

MODULE 1

INTRODUCTION TO C LANGUAGE

Syllabus

- ✓ **Pseudo code Solution to Problem**
- ✓ **Basic Concepts of C Program**
- ✓ **Declaration, Assignment & Print Statements**
- ✓ **Data Types**
- ✓ **Operators and Expressions**
- ✓ **Programming Examples and Exercise.**

1.1 Introduction to C

- C is a general-purpose programming language developed by **Dennis Ritchie** at AT&T Bell laboratories in 1972.

Advantages/Features of C Language

C language is very popular language because of the following features:

1. C is structured Programming Language
2. It is considered a high-level language because it allows the programmer to solve a problem without worrying about machine details.
3. It has wide variety of operators using which a program can be written easily to solve a given problem.
4. C is more efficient which increases the speed of execution and management of memory compared to low level languages.
5. C is machine independent. The program written on one machine will work on another machine.
6. C can be executed on many different hardware platforms.

1.2 Pseudocode: A solution to Problem

• Definition:

- It is a series of steps to solve a given problem written using a mixture of English and C language.
- It acts as a problem solving tool.
- It is the first step in writing a program.

- **Purpose:**
 - Is to express solution to a given problem using mixture of English language and c like code.
- **Advantage:**
 - Easy to write and understand
 - It is relatively easy to convert English description solution of small programs to C program.

Ex 1: Addition of two numbers

1. Get the numbers[a,b]
2. Compute addition [Sum= a + b]
3. Print the results [Sum]

Ex 2: Area of Circle

1. Get the radius[r]
2. Compute area[Area = 3.141*r*r]
3. Print the results [Area]

Disadvantage:

- It is very difficult to translate the solution of lengthy and complex problem in English to C

C Programming Concepts

- **Program:** A program is a group of instructions given by the programmer to perform a specific task.

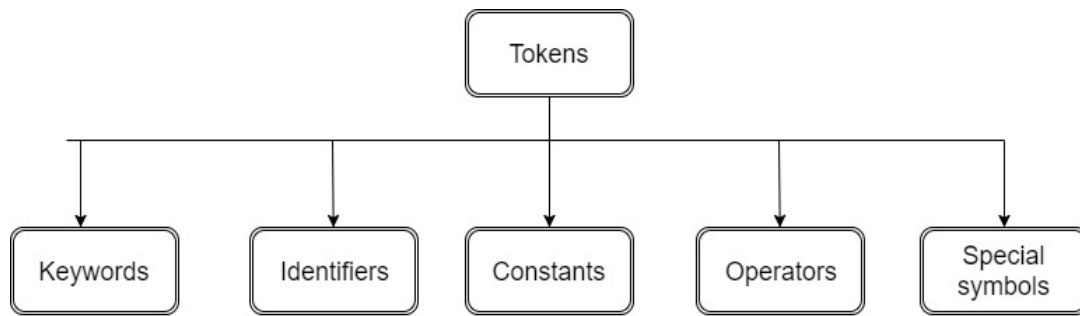
1.3 Character set of C language

- **Definition:** A symbol that is used while writing a program is called a character.
- A character can be: .
 - ➔ **Alphabets/Letters (Lowercase a-z, Uppercase A-Z)**
 - ➔ **Digits (0-9)**
 - ➔ **Special Symbols (~ ‘ ! @ # % & * () - + / \$ = \ {**
 - ➔ **} [] : ; “ “ ? etc)**

Symbol	Name	Symbol	Name	Symbol	Name
~	Tilde		Vertical bar	[Left bracket
#	hash	(Left parenthesis]	Right bracket
\$	Dollar sign)	Right parenthesis	:	Colon
%	Percent sign	_	Underscore	"	Quotation mark
^	Caret	+	Plus sign	;	Semicolon
&	Ampersand	{	Left brace	<	Less than
*	Asterisk	}	Right brace	>	Greater than
'	Single quote	.	dot	=	Assignment
,	comma	\	backslash	/	Division

1.4 C Tokens

- Tokens are **the smallest or basic units of C program.**
- One or more characters are grouped in sequence to form meaningful words. these meaningful words are called as tokens.
- A token is **collection of characters.**
- Tokens are classified in to **5 types** as below:



1.4.1 Keywords

- The tokens which have predefined meaning in C language are called **keywords**.
- They are reserved for specific purpose in C language they are called as **Reserved Words**.
- There are totally **32 keywords** supported in C they are:

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

Rules for keywords

1. Keywords should **not be used** as variables ,function names, array names etc.
2. All keywords should be written in **lowercase letters**.
3. Keywords **meaning cannot be changed** by the users.

1.4.2 Identifiers

Definition:

- Identifiers are the names given to program elements such as variables, constants ,function names, array names etc
- It consists of one or more letters or digits or underscore.

Rules for identifiers

1. The First character should be an **alphabet** or an **underscore** _
Then First character is followed by any number of **letters or digits**.

2. No extra symbols are allowed other than **letters ,digits and Underscore**
3. Keywords cannot be used as an identifier
4. The length can be **31 characters** for external, **63** for internal.
5. Identifiers are **case sensitive**.

Example: Area, Sum_

Ex:-

Identifier	Reasons for invalidity
india06	Valid
_india	Valid
india_06	Valid
india_06_king	Valid
__india	valid
06india	not valid as it starts from digits
int	not valid since int is a keyword
india 06	not valid since there is space between india and 06
india@06	not valid since @ is not allowed

1.4.3 Constants

Definition:

- Constants refers to **fixed values** that **do not change** during the execution of a program
- The different types of constants are:
 1. **Integer constant**
 2. **Real constant/Floating Pointing constant**
 3. **Enumeration constant**
 4. **Character constant**
 5. **String constant**

1. Integer constant

- Definition: An integer is whole number without any decimal point.no extra characters are allowed other than + or _ .

- Three types of integers
- **Decimal integers:** are constants with a combination of digits 0 to 9, optional + or -
Ex: 123 , -345, 0 , 5436 , +79
- **Octal integers:** are constants with a combination of Digits from 0 to 7 but it has a prefix of 0
Ex: 027 , 0657 , 0777645
- **Hexadecimal integers:** Digits from 0 to 9 and characters from a to f, it has to start with 0X or 0x
Ex: 0X2 0x56 0X5fd 0xbdae

2. Real constants/Floating point:

- **Floating point** constants are base 10 numbers that are represented by fractional parts, such as 10.5. They can be positive or negative.
- Two forms are:

i. Fractional form:

- A floating point number represented using fractional form has an integer part followed by dot and a fractional part.
- Ex: 0.0083
- 215.5
- -71.
- +0.56 etc..

ii. Exponential form:

- A floating point number represented using **Exponent form has 3 parts**
- **mantissa e exponent**
- Mantissa is either real number expressed in decimal notation or integer.
- **Exponent must be integer with optional + or –**
- Ex 9.86 E 3 $\Rightarrow 9.86 \times 10^3$
- Ex 9.86 E -3 $\Rightarrow 9.86 \times 10^{-3}$

3. Enumeration constant

- A set of **named integer constants defined using the keyword enum** are called enumeration constants.
- Syntax:
enum identifier{enum list};

- `enum Boolean{NO,YES};`
 NO is assigned with 0
 YES is assigned with value 1
- `enum days{ mon,tue.wed};`
 mon is assigned with 0
 tue is assigned with 1
 wed is assigned with 2

4.Character constant

- A symbol enclosed within pair of single quotes is called character constant.
- Each character is associated with unique value called ASCII value.
- Ex: '9'
- '\$'
- Backslash constants(Escape sequence character)
 - Definition: An escape sequence character begins with backslash and is followed by one character.
 - A backslash along with a character give rise to special print effects.
 - It always start with backslash ,hence they are called as backslash constants.

character	name	Meaning
<code>\a</code>	Bell	Beep sound
<code>\b</code>	Backspace	Cursor moves towards left by one position
<code>\n</code>	Newline	Cursor moves to next line
<code>\t</code>	Horizontal tab	Cursor moves towards right by 8 position
<code>\r</code>	Carriage return	Cursor moves towards beginning of the same line
<code>\"</code>	Double quotes	Double quotes
<code>\'</code>	Single quotes	Single quotes
<code>\?</code>	Question mark	Question mark
<code>\\</code>	Backslash	Backslash

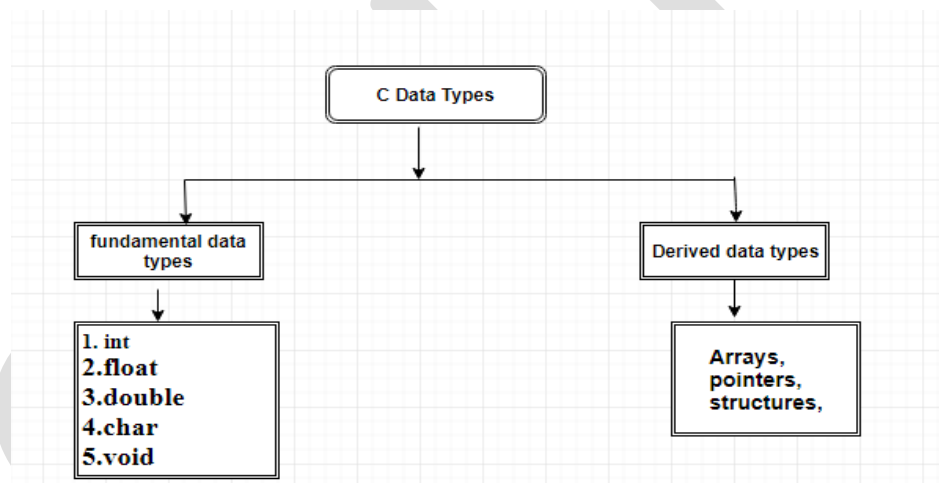
\0	NULL	
----	------	--

5.String constant

- A sequence of characters enclosed within pair of double quotes is called string constant.
- The string always ends with a NULL character.
- Ex: “9”
- “SVIT”

1.4.4 Data types and sizes

- **Definition:** The data type defines the type of data stored in memory location or the type of data the variable can hold.
- The classification of data types are:



- There are few basic data types supported in C.

1. **int**
2. **float**
3. **double**
4. **char**
5. **void**

1.Integer data type (int):

- It stores an integer value or integer constant.
- An int is a keyword using which the programmer can inform the compiler that the data associated with this keyword should be treated as integer.
- C supports 3 different types of integer:
 - short int,
 - int,

- long int

n-bit machine	Type	size	Range of unsigned int 0 to 2^n-1	Range of signed int be -2^n to $+2^n-1$
16-bit machine	short int	2 bytes	0 to $2^{16}-1$ 0 to 65535	-2^{15} to $+2^{15}-1$ -32768 to +32767
	int	2 bytes	0 to $2^{16}-1$ 0 to 65535	-2^{15} to $+2^{15}-1$ -32768 to +32767
	long int	4 bytes	0 to $2^{32}-1$ 0 to 4294967295	-2^{31} to $+2^{31}-1$ -2147483648 to +2147483647

2. Floating data type (float):

- An float is a keyword using which the programmer can inform the compiler that the data associated with this keyword should be treated as floating point number.
- The size is 4 bytes.
- The range is **-3.4e38 to +3.4e38**.
- Float can hold the real constant accuracy up to 6 digits after the decimal point.

3. Double data type (double):

- An double is a keyword using which the programmer can inform the compiler that the data associated with this keyword should be treated as long floating point number.
- The size is 8 bytes.
- The range is from **1.7e-308 to 1.7 e+308**.
- Double can hold real constant up to 16 digits after the decimal point.

4. Character data type (char):

- char is a keyword which is used to define single character or a sequence of character in c language capable of holding one character from the local character set.
- The size of the character variable is 1 byte.
- The range is from **-128 to +127**.
- Each character stored in the memory is associated with a unique value termed as ASCII (American Standard Code for Information Interchange).
- It is used to represent character and strings.

n-bit machine	Type	size	Range of unsigned char 0 to 2^n-1	Range of signed char be -2^n to $+2^n-1$
16-bit machine	short int	2 bytes	0 to 2^8-1 0 to 255	-2^7 to $+2^7-1$ -128 to +127

5. void data type (void):

- It is an empty data type.
- No memory is allocated.
- Mainly used when there are no return values.

1.4.4 variable

- A variable is a name given to memory location where data can be stored.
- Using variable name data can be stored in a memory location and can be accessed or manipulated very easily.

• Rules for variables

- The First character should be an **alphabet** or an **underscore** _
- Then First character is followed by any number of **letters or digits**.
- No extra symbols are allowed other than **letters ,digits and Underscore**
- Keywords cannot be used as an identifier

Example:

Sum – valid

For1- valid

for -invalid (it is a keyword)

1.4.4.1 Declaration of variables:

- Giving a name to memory location is called **declaring a variable**.
- Reserving the required memory space to store the data is called **defining a variable**.
- General Syntax:

datatype variable; (or)

datatype variable1, variable2,.....variablen;

example: **int a;**

float x, y;

double sum;

- From the examples we will come to know that “a” is a variable of type integer and allocates 2 bytes of memory. “x” and “y” are two variable of type float which will be allocated 4 bytes of memory for each variable. “sum” is a double type variables which will be allocated with 8 bytes of memory for each.

1.5 Variable Initialization

- Variables are not initialized when they are declared and defined, they contain garbage values(meaningless values)
- The method of giving initial values for variables before they are processed is called variable initialization.
- General Syntax:

Var_name = expr;

Where,

Var_name is the name of the variable ,

expr is the value of the expression.

- The “expr” on the right hand side is evaluated and stored in the variable name (Var_name) on left hand side.
- The expression on the right hand side may be a constant, variable or a larger formula built from simple expressions by arithmetic operators.

Examples:

- `int a=10;`
// assigns the value 10 to the integer variable a
- `float x;`
`x=20;`
// creates a variable y of float type and assigns value 20 to it.
- `int a=10,b,b=a;`
// creates two variables a and b. “a” is assigned with value 10, the value of “a” is assigned to variable “b”. Now the value of b will be 10.
- `price = cost*3;`
//assigns the product of cost and 3 to price.
- `Square = num*num;`
// assigns the product of num with num to square.

1.6 (OUTPUT FUNCTION)

Displaying Output using printf

- printf is an output statement in C used to display the content on the screen.
- print: Print the data stored in the specified memory location or variable.
- Format: The data present in memory location is formatted in to appropriate data type.
- There are various forms of printf statements.

Method 1: **printf(" format string");**

- Format string may be any character. The characters included within the double quotes will be displayed on the output screen
- Example: **printf("Welcome to India");**

Output:

Welcome to India

Method 2:

printf(" format string", variable list);

- Format string also called as control string.
- Format string consist of **format specifier** of particular data type
- Format specifiers starts with % sign followed by **conversion code**.
- variable list are the **variable names** separated by **comma**.
- Example:

int a=10;

float b=20;

printf(" integer =%d, floating=%f",a,b);

○ output:

integer=10, floating=20.00000

- Number of format specifiers must be equal to number of variables.
- While specifying the variables name make sure that it matches to the **format specifiers** with in the double quotes.

Format Specifiers

- Format specifiers are the character string with % **sign** followed with a character.
- It specifies the type of data that is being processed.
- It is also called **conversion specifier or conversion code**.
- There are many format specifiers defined in C.

Symbols	Meaning
---------	---------

%d	Decimal signed integer number
%f	float point number
%c	Character
%o	octal number
%x	hexadecimal integer(Lower case letter x)
%X	hexadecimal integer(Upper case letter X)
%e	floating point value with exponent(Lower case letter e)
%E	floating point value with exponent (Upper case letter E)
%ld	long integer
%s	String
%lf	double

1.7 Input Function (scanf)

Inputting Values Using scanf

- To enter the input through the input devices like keyboard we make use of scanf statement.

General Syntax:

scanf(“format string”, list of address of variables);

- Where: Format string consists of the access specifiers/format specifiers.
- Format string also called as control string.
- Format string consist of **format specifier** of particular data type
- Format specifiers starts with **%** sign followed by **conversion code**.
- **List of addresses of variables consist of the variable name preceded with & symbol(address operator).**

Example: int a;
 float b;
 scanf(“%d%f”,&a,&b);

Rules for scanf

- **No escape sequences or additional blank spaces** should be specified in the format specifiers.
- Ex: `scanf("%d %f",&a,&b); //invalid`
`scanf("%d\n%f",&a,&b); //invalid`
- **& symbol is must to read the values**, if not the entered value will not be stored in the variable specified. Ex: `scanf("%d%f",a,b);//invalid`.

A Simple C Program

Example 1: Let us discuss a simple program to print the Hello SVIT

```
/*program to print Hello svit*/
```

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

```
void main( )
```

```
{
```

```
    printf(" Hello SVIT");
```

```
}
```

Output: Hello SVIT

Example 2: Consider another example program to find the simple interest

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

```
void main( )
```

```
{
```

```
    float p,t,r;
```

```
    float si;
```

```
    printf("enter the values of p,t,r\n");
```

```
    scanf("%f%f%f",&p,&t,&r);
```

```
    si = (p*t*r)/100;
```

```
    printf("Simple Interest=%f",si);
```

```
}
```

The above program illustrates the calculation of simple interest.

- Firstly we have declared the header files `stdio.h` for including standard input and output which will be `printf` and `scanf` statements.
- Next we start with the main program indicated by `void main()`

- Within the main program we declare the required variables for calculating the simple interest. So we declare p, t and r as float type and si as float type.
- After which the values to p, t and r are read from keyboard using scanf.
- Computed the simple interest by the formula $(p*t*r)/100$ and the result is assigned to si.
- Display the result si.

1.8 Structure of C Program

Comments/Documentation Section
Preprocessor Directives
Global declaration section
void main() [Program Header] { <div style="text-align: center;"> Local Declaration part Execution part Statement 1 ----- ----- Statement n </div> }
User defined Functions

- **Comments/Documentation Section**

- Comments are short explaining the purpose of the program or a statement.
- They are non executable statements which are ignored by the compiler.

- The comment begins **with /* and ends with */**.
- The symbols /* and */ are called **comment line delimiters**.
- We can put any message we want in the comments.

Example: /* Program to compute Quadratic Equation */

Note: Comments are ignored by C compiler, i.e. everything within comment delimiters

- **Preprocessor Directives**

- Preprocessor Directives begins with a **#** symbol
- Provides instructions to the compiler to include some of the files before compilation starts.
- Some examples of header files are

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>       Etc...
```

- **Global Declaration Section**

- There will be some variable which has to be used in anywhere in program and used in more than one function which will be declared as global.

- **The Program Header**

- Every program must contain a main() function.
- The execution always starts from main function.
- Every C program must have only one main function.

- **Body of the program**

- Series of statements that performs specific task will be enclosed within braces { and }
- It is present immediately after program header.

It consists of two parts

1. **Local Declaration part:** Describes variables of program

Ex: int sum = 0;

int a , b;

float b;

2. **Executable part:** Task done using statements.

All statements should end with semicolon(;)

Subroutine Section: All user defined functions that are called in main function should be defined.

1.9 Algorithm

- It is a step by step procedure to solve a problem
- A sequential solution of any problem that is written in natural language.
- Using the algorithm, the programmer then writes the actual program.
- The main use of algorithm is to help us to translate English into C.
- It is an outline or basic structure or logic of the problem.

1.9.1 Characteristics of Algorithm

- Each and every instruction must be precise and unambiguous
- Each instruction execution time should be finite.
- There should be a termination condition.
- It has to accept 0 or more inputs and produce compulsory an output.
- It can accept any type of input and produce a corresponding output.

1.9.2 General Way of Writing the Algorithm

- Name of the algorithm must be specified.
- Beginning of algorithm must be specified as Start
- Input and output description may be included
- Step number has to be included for identification
- Each step may have explanatory note provided with in square bracket followed by operation.
- Completion of algorithm must be specified as end or Stop.

Ex 1: Write an algorithm to add two numbers

Algorithm: Addition of two numbers

Input: Two number a and b

Output: Sum of two numbers

Step 1: Start

Step 2: [input two number]

Read a and b

Step 3: [compute its addition]

Sum = a + b

Step 4: print Sum

Step 5: Stop

Ex 2: Write an algorithm to compute simple interest

Algorithm: To find Simple Interest

Input: p, t, r

Output: SI

Step 1: Start

Step 2: [input a principle p, time t and rate r]

Read p, t, r

Step 3: [compute simple interest]

$SI \leftarrow (p * t * r) / 100$

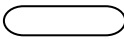
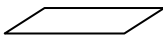
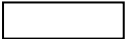

Step 4: display SI


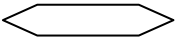
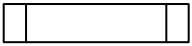


Step 5: Stop

1.10 Flow charts

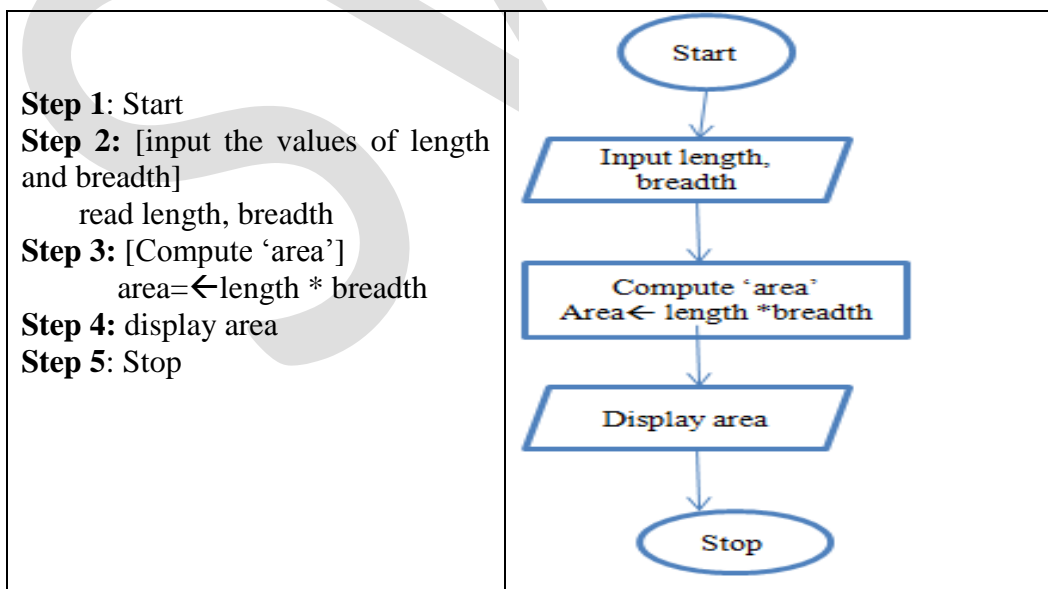
- A flowchart is a pictorial or graphical representation of an algorithm or a program.
- It consist of sequences of instructions that are carried out in an algorithm.
- The shapes represent operations.
- Arrows represent the sequence in which these operations are carried out.
- It is mainly used to help the programmer to understand the logic of the program.

1.10.1 Notations used in flowchart

Shape	Name	Meaning
	Start and Stop	Used to denote start and stop of flowchart
	Input and Output	Parallelogram used to read input and display output
	Processing/computation	Rectangle used to include instructions or processing statements
	Decision/ Condition	To include decision making statements

		rhombus is used
	Connector	Join different flow chart or flow lines
	Repetition/ Loop	Hexagon is used for looping constructs
	Functions	To include functions in a program, the name of function should be enclosed with in the figure
	Control Flow Lines	Arrows denoting flow of control in flow chart
	Comment box	For including the comments, definition or explanation

Ex 1. Algorithm and Flowchart to Input the dimensions of a rectangle and print its area



1.11 Operators and Expressions

1.11.1 Operators:

- “An operator is a symbol that specifies the operation to be performed on various types of operands”. Or in other words “An operator is a symbol that tells the computer to perform certain mathematical or logical manipulations”.
- Operators are used in programs to manipulate data and variable.

Example: +, -, *

1.11.2 Operand:

- The entity on which an operator operates is called an operand. Here the entity may be a constant or a variable or a function.
- An operator may have one or two or three operands.

Example: a+b-c*d/e%f

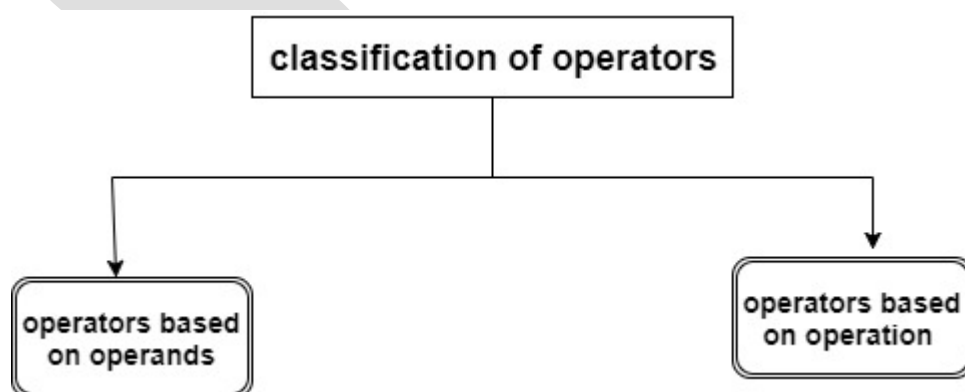
In this example there are **6 operands** i.e a, b, c, d, e, f and **5 operators** i.e +, -, *, / and %.

1.11.3 Expression:

- A combination of operands and operators that reduces to a single value is called an expression.

Eg: a+b/d

Classification of operators



1.12.1 Classification of operators based on operands

- The Operators are classified into **four** major categories based on the number of operands as shown below:

- 1) *Unary Operators*
- 2) *Binary Operators*
- 3) *Ternary Operators*
- 4) *Special Operators*

1. **Unary Operators:** An operator which acts on only **one operand** to produce the result is called unary operator. The Operators precede the operand.

Examples: **-10**

-a

--b

2. **Binary Operators:** An operator which acts on **two operands** to produce the result is called a binary operator. In an expression involving a binary operator, the operator is in between two operands.

Examples: a + b a*b

3. **Ternary Operator:** An operator which acts on **three operands to** produce a result is called a ternary operator.
 - Ternary operator is also called **conditional operator**.
 - **Example:** **a ?b : c**
 - Since there are three operands a, b and c (hence the name ternary) that are associated with operators ?and : it is called ternary operator.

1.12.2 C operators can be classified based on type of operation

- 1. Arithmetic operators**
- 2. Relational operators**
- 3. Logical operators**
- 4. Assignment operators**
- 5. Increment and decrement operators**
- 6. Conditional operators**
- 7. Bitwise operators**
- 8. Special operators**

1) Arithmetic operators

- C provides all basic arithmetic operators.

- The operators that are used to perform arithmetic operation such as addition, subtraction, multiplication, division etc are called arithmetic operators.
- The operators are +, -, *, /, %.
- Let a=10, b=5. Here, **a and b are variables and are known as operands.**

Description	Symbol	Example	Result	Priority	Associativity
Addition	+	a+b = 10 + 5	15	2	L-R
Subtraction	-	a-b = 10 - 5	5	2	L-R
Multiplication	*	a* b = 10 * 5	50	1	L-R
Division(Quotient)	/	a/b = 10/5	2	1	L-R
Modulus(Remainder)	%	a % b = 10 % 5	0	1	L-R

/*Program to illustrate the use of arithmetic operators is given below*/

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

```
void main()
```

```
{
```

```
    int a,b,sum,diff,mul,divres,modres;
```

```
    printf("enter a and b value\n");
```

```
    scanf("%d %d",&a,&b);
```

```
    sum=a+b;
```

```
    diff=a-b;
```

```
    mul=a*b;
```

```
    divres=a/b;
```

```
    modres=a%b;
```

```
    printf("%d, %d, %d, %d, %d\n", sum,diff,mul,divres,modres);
```

```
}
```

Output: 35,25,150,6,0

Converting Mathematical Expression into C Expression

Mathematical Expression	C expression
-------------------------	--------------

$a \div b$	a/b
$S = \frac{a+b+c}{2}$	$S = (a+b+c)/2$
$\text{Area} = \sqrt{s(s-a)(s-b)(s-c)}$	$\text{Area} = \text{sqrt}(s*(s-a)*(s-b)*(s-c))$
$ax^2 + bx + c$	$a*x*x + b*x + c$
$e^{ a }$	$\text{exp}(\text{abs}(a))$

2) Relational Operators

- We often compare two quantities depending on their relation, take certain decisions.
- For example, we compare the age of two persons, or the price of two items, and so on. These comparisons can be done with the help of relational operators.
- The operators that are used to find the relationship between two operands are called as **relational expression**.
- An expression such as
 $a < b$ or $1 < 20$
- **The value of relational expression is either one or zero.**
- It is **one** if the specified relation is **true** and zero if the relation is **false**.

Example: $10 < 20$ is true.

$20 < 10$ is false.

Relational operator in C are:

Operator	Meaning	Precedence	Associativity	Example	Result
$<$	is less than	1	L->R	$20 < 10$	F
$<=$	is less than or equal to	1	L->R	$20 <= 10$	F
$>$	is greater than	1	L->R	$20 > 10$	T
$>=$	is greater than or equal to	1	L->R	$20 >= 10$	T
$==$	is Equal to	2	L->R	$20 == 10$	F
$!=$	is not equal to	2	L->R	$20 != 10$	T

Example:

```

a=10 b=20 c=30
a > b != c <= b > a
10 > 20 != 30 <= 20 > 10
0 != 30 <= 20 > 10
0 != 0 > 0
0 != 0

```

3) Logical Operators

- The operator that are used to combine two or more relational expression is called logical operators.
- Result of logical operators is always true or false.

Logical Operators in C are.

Operator	Meaning	Precedence	Associativity
&&	logical AND	2	L->R
	logical OR	3	L->R
!	logical NOT	1	L->R

3.1 logical AND (&&)

The output of logical operation is true if and only if both the operands are evaluated to true.

operand 1	AND	operand 2	result
True (1)	&&	True(1)	True(1)
True (1)	&&	False(0)	False(0)
False(0)	&&	True(1)	False(0)
False(0)	&&	False(0)	False(0)

Example:

36 && 0

=1 && 0

=0(False)

3.2 logical OR (||)

The output of logical operation is false if and only if both the operands are evaluated to false.

operand 1	OR	operand 2	result
True (1)		True(1)	True(1)
True (1)		False(0)	True(1)
False(0)		True(1)	True(1)
False(0)		False(0)	False(0)

Example:

36 || 0
 =1 || 0
 =0(True)

3.4 logical NOT

It is a unary operator. The result is true if the operand is false and the result is false if the operand is true.

operand	! operand
True (1)	False(0)
False(0)	True (1)

Example:

(a) !0=1

(b) !36

!1=0

NOTE: The logical operators && and || are used when we want to test more than one condition and make decisions.

An example: a>b && x==10

The above logical expression is true only if a>b is true and x==10 is true. If either (or both) of them are false, the expression is false.

4) Assignment Operators

➤ An operator which is used to assign the data or result of an expression into a variable is called Assignment operators. The usual assignment operator is =.

➤ Types of Assignment Statements:

1.Simple assignment statement

2.Short Hand assignment statement

3.Multiple assignment statement

1.Simple assignment statement

Syntax: **Variable=expression;**

Ex:

a= 10; //RHS is constant

a=b; // RHS is variable.

a=b+c; //RHS is an expression

So,

1.A expression can be a constant,variable or expression.

2.The symbol = is assignment operator

3.The expression on right side of assignment is evaluated and the result is converted into type of variable on left hand side of Assignment operator.

4.The converted data is copied into variable which is present on left hand side(LHS) of assignment operator.

Example:

```
int a;
```

```
a=100;        // the value 100 is assigned to variable a
```

```
int a=100;
```

```
int b;
```

```
b=a;        // a value i.e 100 is assigned to b
```

2.Short Hand assignment Statement

C has a set of “**shorthand**” assignment operators of the form.

var op = exp;

where

v is a variable, exp is an expression and op is a C library arithmetic operator.

The operator **op=** is known as the shorthand assignment operator.

The assignment statement

var op= exp; is equivalent to $v = \text{var op (exp)}$. with var evaluated only once.

The operators such as +=, -=, *=, /= are called assignment operators.

Example:

The statement

$x = x + 10;$ can be written as $x += 10;$

Example:

Solve the equation $x * y = z$ when $x=10, y=5$ and $z=3$

Solution: $x * y = z$ can be written as

$$x = x * (y + z)$$

$$x = 10 * (5 + 3)$$

$$x = 10 * 8$$

$$x = 80$$

shorthand assignment	Meaning	Description
a+=2	a=a+2	Find a+2 and store result in a
a-=2	a=a-2	Find a-2 and store result in a
a*=2	a=a*2	Find a*2 and store result in a
a/=2	a=a/2	Find a/2 and store result in a

3. Multiple Assignment Statement

A statement in which a value or set of values are assigned to more than one variable is called multiple assignment statement.

Example:

$i=10;$

```
j=10;
```

```
k=10;
```

can be written as:

```
i=j=k=10; //Multiple assignment statement
```

Note: Assignment operator is right associative operator. Hence in the above statement 10 is assigned to k first, the k is assigned to j and j is assigned to i.

5) Increment and Decrement Operators

- C allows two very useful operators not generally found in other languages. These are the increment and decrement operators:

++ and --

- The operator ++ adds 1 to the operand, while -- subtracts 1. The increment and decrement operators are Unary operators.
- The increment and decrement operators are classified into two categories:

1. Post Increment and Post Decrement:

- If increment or decrement operator is placed immediately after the operand is called Post Increment and Post Decrement.
- As the name indicates Post Increment and Post Decrement means increment or decrement the operand value is used.
- So operand value is used first and then the operand value is incremented or decremented by 1.

Example:

Post Increment: a++ equivalent to a=a+1

Post Decrement: a-- equivalent to a=a-1

Post Increment:

Eg 1:

```
#include<stdio.h>

void main()
{
    int a=20;
    a++;
    printf("%d",a);
}
```

Output:21

Description: Initial value of a is 20 after execution of a++, the value of a will be 21.

Eg2:

```
#include<stdio.h>

void main( )
{
int a=20,b;
b=a++;
printf("a=%d",a);
printf("b=%d",b);
}
```

Output : a= 21

b=20

Post Decrement:

Eg 1:

```
#include<stdio.h>

void main()
{
    int a=20;
    a--;
    printf("%d",a);
}
```

Output:19

Description: Initial value of a is 20 after execution of a--, the value of a will be 19.

2.Pre Increment and Pre Decrement

- If increment or decrement operator is placed before the operand is called Pre Increment and Pre Decrement.
- As the name indicates Pre Increment and Pre Decrement means increment or decrement before the operand value is used.
- So the operand value is incremented or decremented by 1 first then operand value is used .

Example:

Post Increment: ++a equivalent to a+1=a

Post Decrement: a--equivalent to a-1=a

Pre Increment:

Eg 1:

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

Output:21

```

void main()
{
    int a=20;
    ++a;
    printf("%d",a);
}

```

Description: Initial value of a is 20 after execution of ++a, the value of a will be 21.

Pre Decrement:

Eg 1:

```

#include<stdio.h>
void main()
{
    int a=20;
    --a;
    printf("%d",a);
}

```

Output:19

Description: Initial value of a is 20 after execution of --a, the value of a will be 19.

Consider the following(let a=20)

Post increment

b=a++

current value of a used, b=a //b=20
a is incremented by 1 a=a+1 //a=21

Pre increment

b=++a

a is incremented by 1, a=a+1 //a=21
incremented value of a used, b=a //b=21

Post Decrement (let a=10)

b=a--

current value of a used, b=a //b=10
a is decremented by 1 a=a-1 //a=9

Pre decrement(let a=10)

b=--a

a is decremented by 1, a=a-1 // a=9
decremented value of a used, b=a // b=9

6) Conditional Operator (OR) Ternary Operator

The operator which operates on 3 operands are called Ternary operator or conditional operator.

Syntax: (exp1)? exp2: exp3

where exp1, exp2, and exp3 are expressions and

- exp1 is an expression evaluated to true or false
- if exp1 is evaluated to true exp2 is executed
- if exp1 is evaluated false exp3 is executed

For example, consider the following statements.

```
a=10;  
b=15;  
Max = (a>b)? a:b;
```

In this example, Max will be assigned the value of b i.e b=15. This can be achieved using the if... else statements as follows:

```
if (a>b)  
x=a;  
else  
x=b;
```

7) Bitwise Operators

Bitwise operators are used to manipulate the bits of given data. These operators are used for testing the bits, or shifting them right or left. Bitwise operators may not be applied to float or double.

Operator	Meaning	Precedence	Associativity
~	Bitwise Negate	1	R->L
<<	Bitwise Left Shift	2	L->R
>>	Bitwise Right Shift	2	L->R
&	Bitwise AND	3	L->R
^	Bitwise XOR(exclusive OR)	4	L->R
	Bitwise OR	5	L->R

1)Bitwise AND (&)

If the corresponding bit positions in both the operand are 1,then AND operation results in 1 ;otherwise 0.

operand 1	AND	operand 2	result
0	&	0	0
0	&	1	0
1	&	0	0
1	&	1	1

a=10 00001010

b=6 00000110 (a&b)

c=2 00000010

2)Bitwise OR(|)

If the corresponding bit positions in both the operand are 0,then OR operation results in 0 ;otherwise 1.

operand 1	OR	operand 2	result
0		0	0
0		1	1
1		0	1
1		1	1

a=10 00001010

b=6 00000110 (a|b)

c=2 00001110

3) Bitwise XOR(^)

If the corresponding bit positions in both the operand are different, then XOR operation results in 1 ;otherwise the result is 0.

operand 1	XOR	operand 2	result
0	^	0	0
0	^	1	1
1	^	0	1
1	^	1	0

```

a=10  00001010
b=6    00000110 (a^b)
-----
c=12   00001100

```

4) Left shift operator(<<)

The operator that is used to shift the data by a specified number of bit positions towards left is called left shift operator.

After left shift MSB should be discarded and LSB will be empty and should be filled with “0”.

Diagram illustrating the structure of a number literal in C++: `b=a<<num`. The `num` part is identified as the **Second operand**, and the `<` operator is identified as the **First operand**.

The first operand is the value to be shifted.

The second operand specifies the number of bits to be shifted.

The content of first operand are shifted towards left by the number bit positions specified in second operand.

Example:

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

```
void main()
```

OUTPUT: 5<<1=10

```
{
    int a,b;

    a=5;

    b=a<<1;

    printf(“ %d<<1=%d”,a,b);

}
```

5) Right shift operator(>>)

The operator that is used to shift the data by a specified number of bit positions towards right is called right shift operator.

After right shift operation, LSB bit should be discarded and MSB is empty and should be filled with 0.

b=a>>num \longrightarrow **Second operand**
 \downarrow
First operand

The first operand is the value to be shifted.

The second operand specifies the number of bits to be shifted.

The content of first operand are shifted towards right by the number bit positions specified in second operand.

Example:

```
#include<stdio.h>

void main()

{
    int a,b;

    a=10;

    b=a>>1;

    printf(“ %d>>1=%d”,a,b);

}
```

OUTPUT: 10>>1=5

6) Bitwise negate:

The operator that is used to change every bit from 0 to 1 and 1 to 0 in the specified operand is called bitwise negate operator. It is used to find one's complement of an operand.

Example:

```
#include<stdio.h>
void main()
{
```

```
int a=10,b;  
b=~a;  
printf("Negation of a is %d",b);  
}
```

Tracing:

```
a=10      0 0 0 0 1 0 1 0  
b=245     1 1 1 1 0 1 0 1
```

8) Special Operators

C supports some special operators of interest such as comma operator, sizeof operator.

The Comma Operator

- ❖ The comma operator can be used to link the related expressions together. A comma-linked list of expressions are evaluated from left to right. The comma operator has least precedence among all operator.
- ❖ It is a binary operator which is used to:

1. Separate items in the list.

Eg: a=12,35,678;

2. Combine two or more statements into a single statement.

Eg: int a=10;
 int b=20;
 int c=30;

can be combined using comma operator as follows:

int a=10,b=20,c=30;

Example, the statement: value = (x=10,y=5,x+y)

first assigns the value 10 to x, then assigns 5 to y, and finally assigns 15 to value. Since comma operator has the lowest precedence of all operators, the parentheses are necessary.

The sizeof() Operator

The sizeof() is a compile time operator and, when used with an operand, it returns the number of bytes the operand occupies. The operand may be a variable, or a constant or a data type qualifier.

Syntax: **sizeof(operand)**

Examples:

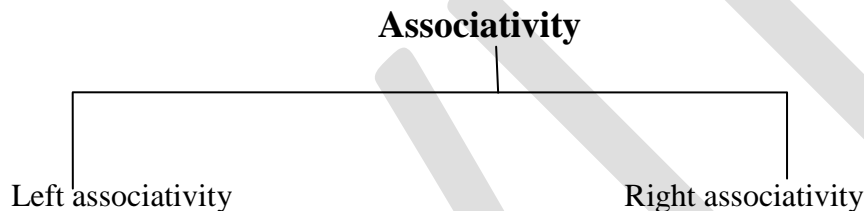
- | | |
|--------------------|---------------|
| 1. sizeof(char); | Output:1 Byte |
| 2 int a; | Output:2 Byte |
| sizeof(a) | |

Precedence[Priority]

It is the rule that specifies the order in which certain operations need to be performed in an expression. For a given expression containing more than two operators, it determines which operations should be calculated first.

Associativity

If all the operators in an expression have equal priority then the direction or order chosen left to right or right to left to evaluate an expression is called associativity.



Left associativity:

In an expression if two or more operators having same priority are evaluated from left to right then the operators are called left to right associative operator, denoted as **L->R**

Right associativity:

In an expression if two or more operators having same priority are evaluated from left to right then the operators are called right to left associative operator, denoted as **R->L**

BODMAS Rule can be used to solve the expressions, where

B: Brackets O: Operators D: Division M: Multiplication A: Addition S: Subtraction

The Precedence (Hierarchy) of Operator

Operator Category	Operators in Order of Precedence (Highest to Lowest)	Associativity
Innermost brackets/Array elements	(), [], {}	Left to Right(L->R)

reference		
Unary Operators	++, --, sizeof(), ~, +, -	Right to Left(R->L)
Member Access	-> or *	L->R
Arithmetic Operators	*, /, %	L->R
Arithmetic Operators	-, +	L->R
Shift Operators	<<, >>	L->R
Relational Operators	<, <=, >, >=	L->R
Equality Operators	==, !=	L->R
Bitwise AND	&	L->R
Bitwise XOR	^	L->R
Bitwise OR		L->R
Logical AND	&&	L->R
Logical OR		L->R
Conditional Operator	?:	R->L
Assignment Operator	=, +=, -=, *=, /=, %=	R->L
Comma Operator	,	L->R

Modes of Arithmetic Expressions

- An expression is a sequence of operands and operators that reduces to a single value. Expressions can be simpler or complex. An operator is a syntactical token that requires an action to be taken. An operand is an object on which an operation is performed.
- Arithmetic expressions are divided into 3 categories:
 1. Integer Expressions
 2. Floating Point Expressions
 3. Mixed mode Expressions

1. Integer Expressions

- If all the operands in an expression are integers then the expression is called Integer Expression.
- The result of integer expression is always integer.

Ex: $4/2=2$

2. Floating Point Expressions

- If all the operands in an expression are float numbers then the expression is called floating point Expression.
- The result of floating point expression is always float

Ex: $4.0/3.0=1.3333$

3. Mixed Mode Expressions

- An expression that has operands of different types is called Mixed Mode Expression.

Ex: $4.0/2$ i.e float/int

Evaluation of Expressions involving all the operators

The rules to be followed while evaluating any expressions are shown below.

- Replace the variables if any by their values.
- Evaluate the expressions inside the parentheses
- Rewrite the expressions with increment or decrement operator as shown below:
 - a) Place the pre-increment or pre-decrement expressions before the expression being evaluated and replace each of them with corresponding variable.
 - b) Place the post-increment or post-decrement expressions after the expression being evaluated and replace each of them with corresponding values.
- Evaluate each of the expression based on precedence and associativity.

1.10 Type Conversion

The process of converting the data from one data type to another data type is called Type conversion.

Type conversion is of two types

(i) Implicit type conversion

(ii) Explicit type conversion

i). Implicit Conversion

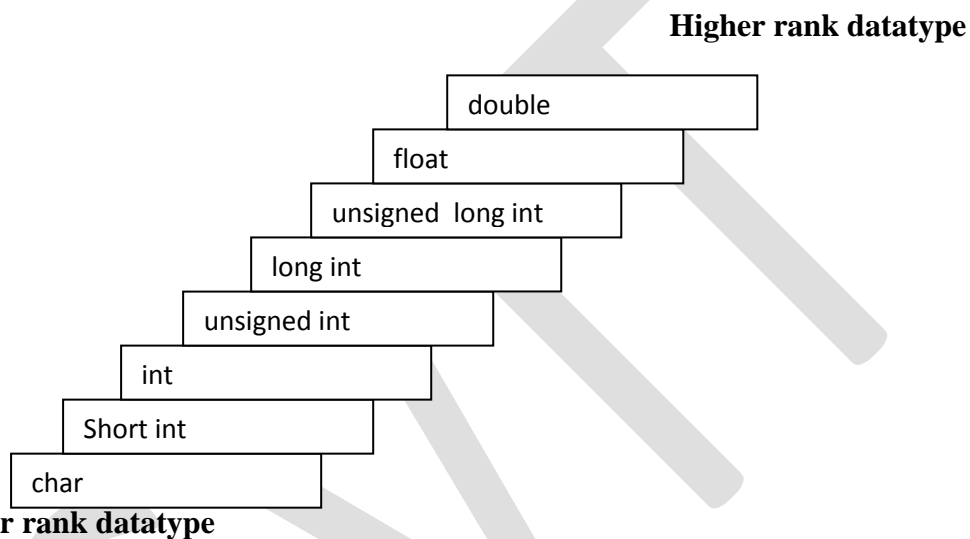
- C can evaluate the expression if and only if the data types of two operands are same.
- If operands are of different type, it is not possible to evaluate expression.
- In such situation to ensure that both operands are of same type, C compiler converts data from lower rank to higher rank automatically.

- This process of converting data from lower rank to higher rank is called as Implicit type conversion(ITC).

For example,

- If one operand type is same as other operand type, no conversion takes place and type of result remains same as the operands
i.e $\text{int} + \text{int} = \text{int}$, $\text{float} + \text{float} = \text{float}$ etc.
- If one operand type is int and other operand type is float, the operand with type int is promoted to float.

Promotion of hierarchy is as follows:



Examples:

- i. $5 + 3 = 8$
 $\text{int} + \text{int} = \text{int}$
- ii. $5 + 3.5 = 8.5$
 $\text{int} + \text{float} = \text{float}$

ii) Explicit type conversion

- If the operands are of same datatype no conversion takes place by compiler. Sometimes the type conversion is required to get desired results.
- In such situation programmer himself has to convert the data from one type to another. This forcible conversion from one datatype to another is called as explicit type conversion(ETC).

(type) expression

Syntax:**Example:** (int) 9.93=9

The data conversion from higher data type to lower data type is possible using explicit type conversion. The following example gives the explicit type conversion.

x=(int) 7.5	7.5 is converted to integer by truncation
a=(int)21.3 / (int) 4.5	Evaluated as 21/4 and the result would be 5.
b=(double) sum/n	Division is done in floating point mode
y=(int)(a+b)	The result a + b is converted to integer
z=(int)a +b	a is converted to integer and then added to b

“What is type casting? Explain with example?”

Definition: C allows programmers to perform typecasting by placing the type name in parentheses and placing this in front of the value.

For instance

```
main()
{
    float a;
    a = (float)5 / 3;
}
```

Gives result as 1.666666 . This is because the integer 5 is converted to floating point value before division and the operation between float and integer results in float.

Difference between ITC and ETC

ITC	ETC
<p>1.It is performed by Compiler.</p> <p>2.ITC performs only lower to higher data conversion</p> <p>3.ITC is performed when both the operands are of different type.</p> <p>4.Example</p> $2+3.4=5.4$	<p>1.It is performed by programmer.</p> <p>2.ETC perform both lower to higher and higher to lower.</p> <p>3.ETC is performed when both the operands are of same type.</p> <p>4.Example</p> <pre>float a=10.6 (int)a; Ans:10</pre>

Question Bank

Module 1				
Sl.No	Questions	Appeared in VTU qp month/year	CO mapping	BT Level Attained
1.	Define pseudocode .Write a pseudocode to find the sum and average of three numbers.	June/July 2016 Dec/Jan 2015	CO1	BT1

2	Classify the following identifiers as valid/invalid. a) num2 b) \$num1 c) +add d) a_2	June/July 2016	CO1	BT4
3	Explain scanf and printf functions with example.	June/July 2016	CO1	BT1
4	List all the operators used in C give examples	June/July 2016	CO1	BT1
5	Write the output for the following code. <pre> 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=%d res2=%f",res1,res2); } ii) void main() { int i=5,j=6,m,n; m = ++ i + j ++; n= --i + j--; printf("m=%d n=%d",m,n); } </pre>	June/July 2016	CO1	BT1
6	Explain the structure of C program with an programming example.	Dec/Jan 2015, Dec 2011	CO1	BT1
7	Explain any five operators used in C language	Dec/Jan 2015, Jun/July-2015	CO1	BT1
8	What are type conversions? Explain two type of type conversions with example	Dec/Jan 2015, June 2012	CO1	BT1
9	write a program in C to find area and perimeter of rectangle	Dec/Jan 2015.- jun/july 2016	CO1	BT1
10	Define i) variable ii) constant iii) Associativity iv) Precedence (8M) - Dec/Jan 2015. What are data types? Mention the different data types supported by C language, giving an example to each	Jun/July-2015	CO1	BT1
11	Write a c program which takes as input p,t,r and computes simple interest	Jun/July-2015.	CO1	BT1

12	What is an operator? List and explain various types of operators	Jun/July-2015	CO1	BT1
13	What is a token? What are the different types of tokens available in C language? Explain	Jun/July-2015	CO1	BT1
14	What is the value of x in the 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;	Jun/July-2015	CO1	BT1
15	What are identifiers? Discuss the rules to be followed while naming identifiers? Give examples.(06 marks)	June/July 2013 Jun/July-2015	CO1	BT1
16	Explain format specifiers used in scanf() function to read int , float,char,double and long int data types.(06 marks)	June/July 2013	CO1	BT1
17	Write a c program to swap values of two integers without using a third variable and write flowchart for the same.(06 marks)	June/July 2013	CO1	BT1
18	Find the result of each of the following expression with i=4,j=2,k=6,a=2.(10 marks) i)k*=i+j ii)j=i/=k iii)i%=i/3 iv)m=i+(j=2+k)	June/July 2013	CO1	BT1
19	Explain the scope of local and global variables with examples(04 marks)	June/July 2013	CO1	BT1
20	Briefly explain how to create and run the program(04 marks) –	Jan 2013	CO1	BT1
21	Explain 5 types of data with its range and values(04 marks) –	Jan 2013	CO1	BT1
22	Explain scanf() and printf() functions with syntax. (06 marks)	Jan 2013	CO1	BT1
23	What do you mean by type conversion? Explain explicit type conversion with examples(04 marks)	Jan 2013	CO1	BT1

24	Explain the following operators with examples (09 marks) i)conditional ii)bitwise iii)sizeof	Jan 2013	CO1	BT1
25	Determine the value of each of the following logical expressions where a=5,b=10 and c=-6.(03 marks) i)a>b&&a>c ii)b>15&&c<0 a>0 iii)(a/2.0==0.0&&b/2.0!=0.0) c<0.0	Jan 2013	CO1	BT1
26	What are c tokens? Mention then? and explain any two tokens(08 marks)	Dec 2011	CO1	BT1
27	What is a datatype? explain the basic data types available in c (04 marks)	Dec 2011	CO1	BT1
28	What are variables how they are declared(04 marks)	Dec 2011	CO1	BT1
29	Write a program to find area of triangle given three sides(06 marks)	Dec 2011	CO1	BT1
30	With examples illustrate any four programming errors (04 marks)	Dec 2011	CO1	BT1
31	Write an algorithm and flowchart to calculate factorial of a number(6 marks)	June 2012	CO1	BT1
32	Explain precedence and associativity of operators in c with an example(08 marks) –June 2012	June 2012	CO1	BT1
33	List and explain coding constant(06 marks)	June/July 2011	CO1	BT1
34	If a=2,b=8,c=4,d=10 ,what is value of each of the following(04 marks) i)a+b/c*d-c/a ii)(b/a)%c iii)a++ + b-- + d++	June/July 2011	CO1	BT1
35	Write a c program to convert degrees into radians accepting the value from the user(06 marks)	Jan 2011	CO1	BT1
36	Explain the following operators with examples(08 marks) a. Relational ii)conditional iii)increment iv)special	May/June 2010	CO1	BT1

	operators			
37	What is the value of x if a=10,b=15 and (x>b)?a:b;(02 marks)	-Dec 2010	CO1	BT1
38	What would be the value of c if a=1,b=2?mention the steps involved(08 marks) C=(a>0&&a<=10)?a+b:a/b b. C=(a<0&&a<=10)?a+b:a/b	-Dec 2010	CO1	BT1
39	Draw a neat flowchart to exchange two numbers without using temporary variable(5 marks)	-Dec 2010	CO1	BT1
40	Evaluate the following expressions independent to each other, the declaration and initialization is as follows(05 marks): int i =3, j=4, k=2; i) i ++ -j-- ii) ++k %--j iii) j+1/i-1 iv) j++/i-- v) ++i/++j+1	-Dec 2010	CO1	BT5
41	What are escape sequences? why they are used? give examples(04 marks)	-June/July 08	CO1	BT1
42	What do you mean by mixed mode operation? Explain with an example(06 marks)	Dec 2010	CO1	BT1
43	Write a C program to find area of circle	Dec 2010	Co1	BT1
	Dec 2016/Jan 2017			
44	Define an Algorithm .write an algorithm to find area and perimeter of a rectangle.6M			
45	Write general structure of C ?Explain with an example. 6M			
46	Explain different types of input output function in c with syntax and examples. 6M			
47	Explain the following operators : Unary,binary conditional 6M			
48	Write an flowchart and C program to find simple interest. 4M			
	JUNE/JULY 2017			
49	Define pseudo code .Explain with an example. 5M			
50	Write a c program to find biggest among three numbers using ternary operator (5M)			

51	Explain the following constants with example(6M) i)integer constant ii)floating constant iii)character constant.			
52	List the formatted input/output functions of c language.explain the basic structure of c program with proper syntax and example.(6M)			
53	Define an algorithm. write an algorithm to find the area of circle and triangle.(6M)			
54	Evaluate the following expression: i) $22+3<6\&\&!5 \mid 22==7\&\&22-2>=5$ ii) $a+2>b \mid !c \&\&a==d \mid a-2<=e$ where a=11,b=6,c=0,d=7,and e=5			
55	List all the restrictions on the variable names(06 Marks)			
56	Explain the block structure of a "C Program" (08 Marks)			
57	What are the basic data types available in "C"? Write the significance of each datatype (06 Marks)			
58	What is an assignment statement? Give the general form of an assignment statement? (5 Marks)			
59	Explain with example, the various constants available in 'C' program(5 Marks)			
60	List and explain any five operators used in "C" programming Language(10 marks)			