2. INTRODUCTION TO C LANGUAGE

BRIEF HISTORY OF DENNIS MACALISTAIR RITCHIE - FATHER OF C AND UNIX

Dennis MacAlistair Ritchie (1941 - 2011) was an American Computer Scientist also popularly known as “dmr” is best known as the creator of the C Programming Language and co-creator of UNIX along with his colleague Ken Thompson at Bell Laboratories.

Education:

1. Harvard University, Bachelor's degrees in physics and applied mathematics, 1963
2. Harvard University, PhD, 1968 Today, C remains the second most popular programming language in the world.

2.1 BRIEF HISTORY OF C PROGRAMMING LANGUAGE

1. C was developed at Bell Laboratories in 1972 by Dennis Ritchie.
2. C is a general-purpose computer programming language and was the descendent of CPL (Combined Programming Language and BCPL (Basic CPL).
3. Dennis Ritchie wrote the improved and portable version named C that could run on any machine.
4. By virtue of C's portability, UNIX was rewritten in 1973 was also a portable operating system that could work on different computers.

2.1.1 Significant Features of C Language

1. C is a powerful, flexible language that provides fast program execution.
2. C is a Procedural Language i.e. the programmer is required to provide step-by-step instructions for the CPU (central processing unit).
3. The success of C is due to its simplicity, efficiency, flexibility and small memory requirements.
4. **Low Level features**: C's power and fast program execution come from its ability to access low level commands, similar to assembly language, but with high level syntax.
5. **Portability**: C programs are portable i.e. they can run on any compiler with little or no modification. **Compiler and Preprocessor** makes it possible for C program to run it on different PC.
6. **Bit Manipulation**: C Programs can be manipulated using bits and it provides wide variety of **bit manipulation operators**.

7. **Modular Programming**: It is a software design technique that increases the extent to which software is composed of separate parts, called **modules**. C program consist of different modules that are integrated together to form a complete program.
8. **Efficient Usage of Pointers**: C supports efficient use of pointers and pointers has direct access to memory.
9. **Standard Library Concept**:
10. **Powerful and varied repertoire of Operators**
11. **Elegant Syntax**:
12. **Ready Access to Hardware when needed**:
13. **Structured Programming Language:** C supports

2.1.2 Areas of Application

The C programming language is used in many different areas of application, but the most prolific area is UNIX operating system applications.

- Computer Games
- System level Programming – making Operating System Assemblers, Compilers, Interpreter, Cross-Compilers, Text Editors, and Device Drivers.
- Application Development – e.g. making Reservation System, Library System, Inventory Control System etc.
- Writing Embedded Software/firmware for various electronics, industrial and communication products.
- Used in developing verification software, test code, and simulators for various applications and hardware products.

2.2 BASIC STRUCTURE OF C LANGUAGE

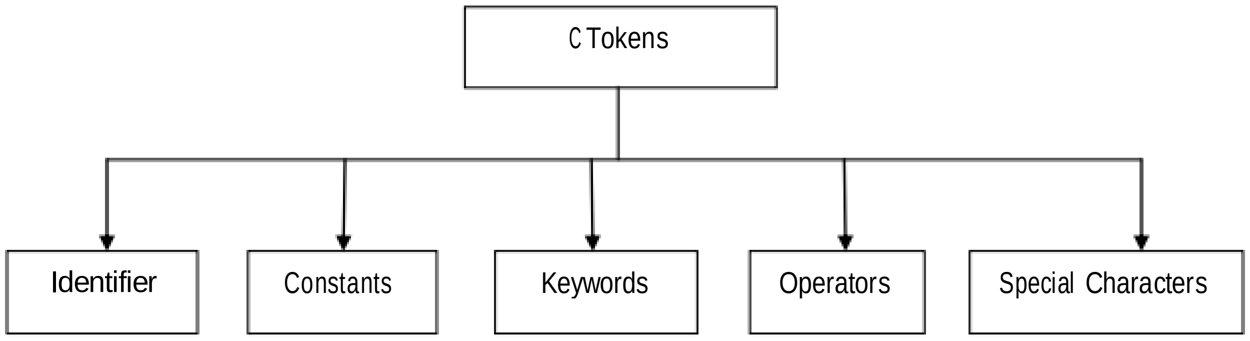
Documentation Section (Optional)	**The documentations section** consists of comment lines giving the name of the program, the author and other details which the programmer would like to use later.**Single Line Comment**: // This is a sample program **Multiple Line Comment**: * This is a sample Program*\
Preprocessor Statements	The preprocessor statements begin with # symbol and are also called the preprocessor directive. These statements instruct the compiler to include C preprocessors such as **header files** and **symbolic constants or macros** before compiling the C program. e.g.

	# include<stdio.h> # include<conio.h> } Header Files # define PI 3.142 # define MAX 100 } Symbolic Constants
Global Declaration Section(Optional)	The variables are declared before the main () functions as well as user defined functions are called global variables.

main() Function Section (must)	The C program execution starts with main() function. The main() function should be written in small (lowercase) letters and it should not be terminated by semicolon.
	
	Syntax:
	main()
	{
	Local Declarations;
	Processing
	Statements;
	}

User Defined Functions (Optional)	This section contains all the user defined functions that are called in the main() function. User defined functions are generally placed immediately before /after the main function.

2.3 C TOKENS



A token is the basic building block of a C program which can be recognized by the compiler.

2.3.1 Identifier

An identifier is used to give a name to an object. An identifier refers to the names of variables, constants, functions, arrays, types (typedefs, structs, unions, enums), type members and macros etc. These are user-defined names and consist of sequence of letters and digits, with a letter or underscore as a first character. Both uppercase and lowercase letters are permitted.

Rules for Identifiers:

1. First character must be an alphabet or underscore.
2. Identifier names must consist of only letters, digits or underscore.
3. It must not be a keyword.
4. It must not contain white space, operators and special
5. characters. There is no rule for the length of an identifier.
6. The first 31 characters of an identifier are discriminated by the compiler.

2.3.2 Constants

A constant is a quantity that doesn't change. There are mainly to types of constants.

1. Numeric Constants
   a. Integer Constants e.g. 5, 9, 35
   b. Floating Point Constants e.g. 5.25,0.22E-5, -2.0
2. Non numeric or Character Constants
   a. Single Character Constants e.g. 'a','x','p'
   b. String Constants e.g. "hello", "sample","good"

2.3.3 Keywords

These are reserved words used in programming. Each keyword has fixed meaning and that cannot be changed by user.

e.g. int n;

Here **int** is keyword and it indicates **n** is of type integer.

Here is the list of all keywords predefined by ANSI C.

Standard Keywords in C Language:

Keywords used while declaring variables	**int** **char** **float** **double** **long** **short** **signed** **unsigned** **volatile**

	void **const**

Control flow related Keywords	**if**
	else
	switch
	case
	default
	do
	while
	for
	return
	break
	continue
	goto

Storage classes related Keywords	**auto** **static** **register** **extern**
User defined data type related Keywords	**enum** **typedef** **struct** **union**
Special operator related Keyword	**sizeof**

2.4 OPERATORS

An operator is used to describe an operation applied to one or several objects.

1. Arithmetic Operators
2. Increment and Decrement Operators
3. Assignment Operators
4. Relational Operators
5. Logical Operators
6. Conditional operators
7. Bitwise Operators
8. Special Operators

Arithmetic Operators:

Operator	Meaning of Operator
+	Addition or unary plus
-	Subtraction or unary minus
*	Multiplication
/	Division
%	Remainder after division (modulo division)

Increment and Decrement Operators:

Operator	Meaning of Operator
++	Increment operator (unary) a++; //post increment ++a; //pre increment
--	Decrement operator (unary) a--; //post decrement [illegible]

Assignment Operators:

There are two types of assignment perators.

1. Simple Assignment
2. Compound Assignment

Operator	Meaning of Operator
Simple Assignment	
=	Assignment Operator e.g. x=5 5 is assigned to x
Compound Assignment	
+=	a += b is equivalent to a = a + b
-=	a -= b is equivalent to a = a - b
*=	a *= b is equivalent to a = a * b
/=	a /= b is equivalent to a = a / b
%=	a %= b is equivalent to a = a % b
&=	a &= b is equivalent to a = a & b
|=	a |= b is equivalent to a = a | b

^=	a ^= b is equivalent to a = a ^ b
<<=	a <<= b is equivalent to a = a << b
>>=	a >>= b is equivalent to a = a >> b

Relational Operators:

Relational Operators are used to check the relationship between two operands. If the relation is true, it returns value 1 and if the relation is false, it returns value 0. Relational operators are used in decision making and loops in C programming.

Operator	Meaning of Operator
==	Equal to e.g. 5 == 3 returns false or 0
>	Greater than e.g. 5 > 3 returns true or 1
<	Less than e.g. 5 < 3 returns false or 0
!=	Not equal to 5 != 3 returns true or 1
>=	Greater than or equal to e.g. 5 >= 3 returns true or 1
<=	Less than or equal to e.g. 5 <= 3 returns false or 0

Logical Operators:

Logical operators are used to combine expressions containing relation operators. In C, there are 3 logical operators:

Operator	Meaning of Operator
&&	Logical AND e.g. if c=5 and d=2 then ((c == 5) && (d > 5)) returns false or 0
||	Logical OR e.g. if c=5 and d=2 then ((c == 5) || (d > 5)) returns true or 1
!	Logical NOT e.g. if c=5 then !(c == 5) returns false or 0

Conditional Operator:

Conditional operator takes three operands and consists of two symbols **?** and **:** .
Conditional operators are used for decision making in C.
For example: c = (c > 0) ? 10 : 20;
If c is greater than 0, value of c will be 10 but, if c is less than 0, value of c will be 20.

Bitwise Operators:

A bitwise operator works on each bit of data.

Operator	Meaning of Operator
&	Bitwise AND
|	Bitwise OR
^	Bitwise Exclusive OR
<<	Shift Left
>>	Shift Right
~	Bitwise Complement (One's Complement)

Special Operators:

Operator	Meaning of Operator
,	Comma operators are used to separate the variables or expressions. Example: int a,b,c =5*3;
sizeof	It is a unary operator which is used in finding the size of data type, constant, arrays, structure etc. e.g. sizeof int = 2 bytes sizeof float = 4 bytes (based on MS-DOS OS) sizeof double = 8 bytes sizeof char = 1 byte
.	Access Operator is used to access any member of structure or union.
→	This operator is used to access any member of structure or union using pointer.

2.5 OPERATOR PRECEDENCE TABLE

Operator	Description	Associativity
( )	Parentheses (function call) (see Note 1)	left-to-right

[ ] . ++ --	Brackets (array subscript) Member selection via object name Member selection via pointer	
+ - (*type*)	Prefix increment/decrement Unary plus/minus Logical negation/bitwise complement Cast (convert value to temporary value of *type*) Dereference Address (of operand) Determine size in bytes on this implementation	right-to-left
* / %	Multiplication/division/modulus	left-to-right
+ -	Addition/subtraction	left-to-right
<< >>	Bitwise shift left, Bitwise shift right	left-to-right
< <= > >=	Relational less than/less than or equal to Relational greater than/greater than or equal to	left-to-right
== !=	Relational is equal to/is not equal to	left-to-right
&	Bitwise AND	left-to-right
^	Bitwise exclusive OR	left-to-right
|	Bitwise inclusive OR	left-to-right
&&	Logical AND	left-to-right
||	Logical OR	left-to-right
? :	Ternary conditional	right-to-left
= ^= |=	Assignment Addition/subtraction assignment Multiplication/division assignment Modulus/bitwise AND assignment Bitwise exclusive/inclusive OR assignment Bitwise shift left/right assignment	right-to-left
,	Comma (separate expressions)	left-to-right

Note 1:

Parentheses are also used to group sub-expressions to force a different precedence; such parenthetical expressions can be nested and are evaluated from inner to outer.

Note 2:

Postfix increment/decrement have high precedence, but the actual increment or decrement of the operand is delayed (to be accomplished sometime before the statement completes execution). So in the statement **y = x * z++;** the current value of **z** is used to evaluate the expression (*i.e.,* **z++** evaluates to **z**) and **z** only incremented after all else is done.

2.6 EXPRESSIONS AND EVALUATION

2.6.1 Rules for Evaluation of Arithmetic expressions

1. All expressions in parentheses must be evaluated separately. Nested parenthesized expressions must be evaluated from the inside out, with the innermost expressions evaluated first.
2. The operator precedence rule:
Operators in the same sub expression are evaluated in the following order:
Unary + and – are evaluated first.
*, /, % are evaluated next .
binary operator + and – are evaluated last .
3. The associativity rule:
Unary operators in the same sub expression and at the same precedence level (such as + and -) are evaluated right to left (right associativity).
Binary operators in the same sub expression and at the same precedence level are evaluated left to right (left associativity).

Example:
Consider the expression
-a + (c + b * (c + a) / c – b / a) + a – b / 2
In the above expression the innermost parenthesis (c + a) is evaluated first, then the next innermost parenthesis (c + b * (c + a) / c – b / a) is evaluated. In this sub-expression, first b * (c + a) is evaluated then b * (c + a) / c is evaluated, then b / a is evaluated, then the whole expression is evaluated. Then –a is evaluated, then b / 2 is evaluated and finally the whole expression is evaluated.

2.6.2 Type Conversions

C allows types to be mixed in expressions, and permits operations that result in type conversions happening implicitly.In the C programming language, a type conversion is the conversion two different sorts of data type into a common form, in order for them to be manipulated. There are different basic data types, such as int, char, float, double; there are also some user defined data types such as structures, arrays, etc. If the operator is taking operands of different data types, then they are converted to a common data types by certain rules. Generally, automatic conversions are those which can convert a *narrower* operand into a *wider* one without loss of information. For example, converting an integer to floating point in examples like *float + integer* (on 64-bit machine). A *char* is simply a small integer, so chars may be freely used in arithmetic expressions.

2.7 INPUT AND OUTPUT STATEMENTS

Input : In any programming language input means to feed some data into program. This can be given in the form of file or from command line. C programming language provides a set of built- in functions to read given input and feed it to the program as per requirement.

Output : In any programming language output means to display some data on screen, printer or in any file. C programming language provides a set of built-in functions to output required data.

Reading and writing Characters: The simplest of the console I/O functions are getche (), which reads a character from the keyboard, and putchar (), which prints a character to the screen. The getche () function waits until a key is pressed and then returns its value. The key pressed is also echoed to the screen automatically. The putchar () function will write its character argument to the screen at the current cursor position. The prototypes for getche () and putchar () are shown here:

```
int getche (void);
int putchar (int c);
```

The header file for getche () and putchar () is in CONIO.H.

The following programs inputs characters from the keyboard and prints them in reverse case. That is, uppercase prints as lowercase, the lowercase prints as uppercase. The program halts when a period is typed. The header file CTYPE.H is required by the islower() library function, which returns true if its argument is lowercase and false if it is not.

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

main(void)
{
        char ch;
        printf (“enter chars, enter a period to stop\n”);
        do
        {
                ch = getche ();
                if ( islower (ch) )
                        putchar (toupper (ch));
                else
                        putchar (tolower (ch));
        } while (ch! = ‘.’);                    /* use a period to stop */
        return 0;
}
```

There are two important variations on getche().

- The first is getchar(), which is the original, UNIX-based character input function.
    - ❑ The trouble with getchar() is that it buffers input until a carriage return is entered. The reason for this is that the original UNIX systems line-buffered terminal input, i.e., you had to hit a carriage return for anything you had just typed to actually be sent to the computer.
    - ❑ The getchar() function uses the STDIO.H header file.
- A second, more useful, variation on getche() is getch(), which operates precisely like getche () except that the character you type is not echoed to the screen. It

CONIO.H header.

Reading and writing Strings:

On the next step, the functions gets() and puts() enable us to read and write strings of characters at the console.

The gets() function reads a string of characters entered at the keyboard and places them at the address pointed to by its argument. We can type characters at the keyboard until we strike a carriage return. The carriage return does not become part of the string; instead a null terminator is placed at the end and gets() returns. Typing mistakes can be corrected prior to striking ENTER. The prototype for gets() is:

```
char* gets (char *str);
```

Where, str is a character array that receives the characters input by the user. Its prototype is found in STDIO.H. The following program reads a string into the array str and prints its length.

```
# include <stdio.h>
# include <string.h>
main(void)
{
        char str[80];
        gets (str);
        printf (“length is %d”, strlen(str));
        return 0;
}
```

The puts() function writes its string argument to the screen followed by a newline. Its prototype is.

```
int puts (char str);
```

It recognizes the same backslash codes as printf(), such as “\t” for tab. It cannot output numbers or do format conversions. Therefore, puts() takes up less space and runs faster than printf(). Hence, the puts() function is often used when it is important to have highly optimized code. The puts() function returns a non negative value if successful, EOF otherwise. The following statement writes “hello” on the screen.

```
puts (“hello”);
```

The puts() function uses the STDIO.H header file.

Basic console I/O functions:

Function	**Operation**
getchar()	Reads a character from the keyboard and waits for carriage return
getche()	Reads a character with echo and does not waits for carriage return
getch()	Reads a character from the keyboard with out echo and not waits for carriage return
Putchar()	Writes a character to the screen
gets()	Reads a string from the keyboard

puts()	Writes a string to the screen

Distinguishion between getchar() and gets() functions:

getchar()	gets()
Used to receive a single character.	Used to receive a single string, white spaces and blanks.
Does not require any argument.	It requires a single argument.

Reading Character Data in a C Program

All data that is entered into a C program by using the scanf function enters the computer through a special storage area in memory called the **standard input buffer** (or **stdin**). A user's keystrokes (including the new line character \n generated when the Enter key is pressed) are stored in this buffer, which can then be read using functions such as scanf. When numbers are read from this area, the function converts the keystrokes stored in the buffer (except for the \n) into the type of data specified by the control string (such as "%f") and stores it in the memory location indicated by the second parameter in the scanf function call (the variable's address). The \n remains in the buffer. This can cause a problem if the next scanf statement is intended to read a *character* from the buffer. The program will mistakenly read the remaining \n as the character pressed by the user and then proceed to the next statement, never even allowing the user to enter a character of their own.

You can solve this problem when writing C statements to read *character* data from the keyboard by adding a call to a special function named **fflush** that clears all characters (including \n's) from the given input buffer. The statement would be place ahead of each statement in your program used to input characters, such as:

```
fflush(stdin); scanf("%c", &A);
        or
fflush(stdin); A=getchar();
```

Formatted Console I/O (printf() and scanf()):

printf() function

This is one of the most frequently used functions in C for output.

```
#include <stdio.h>

main()
{
  int dec = 5;
  char str[] = "abc";
  char ch = 's';
  float pi = 3.14;

  printf("%d %s %f %c\n", dec, str, pi,  ch);
}
```

The output of the above would be:

```
    5 abc 3.140000 c
```

Here %d is being used to print an integer, %s is being usedto print a string, %f is being used to print a float and %c is being used to print a character.

scanf() function

This is the function which can be used to to read an input from the command line.

```
#include <stdio.h>

main()
{
  int x;
  int args;

  printf("Enter an integer: ");
  if (( args = scanf("%d", &x)) == 0) {
      printf("Error: not an integer\n");
  } else {
      printf("Read in %d\n", x);
  }

}
```

Here %d is being used to read an integer value and we are passing &x to store the vale read input. Here &indicates the address of variable x.

This program will prompt you to enter a value. Whatever value you will enter at command prompt that will be output at the screen using printf() function. If you enter a non-integer value then it will display an error message.

```
Enter an integer: 20
Read in 20
```

Questions on Variables, Constant, Data Types, Arithmetic Expressions, Input & Output Functions:	
Q1)	What is input function?
Ans:	It is used to read values from input device.
Q2)	What is output function?
Ans:	It is used to display output of a program.
Q3)	What is a Variable?
Ans:	A variable is a data name that is used to store a value.
Q4)	What is Constant?
Ans:	Constant is a fixed values that do not change during the execution of a program.
Q5)	What is data type?
Ans:	This indicates type of data.
Q6)	What are the rules to define a variable?
Ans:	• It must begin with a letter, or with underscore. • It must not be a keyword. • It must not contain white space or special characters. It can have digits at middle or end.
Q7)	Give few examples for valid variable names?
Ans:	T_raise, delhi, x1, ph_value, mark, sum1, distance
Q8)	Is char a valid variable name?
Ans:	No, char is a keyword
Q9)	Is price$ a valid variable name?
Ans:	NOT VALID, dollar sign is illegal.
Q10)	Is 'group one' a valid variable name?
Ans:	NOT VALID blank space not permitted.
Q11)	Is int_type a valid variable name?
Ans:	Valid, keyword may be a part of a name.
Q12)	What are the data types that ANSI c supports?
Ans:	ANSI c supports three classes of data types .they are 1. Primary (or fundamental/ primitive)data type 2. Derived data types 3. User defined data types.

Q13) What are primary data types?

Ans : char , int , long , float & double.

Q14) What are derived data types?

Ans : Array , pointer.

Q15) What are user defined data types.

Ans : Structure, union and enum.

Q16) What is the size and range of the basic data types?

Ans: The size and range of the basic data types are:

Data Type	size (bytes)	range
int	2	-32,768 to +32767
long int	4	$-2^{31}$to $2^{31}-1$
char	1	-128 to +127
float	4	3.4e-38 to 3.4e+38
double	8	1.7e-308to1.7e+308

Q17) What is the size and range of short int(or)signed short int, long int (or)signed long int?

Ans:

Data Type	size (bytes)	range
Short int or signed short int	2	-2^{15} to $+2^{15}-1$
Long int or signed long int	4	-2^{31} to $+2^{31}-1$

Q18) How many types of constants exist in C?

Ans: C has two types of constants. Those are

1. Numeric constants
2. Character constants

Q19) How many types of numeric constants exist in C?

Ans: There are two types of numeric constants. They are:

1. Integer constants
2. Real constants

Q20) How many types of character constants exist in?

Ans: There are two types of character constants. Those are

1. Single character constants
2. String constants

Q21) Is embedded spaces, commas and non digit characters are permitted between decimal integer constants?

Ans:	No, embedded spaces, commas and non digit characters are NOT permitted between decimal integer constants.
Q22)	Is '20,000' a valid decimal integer constant?
Ans:	Not valid
Q23)	Give few examples of octal integer constants?
Ans:	037, 0435, 0551
Q24)	Give few examples of hexa decimal integer constants?
Ans:	0x2, 0x9f, 0xbcd, ox1234.
Q25)	Give few examples of real type constants?
Ans:	0.0083,-.75,435.36,+247.008
Q26)	Give few examples of single character type constant?
Ans:	'5', 'x', ';'
Q27)	Give few examples of string constant?
Ans:	”rana”, ”123”, ”seetha123”. ”Rama Rao”
Q28)	What are the basic escape sequence characters & meaning ?
Ans:	’\a’ ---- alarm ’\b’ ----- BackSpace ’\f’ ---- form feed ’\n’ ---- newline ’\r’ ---- carriage Return ’\t’ ---- tab ’\v’ ---- vertical Tab
Q29)	What is an integer expression?
Ans:	When both the operands in an expression are integers then the expression is known as an integer expression.
Q30)	Let a=14, b=4 what will be the output for the following a) a-b b) a+b c) a*b d) a/b e) a%b
Ans:	a) 10 b) 18 c) 56 d) 3 e) 2
Q31)	When an expression is called mixed mode arithmetic?
Ans:	When one of the operands is real and the other is integer, the expression is called mixed mode arithmetic.
Q32)	What will be the value of c,d,e & f in the below code. float c=15/10.0 & int d=15/10;

	float e = 15/10 and float f= 15.0/10.0;
Ans:	c = 1.5 d = 1 e = 1.0, f = 1.5.
Questions on different types of Operators	
Q1)	How many arithmetic operators exist in C ?
Ans:	Five.
Q2)	What are the different arithmetic operators?
Ans:	The different arithmetic operators are: + (addition), - (subtraction), * (multiplication), / (division), % (modulo division).
Q3)	What is precedence of an operator?
Ans:	Precedence of an operator decides the order in which different operators are applied.
Q4)	What is associativity ?
Ans:	Associativity decides the order in which operands are associated with operator.
Q5)	How many types of associativity exist in C ?
Ans :	Two types.
Q6)	What are two types of associativity ?
Ans :	Left to Right & Right to Left.
Q7)	Which arithmetic operator/s has highest precedence?

Ans:	* , / and %
Q8)	Which arithmetic operator/s has lowest precedence?
Ans:	+ and -
Q9)	What is the associativity of arithmetic operators?
Ans:	Left to Right.
Q10)	Which operator has highest precedence level in c?
Ans:	( ) [] has highest precedence level in c .

Q11)	What will be the value of x, y, z for a=9,b=12,c=3 (all are declared as float data type)
	x= a-b/3+c*2-1;
	y= a-b/(3+c)*(2-1);
	z= a-(b/(3+c)*2)-1;

Ans:	x = 10.0000
	y= 7.0000
	z= 4.000.

Q12)	How many relational operators exist in C?
Ans:	Six
Q13)	What are the different relational operators?
Ans:	The different relational operators are: < (is less than), <= (less than or equal to), > (greater than), >= (greater than or equal to), == (equal to), !=(not equal to).
Q14)	Which relational operators have highest precedence?
Ans:	>, <, >=, <=
Q15)	Which relational operator has lowest precedence?
Ans:	= =, !=
Q16)	What is the associativity of relational operators?
Ans:	Left to Right.
Q17)	What are the different assignment operators?
Ans:	=, +=, -=, *=, /=, %=, &=, ^=, |=, <<=, >>=
Q18)	What is the associativity of assignment operators?
Ans:	Right to Left

Q19)	What will be the values of a, b, c, d (all are integer data types) and initially a=10,b=3,c=2,d=
	8 a+=1;
	b-=1;
	c*=2;
	d/=4;

Ans:	a=11, b=2, c=8, d=2.
Q20)	How many Logical operators exist in C?
Ans:	Three
Q21)	What are the different logical operators?
Ans:	&& (LOGICAL AND), || (LOGICAL OR), ! (LOGICAL NOT)
Q22)	What is the associativity of logical and (&&), logical or (||) ?
Ans:	Left to Right
Q23)	What is the associativity of logical not ?
Ans:	Right to Left
Q24)	What is the associativity of increment and decrement operators?
Ans:	Right to Left
Q25)	What is an increment operator?
Ans:	The operator which automatically increments the the value by one.
Q26)	How many types of increment operators exist in C ?
Ans:	Two types.
Q27)	What are the two types of increment operators exist in C ?
Ans :	Post increment and pre increment.
Q28)	Which is a pre / post increment operator?
Ans:	++
Q29)	Give an example for usage of pre increment operator
Ans:	++a
Q30)	Give an example for usage of post increment operator
Ans:	a++
Q31)	What is post increment operator?
Ans :	An operator which increments the value of a variable after the operation

Q32)	What is pre increment operator?
Ans :	An operator which increments the value of a variable before the operation.
Q33)	What is a decrement operator?
Ans:	The operator which automatically reduce the value by one.
Q34)	How many types of decrement operators exist in C ?
Ans:	Two types.
Q35)	What are the two types of decrement operators exist in C ?
Ans :	Post decrement and pre decrement.
Q36)	Give an example for usage of pre decrement and post decrement operator?
Ans:	--a & a--;
Q37) What is the value of m and y in the below expression? m= 5; y= ++m;	
Ans:	m=6,y=6.
Q38) What is the value of m and y in the below expression? m= 5; y= m++;	
Ans:	y=5, m=6.
Q39)	Which is a conditional operator?
Ans: A ternary operator (? :) is a conditional operator. Which is an alternate for **if else** statement.	
Q40)	What is the associativity of a conditional operator?
Ans:	Right to Left
Q41)	What will be the value of x after evaluating the following expression?
a=10; b=15; x= (a>b)?a:b;	
Ans:	15
Q41)	How many bitwise operators exist in C ?
Ans:	Six.
Q42)	What are the different Bit wise operators?
ans: Bit wise AND(&), bit wise OR(|), bit wise exclusive OR(^), shift left(<<), shift right (>>), and One's complement (~) are called Bit wise operators.	
Q43)	What is the associativity of bitwise operators?
Ans:	Left to Right.