseudocode is popularly referred to as algorithm, and is invented by Abu Jafar Muhammad ibn Mūsā al-Khwarizmī. It is a way of describing computer algorithms or a specific task as a set of statements using a combination of natural language and programming language. Pseudocode is an informal way of representing a program or a task without following any programming syntax. It is a generic way of describing an algorithm without use of any specific programming language/ syntax, means it cannot be executed on a computer, but it resembles the real programming code.

Characteristics

- The pseudocode starts with zero or more inputs.
- After doing computational operation, pseudocode delivers atleast one output.
- The statements mentioned in the pseudocode must be precise.
- The statements must get completed in finite time interval.
- The complexity of the pseudocode must be less and simple to understand.

Flowcharts

These are graphical representations of algorithms (or pseudocodes). They form an intermediate step between human understanding of the task to be accomplished and the coded program that directs a computer to complete the various steps of the task.

It is a diagram made up of boxes, diamonds and other shapes, connected by arrows, each shape represents a step in the process, and the arrows show the order in which they occur.

Advantages

i. Flowcharts are easier to understand than algorithm/pseudocodes since it is the pictorial representation of problem or the task to be solved .

ii. Flowcharts are independent of programming language.

Flowcharts are classified into two types (i) program flowchart and (ii) system flowchart. The program flowchart is used to represent a set of activities involved in the execution of the program whereas the system flowchart is used to represent a set of system related activities.

Basic flowchart symbols: Table 1.1 shows the commonly used symbols in a flowchart to represent some actions or operations for accomplishing a task. The flow is normally from top to bottom.

Table 1.1: Flowchart symbols

Symbol	*Name*	*Description*
	Start/Stop	Terminal points to indicate both start and stop
	Process	Represents the process steps
	Data(I/O)	Input/output from device (i.e. to accept data and display output)
	Decision	Indicates decision and branching
	Connector	Shows continuation of one point in process to other
	Flow line	Shows direction and connects the flowchart symbols
	Repetition	Shows the repetition of statements in a loop

Example 1.1: Design pseudocode and draw flowchart to find product of two numbers.

Pseudocode

1. Start
2. Set product = 1
3. Read two numbers Num1 and Num2
4. Product = Num1 * Num2
5. Output product
6. Stop

Flowchart

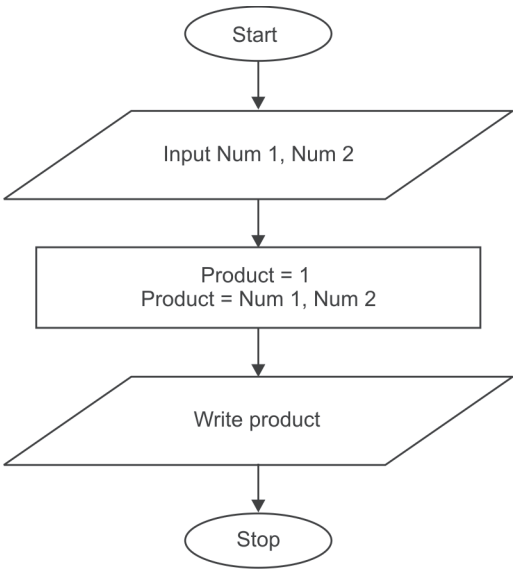

Fig. 1.2: Flowchart for product of two numbers

Example 1.2: Design pseudocode and draw flowchart to find sum and average of three numbers.

Pseudocode

1. Start
2. Set Average = 0, Sum = 0
3. Read three numbers Num1, Num2, Num3
4. Sum = Num1 + Num2 + Num3
5. Average = Sum/3
6. Output Sum, Average
7. Stop

Flowchart

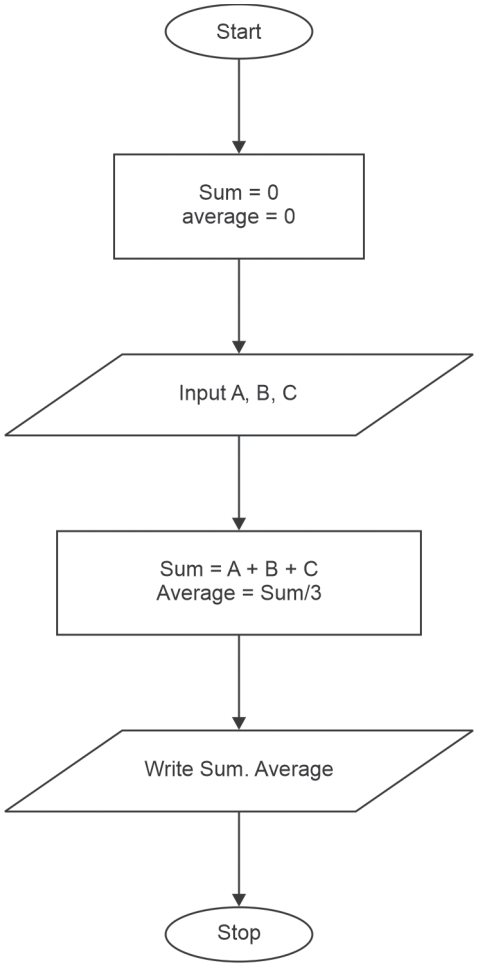

Fig. 1.3: Flowchart for finding sum and average of three numbers

Example 1.3: Design pseudocode and draw flowchart to find greatest of two numbers.

Pseudocode

1. Start
2. Read two numbers A and B
3. If (A> B)
   Output A is greater
   else
   Output B is greater
4. Stop

Flowchart

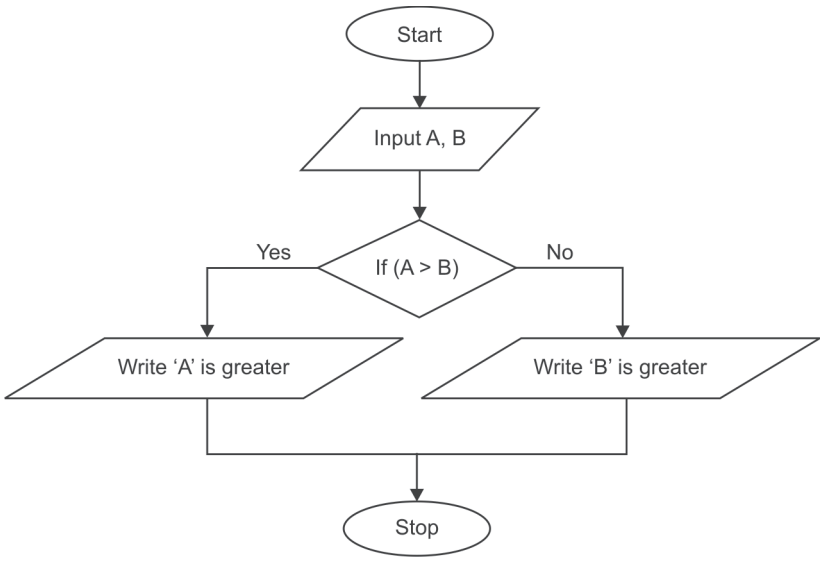

Fig. 1.4: Flowchart to compare two numbers and print the greater one

Example 1.4: Design pseudocode and draw flowchart to compute simple interest.

Pseudocode

1. Start
2. Set Interest I = 0
3. Read Principle as P, Rate of interest as R and Duration as N
4. I = (P * R * N)/100
5. Output I
6. Stop

Flowchart

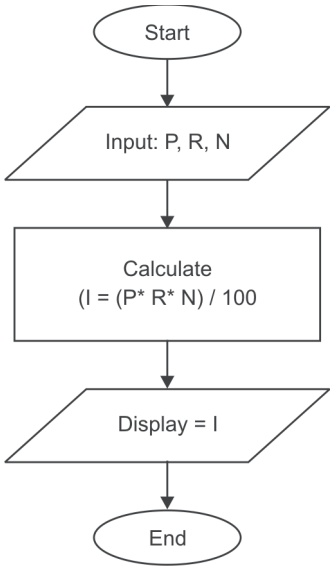

Fig. 1.5: Flowchart to compute simple interest

Example 1.5: Design pseudocode and draw flowchart to find area of a rectangle.

Pseudocode

1. Start
2. Set Area = 0
3. Read Length and Breadth
4. Area = Length * Breadth
5. Output Area
6. Stop

Flowchart

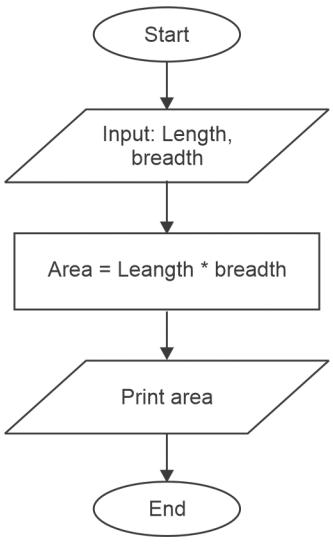

Fig. 1.6: Flowchart to compute area of a rectangle

Example 1.6: Design pseudocode and draw flowchart for adding first N natural numbers.

Pseudocode

1. Set I = 1, N = 0, Sum = 0
2. Enter N
3. Repeat the following:
   a. If I > N, terminate the repetition and go to step 4 otherwise go to step 3b.
   b. Add I to Sum and set equal to Sum.
   c. Increment I by one.
4. Print Sum.
5. Stop

Flowchart

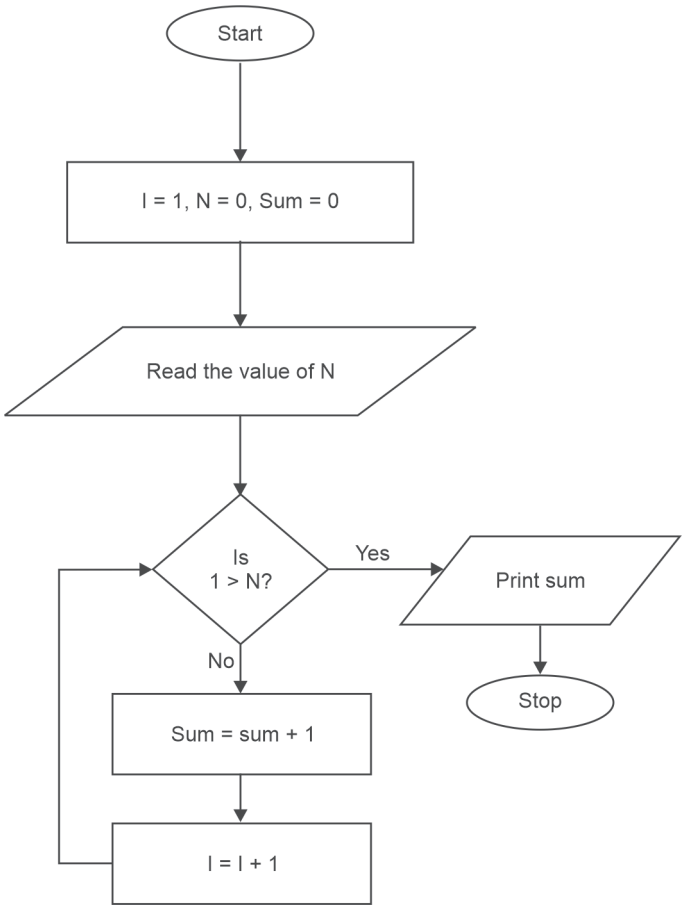

Fig. 1.7: Flowchart to find sum of first N natural numbers

Example 1.7: Design a pseudocode and draw flowchart to convert a temperature in degrees Fahrenheit to degrees Celsius (*Hint*: Celsius = 5/9 * (Fahrenheit - 32).

Pseudocode

1. Start
2. Read F (Fahrenheit's degree)
3. Celsius = 5/9 * (F - 32)
4. Output Celsius
5. Stop

Flowchart

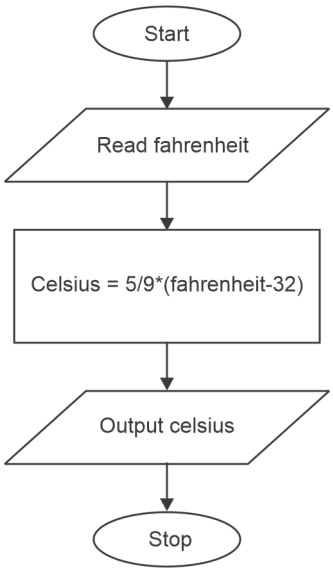

Fig. 1.8: Flowchart to convert temperature in degree Fahrenheit to degree Celsius

Example 1.8: Design a pseudocode and draw flowchart to find sum, difference, product and quotient for given numbers.

Pseudocode

1. Start
2. Read Num1 and Num2
3. Sum = Num1 + Num2
4. Difference = Num1–Num2
5. Product = Num1* Num2
6. Quotient = Num1/Num2
7. Output Sum, Difference, Product and Quotient
8. Stop

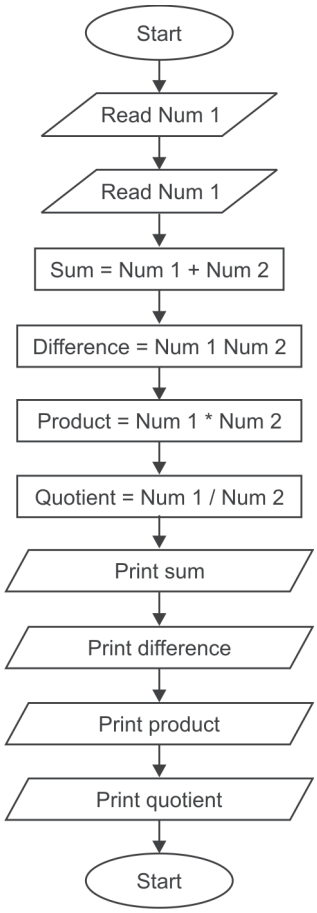

Fig. 1.9: Flowchart to find sum, difference, product and quotient for given numbers

Example 1.9: Design a pseudocode and draw flowchart to find the entered number is positive, negative or neutral.

Pseudocode

1. Start
2. Set Num = 0
3. Input Num
4. If Num > 0 goto step 6, otherwise go to step 5
5. If Num < 0 goto step 7, otherwise go to step 8
6. Output "Num is positive", go to step 9
7. Output "Num is negative", go to step 9
8. Output "Num is zero"
9. Stop

Flowchart

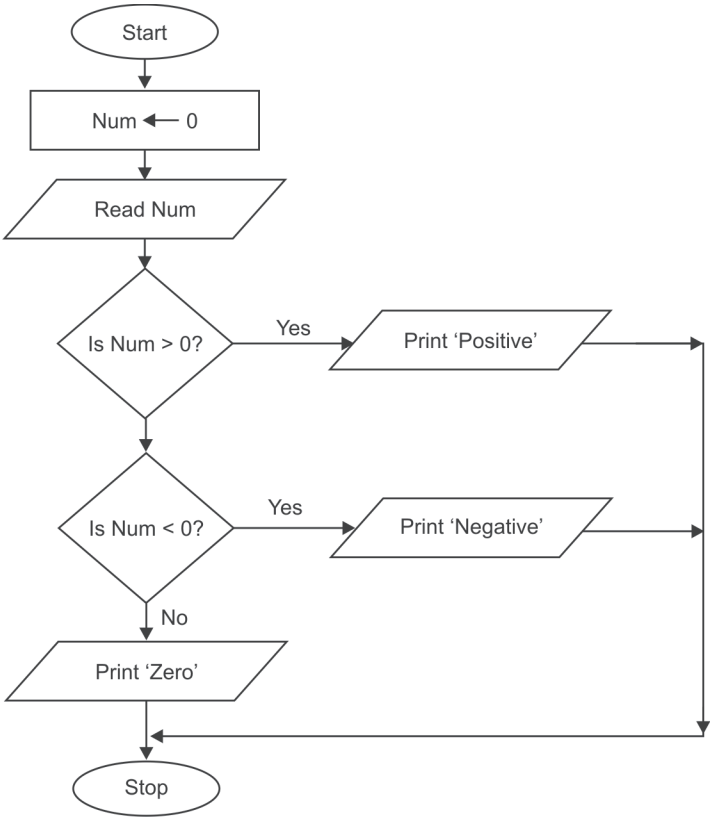

Fig. 1.10: Flowchart to find if entered number is positive or negative or neutral

Example 1.10: Design a pseudocode and draw flowchart to find largest of three numbers.

Pseudocode

1. Start
2. Input A, B, C
3. If A > B go to step 4, otherwise go to step 5
4. If A> C go to step 6, otherwise go to step 8
5. If B > C go to step 7, otherwise go to step 8
6. Output "A is the largest", go to step 9
7. Output "B is the largest", go to step 9
8. Output " C is the largest"
9. Stop

Flowchart

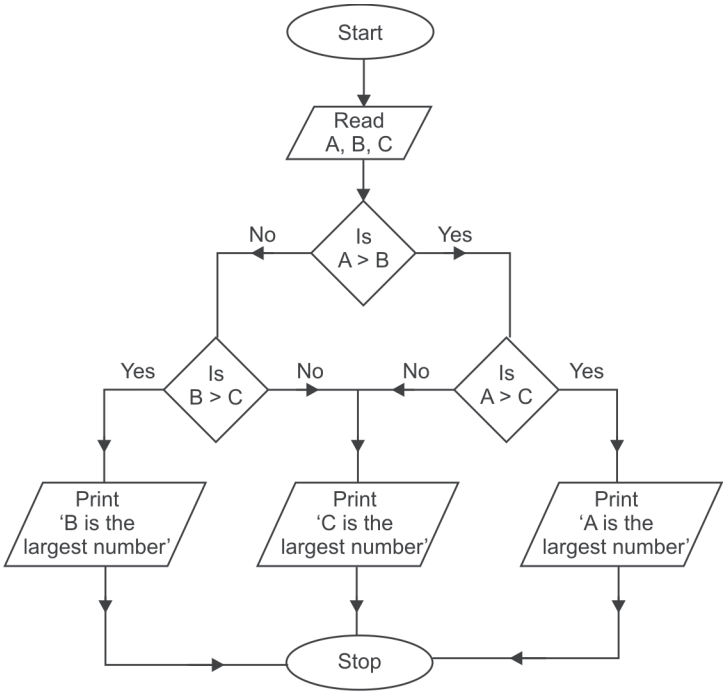

Fig. 1.11: Flowchart to find largest of three numbers

1.3 FEATURES OF C LANGUAGE

C is a procedural (structured) programming language, initially developed by Dennis Ritchie between 1969 and 1973. It was mainly developed at the Bell Telephone Laboratories to develop the UNIX operating system. C language features were derived from an earlier language called "B" (basic combined programming language–BCPL).

Characteristic Features

- C is a structured programming language, i.e. we can divide the larger program into smaller parts using functions and hence easy to understand and modify.
- It is a robust language with rich set of built-in functions and operators that can be used to write complex programs.
- The C compiler combines the capabilities of an assembly language with features of a high-level language.
- Programs written in C are efficient and fast. This is due to its variety of data type and powerful operators.
- C is highly portable, i.e. programs once written can be run on another machines with little or no modification.
- Another important feature of C program is its ability to extend itself.
- A C program is basically a collection of functions that are supported by C library. We can also create our own function and add it to C library.

- C language is the most widely used language in operating systems and embedded systems develop today.
- 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.
- 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.

1.4 CHARACTER SET OF C

Character set of C is a set of alphabets, digits, white space and some special characters as described in Table 1.2.

Table 1.2: List of character set in C

Alphabets	A, B, …, Z (upper case) a, b,…, z (lower case)		
Digits	1, 2 ,3, …, 9		
White space characters	Blank space, new line, horizontal tab, carriage return and form feed		
	Special Characters		
,	Comma	.	Period
;	Semicolon	#	Hash
:	Colon	$	Dollar sign
?	Question mark	%	Percentage sign
‘	Apostrophe	|	Pipeline character
"	Quotation mark	/	Slash
_	Underscore	\	Backslash
&	Ampersand	<	Less than
~	Tilde	>	Greater than
^	Caret	(	Left parenthesis
*	Asterisk	)	Right parenthesis
+	Plus	[	Left bracket
-	Minus	]	Right bracket
{	Left brace	}	Right brace

1.5 C TOKENS

The smallest individual elements or units in a program are called as tokens. C has the following tokens.

- Keywords
- Identifiers
- Constants
- Operators

1.5.1 Keywords

Keywords are also known as reserved words. *These are words which have special meanings to the compiler*. They are meant for doing specific tasks hence called as reserved words. These meaning cannot be changed and hence keywords cannot be used as identifiers for naming functions or variables (identifiers are explained in next section). C compilers support 32 keywords as listed in Table 1.3. All keywords are in lower case and when used should be in lower case as C is a case sensitive language.

Table 1.3: Keywords in C programming

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

1.5.2 Identifiers

Identifiers in C refer to as the **name given to variables and the functions** used in the program. Keywords cannot be used as identifiers. Identifiers are case sensitive, i.e. the identifier amount and amount are different.

Rules for writing an identifier are as follows:

- An Identifier can only have alphanumeric characters (a-z , A-Z, 0–9) and underscore (_).
- The first letter of an identifier should be either a letter or an underscore.
- Usually the length of the letter is no rule on length of an identifier. However, the first 31 characters of identifiers are discriminated by the compiler.
- Keywords are not allowed to be used as identifiers.
- No special character other than underscore is allowed for identifiers.

Few examples of valid identifiers: Amount, _xyz, Student_Name, Stud1, Emp_Dept_No, Num1, Avg_Score5

Few examples of invalid Identifiers:

- 9Amount, invalid since identifier must not begin with number
- Tot-Amount, invalid since identifier can have only underscore (_) as special character, - not allowed
- TotAmount_in$, invalid since identifier can have only underscore (_) as special character, $ not allowed
- Total Amount, invalid since blank space between Total and Amount not allowed in identifier

Difference between keywords and identifiers is shown in Table 1.4.

Table 1.4: Difference between keywords and identifiers

Keywords	*Identifiers*
Keywords are words which have special meanings (fixed meaning) to the compiler	Identifiers are user defined names for variables and functions
Keywords cannot be used as identifiers. All keywords contain lower case alphabets with no digits and no special characters.	Since identifiers are user defined they can have lower and upper case alphabets, digits and only underscore as special character.
Examples: for, while, int, break, sizeof(), etc.	**Examples:** Tot_Amount, sum, name, AB123, GCD, etc.

1.5.3 Constants

Constants define fixed values which do not change during the execution of a program. C supports four types of the constants: Integer constants, character constants, real or floating constants and string constants.

a. Integer Constant

It is a numeric constant value without any fractional or exponential part, i.e. it is an whole number with no decimal point. No commas or blanks are allowed within an integer constant. There are three types of integer constants in C programming: Decimal constant (base 10), octal constant (base 8) and hexadecimal constant (base 16).

i. **A decimal constant** consists of any combination of digits taken from the set 0 through 9

   Valid decimal constants: –5, 10, 0, 10024, etc.

   Invalid decimal constants: 5.5 (decimal point is not allowed)

   55,450 (, (comma) not allowed)

ii. **A hexadecimal integer constant** must begin with either 0x or 0X followed by any combination of digits from the set 0 through 9 and characters from the set A through F (either uppercase or lowercase).

   Valid examples of hexadecimal constant (base 16): Ox4a, 0x51

   Invalid examples of hexadecimal constants:

   45A (invalid since it must begin with Ox or OX)

   Ox45A.30 (invalid since it must not have decimal point)

   Ox12M (invalid since it does not allow M)

iii. **An octal integer constant** must begin with O, followed by any combination of digits from the set 0 through 7.

   Valid examples of octal constant (base 8): O4, O124

   Invalid examples of octal constants:

   O45A (invalid since A is not allowed)

   O45.30 (invalid since it must not have decimal point)

   O198 (invalid since it does not allow digits 8 and 9)

b. Floating Point Constant

It is a base 10 number that contains either a decimal point or an exponent or both. A floating point constant can be represented in either decimal form or exponential form. A floating point constant in a decimal (fractional) form must have at least one digit on either side of decimal point. A floating point in exponent form consists of a mantissa and an exponent. The mantissa itself is represented as a decimal integer constant or a decimal floating point constant in fractional form. The letter E or e and the exponent follow the mantissa. The exponent must be a decimal positive or negative integer.

Examples of valid floating point constants

18.345, 18.3E + 2, 0.1834E + 4, 18.0, 18345.0E-3

Examples of invalid floating point constants

18. 24E + 1.5 (exponent must be positive or negative integer, invalid since exponent is not integer)

18.43E (exponent must be positive or negative integer, invalid since exponent integer value is missing)

1843 (floating point constant must have decimal point, invalid since decimal point is missing)

1,870.95 (special character, (comma) not allowed)

c. Character Constants

Character constants are any valid character of C enclosed in single quotes.

Example of valid character constants: 'A', 'a','1', '(', '$', etc

Examples of invalid character constants: 'Ab' (invalid since it has two characters).

"A" (invalid since it is enclosed within double quotes)

d. String Constants

A string constant consists of any set of character enclosed in double quotes. The last character of every string constant is a special character call null value, i.e. '\0' which indicates the end of the string. For example 'A' and "A" are not one and the same. 'A' is character constant storing only one single character, i.e. capital A. "A" is string constant with two characters, i.e. A followed by '\O' character.

Example of valid string constants: "Ghandhi", "1SI11AT002", "India", "145","*$$$++ %%*", "CProgramming"

Examples of invalid string constants: 'Asha' (invalid since string must be enclosed with double quotes).

Note: More about string is explained in chapter 4 on arrays.

1.5.4 Operators in C

The various operators in C are summarized in Table 1.5. The operators are called unary operators if they operate on only one operand. Binary operators are the operators which operate on two operands. Examples of different types of operators: arithmetic,

Table 1.5: Operators in C

Type of operators	*Examples*
Arithmetic	+ (addition) , _ (subtraction) ,/(division) , * (multiplication), % (modulus operator to get remainder)
Relational operators	>, <, >=, <=, != (not equal to) , = = (equal)
Logical operators	&& (AND), || (OR), ! (NOT)
Increment and decrement operators	++, —
Bitwise operators	& (bitwise AND), | (bitwise OR), ^ (Bitwise XOR), << (left shift), >> (right shift), ~ (binary ones complement unary)
Assignment Operators	=
Shorthand assignment operators	+=, -=, *=, /=, %= , &=, |=, ^=, <<=., >>=
Special operators	? : (ternary operator), sizeof() operator, , (comma operator)

relation, logical, increment and decrement operators, bitwise operators, shorthand assignment operators, and special operators are illustrated in chapter 2.

Arithmetic operators: Arithmetic operators are binary operators used to perform arithmetic operations. The result of each of the arithmetic operator on operand A with value 50 and operand B with value 10 is briefed in Table 1.6.

Relational operators: Relational operators are binary operators used to compare two operands. The outcome of comparisons is either true or false depending on the value of the left and right operands to be compared. The result of each of the relational operator on operand A with value 50 and operand B with value 10 is briefed in Table 1.6.

Logical operators: Logical operators are used when there is a need for comparing two or more conditions together. The examples of the logical operators are briefed in Table 1.6.

Increment and decrement operator: Increment operators are **unary operators** used to increase the value of the variable by 1 and decrement operators are **unary operators** used to decrease the value of the variable by 1. Assume A holds value 15. Outcome of increment operator on A, i.e. A++ is 16. Let A holds value 20. Outcome of decrement operator on A, i.e. A is 19.

Bitwise operators: Bitwise operators work on individual bits. The truth table for bitwise AND, OR and XOR is given in (Table 1.7). [Note truth table for logical AND && and bitwise AND & is same. Similarly truth table for logical OR || and bitwise OR | is same.] Examples and description of various C bitwise operators are briefed in Table 1.8.

Assignment and shorthand assignment operators: Assignment operators are used to assign the value of right hand operand to left hand operand. For example, we need to assign the value 100 to a variable say, 'total'. This is done using assignment operator as shown below [*Note:* With assignment the left operand cannot be a constant].

Total = 100

In addition, C supports a short variant of assignment operator called *compound assignment* or *shorthand assignment*. Shorthand assignment operator combines one of

Table 1.6: Examples of arithmetic operators, relational operators and logical operators

Arithmetic operator (binary operators)	*Use of arithmetic operator*	*Example (assume A = 100, B = 200, C = 4)*
+ (addition)	For adding two operands	A + B = 50 +10 = 60
- (subtraction)	For subtracting two operands	A - B = 50-10 = 40
* (multiplication)	For multiplying two operands	A * B = 50 * 10 = 500
/ (division)	For dividing one operand by another operand	A/ B = 50 / 10 = 5
% (modulus)	For getting remainder after division	A % B = 50 % 10 = 0A%C = 50 % 4 = 2
Relational operator (binary operators)	*Use of relational operator*	*Example (Assume A = 100, B = 200)*
> (greater than)	Returns true if the value of left operand is greater than the value of right operand else returns false	(A > B) returns false (B>A) returns true
< (less than)	Returns true if the value of left operand is less than the value of right operand else returns false	(A < B) returns true (B<A) returns false
>= (greater than or equal to)	Returns true if the value of left operand is greater than or equal to the value of right operand else returns false	(A >= B) returns false (B>=A) returns true
<= (less than or equal to)	Returns true if the value of left operand is less than or equal to the value of right operand else returns false	(A <= B) returns true (B<=A) returns false
== (equal to)	Returns true if the values of two operands are equal else returns false	(A == B) returns false
!= (not equal to)	Returns true if the values of two operands are not equal else returns false	(A != B) returns true
Logical operator	Use of logical operator	Example (Assume A = 100, B = 200, C =40)
&& (logical AND) Binary operators	Returns true if the outcome of all conditions are true Returns false if one or more outcomes of condition is false	(B>A) && (B> C) returns true since both (B>A) and (B>C) are true (A>B) && (A>C) returns false since the condition (A>B) is false
|| (logical Or) Binary operators	Returns false if the outcome of all conditions are false. Returns true even if any one or more than one outcome of condition is true	(C>A) || (C> B) returns false since both (C>A) and (C>C) are false (A>B) || (A>C) returns true since the outcome of condition (A>C) is true
! (logical NOT) **Unary operator**	Used to reverse the outcome of any condition. If the outcome of condition is true, then logical NOT makes it false. If the outcome of condition is false, then logical NOT makes it true	(A>B) returns false !(A>B) returns true (A>C) returns true! (A>C) returns false

Table 1.7: Truth table for bitwise AND, OR and XOR

A	*B*	*A and B (Bitwise AND)*	*A | B (Bitwise OR)*	*A^B (Bitwise XOR)*
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Table 1.8: Examples of bitwise operators in C

Bitwise operators (binary opertors)	*Use of bitwise operator*	*Example*
& (bitwise AND operator)	Outcome of bitwise AND operator on two bits is one only both the bits are one else outcome is 0	Let A = 7, B = 4 A in binary: 1 1 1 B in binary: 1 0 0 (A&B) = 1 0 0 = 4
| (bitwise OR operator)	Outcome of bitwise OR operator on two bits is zero only if both the bits are ZERO else outcome is 1	Let A = 7, B = 4 A in binary: 1 1 1 B in binary: 1 0 0 (A | B) = 1 1 1 = 7
^ (bitwise XOR operator) (binary operator)	Outcome of bitwise XOR operator on two bits is one if both bits are not same else zero if both bits have same value	Let A = 7, B = 4 A in binary: 1 1 1 B in binary: 1 0 0 (A^B) = 0 1 1 = 3
~ (bitwise ones complement) **(unary operator)**	Binary ones complement changes bit 1 to 0 and 0 to 1	Let A = 5 A in binary: 1 0 1 (~A) = 0 1 0 = 2
<< (binary left shift operator)	The bits of left operand are shifted toward left by the number of bits specified in the right operand. **Note:** Left shift by 2 bits is equivalent to multiplication of left operand by 2	Let A = 3 A in binary: 0 1 1 (A<<2) = 1 1 0 = 6
>>(binary right shift operator)	The bits of left operand are shifted toward right by the number of bits specified in the right operand **Note:** Right shift by 2 bits is equivalent to division of left operand by 2	Let A = 6 A in binary: 1 1 0 (A>>2) = 0 1 1 = 3

the arithmetic or bitwise operators with assignment operator. Examples of assignment and shorthand assignment operators are briefed in Table 1.9.

Special operators: Comma operator (,), ternary operator (? :), sizeof() operator are a few of the examples of special operators in C.

i. **Comma operator (,)** is used when we need to have multiple assignment in the same statement.

   Example: If we want to assign three variables say A, B and C three values say 10, 20 and 30 in one statement, then it can be written as follows using comma operator.

Table 1.9: Shorthand assignment operators in C

Shorthand assignment operator	*Use of shorthand assignment operator*	*Examples of short-hand assignment operator*	*Equivalent statement using simple assignment*
+=	Add and assign. The right operand is added to value of left operand and the result is stored in left operand	Let A = 10, B = 5 A + = B Now A = 15, B = 5	A = A + B
-=	Subtract and assign. The right operand is subtracted from to value of left operand and the result is stored in left operand	Let A = 10, B = 5 A -=B Now A = 5, B = 5	A = A - B
*=	Multiply and assign. The right operand is multiplied to value of left operand and the result is stored in left operand	Let A = 10, B = 5 A *=B Now A = 50, B = 5	A = A * B
/=	Divide and assign. The right operand divides the value of left operand and the result is stored in left operand	Let A = 10, B = 5 A /=B Now A = 2, B = 5	
%=	Modulus and assign. The right operand divides the value of left operand and the remainder is stored in left operand	Let A = 10, B = 5 A %=B Now A = 0, B = 5	A = A% B
<<=	Left shift and assignment operator. Shifts lefts the bits of left operand by the number of bits specified in right operand and stores the result in left operand	Let A = 3, B = 2 A <<= B Now A = 6, B = 2	A = A<<B
>>=	Right shift and assignment operator. Shifts right the bits of left operand by the number of bits specified in right operand and stores the result in left operand	Let A = 6, B = 2 A >>= B Now A = 3, B = 2	A = A>>B
&=	Bitwise and (AND)and assignment operator	Let A = 6, B = 2 A&=B Now A = 2, B = 2	A = A &B
|=	Bitwise | (OR)and assignment operator	Let A = 6, B = 2 A|=B Now A = 6, B = 2	A = A |B
^=	Bitwise ^ (XOR) and assignment operator	Let A = 6, B = 2 A^ =B Now A = 4, B = 2	A = A ^B

A = 10, B= 20, C= 10 (instead of writing these three assignment on three separate lines) C program to illustrate the use of comma operators is provided in chapter 2.

ii. **Ternary operators (?):** Commonly called as *conditional operator*. The ternary operand requires three operands: condition expression, Statement1 and Statement2. The condition expression is followed by first operator (?) followed by statment1. Statement1 is followed by second operator (:) followed by Statement2.

The syntax of ternary operator is as follows:

(Condition expression)? Statement1: Statement2

First the condition expression is evaluated to find condition is true or false. If the condition is true then statement1 is executed. If condition expression is false then statement2 is executed.

Suppose we need to find the largest of two numbers, say A = 10, B = 20 using ternary operator.

Large = (A>B) ? A : B

Here A > B is false hence the second statement, i.e. the value of B is assigned to large.

Large = (B> A) ? B : A

Here B> A is true hence the first statement, i.e. the value of B is assigned to large.

Few more C programs to illustrate the use of ternary operators is provided in chapter 2.

Note: A few examples of arithmetic expression representation in C is illustrated in Table 1.10.

Table 1.10: A few examples of arithmetic expression representation in C

Arithmetic expression	*Equivalent representation of arithmetic expression in C*
$\text{Area} = \sqrt{s(s-a)(s-b)(s-c)}$	area = sqrt(s(s-a) *(s-b)*(s-c))
$D = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$	D = (-b + sqrt(b*b - 4*a*c)) / (2*a)
$\frac{5x + 3y}{a + b}$	(5*x+3*y)/ (a+b)
$\sqrt{s(s-a)(s-b)(s-c)}$	sqrt(s* (s-a) * (s-b) * (s-c))
$e\ \lvert x = y - 10 \rvert$	exp(abs(x+y-10))
$\frac{e\sqrt{x} + e\sqrt{y}}{x \sin \sqrt{y}}$	(exp(sqrt(x)) + exp(sqrt(y)))/ (x *sin (sqrt(y)))
$x^{25}+y^{35}$	pow(x,25) + pow(y,35)
$\frac{x}{b+c} + \frac{y}{b-c}$	(x/(b+c))+(y/(b-c))
$a + \frac{b(ad+e)}{b-a} - \frac{c}{d}$	a + (b* ((a*d)+e)) /(b-a) -(c/d)

iii. The sizeof() operator is a unary operator which is used to find the size of operand (i.e. variable or data type) in bytes. sizeof() returns the amount of memory for the operand in bytes. The operand can be primitive or user defined data type. To

understand sizeof() we will first understand the data types in C. C program to illustrate the use of sizeof() are explained in Chapter 2.

1.6 OPERATOR PRECEDENCE AND ASSOCIATIVITY IN C PROGRAMMING LANGUAGE

Operator precedence determines which operator is evaluated first when an expression has more than one operator. For example 10–60/5 would yield –2, because it is evaluated as follows: (60/5) = 12, followed by 10–12 = 2. The reason is that division/ has higher precedence than subtraction (-).

Associativity is used when there are two or more operators of same precedence are present in an expression. For example multiplication and division arithmetic operators have same precedence, lets say we have an expression 10*60/5, this expression would be evaluated as (10*60) = 600, followed by 600/5 = 120, because the associativity is left to right for these operators.

10*60/5 = 600/5 = 12.

Table 1.11 shows all the operators in C with precedence and associativity.

Table 1.11: Operator precedence and associativity in C programming language

Operator	*Meaning of operator*	*Associativity*	*Precedence/ rank*
()	Functional call	Left to right	1
[]	Array element reference		
->	Indirect member selection		
.	Direct member selection (dot operator)		
!	Logical negation	Right to left	2
~	Bitwise(1 ‘s) complement		
+	Unary plus		
-	Unary minus		
++	Increment		
--	Decrement		
&	Dereference Operator (Address)		
*	Pointer reference		
size of	Returns the size of an object		
(type)	Type cast (conversion)		
*	Multiply	Left to right	3
/	Divide		
%	Remainder		
+	Binary plus (Addition)	Left to right	4
-	Binary minus (subtraction)		
<<	Left shift	Left to right	5
>>	Right shift		
<	Less than	Left to right	6
<=	Less than or equal		
>	Greater than		
>=	Greater than or equal		
==	Equal to	Left to right	7
!=	Not equal to		
&	Bitwise AND	Left to right	8

(Contd…)

Table 1.11: Operator precedence and associativity in C programming language *(Contd…)*

Operator	*Meaning of operator*	*Associativity*	*Precedence/ rank*
^	Bitwise exclusive OR	Left to right	9
|	Bitwise OR	Left to right	10
&&	Logical AND	Left to right	11
||	Logical OR	Left to right	12
?:	Conditional Operator	Right to left	13
=	Simple assignment	Right to left	14
*=	Assign product		
/=	Assign quotient		
%=	Assign remainder		
+=	Assign sum		
-=	Assign difference		
&=	Assign bitwise AND		
^=	Assign bitwise XOR		
|=	Assign bitwise OR		
<<=	Assign left shift		
>>=	Assign right shift		
,	Separator of expressions	Left to right	15

1.7 DATA TYPES IN C

Data types can be broadly classified into three categories:

- Fundamental (primitive/primary) data type: Examples: int, float, char, void (represents no data)
- Derived data type: Examples: Arrays, pointers, functions
- User defined data type: Examples: Structures, union, typedef and enum.

i. Fundamental Data Type

The fundamental data type include: Integer, char, float and void. The derived data type include: arrays and pointers. User defined data type include unions, structures, enum and typedef. Arrays are explained in Chapter 4. Structures and unions are explained in Chapter 6. Pointers are explained in Chapter 8. In this chapter, we will understand fundamental data types, and two user defined data types: enum and typedef. The fundamental data types are briefed assuming 16 bit machines.

Integer data types are used to hold positive or negative whole numbers without decimal point. They are further classified as short int, int and long int in both signed and unsigned integers. In 16 bit machine, for signed integer, one bit is reserved for sign (positive or negative0 and remaining 15 bits for integer number. Unsigned integers use all the 16 bits for storing integer number. Short int are used to represent small integer values and requires half the amount of storage compared to regular integer number. Large int are used to represent large integer values and requires double the amount of storage compared to regular integer numbers. The range and size of fundamental data types are briefed in Table 1.12.

Floating data types are used to store real numbers using keyword float (32 bits with 6 digits precision). Double data type is used when real numbers with more precision (64 bits with 14 digits) are needed. The highest precision is provided by long double data type with 80 bits.

Table 1. 12: Size (in bits) and range of fundamental data types: integer, float and char on 16 bit machine

C data types/storage size	*Size in bits*	*Range*
char or signed char	8	–127 to 127
unsigned char	8	0 to 255
int or signed int	16	–32,767 to 32,767
unsigned int	16	0 to 65535
short int or short signed int	8	–127 to 127
short unsigned int	8	0 to 255
long int or long signed int	32	-2,147,483,648 to 2,147,483,647
long unsigned int	32	0 to 4294967295
float	32	3.4 e-38 to 3.4 e+38
double	64	1.7 e-308 to 1.7 e+308
long double	80	3.4 e-4932 to 3.4 e+4932

Char data type are used to store single character type data using 8 bits of space. Unsigned chars have values between 0 and 255. Signed chars have value from –128 to 127.

Void data type has no values. This is usually used with functions to indicate that the called function is not going to return any data.

ii. Derived Data Type

These types of data are derived from primitive data types. The derived data type includes arrays and pointers. Arrays are a collection of similar data type with

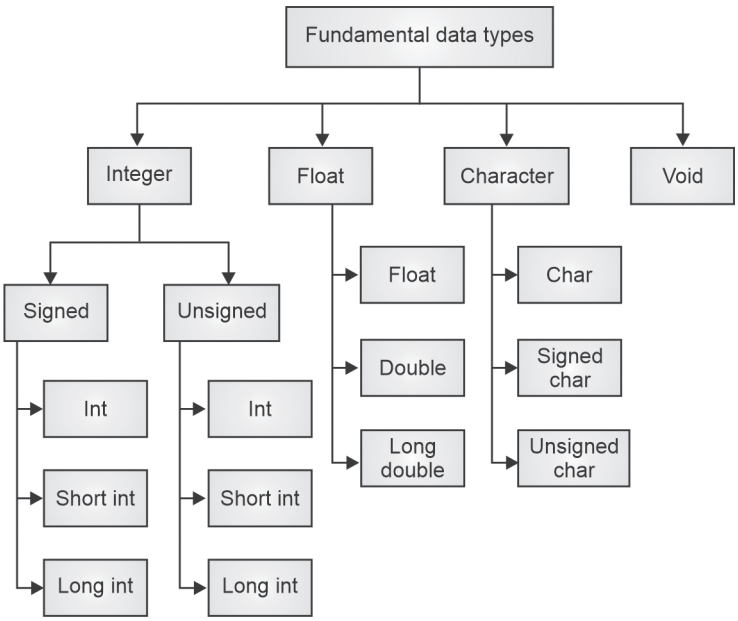

Fig. 1.12: Fundamental/primitive data types in C

contiguous memory allocations. Arrays are explained in chapter 4. Pointers are used to hold the address of another variable. Pointers are discussed in chapter 8.

iii. User Defined Data Type

Sometimes the primitive data type alone may not serve our need of programming. In such cases C provides user to define more complex data types using primitive data types. These complex data types defined by user are called as user defined data types. Structures, union, typedef and enum are examples of user defined data types. Structures and unions are discussed in chapter 6. The enum and typedef user defined data types are explained as follows:

- **The enumerated data type:** enum is a user defined data type. It allows a variable to have one of the values enclosed within set of braces. The collections of values are called as *enumerations constants/enumeration member*. Default value of the first enumerated member is zero. enums are mainly used for improving the readability of code. The following code snippet illustrates enumerated data type.

1.8 VARIABLES

Variables are user defined names used to refer to some location in memory. The data stored in the memory location can be accessed using variables. The value of the constants cannot be manipulated during execution of the program, but during execution of the program the value of the variables can be manipulated. We can declare the variables based on what type of data are to be stored in variables. Different kind of data types are briefed in next section. The rules for naming the variables are same as rules for naming identifiers.

Examples of valid variables: Sum, sum, A, N, n, Add_Num, TotalM123, etc.

Examples of invalid variables: main (since keywords cannot be used as variables), 1Sum (since variable must not begin with numbers), while (since keywords cannot be used as variables), etc.

1.8.1 Declaring Variable and Initializing Variables

Variables must be declared before using variables in C programs. The general syntax for declaring variables is as follows:

Data type, list of variables; where DataType specifies the type of data (fundamental/ derived/user defined data type) and list of variables is set of one or more variables separated by comma.

Suppose we need to find sum and average of three integer numbers. Firstly we need to declare four integer variables (three integer variables to hold 3 integers values and one more integer variable to store the sum of three integer variables) and one float type variable to hold the average of three integer numbers as follows.

```
int A, B, C, sum ;  // declare four integers data type variables
float average; // declare average as float data type variable
```

Initializing variables means assign some values to the declared variables. Assignment of values can be dome usng equal operator (=). The general syntax is as follows:

Variable = constant or variable.

For examples to initialized three integer variables A, B, C and variable sum to find sum and average is illustrated as follows:

A = 10;

B= 5;

C= 10;

Sum = 0;

Sum = A+B+C; // assigning sum of three variables to sum

Average = Sum/3;

Suppose we want to same value 10 to both A and B then it can be done in different ways

A = 10;

B = A or B =10 are equivalent; the four variables can be assigned in the same line using comma operator as follows:

A = 10, B= 5, C = 10, Sum = 0;

Variables can be assigned during declaration of variables also as follows:

int A= 10, B= 5, C = 10, Sum = 0;

Program 1.1: Write C program to illustrate enumerated data type.

```
#include <stdio.h>
enum WeekDay {Sunday, Monday, Tuesday, Wednesday, Thursday,
Friday, Saturday};
int main()
{
  enum WeekDay D =Friday;
  printf("The day number stored in D is %d", D);
  return 0;
}
```

Output

The day number stored in D is 5

Note: If user does not explicitly assign values to enum names, the compiler by default assigns values starting from 0. For example, in the Program 1, sunday gets value 0, Monday gets 1, and hence Friday assigned to D results in output 5.

Program 1.2: Write C program to illustrate use of enumerated data type.

```
#include <stdio.h>
enum Course {Mechanical, Electrical, Electronics, Computer,
Chemical, Instrumentation, Information Science, Architecture} ;
// define Course as enumerated data type
int main()
```

```
{
     enum Course C =Architecture;
//declare C as enum Course type & initialize with Architecture
     printf("The course number stored in C is %d", D);
      return 0;
}
```

Output

The course number stored in C is 7

Program 1.3: Write C program to illustrate use of enumerated data type.

```
#include <stdio.h>
enum WeekDay {Sunday = 1, Monday, Tuesday, Wednesday, Thursday,
Friday, Saturday};
int main()
{
  enum WeekDay D =Friday;
  printf("The day number stored in Saturday is %d", D+1);
  return 0;
}
```

Output

The day number stored in Saturday is 7

Note: If user has explicitly assigned value 1 to Sunday, hence the compiler by assigns values starting from 1. For example, in the Program 1, Sunday gets value 1, Monday gets 2, and Friday assigned to 6 and hence Saturday results in Friday + 1, i.e. 6+1 = 7.

Program 1.4: Write C program to illustrate use of enumerated data type.

```
#include <stdio.h>
enum WeekDay {Sunday = 1, Monday, Tuesday, Wednesday= 10,
Thursday, Friday, Saturday};

int main()
{
  enum WeekDay D =Friday;
  printf("The day number stored in D is %d", D);
  return 0;
}
```

Output

The day number stored in D is 12

Note: If user has explicitly assigned value 1 to Sunday, hence the compiler by assigns values starting from 1. For example, in the Program 1. Sunday gets value 1, Monday gets 2, and Tuesday gets 3. But since Wednesday has been assigned explicitly value 10, Thursday gets value 11 and hence Friday assigned to D results in output 12.

The typedef keyword

The typedef keyword allows the programmer to create new data types. One of the advantages of typedef is to improve the readability of code.

For example

typedef long int LI;

creates a new datatype called LI which is shorthand for long int. You can use this new type to declare long int variables as follows:
LI x;

Example in program 1.1, the statement

typedef enum WeekDay WDay ;

Program 1.5: Creates a new datatype called WDay which is shorthand for enum WeekDay.

```
#include <stdio.h>
enum WeekDay {Sunday, Monday, Tuesday, Wednesday, Thursday,
Friday, Saturday};
typedef enum WeekDay WDay;
// use of typedef to define new data type WDay
int main()
{
  WDay D =Friday;
  // use of user defined data type WDay to declare variable D
  printf("The day number stored in D is %d", D);
  return 0;
}
```

Output

The day number stored in D is 5

SUMMARY

- Providing basic understanding of the design of the program in terms of pseudocode and flowcharts. Pseudocode explains the stepwise representation of program in natural language. Flowchart contains graphical representation of program containing the symbols used for writing flowchart along with its advantages and disadvantages. Sample examples are added to understand the basics of pseudocode and flowcharts.
- The introduction to C language is briefly explained stating the inventor of the C language, year of introduction and the basic characteristics of C language.
- The character set including the special characters supported by C language is tabulated with the symbols.
- The smallest individual unit of the program is detailed by considering the varieties of the tokens like keywords, identifiers, constants and operators. 32 keywords supported by C compiler are tabulated. Basics of identifiers and rules defined to declare identifiers are explained with appropriate valid and invalid example. Four

different types of constants that are integer, character, floating point and string are described with valid examples. Different types of operators, i.e. arithmetic, relational operators, logical operators, increment and decrement operators, bitwise operators, assignment operators, shorthand operators and special operators are explained with examples and even the associativity and precedence of the operators during expression evaluation is also discussed.

- The data type in C language alongwith its categories, i.e. fundamental data type, derived data type and user defined data type is explained with its size and range. Simple C programs covering the concepts of data types are also given.
- The basic ways to declare and define variables is described with their syntax.

QUIZ QUESTIONS

1. **Define Pseudocode.**
2. **Define flowchart.**
3. **Tell the advantages of flowchart.**
4. **Tell the advantages of pseudocode.**
5. **Which symbol is used to represent start and stop in flowchart?**
6. **Which symbol is used to represent input and output from the device?**
7. **Which symbol is used to represent flowline?**
8. **Design pseudocode and flowchart to display "Welcome to the world of C".**
9. **C is a _____ language.**
10. **Who invented C language?**
11. **BCPL stands for _____**
12. **Which of the following are valid and invalid identifiers in C with proper reasons?**
    a. area of triangle
    b. main
    c. for
    d. 123ijk
    e. if
    f. a*b
    g. _computer
    h. Name&usn
    i. –SI
    j. Dist in KM
    k. Float
    l. Simple-interest

13. The C language was invented in the year _____

14. Recall any four special operators supported by C compiler.

15. The smallest individual elements or units in a program are known as _____

16. List any four rules for declaring identifiers.

17. Differentiate between keyword and identifier.

18. List any four arithmetic operators.

19. List any four relational operators.

20. List any four logical operators.

21. List any four increment and decrement operators.

22. List any four bitwise operators.

23. List any four short hand assignment operators.

24. List any two special operators.

25. Omit the truth table for bitwise AND operator.

26. Omit the truth table for bitwise OR operator.

27. Omit the truth table for bitwise XOR operator.

28. What is the syntax for ternary operator?

29. What is the associativity of function call operator?

30. Which operator has highest precedence?

31. Recall the range of integer data type.

32. Recall the range of floating point data type.

33. Recall the range of double datatype.

34. _____ keyword is used to create new datatype.

35. Write equivalent C expression

a. $d = b^2–4ac$

b. $d = \sqrt{(x2-x1)^2-(y2-y1)^2}$

c. $TE = \sum_{i=1}^{i-n}\left[A(i)+c(i)\right]$

d. $ax^2 + bx + c$

e. $(m + n)(a + b)$

f. $8.8(a + b)2/c – 2a/(a + b)(1/m)$

g. $2v + 6.22(c + d)/g + v$

h. $a + bc + bp/d + et$

36. If x = 5 and y = 6, what is the value of x and y after each of the following expressions.

a. ++x + 5
b. x++ + y
c. —x – y--
d. x++ – y—

37. Determine the end result of the following expressions.

a. 3.4 + 9.4 * 3.3
b. 2.9 + 10.5/2
c. 6.0/(4.5 + 4.0 * 1.2)
d. 4.9 * (4.0 + 25/6)
e. a = ++2

38. If a = 3000, what is the value of following expressions?

a. a%10
b. a/10
c. (a/10)%10
d. a/100
e. (a/100)%10

39. Find the errors in the following C statements.

i.
```
int main
{
      return 0;
}
```

ii.
```
#include(stdio.h)
int main()
{
      return 0;
}
```

iii.
```
int main()
{
      printf("%7.2",avg);
      printf("7.2f",num1);
      printf("$3d",num2); return 0;
}
```

iv.
```
int main(void)
{
      a int;
      b float, double; c,d char;
  return 0;
}
```

40. Point out the errors, if any, in the following C statements.

```
int = 314.562*150;
name='Ajay';
```

```
3.14*r*r=area;
k=a*b+c (2.5a+b);
m_inst=rate of interest* amount in rs;
si=p*t*r/100;
area=3.14*r**2;
int 1,2,3;
int 7.8,9.2,34.5;
age='123' ;
address=SIT TUMKUR;
```

41. Evaluate the following expressions.

```
g=big/2+big*4/big-big+abc/3;(abc=1.5, big=3, assume g to be float)
on=ink*act/2+3/2*act+2+tig; (ink=3, act=2, tig=3.2, assume on to be int)
s=qui*add/4-6/2+2/3*6/god;(qui=2, add=4, god=3, assume s to be an int)
hi=s+i-t/e;  (s=2, i=7, t=19, e=5, assume hi to be float)
a=5/2; (assume a to be int)
a=7/22*(3.14+2)*3/5    (assume a to be float)
```

IMPORTANT QUESTIONS

1. **List features of C language.**
2. **What are tokens in C language? Explain the kinds of tokens.**
3. **What are keywords?**
4. **Differentiate between keywords and identifiers and give two examples for each.**
5. **Define identifiers, constants, keywords.**
6. **Explain the various data types available in C.**
7. **Explain the different types of operators in C with appropriate examples.**
8. **Explain relation and logical operators in C.**
9. **Explain bitwise operators in C.**
10. **Explain any two special operators in C with examples.**
11. **What are enumerated data types?**
12. **What is difference between fundamental data type, derived data type and user defined data type?**
13. **Explain constants in C.**
14. **With an example explain ternary operator.**
15. **Write pseudocode and draw flowchart for finding area of triangle.**
16. **Write pseudocode and draw flowchart for finding sum and average of five subject marks.**
17. **Develop pseudocode and draw flowchart for finding smallest of two numbers.**

18. Write C expressions corresponding to the following (assume all quantities are of same type).

$A = \frac{5x+3y}{a+b}$

$B = \sqrt{s(s-a)(s-b)(s-c)}$

$C = e$

$D = e^{|x+y-10|}$

$X = \frac{e\sqrt{x}+e\sqrt{y}}{x \sin\sqrt{y}}$

$X = \frac{-b+\sqrt{b^2-4ac}}{2a}\frac{x}{b+c}+\frac{y}{b-c}$

19. Convert the following mathematical expression into C expressions:

a. $\frac{x}{a+b}+\frac{y}{a-b}$

b. $a+\frac{b(ad+e)}{b-c}-\frac{c}{d}$

c. (x/b+C)+y/(b-c)

d. area=sqrt(s(s-a)(s-b)(s-c))

e. X=-b+sqrt(b*b-4*a*c)/2*a

20. Define algorithm. Write an algorithm to find the area and perimeter of a rectangle.

21. Convert the following mathematical expression into C equivalent:

i. $\text{area} = \sqrt{s(s-a)(s-b)(s-c)}$

ii. $X = \frac{-b+\sqrt{b^2-4ac}}{2a}\frac{5x+3y}{a+b}$

22. Explain the following operators:

i. Unary
ii. Bitwise
iii. Conditional

23. Draw the flowchart and write a C program to compute simple interest.

24. What is pseudocode? Explain with an example.

25. Explain any 5 operators used in C language.

26. What is type conversion? Eplain 2 types of type conversion with examples.

27. Define variable, contstant, assocoativity, precedence.

28. What is an identifier? What are the rules to construct identifier?

29. Classify the following as valid/invalid identifiers : num2, $num1, +add, a_2.

30. Write the output of the following C code.

i.

```
void main()
{     int a=5,b=2,res1;
      float  f1=5.0,f2=2.0,res2;
      res1=5/2.0+a/2+a/b;
      res2=f1/2*f1-f2;
      printf("res1=%dres2=%f",res1,res2);
}
```

ii.

```
void main()
{
      int i=5,=6,m,n;
      m=++i+j++;
      n=-i+-;
      printf("m=%d n=%d",m,n);
}
```

31. What are data types? Mention the different data types supported by 'C' language, giving an example to each.

32. What is a token? What are different types of token available in C language? Explain.

33. What is the value of 'x' in following code segments? Justify your answers.

i.

```
int a,b;
        float x;
        a=4;
        b=5;
        x=b/a;
```

ii.

```
int a,b;
      float x;
      a=4;
      b=5;
      x=(float)b/a;
```

34. Write C expressions corresponding to the following (assume all quantities are of same type).

i. $A = \frac{5x + 3y}{a + b}$

ii. $B = \sqrt{s(s-a)(s-b)(s-c)}$

iii. $C = e^{|x+y-10|}$

iv. $D = X25 + y35$

v. $X = \dfrac{e\sqrt{x} + e\sqrt{y}}{x \sin\sqrt{y}}$

vi. $x = \dfrac{-b + \sqrt{b^2 - 4ac}}{2a} \dfrac{x}{b+a} + \dfrac{y}{b+c}$

35. What is a variable? Explain the rules for constructing variables in C language. Give examples for valid and invalid variable.

36. Convert the following mathematical expression into C expressions.

i. $\dfrac{x}{b+c} + \dfrac{y}{b-c}$

ii. $\dfrac{b(ad+e)}{b-a} - \dfrac{c}{d}$

37. Write a pseudocode to find sum and average of given 3 numbers.

38. Evaluate the following expressions:

i. `100%20<=20-5+100%10-20==5>=1!=20`

ii. `a+=b*=c-=5 where a=3 b=5 and c=8.`