

Programming in C and Data Structures (15PCD13/23)

Module 1

Introduction to C Language

Topics:

Pseudo code solution to problem, Basic concepts in a C program, Declaration, Assignment & Print statements, Data Types, operators and expressions etc., Programming examples and exercise.

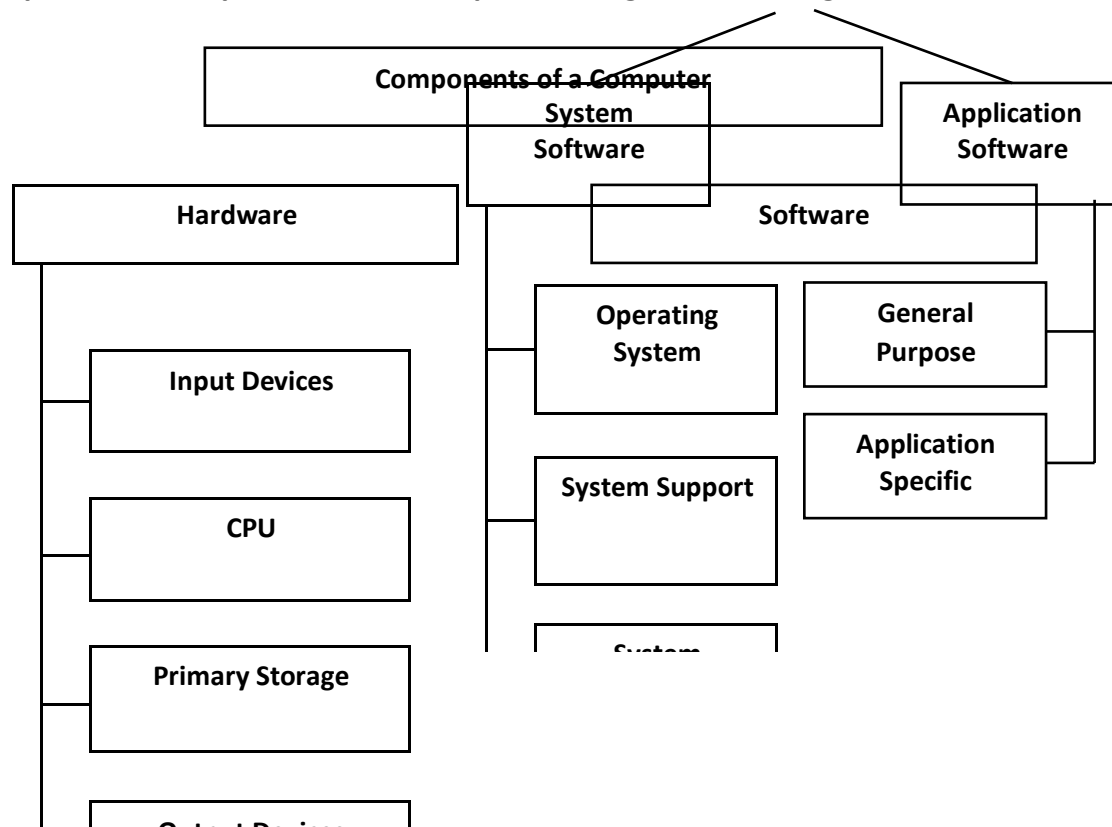
Course Outcome:

Understand the problem solving techniques in C programming.

1. What is a computer?

An electronic device which is capable of receiving information (data) in a particular form and of performing a sequence of operations in accordance with a predetermined but variable set of procedural instructions (program) to produce a result in the form of information.

2. Explain the components of a Computer along with the diagram.



Hardware	Software	
	System Software	Application Software
Input Devices Used to input the data. Ex : Keyboard, Mouse, Scanner	Operating System Interface, File access from database	General Purpose Ms-Office, Notepad, Ms-paint

CPU Used for Arithmetic calculations, comparing values, movement of data	System Support Provides System utilities, Disk fragmentation	Application Specific Ledger maintenance, Talley
Primary Storage It is the Main memory which is temporary. Also known as RAM	System Development Translators, Debugging tools	-
Output Devices Used to get the processed results. Ex : Monitor, Printer, Plotter	-	-
Auxillary Storage Secondary memory, Permanent memory. Ex : Hard Disk, Pen Drive	-	-

3. List the different Computer Languages.

1. Machine Level Language

- Binary language
- Contains only 0s and 1s
- Understood by Computer only

2. Symbolic Level Language

- Assembly Language
- Mnemonics

3. High Level Language

- Contains English words
- Understood by Humans
- FORTRAN(Formula Translation) , COBOL(Common Business Oriented Language)

4. Who invented C programming Language and what are its features?

C was designed and developed by Dennis Ritchie at Bell Laboratories in 1972.

Features of C are

1. Simplicity:

C has a good collection of inbuilt functions, keyword and data types. The keyword and library functions available in C resembles common English words thus it helps to improve the clarity and simplicity of the program.

2. Modularity:

Ability to breakdown a large module into manageable sub modules called as modularity that is an important feature of C programming languages.

3. Portability:

The ability to port i.e. to install the software in different platform is called portability. Highest degree of portability: 'C' language offers highest degree of portability i.e., percentages of changes to be made to the sources code are at minimum when the software is to be loaded in another platform. Percentage of changes to the source code is minimum.

4. Extendibility:

Ability to extend the existing software by adding new features is called as extendibility.

5. Speed:

C' is also called as middle level language because programs written in 'c' language run at the speeds matching to that of the same programs written in assembly language so 'c' language has both the merits of high level and middle level language and because if this feature it is mainly used in developing system software.

6. Case sensitive:

C language is a case sensitive language and it can differentiate the character either in upper case or lower case. All types of words in C language are case sensitive.

5. Define an algorithm?

Algorithm is a step by step procedural description of the programming logic. An algorithm must have finite number of steps, must be complete and unambiguous and error free.

6. What are the characteristics of algorithm?

1. Finiteness:

An algorithm must terminate after a finite number of steps and further each step must be executable in finite amount of time that it terminates (in finite number of steps) on all allowed inputs.

2. Definiteness (no ambiguity):

Each steps of an algorithm must be precisely defined; the action to be carried out must be rigorously and unambiguously specified for each case.

3. Input:

An algorithm has zero or more but only finite, number of inputs.

4. Output:

An algorithm has one or more outputs. The requirement of at least one output is obviously essential, because, otherwise Effectiveness cannot know the answer/ solution provided by the algorithm.

5. Effectiveness:

An algorithm should be effective. This means that each of the operation to be performed in an algorithm must be sufficiently basic that it can, in principle, be done exactly and in a finite length of time, by person using pencil and paper.

7. What is the need for an algorithm?

- 1. Effective communication:-** Since algorithm is written in English, it is easy to communicate to











people.

2. **Effective analysis:-**with the help of an algorithm, problem can be analyzed effectively.
3. **Proper documentation:-** they serve as a good documentation which is required to identify logical errors.
4. **Easy and efficient coding:-** the algorithm acts as a blueprint during program development.
5. **Program debugging:-** they help in debugging so that we can identify logical errors.
6. **Program maintenance:-** maintaining the software becomes easier.

8. Define flowchart? Explain the meaning of symbols used.

Pictorial representation of flow of logical steps followed in an algorithm.

Flow chart symbols

	Oval	Terminal
	Parallelogram	Input/output
	Rectangle	Process
	Document	Hard copy
	Diamond	Decision
	Circle	Connector
	Double sided Rectangle	Sub program
	Hexagon	Iteration
	Trapezoid	Manual Operation
	Cylinder	Magnetic Disk Storage

9. Define Pseudocode

Pseudocode is a problem solving tool which contains statements written in English and C language. It is a series of steps which describes what must be done to solve a problem. It is used for developing a program and written before the program. More closer to the programmers.

10. What are the advantages and disadvantages of flowchart?**Advantages:**

Effective communication:-Flowcharts are a better way of communicating logic to the people

Effective analysis:-with the help of a flowchart, problems can be analysed effectively

Proper documentation:-they play an important role in understanding the logic of complicated and lengthy problems.

Easy and efficient coding:- by looking at the flowchart writing programs is a straight forward method.

Program debugging:- flowcharts help in debugging process.

Program maintenance:- maintaining the software becomes much easier.

Disadvantages:

Complex logic:-for complicated logic, the flowchart becomes complex and clumsy.

Alterations and modifications:- if there is a change in the logic, the flowchart has to be completely rewritten and requires a lot of time.

Reproduction:- in case of any problem, reproduction of flowchart becomes problem since the flowchart symbols cannot be typed.

11. Explain the basic structure of a C program

The list of components of a C program is as follows

1. Comments
2. The program header
3. The body or the action part

Comments:

Comment lines in a program gives short description OR purpose of the program or any statement of the program.

Comment is not strictly necessary because a program without it is technically correct however if included it is considered as a good programming style.

Comments can be of 2 forms:

1. **Single line comment:** it is used to write single line description and the symbol used is “//”
2. **Multi line comment:** it is used to write descriptions in many lines and the symbol used are
/*→ comment begins, */→comment ends.

The program Header:

The program header includes the following sections

1. Pre-processor directives
2. main program header

Pre-processor directives:

The pre-processor directives are the statements which start with symbol #. This statement instructs the compiler to include some of the files in the beginning of the program.

Ex: #include<stdio.h>

→ instruction to the compiler

include→ to include files as a part of our program

<stdio.h>→<> pointed brackets tell the compiler, the exact location of this header file.

stdio :Standard input/ output

. h : header file.

main program header:

The main statement instructs the compiler that execution always starts from function main and it is called main program header. The pair of brackets denoted () must follow after the main to indicate it as a function. Every C program must have only one main.

The body or the action part:

Inside every C main program is a set of braces {and} containing a series of C statements which comprise the body. This is called the action part of the program.

The body consists of two essential parts

Declaration section

Execution section

Declaration section:

The variables that are used inside the main function or sub function should be declared in the declaration section. The variables can be initialized during declaration.

Ex:

```
main()
{
    int sum =0;           // initialization
    int a,b;              // declaration
    .....
}
```

Executable section:

This section includes the instructions given to the computer to perform some specific task. An instruction may represent an expression to be evaluated, input /output statements etc. Each executable statement ends with ;

Ex:

```
main()
{
    .....
    printf("enter two numbers");
    scanf("%d%d", &a, &b);

    sum = a+b;
    printf("sum = %d", sum);
}
```

Alphabets, Tokens, Data Types, variables and Constants

Character set (Alphabets of C):

A symbol that is used while writing a program is called a character or an alphabet. A character can be a letter, digit or any special symbol. Using these alphabets, we can obtain meaningful numbers, statements and expressions.

- Alphabets A - Z, a - z
- Digits 0 - 9
- Special symbols * , @ , % , { , [,) , #

- White spaces new line, horizontal tab

Tokens:

A token is the smallest or basic unit of a C program. One or more characters are grouped in sequence to form meaningful words. These meaningful words are called tokens.

Examples

i) $c = a + b * 10$; Here, c , $=$, a , $+$, b , $*$, 10 each one is called as a token.

ii) $\text{Result} = (x * y) * (x - y)$;

The tokens in C language are broadly classified as shown below:

Types of Tokens:

1. Keywords
2. Identifiers

3. Constants

4. Operators

1. Keywords (Reserved words)

- Reserved words which have a pre-defined meaning in C compiler.
- Used for specific task in C language.
- The keywords in C are in lowercase.

Some keywords

int	float	if	else
for	while	switch	break
long	void	char	double
continue	do	default	short
delete	static	const	unsigned

2. Identifiers

Names provided to the elements of the program such as variables, functions and arrays.

Rules for naming an identifier

1. Can consist of alphabets, digits and _ (underscore).
2. Must always begin with an alphabet or _ and can be followed by digits.
3. No special symbols allowed.
4. Case sensitive.
5. Cannot be a keyword.

Valid examples of Identifiers

Total	sum1
Voter_ID	roll_no

Invalid Identifiers

- for - keyword
- 1stname - beginning with a digit

- sum-of-digits - illegal use of special symbol -
- roll number - illegal use of special symbol space
- #USN NO - illegal use of special symbol #
- I Class - illegal use of special symbol space
- void - keyword
- stream(branch) - illegal use of special symbols ()

3. Constants(Literals)

A fixed value assigned to an identifier whose value cannot be changed.

Types of constants

1. Integer constant
2. Floating point constant
3. Character constant

4. String constant
5. Backslash constants (escape sequence)

1. Integer constant

It represents a whole number without decimal part.

Subtypes

- a) Decimal
- b) Octal
- c) Hexadecimal

a) Decimal

- Constants having digit combination between 0 – 9.
- Can be + or –

Valid ex: -45 756 1234

Invalid ex : 102 56 4.50 -+56

b) Octal

- Constants having digit combination between 0 – 7.
- Can be + or –
- The constant must have the prefix 0 (Zero).

Valid ex: 077 -0756 01234

Invalid ex : 0102 56 -017.7 856

c) Hexadecimal

- Constants having digit combination between 0 – 9 and A – F.
- A represents 10, B represents 11, C represents 12, D represents 13, E represents 14, F represents 15
- Can be + or –
- The constant must have the prefix 0X / 0x.

Valid ex: 0X917 -0x1AF 0x304a

Invalid ex : 18C 0X1GA 0x4.6E

2. Float constant

- Represents a number having fractional part.
- May be + or –

Examples

-221.45

44567.34

0.000009

67.89E2---> $67.89 * 10^2$

67.89e-3 ----> $67.8 * 10^{-3}$

12.0

3. Character constant

- Represents a single character within a pair of ‘ ‘
- Stored as integers in computer’s memory.

Examples

```
char op = '+';char choice = 'Y';  
charnum = '7'; char ch='n';
```

4. String constant

Represents a sequence of characters within a pair of “ ”

Examples

```
“Good Morning”          “10 + 40”  
“G.A.T”                 “#163/13, Bangalore”
```

5. Backslash constants (Escape sequence)

Non printing characters can be printed using escape sequence.
It always starts with \ followed by a character.

Escape sequence	Description
\a	System alarm
\b	Backspace
\n	Newline
\t	Horizontal tab
\"	Double quote
\'	Single quote
\\	\ (backslash)
\0	Null character used to terminate a string

12. Define data type. List the primitive (built-in) data types.

Defines the type of data that is stored in a variable.

Determines how much storage should be allocated to a variable.

Built-in data types in C

Data type	Description	Size (bytes)	Range
bool	True/false	1	-
char	Stores single character	1	-128 to 127
int	Stores integer values	2	- 32768 to 32767
float	Stores real numbers	4	3.4×10^{-38} to 3.4×10^{38}
double	Stores real numbers	8	1.7×10^{-308} to 1.7×10^{308}
void	Stores no values Returns no values	-	-

1. bool

It is used to store the value wither true or false. True is represented as 1 and false as 0.

Ex ample:

```
bool a;  
a=true;
```

2. char

Each character constant is stored internally in the computer as integer constant. This integer constant is called as ASCII (American Standard Code for Information Interchange)

Example :

```
void main()  
{  
    charvar1 ='a';  
    char var2 = 'E'  
  
    char var3 = '7', var4 = '*';  
}
```

3. int

An integer datatype is used along with a variable to store a whole number.

signedint : +ve or -ve values

unsignedint : only +ve values

Example :

```
void main()  
{  
    int a = 10;  
    int b, sum;  
    b=60;  
  
    sum=a+b;  
}
```

4. float

A float datatype is used along with a variable to store a real number.

Example :

```
void main()
```

```
{    float a, b, c;  
    a=7.1;  
    b=77.89;  
  
    c=a-b;  
  
}
```

5. double

A double datatype is used along with a variable to store a real number having a range higher than floating point number.

Example :

```
void main()

{
    double a, b, c;
    a=78907.1976;
    b=751237.89;

    c=a*b;

}
```

6. void

Stores no values, non-specific data type.

13. What is a variable? List the rules in naming a variable.

Variable

- An identifier which can store a value.
- The value may change later in the program.
- Refers to a specific memory location where the data can be stored.

Rules for naming a variable

- Can consist of alphabets, digits and _ (underscore).
- Must always begin with an alphabet or _ and can be followed by digits.
- No special symbols allowed.
- Case sensitive.
- Cannot be a keyword.

Declaring a Variable

Every variable has to be declared before its first use.

Syntax:

Data_type variable name;

Example:

int a;

float num1, num2;

char grade, name[21];

Initializing a Variable

Here, the value can be assigned to a variable by the programmer.
For every execution, the value of the variable may be same.

Example:

```
int a =40;
```

```
float num1, num2;
```

```
num1=67;
```

```
num2=6.8;
```

```
char grade='A';
```

Note:

When a variable is defined, it is not initialized. We must initialize any variable requiring prescribed data when the function starts.

Example of uninitialized variable

```
#include<stdio.h>
void main()
{
    int x, y, z;
    z=34;

    printf("x=%d y=%d z=%d\n", x, y, z);
    x=z;

    y = x + 10;

    printf("x=%d y=%d z=%d\n", x, y, z);
}
```

Output

```
x=5678(garbage) y=678(garbage) z=34
x=34    y=44    z=34
```

Formatted Input/output functions

1. printf()
2. scanf()

printf()

The statement which instructs the compiler to display or print the string, expression value or variable value to the output device is called print statement. The printed results of running program are called output. In C programming language the output is printed using formatted output statement **printf()**.

There are two forms of **printf()**

1. That has a literal string – a sequence of characters within quotation marks.

Syntax:

```
printf("literal string");
```

Example:

```
printf("welcome to C programming\n");
```

2. That has literal string, conversion specifier and any or all of the following: variables, constants and expressions values to be printed.

Syntax:

```
printf("Literal string/ conversion specifier" , list of variables);
```

“Format string”

Example:

```
printf(“%d %d” , number, square);  
printf(“ the sum = %d”, sum);
```

Note:

The no of variable names should be equal to the no of format specifiers.

The various format specifiers /conversion specifiers used in printf() are as follows:

int	%d	data is displayed as d ecimal signed integer
	%o	data is displayed as o ctal integer
	%x	data is displayed as hexadecimal integer (lowercase letters)
	%X	data is displayed as hexadecimal integer (uppercase letters)
	%i	data is displayed as decimal integer, octal or hexadecimal.
	%u	data is displayed as unsigned integer

	%h %ld	data is displayed as short integer data is displayed as long integer
float	%f %e %E %g	data is displayed as floating point value without exponent. data is displayed as floating point value with exponent (lowercase e) data is displayed as floating point value with exponent (uppercase e) data is displayed as floating point value with or without exponent.
char	%c %s	data is displayed as a character data is displayed as a string (sequence of characters)
double	%lf	data is displayed as long float.

Examples: on printf()

```
printf("%d%c%f", 23, 'z', 4.1);printf("%d %c %f", 23, 'z', 4.1);
```

23z4.100000

23 z 4.100000

```
int num1 = 23;
```

```
char zee = 'z';
```

```
float num2 = 4.1;
```

```
printf("%d %c %f", num1, zee, num2);
```

23 z 4.100000

```
int a=5,b=10;
```

```
printf("a=%d\tb=%d", a,b);
```

output

a=5 b=10

```
int a=5,b=10;
```

```
printf("a=%d\nb=%d", a,b);
```

output

a=5

b=10

```
printf("The tax is %8.2f this year",233.12);
```

output

The tax is 233.12 this year

```
printf("The tax is %08.2f this year",233.12);
```

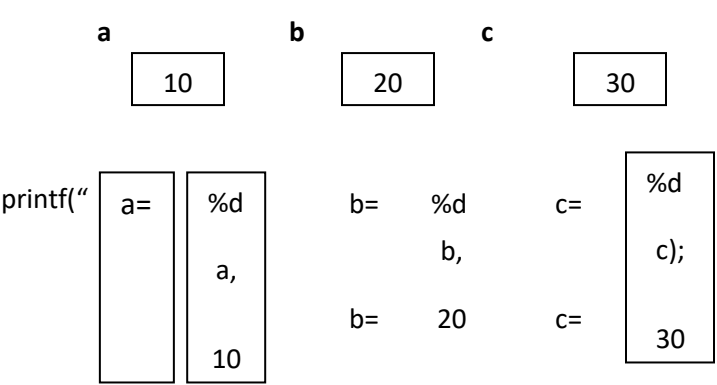
output

The tax is 00233.12 this year

Pictorial representation for the program segment

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

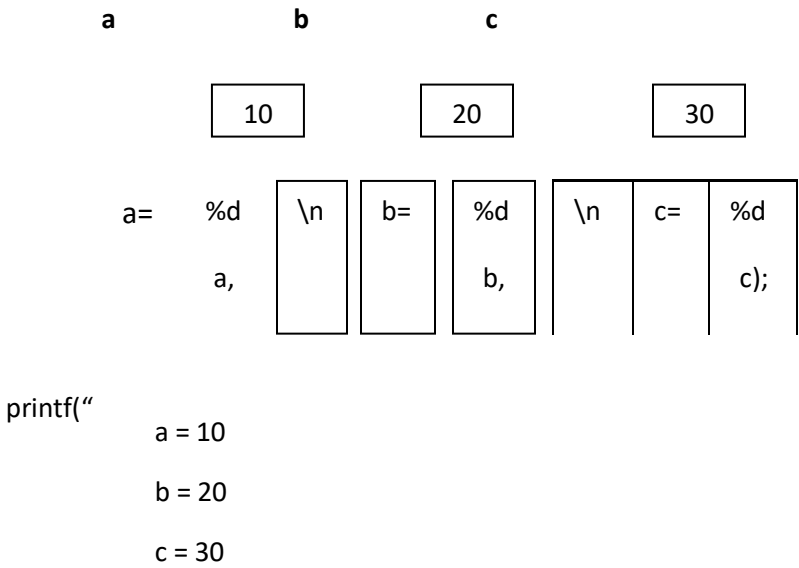
printf("a=%d b=%d c=%d", a,b,c);
```



Pictorial representation for the program

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

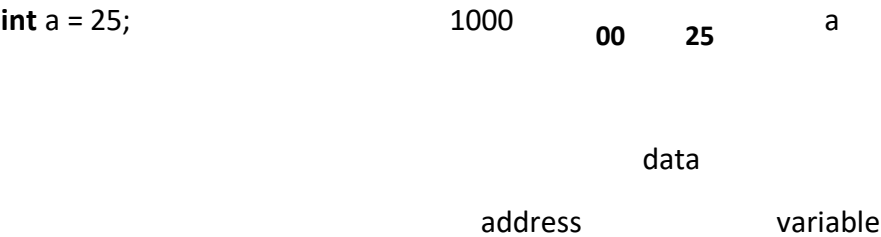
printf("a=%d\n b=%d\n c=%d", a,b,c);
```



The Address operator (&)

The operator that is used to obtain the address of a variable is called address operator and it is denoted by the symbol `&`.

Consider the following statement:



the two bytes of memory space is reserved for the variable `a` starting from the address 1000 and the integer data 25 is stored in memory as shown above

now , to print the value of `a` and the address of `a`, we use the following statements:

```
printf("Value of a = %d\n", a);

printf("Address of a = %u\n", &a);
```

scanf()

Data read by a program can come from two places: the keyboard or a data file. In C programming language the data is read from the keyboard using formatted input statement *scanf()*.

Syntax:

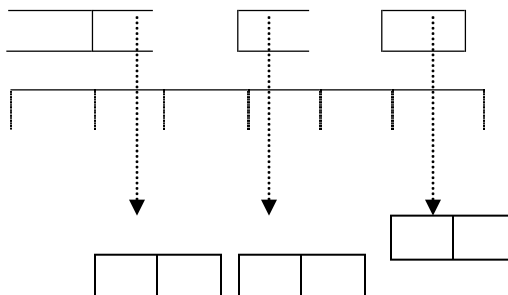
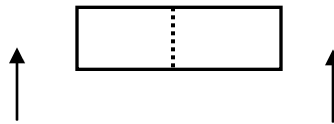
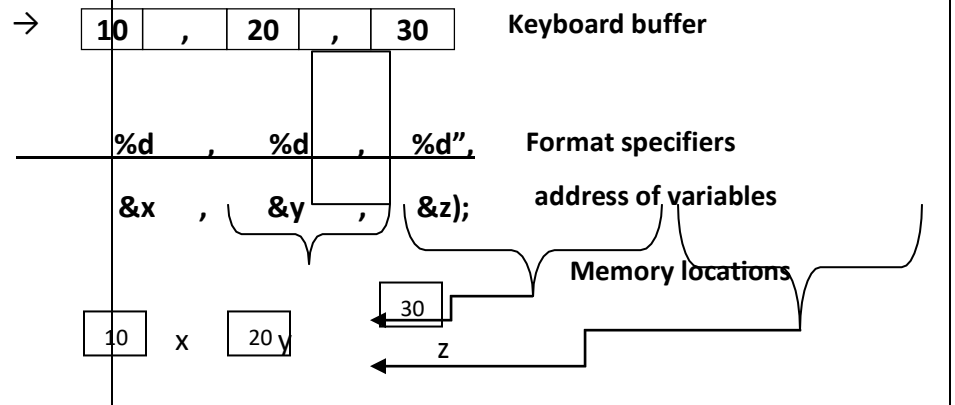
```
scanf("format string", address_list);
address_list----- >&variablename
```

Explain the instruction for the input data:

10, 20, 30

```
scanf ("%d, %d, %d", &x, &y, &z);
```

scanf(")



Examples

10 10.5 A

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

3.1416, 10 Z

```
scanf("%f,%d %c",&k, &j, &h);
```

```
scanf("%d#%d$%f",&a,&b,&c);
```

9#10\$3.1416

10,20,30

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

9-10-2010

```
scanf("%d-%d-%d",&day,&month,&year);
```

```
scanf("%1d %2d %3d",&x,&y,&z);
```

123456789

X = 1

Y = 23

Z = 456

Differentiate between inputting a variable and initialize a variable.

Input	Initialization
Used to input the value to a variable.	Used to assign a value to a variable.
The value is input by the user.	The value is initialized by the programmer.
Scan() is used to input the value.	= sign is used to initialize a variable.
The input value may be different for each execution.	The initialized value will be same for all executions.
Ex: int a; scanf("%d",&a);	Ex: Int a; a = 67;

Sample programs

1. Write a program to input length and breadth and find the area of a rectangle.

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

```
void main()
```

```
{ float length, breadth, area;
```

```
printf("Enter length and breadth\n");
scanf("%f %f",&length, &breadth);
area= length * breadth;

printf("area=%f",area);

}
```

2. Write a program to input P, T and R, calculate simple interest.

```
#include<stdio.h>
void main()
{ float P, T, R, SI;

  printf("Enter P,T,R\n");

  scanf("%f %f %f", &P, &T, &R);
  SI = (P*T*R)/100;

  printf("Simple Interest=%f",SI);

}
```

3. Input 2 values, simulate a calculator.

```
#include<stdio.h>
void main()
{
    float a, b, sum, diff, pro, div;
    printf("Enter 2 numbers\n");
    scanf("%f %f",&a,&b);
    sum=a+b;

    diff=a-b;
    pro=a*b;
    div=a/b;

    printf("Sum=%f\nDifference=%f\n",sum,diff);
    printf("Product=%f\nQuotient=%f\n",pro, div);
}
```

4. Input 2 values, swap them using a temporary variable and print the new values.

```
#include<stdio.h>
void main()
{ int A, B, temp;

    printf("Enter 2 integers\n");
    scanf("%d %d", &A, &B);
    temp = A;

    A = B;

    B = temp;

    printf("A=%d\tB=%d",A, B);
}
```

5. Input 2 values, swap them without using a temporary variable and print the new values.

```
#include<stdio.h>
void main()
{ int A, B;

    printf("Enter 2 integers\n");
    scanf("%d %d", &A, &B);

    A = A + B;

}
```

```
B = A - B;  
A = A - B;  
printf("A=%d\tB=%d",A, B);  
}
```

Symbolic Constants

Name that can be used as a substitute for a numeric constant, a character constant or string constant.

Ex:

```
#include<stdio.h>  
#define PI 3.142  
void main()  
{   float radius, area;  
    radius = 7;  
  
    area = PI * radius * radius;  
    printf("%f",area);
```

```
}
```


Assignments – C Programming

1. Input radius of a circle, find the area and circumference of the circle.

```
#include<stdio.h>
void main()
{
    float r,area,cir;

    printf("Enter radius of a circle\n");
    scanf("%f",&r);

    area=3.14*r*r;
    cir=2*3.14*r;

    printf("Area=%f\nCircumference=%f",area,cir);
}
```

2. Input marks in 5 subjects, find the total and average.

```
void main()
{
    float m1,m2,m3,m4,m5,total,avg;
    printf("Enter 5 marks\n");

    scanf("%f%f%f%f%f",&m1,&m2,&m3,&m4,&m5);
    total=m1+m2+m3+m4+m5;

    avg=total/5;

    printf("Total=%f\nAverage=%f",total,avg);
}
```

3. Input value in degrees and convert to radians radians = degrees * π / 180

```
void main()
{
    float deg,rad;

    printf("Enter value in degrees\n");
    scanf("%f",&deg);
    rad=deg*3.14/180;

    printf("Radians=%f",rad);
}
```

4. Input 3 sides of a triangle a, b, c and find the area. area = $\sqrt{s(s-a)(s-b)(s-c)}$ s = (a + b + c)/2

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

```
#include<math.h>
void main()
{ float a,b,c,s,area;

    printf("Enter 3 sides of a triangle\n"); scanf("%f%f%f",&a,&b,&c);
    s=(a+b+c)/2;

    area=sqrt(s*(s-a)*(s-b)*(s-c));
    printf("Area=%f",area);

}
```

5. Input value in Fahrenheit and convert to Celsius $C = 5/9 * (F - 32)$

```
void main()
{ float f,c;

    printf("Enter temperature in Fahrenheit\n");
    scanf("%f",&f);

    c=5.0/9 *(f-32);
    printf("Celcius=%f",c);
```

```
}
```

6. Input an integer, find the right most digit and print it. Ex : n = 6745 right most digit = 5

```
void main()
{
    int n,rem;

    printf("enter an integer\n");
    scanf("%d",&n);

    rem=n%10;

    printf("the right most digit=%d",rem);
}
```

Operators and Expressions:

Operator: An operator is a symbol that tells the compiler to perform specific mathematical or logical functions using the operands.

Operand: A constant or a variable or a function which returns a value is an operand.

Expression: A sequence of operands and operators that reduces to a single value is known as expression.

Example:

Expression

c = a + b ;

Operands

Operators

Operator classification:

The operators in C language are classified based on:

1. The number of operands
2. The type of operation

Classification based on the number of operands:

The operators are classified into

a. Unary operators:

An operator that operates on one operand to produce a result is called unary operator.

Eg: ++, --, sizeof(), casting

Contains single operator, single operand

Ex :+ : 4, +6
- : -10, -(6+10)

Example

int a=114, b= -3 ,c;

c = - (-a + b);

working : $c = - (- (a) + b) = - (-114 + (-3))$

$c = - (-114 - 3) = - (-117)$

c = 117

b. Binary operators:

An operator that operates on two operands in an expression to produce a result is called a binary operator.

- Eg: a+b, 10/5, +, -, *, /, %

Examples

- | | |
|-------------------------------|----------------------------|
| 1. $7 + 3 = 10$ | 8. $'1' + 3 = 49 + 3 = 52$ |
| 2. $'A' * 2 = 65 * 2 = 130$ | 9. $-5 / 2 = -2$ |
| 3. $7 / 2 = 3$ | 10. $-5 \% 2 = -1$ |
| 4. $7.5 / 2 = 3.75$ | 11. $5 / -2 = -2$ |
| 5. $7.0 / 2 = 3.5$ | 12. $5 \% 2 = 1$ |
| 6. $10 \% 3 = 1$ | 13. $-5 / -2 = 2$ |
| 7. $'a' \% 10 = 97 \% 10 = 7$ | 14. $-5 \% -2 = -1$ |

c. Ternary operators :

An operator that operates on three operands to produce a result is called ternary operator. also called as the conditional operators.

Eg: $a ? b : c$

Examples :

<code>int x , y =7, z = 10;</code>	Working : Since $7 \geq 10$ is false, 100 is evaluated to x;
<code>x = y >= z ? 70 : 100;</code>	<code>x = 100</code>
<code>x = 7 >= 10 ? 70 : 100</code>	

d. Special operators:

Ex: `comma(,)` `sizeof()` etc.

Classification based on the type of operation:

- | | |
|--------------------------------------|-------------------------------|
| a. Arithmetic operators | $+, -, *, / , \%$ |
| b. Relational operators | $< , <= , > , >= , == , !=$ |
| c. Logical operators | $! , \&\& , $ |
| d. Assignment operators | $= , += , -= , *= , /= , \%=$ |
| e. Increment and decrement operators | $++ , --$ |
| f. Conditional operators | |
| g. Bitwise operators | $\sim , << , >> , \& , , ^$ |
| h. Special operators | |

a. Arithmetic operators

The operators that are used to perform arithmetic operations such as addition, subtraction, multiplication, division and modulus operations are called as arithmetic operators. These operators perform operations on two operands and hence they are binary operators.

Arithmetic operators	Addition	+	add	4+2	6
	Subtraction	-	subtract	4-2	2
	Multiplication	*	multiply	4*2	8
	Division	/	divide	4/2	2
	Modulus	%	remainder	4%2	0

Note: Modulus operation denoted by % divides first operand by the second operand and returns the remainder. Remainder is the result obtained after modulus operation.

Modulus operation is allowed on integers and not allowed on other data type.

C program showing the usage of arithmetic operators

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

```
void main( )
```

```
{
```

```
    int a, b, c, d, e, f, g ;
```

```
    a=10, b=2;
```

```
    c=a+b;
```

```
    d=a-b;
```

```
    e=a*b;
```

```
    f=a/b;
```

```
    g=a%b;
```

```
    printf("%d + %d = %d\n", a, b, c);
```

```
    printf("%d - %d = %d\n", a, b, d);
```

```
    printf("%d * %d = %d\n", a, b, e);
```

```
    printf("%d / %d = %d\n", a, b, f);
```

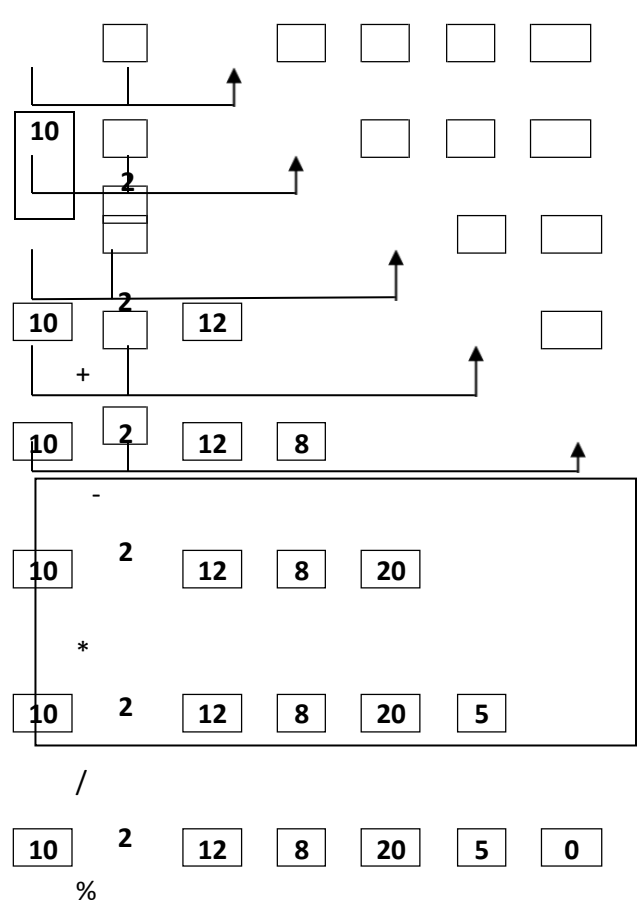
```
    printf("%d %% %d = %d\n", a, b, g);
```

```
}
```

Note: prints one symbol %

TRACING

a b c d e f g



Output

10 + 2 = 12

10 - 2 = 8

10 * 2 = 20

10 / 2 = 5

Conversion of expressions

Expressions that we use in mathematics are different from the expression that we use in C language. The expressions must be written in single line.

The rules to be followed while converting from mathematical expressions to C expressions are:

- a. if numerator has more than one operand, enclose the entire numerator inside the parentheses
- b. if denominator has more than one operand, enclose the entire denominator inside the parentheses

Mathematical expression	C expression
$s = \frac{a + b + c}{2}$	<code>s = (a + b + c) / 2</code>
$x = \frac{-b}{2a}$	<code>x = -b / (2*a)</code>
$\text{area} = \sqrt{s(s-a)(s-b)(s-c)}$	<code>area = sqrt(s* (s-a)*(s-b)*(s-c))</code>

$ax^2 + bx + c$	$a*x*x + b*x + c$
$\sin \left(\frac{b}{\sqrt{a^2 + b^2}} \right)$	$\sin (b / \text{sqrt} (a*a + b*b))$

Type conversion:

In C language, the programmer can instruct the compiler to convert the data from one data type to another data type. Sometimes, the compiler itself will convert the data from one type to another data type. This process of converting the data from one type to another data type is called type conversion.

Implicit conversion $1 / 2.0$ $1.0 / 2.0$ 0.500000

Type conversion

Explicit conversion $1 / (\text{float})2$ $1.0 / 2.0$ 0.500000

C can evaluate expressions if and only if the data types of two operands are same. So, if operands are of different data type, it is not possible to evaluate the expression. In such situation, to ensure that both the operands are same the compiler converts the data from lower rank to higher rank. This process of conversion of data from lower rank to higher rank automatically by the C compiler is called **implicit conversion**.

Example:

```
int + int = int
5 + 3 = 8

8
(int)
```

```
int + float = float
5 + 3.5 = 8.5

5.0 + 3.5
```

```
int + double = double
5 + 2.112 = 7.112

5.0 + 2.112
```

Explicit conversion/ Type casting:

If the operands are of the same data type, no conversion takes place by the compiler. Sometimes, the type conversion is required to get the desired results. In such case, the programmer can instruct the compiler to change the type of the operand from one data type to another data type. This forcible conversion from one data type to another data type is called explicit type conversion.

Syntax:

(type)expression

Examples:

`i = (int) 8.9999` 8.9999 is truncated to 8 and is stored in i so the value of **i = 8**

`i = (int)5.99 / (int) 2.4` 5.99 is truncated to 5 and 2.4 is truncated to 2 which results in $5 / 2 = 2$.
Hence **i= 2**

$i = (\text{float}) 1 / 2$ 1 is converted to 1.0. Next implicit conversion is done on 2. Thus we get $1.0 / 2.0 = 0.5$. Hence $i = 0.500000$

Modes of arithmetic expression:

The mode of expression depends on the type of operands used in the expression. Based on the type of the operands in the expression, there are 3 modes of expressions

a. Integer expressions

$$4 / 2 = 2$$

b. Floating point expressions

$$4.0 / 2.0 = 2.000000$$

c. Mixed mode expressions

$4.0 / 3 = 1.333333$ →
 i.e. $4.0 / 3.0 = 1.333333$ after conversion

Precedence of operators:

Arithmetic operators	Addition	+	add	2	left to right	Precedence
	Subtraction	-	subtract	2	left to right	
	Multiplication	*	multiply	1	left to right	

}

}

Division	/	divide	1	left to right
Modulus	%	remainder	1	left to right

}

Example:

Evaluate the expression $2 * ((a \% 5) * (4 + (b - 3) / (c + 2)))$ assume $a=8$, $b=15$ and $c=4$

$$2 * ((a \% 5) * (4 + (b - 3) / (c + 2))) \quad \text{substitute value for a, b, c}$$

$$2 * ((8 \% 5) * (4 + (15 - 3) / (4 + 2)))$$

$$2 * (3 * (4 + (15 - 3) / (4 + 2)))$$

$$2 * (3 * (4 + 12 / (4 + 2)))$$

$$2 * (3 * (4 + 12 / 6))$$

$$2 * (3 * (4 + 2))$$

$$2 * (3 * 6)$$

$$2 * 18$$

36

Associativity: Associativity determines how operators with the same precedence are evaluated in an expression. It can be classified into two types

1. Left associative : expression is evaluated from left to right
2. Right associative : expression is evaluated from right to left

1. Write a program to input an integer and find the right most digit.

```
void main()
{
    int num,rem;

    printf("enter an integer\n");
    scanf("%d",&num);
    rem=num%10;

    printf("The right most digit is %d", rem);
}
```

2. Write a program to input an integer and find its digit at 10's place.

```
void main()
{
    int num,rem;

    printf("enter an integer\n");
    scanf("%d",&num);
    num=num/10;
    rem=num%10;

    printf("The digit at ten's place is %d",rem);
}
```

3. Write a program to input 2 integers and find the remainder without using % operator.

```
void main()
{
    int a, b, q, rem;

    printf("enter 2 integer\n");
    scanf("%d %d",&a, &b);

    q = a/b;

    rem = a - (q * b);

    printf("Quotient=%d\nRemainder=%d", q, rem);
}
```

}

b. Relational operators

The operators that are used to find the relationship between two operands are called relational operators. The relationship between two operand values results in true or false

The relational operators are

	Description	operator	priority	Associativity
Relational Operators	Less than	<	1	left to right
	Less than or equal to	<=	1	left to right
	Greater	>	1	left to right
	Greater than or equal to	>=	1	left to right
	Equal	==	2	left to right
	Not equal	!=	2	left to right

Equality operators

Examples:

$a + b > c$	Valid
$b * b - 4 * a * c == 0$	Valid
$b * b - 4 * a * c > 0$	Valid
$a < b$	Invalid – operators used are not in order
$a < = b$	Invalid – space between < and = not allowed

Precedence of operators:

The rules to be followed while evaluating the relational expressions are shown below:

- Evaluate the expression within parentheses
- Evaluate unary operators
- Evaluate arithmetic expressions
- Evaluate relational expressions

Example:

Evaluate the following expression $100 / 20 <= 10 - 5 + 100 \% 10 - 20 == 5 >= 1 != 20$

$100 / 20 <= 10 - 5 + 100 \% 10 - 20 == 5 >= 1 != 20$

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

5 <= 10 - 5 + 0 - 20 == 5 >= 1 != 20

5 <= 5 + 0 - 20 == 5 >= 1 != 20

5 <= 5 - 20 == 5 >= 1 != 20

5 <= -15 == 5 >= 1 != 20

0 == 5 >= 1 != 20

$$\underbrace{0 == 1}_{\text{true}} != 20$$

$$0 \quad != 20$$

$$1$$

$$\underbrace{\quad}$$
Examples

1. int a = 4 , b = 78 , c;

c = a == b ; printf("%d " , c);

c = 4 != 78 ; printf("%d" , c);

Output :

0 1

2. int a=12, b=3;

printf("%d\n" , (a==b));

printf("%d" , (a=b));

Output :

0

3

3. int a=6,b=8, c;

c= a+10 >=b;

printf("%d" , c);

Output :

1

c. Logical operators

Used to combine two expressions. The result will be either true or false.

Types

- Logical NOT ! 1
- Logical AND && 2
- Logical OR || 3

Logical NOT!

It reverses the logical value of a variable or a constant or an expression.

Truth table

A	! A
0	1
1	0

Examples

$!(3 > 4) \text{ ----} \rightarrow !(false) = true = 1$

$!(4==4) \text{ -----} \rightarrow !(true) = false = 0$

$!7 \text{ ----} \rightarrow !(true) = false = 0$

$!0 \text{ -----} \rightarrow !(false) = true = 1$

$!(-45) \text{ -----} \rightarrow !(true) = false = 0$

In C, 0 is considered as false and any non-zero value is considered as true

Logical AND &&

- Works on 2 operands.
- Result is true if values of both expressions are true.
- Result is false, if any one expression value is false.

Truth table

A	B	A && B
0	0	0
0	1	0
1	0	0
1	1	1

Example

78 > 3 && 4 <=3;
 (78>3) && (4<=3) = 1 && 0 = 0

Write a C expression to check whether marks is between 75 and 100

marks >= 75 && marks<=100;

Logical OR | |

- Works on 2 operands.
- Result is false if values of both expressions are false.
- Result is true, if any one expression value is true.

Truth table

A	B	A B
0	0	0
0	1	1
1	0	1
1	1	1

Example

(10!=10) | | (5<4) && 8

0 | | 0&&1

0 | | (0&& 1)

0 | | 0

0

2. a>=b && ! (a+b > a+1) if
 a=3, b=0 3>=0 && !(3+0 >
 3+1)

1 && !(3 > 4)

1 && !(0)

1 && 1

1

Evaluate the following expressions

1. 10!=10 | | 5<4 && 8

$78 > 3 \mid\mid 4 \leq 3;$

$(78 > 3) \mid\mid (4 \leq 3) = 1 \mid\mid 0 = \mathbf{1}$

3. $a \geq b \ \&\& \ ! \ (a+b) > (a+1)$ if $a=3, b=0$

$3 \geq 0 \ \&\& \ !(3+0) > (3+1)$

1 $\&\& \ (!3) > 4$

1 $\&\& \ 0 > 4$

1 $\&\& \ 0 > 4$

1 $\&\& \ 0$

0

d. Assignment operators

It is used to assign the value of a constant or variable or value of an expression to a variable on the left side of = operator.

Types

- Simple assignment statement Ex : `int x; x = 40;`
- Multiple assignment statement Ex : `int x, y, z; x = y = z = 78;`
- Shorthand expression `+= , -= , *= , /= , %=`

Examples on shorthand expressions

- | | |
|---|--|
| 1. $a += 3 \text{ -----} \rightarrow a = a + 3$ | 4. $b * = (b + 4 * d) \text{ -----} \rightarrow b = b * (b + (4 * d))$ |
| 2. $z -= x * y \text{ -----} \rightarrow z = z - (x * y)$ | 5. $a \% = 2 \text{ -----} \rightarrow a = a \% 2$ |
| 3. $n /= 10 \text{ -----} \rightarrow n = n / 10$ | 6. $x += a / b * c \text{ -----} \rightarrow x = x + ((a / b) * c)$ |

e. Increment and Decrement operators

Used to add/subtract 1 from the same variable.

Ex : $a = a + 1 \text{ =====} \rightarrow a++ \text{ or } ++a$

Types

- Prefix expression
- Postfix expression

Prefix expression

- Consists of only ++ and --
- Increments / decrements value by 1 only
- Consists of one operand preceded by the operator
- ***Rule of prefix : change the value of the variable and then use it in the expression.***

Examples

1. int a = 4;	Output :
printf("a=%d\n",a);	a = 4
printf("++a=%d\n",++a);	++a = 5
printf("a=%d",a);	a = 5

2. `int a = 6, b = 4, c;` **Output :**
`c = ++a + ++b;` 7 5 12
`printf(“%d %d %d”, a, b, c);`

3. `int x = 6, y;` **Output :**
`y = 3 * x--;` 5 18
`printf(“%d %d\n”, x, y);` 19 19
`x = ++y;`
`printf(“%d %d\n”, x, y);`

4. `int x = 3, y=5, z;` **Output :**
`z = ++x - --y;` 4 4 0
`printf(“%d %d %d”, x, y, z);`

Postfix expression

- Consists of only ++ and --
- Increments / decrements value by 1 only
- Consists of one operand followed by the operator
- **Rule of postfix : Use the current value in the expression and then change it.**

Examples

1. `int a = 4;` **Output :**
`printf(“a=%d\n”, a);` a = 4
`printf(“a++=%d\n”, a++);` a++ = 4
`printf(“a=%d”, a);` a = 5

2. `int a = 6, b = 4, c;` **Output :**
`c = a++ + b++;` 7 5 10
`printf(“%d %d %d”, a, b, c);`

3. `int x = 6, y;` **Output :**

y = 3 * x++;	7 18
printf(“%d %d\n”, x, y);	7 19
y++;	
printf(“%d %d\n”, x, y);	

4. int x = 3, y=5, z;	Output :
z = x++ - y--;	4 4 -2
printf(“%d %d %d”, x, y, z);	

Evaluate the following expressions**1. $3 * 8 / 4 \% 4 * 5$**

$$(3 * 8) / 4 \% 4 * 5$$

$$(24 / 4) \% 4 * 5 = 6 \% 4 * 5$$

$$(6 \% 4) * 5$$

$$2 * 5 = 10$$

2. `int a = 3, b = 5, c = 8`

$$a += b * = c - = 5$$

$$a += b * = c = c - 5$$

$$a += b * = c = 8 - 5$$

$$a += b * = c = 3$$

$$c = 3$$

$$a += b * = 3$$

$$a += b = b * 3$$

$$a += b = 5 * 3$$

$$a += b = 15$$

$$b = 15$$

$$a += 15$$

$$a = a + 15$$

$$a = 3 + 15$$

$$a = 18$$

$$a = 18$$

3. `int a = 5;`

$$a += a++ + ++a;$$

$$a+ = (a++) + (++a)$$

$$a+ = (a++) + 6$$

$$a+ = 6 + 6 = 12$$

$$a = a + 12 = 7 + 12$$

$$a = 19$$

5. `int a = 5, b;`

$$b = a++ * a++; b = 5 * 5$$

$$b = 25 a = 7$$

4. `int a = 3, b;`

$$b = a++ + a++ + a++;$$

$$b = 3 + 3 + 3$$

$$b = 9 a = 6$$

6. i
n
t
a
=
5
,
b
;
b
=
-
-
a
-
a
—
;
b
=
4
-
(
a
-
-
)
b
=
4
-
4

b = 0 a = 3

; a - = a * (6)
a - = 6 * 6

a - = 36

a = a - 36
a = 6 - 36

a = -30

8. int a = 4, b;
b = a - ++a;
b = (a) - 5
b = 5 - 5

b = 0, a = 5

7. int a = 5;

a
-
=
a
*
+
+
a

f. Conditional operator / Ternary operator

Decision making statement.

Syntax :

Expression ? True Expression : False expression;

If the value of the expression is true, the True expression is executed, otherwise the False expression is executed, not both.

Examples :

int x , y =7, z = 10;

Working : Since 7 >= 10 is false, 100 is evaluated to x;

x = y>=z ? 70 : 100;

x = 100

x = 7 >= 10 ? 70 : 100

- 1. Write a program to input principal and time. If time is more than 8 years, assign rate of interest as 12% , else as 18%. Calculate simple interest and display.**

```
void main()
{ float P, SI;
  int T, R;

  printf("Enter principal and time\n");
  scanf("%f%d",&P, &T);

  R = T > 8 ? 12 : 18;

  SI = (P * T * R) / 100;

  printf("SI = %f", SI);
}
```

- 2. WAP to input marks, if marks is more than 80, assign grade as 'A' otherwise as 'B'**

```
void main()
{ int marks;
  char grade;

  printf("enter marks:");
  scanf("%d",&marks);

  grade = marks >= 80 ? 'A' : 'B';
  printf("grade = %c", grade);
}
```

}

g. Bitwise operator

Used to manipulate bits of the given data

Work only on integers

Types

- Bitwise negate (one's compliment) ~
- Bitwise AND &
- Bitwise OR |
- Bit left shift <<
- Bit right shift >>

Bitwise negate (one's compliment) ~

Interchanges 0s and 1s in the specified operand

Examples

1.

```
unsigned char a=12,b;  
b=~a;  
  
printf("a=%u b=%u",a,b);
```

Working

a = 12 Since a is of type unsigned char, it takes 1 byte (8 bits) of memory space.

Binary equivalent a = 0 0 0 0 1 1 0 0

~a = 1 1 1 1 0 0 1 1

b = ~a = 243

Output

a=12 b=243

2.

```
unsigned char a=20,b;  
b=~a;  
  
printf("a=%u b=%u",a,b);
```

Working

a = 20 Since a is of type unsigned char, it takes 1 byte (8 bits) of memory space.

Binary equivalent a = 0 0 0 1 0 1 0 0

~a = 1 1 1 0 1 0 1 1

b = ~a = 235

Output

a=12 b=235

Bitwise AND &

If the corresponding bit positions in both the operands are 1, result is 1 , else result is 0.

Example

```
int a=10, b=6, c;
```

```
c = a & b;
```

```
printf("%d",c);
```

working

```
a = 10 -----> 0 0 0 0 1 0 1 0
```

```
b = 6  -----> 0 0 0 0 0 1 1 0
```

```
a & b -----> 0 0 0 0 0 0 1 0 = 2
```

Hence 10 & 6 = 2

Output: 2

Bitwise OR |

If the corresponding bit positions in both the operands are 0, result is 0, else result is 1.

Example

```
int a=10, b=6, c;
```

```
c = a | b;
```

```
printf("%d",c);
```

working

```
a = 10-----> 0 0 0 0 1 0 1 0
```

```
b = 6-----> 0 0 0 0 0 1 1 0
```

```
a | b -----> 0 0 0 0 1 1 1 0 = 14
```

Hence $10 | 6 = 14$

Output: 14

Bitwise XOR ^

If the corresponding bit positions in both the operands have odd number of 1's, result is 1, else result is 0.

Example

```
int a=10, b=6, c;
```

```
c = a ^ b;
```

```
printf("%d",c);
```

working

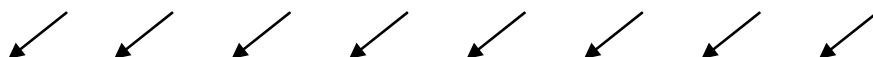
```
a = 10-----> 0 0 0 0 1 0 1 0
```

```
b = 6-----> 0 0 0 0 0 1 1 0
```

```
a ^ b -----> 0 0 0 0 1 1 0 0 = 12
```

Hence $10 ^ 6 = 12$

Output: 12

**Bit Left Shift <<**

Used to shift the data by a specified number of bit positions towards left.

Example : $5 << 1$ means 5 is shifted towards left by 1 bit position.

$7 << 2$ means 7 is shifted towards left by 2 bit positions.

$5 << 1$



5 =

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	1	0
---	---	---	---	---	---	---	---

$00001010 = 10$ Hence $5 << 1 = 10$

$12 \ll 2$

12 =

0	0	0	0	1	1	0	0
---	---	---	---	---	---	---	---

0	0	1	1	0	0	0	0
---	---	---	---	---	---	---	---

00110000 = 48 Hence $12 \ll 2 = 48$

Shortcut method to remember

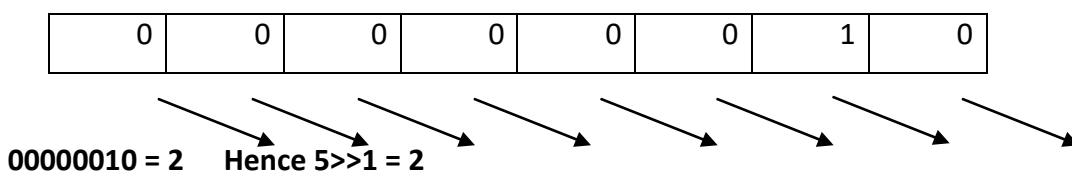
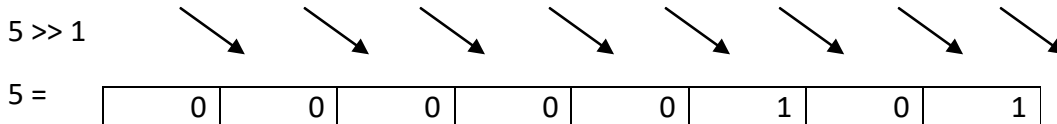
- $5 \ll 1$ -----> $5 * 1 * 2 = 10$ [multiply 5 and 1 and by 2 since we are working on binary bits]
- $4 \ll 2$ -----> $4 * 2 * 2 = 16$
- $12 \ll 2$ -----> $12 * 2 * 2 = 48$
- $3 \ll 3$ -----> $3 * 3 * 2 = 18$

Bit Right Shift >>

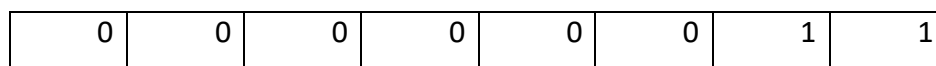
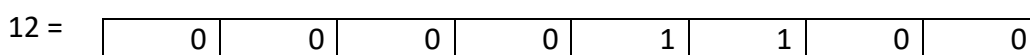
Used to shift the data by a specified number of bit positions towards right.

Example : $5 \gg 1$ means 5 is shifted towards right by 1 bit position.

$7 \gg 2$ means 7 is shifted towards right by 2 bit positions.



$12 \gg 2$



$00110011 = 3$ Hence $12 \gg 2 = 3$

Shortcut method to remember

- $5 \gg 1$ -----> $5 / (1 * 2) = 5/2 = 2$ [take integer value]
[Multiply 1 and 2 and divide 5 by the result, since we are working on binary bits]
- $4 \gg 1$ -----> $4 / (1 * 2) = 4/2 = 2$
- $4 \gg 2$ -----> $4 / (2 * 2) = 4/4 = 1$
- $12 \gg 2$ -----> $12 / (2 * 2) = 12 / 4 = 3$
- $3 \gg 3$ -----> $3 / (3 * 2) = 3/6 = 0$

h. Special operator**sizeof() operator**

It returns the number of bytes the datatype/variable/constant occupies in the RAM.

Ex : float x ;

int y = sizeof(x); ----- > y = 4

y = sizeof(double); ----- > y = 8

y = sizeof(800); ----- > y = 2

y = sizeof('T'); ----- > y = 1

Precedence of operators (from highest to lowest)

Operator	Name	Associativity
++, --	Postfix increment, decrement	L - R
++, --	Prefix increment, decrement	R - L
sizeof()	Size of the given datatype	-

~	One's compliment	R – L
!	Logical Not	R – L
+, -	Unary +, unary -	R – L
()	Type casting	R – L
*, /, %	Multiplication, division, modulus	L – R
+, -	Addition, subtraction	L – R
<<, >>	Bit shift left, bit shift right	L – R
<, <=, >, >=	Comparison	L – R
=, !=	Equal to, not equal to	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
=, shorthand expressions	Assignment	R – L
,	Comma operator	L - R

Question papers Solution

1. What is pseudocode? Explain with an example.

Pseudocode is a problem solving tool which contains statements written in English and C language. It is a series of steps which describes what must be done to solve a problem. It is used for developing a program and written before the program. More closer to the programmers.

Example:

Step 1: Start

Step 2: Declare variables num1, num2 and sum.

Step 3: Read values num1 and num2.

Step 4: Add num1 and num2 and assign the result to sum.

sum ← num1 + num2

Step 5: Display sum

Step 6: Stop

2. Explain the structure of C program with an example.

The list of components of a C program is as follows

- Comments
- The program header
- The body or the action part

Comments:

Comment lines in a program gives short description OR purpose of the program or any statement of the program.

Comment is not strictly necessary because a program without it is technically correct however if included it is considered as a good programming style.

Comments can be of 2 forms:

- **Single line comment:** it is used to write single line description and the symbol used is “//”
- **Multi line comment:** it is used to write descriptions in many lines and the symbol used are /* → comment begins, */ → comment ends.

The program Header:

The program header includes the following sections

- Pre-processor directives
- main program header

Pre-processor directives:

The pre-processor directives are the statements which start with symbol #. This statement instructs the compiler to include some of the files in the beginning of the program.

Ex: #include<stdio.h>

→ instruction to the compiler

include → to include files as a part of our program

<stdio.h> → <> pointed brackets tell the compiler, the exact location of this header file.
stdio :Standard input/ output

. h : header file.

main program header:

The main statement instructs the compiler that execution always starts from function main and it is called main program header. The pair of brackets denoted () must follow after the main to indicate it as a function. Every C program must have only one main.

The body or the action part:

Inside every C main program is a set of braces {} containing a series of C statements which comprise the body. This is called the action part of the program.

The body consists of two essential parts

Declaration section

Execution section

Declaration section:

The variables that are used inside the main function or sub function should be declared in the declaration section. The variables can be initialized during declaration.

Ex:

```
main()
{
int sum =0;           // initialization
inta,b;              // declaration
.....
}
```

Executable section:

This section includes the instructions given to the computer to perform some specific task. An instruction may represent an expression to be evaluated, input /output statements etc. Each executable statement ends with;

Ex:

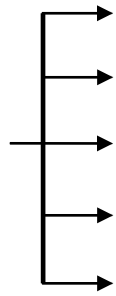
```
main()
{
.....
```

```
printf("enter two numbers");  
scanf("%d%d", &a, &b);  
  
sum = a+b;  
printf("sum = %d", sum);  
}
```

3. Explain any five operators used in C language.

1. Arithmetic operators

The operators that are used to perform arithmetic operations such as addition, subtraction, multiplication, division and modulus operations are called as arithmetic operators. These operators perform operations on two operands and hence they are binary operators.



Addition	+	add	4+2	6
Subtraction	-	subtract	4-2	4

Arithmetic operators	Multiplication *	multiply	4*2	8
	Division /	divide	4/2	2
	Modulus %	remainder	4%2	0

Note: Modulus operation denoted by % divides first operand by the second operand and returns the remainder. Remainder is the result obtained after modulus operation.

Modulus operation is allowed on integers and not allowed on other data type.

2. Unary operators:

An operator that operates on one operand to produce a result is called unary operator.

Eg: a++, a--, sizeof(), casting

Contains single operator, single operand

Ex : + : 4, +6

- : -10, -(6+10)

Example

int a=114, b= -3 ,c;

c = - (-a + b);

working : c = - (- (a) + b) = - (-114 + (-3))

c = - (-114 - 3) = - (-117)

c = 117

3. Conditional operator / Ternary operator

Decision making statement.

Syntax : Expression ? True Expression : False expression;

If the value of the expression is true, the True expression is executed, otherwise the False expression is

executed, not both.

Examples :

```
int x, y = 7, z = 10;
```

```
x = y >= z ? 70 : 100;
```

```
x = 7 >= 10 ? 70 : 100
```

Working : Since $7 \geq 10$ is false, 100 is evaluated to x;

x = 100

4. Assignment operators

It is used to assign the value of a constant or variable or value of an expression to a variable on the left side of = operator.

Types

- Simple assignment statement Ex : `int x; x = 40;`
- Multiple assignment statement Ex : `int x, y, z; x = y = z = 78;`
- Shorthand expression `+= , -= , *= , /= , %=`

Examples on shorthand expressions

1. $a += 3 \rightarrow a = a + 3$
2. $z -= x * y \rightarrow z = z - (x * y)$
3. $n /= 10 \rightarrow n = n / 10$
4. $b * = (b + 4 * d) \rightarrow b = b * (b + (4 * d))$
5. $a \% = 2 \rightarrow a = a \% 2$
6. $x + = a / b * c \rightarrow x = x + ((a / b) * c)$

5. Relational operators

The operators that are used to find the relationship between two operands are called relational operators.

The relationship between two operand values results in true or false

The relational operators are

Relational Operators	Description	operator	priority	Associativity
	Less than	<	1	left to right
	Less than or equal to	<=	1	left to right
	Greater	>	1	left to right
	Greater than or equal to	>=	1	left to right
	Equal	==	2	left to right
	Not equal	!=	2	left to right

Note: you can explain any of the operators of your choice.

4. What is type conversion? Explain two types of type conversions with examples

In C language, the programmer can instruct the compiler to convert the data from one data type to another data type. Sometimes, the compiler itself will convert the data from one type to another data type. This process of converting the data from one type to another data type is called type conversion.

Type conversion	Implicit conversion	1 / 2.0	1.0 / 2.0	0.500000
	Explicit conversion	1 / (float)2	1.0 / 2.0	0.500000

C can evaluate expressions if and only if the data types of two operands are same. So, if operands are of different data type, it is not possible to evaluate the expression. In such situation, to ensure that both the

operands are same the compiler converts the data from lower rank to higher rank. This process of conversion of data from lower rank to higher rank automatically by the C compiler is called **implicit**

conversion.

Example:

```
int + int = int
5   + 3   = 8
```

8

(int)

int + float = float

$$5 + 3.5 = 8.5$$
$$5.0 + 3.5$$

8.5

(float)

int + double = double

$$5 + 2.112 = 7.112$$
$$5.0 + 2.112$$

7.112

(double)

to d

Explicit conversion/ Type casting:

If the operands are of the same data type, no conversion takes place by the compiler. Sometimes, the type conversion is required to get the desired results. In such case, the programmer can instruct the compiler to change the type of the operand from one data type to another data type. This forcible conversion from one data type to another data type is called explicit type conversion.

Syntax:

(type)expression

Examples:

`i = (int) 8.9999` 8.9999 is truncated to 8 and is stored in i so the value of **i = 8**

`i = (int)5.99 / (int) 2.4` 5.99 is truncated to 5 and 2.4 is truncated to 2 which results in $5 / 2 = 2$.
Hence **i= 2**

`i= (float) 1 / 2` 1 i converted to 1.0. Next implicit conversion is done on 2. Thus we get $1.0 / 2.0 = 0.5$.
Hence **i = 0.500000**

5. Write a program in C to find the area and perimeter of a rectangle.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    float length,breadth,area,perimeter;
    clrscr();

    printf("\nEnter the length of the rectangle:");
    scanf("%f",&length);

    printf("\nEnter the breadth of the rectangle:");
    scanf("%f",&breadth);

    area = length * breadth;           //Calculating the area using the formula.
    perimeter = 2*(length + breadth);  //Calculating the perimeter using the formula.
    printf("Area of the rectangle is %f", area);

    printf("Perimeter of the rectangle is %f", perimeter);
    getch();
}
```

Output:

```
Enter the length of the rectangle: 10
Enter the breadth of the rectangle: 20
Area of the rectangle is 200.000000
Perimeter of the rectangle is 60. 000000
```

6. Define: i) Variable ii) Constant iii) Associativity iv) precedence.**Variable**

- An identifier which can store a value.

- The value may change later in the program.
- Refers to a specific memory location where the data can be stored.

Rules for naming a variable

- Can consist of alphabets, digits and _ (underscore).
- Must always begin with an alphabet or _ and can be followed by digits.
- No special symbols allowed.
- Case sensitive.
- Cannot be a keyword.

Declaring a Variable

Every variable has to be declared before its first use.

Syntax:

```
Data_type variable name;
```

Example:

```
Inta;
```

```
float num1, num2;
```

```
char grade, name[21];
```

Initializing a Variable

Here, the value can be assigned to a variable by the programmer.

For every execution, the value of the variable may be same.

Example:

```
Inta =40;
```

```
float num1, num2;
```

```
num1=67;
```

```
num2=6.8;
```

Constant

A fixed value assigned to an identifier whose value cannot be changed.

Types of constants

Integer constant

Floating point constant

Character constant

String constant

Backslash constants (escape sequence)

Integer constant

It represents a whole number without decimal part.

Subtypes

Decimal

Octal

Hexadecimal

Decimal

Constants having digit combination between 0 – 9.

Can be + or –

Valid ex: -45 756 1234

Invalid ex : 102 56 4.50 -+56

Octal

Constants having digit combination between 0 – 7.

Can be + or –

The constant must have the prefix 0 (Zero).

Valid ex: 077 -0756 01234

Invalid ex : 0102 56 -017.7 856

Hexadecimal

Constants having digit combination between 0 – 9 and A – F.

A represents 10, B represents 11, C represents 12, D represents 13, E represents 14,

F represents 15

Can be + or –

The constant must have the prefix 0X / 0x.

Valid ex: 0X917 -0x1AF 0x304a

Invalid ex : 18C 0X1GA 0x4.6E

Float constant

Represents a number having fractional part.

May be + or –

Examples

-221.45	44567.34
0.000009	67.89E2---> 67.89 * 10 ²
67.89e-3 ----> 67.8 * 10 ⁻³	12.0

Character constant

Represents a single character within a pair of ‘ ‘

Stored as integers in computer’s memory.

Examples

```
char op = '+'; char choice = 'Y';
char num = '7'; char ch = 'n';
```

String constant

Represents a sequence of characters within a pair of “ “

Examples

"Good Morning"

"10 + 40"

"G.A.T"

"#163/13, Bangalore"

Backslash constants (Escape sequence)

Non printing characters can be printed using escape sequence.

It always starts with \ followed by a character.

Escape sequence	Description
\a	System alarm
\b	Backspace
\n	Newline

<code>\t</code>	Horizontal tab
<code>\"</code>	Double quote
<code>'</code>	Single quote
<code>\\</code>	<code>\</code> (backslash)
<code>\0</code>	Null character used to terminate a string

Associativity: Associativity determines how operators with the same precedence are evaluated in an expression. It can be classified into two types

Left associative : expression is evaluated from left to right

eg: arithmetic operators

Right associative : expression is evaluated from right to left

eg: assignment operators

Precedence:

In C, 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.

Eg: if an expression has arithmetic and relational operators then arithmetic operators are evaluated first and then the relational operators.

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

Defines the type of data that is stored in a variable.

Determines how much storage should be allocated to a variable.

Bool

It is used to store the value wither true or false. True is represented as 1 and false as 0.

Ex ample:

```
bool a;
```

```
a=true;
```

char

Each character constant is stored internally in the computer as integer constant. This integer constant is called as ASCII (American Standard Code for Information Interchange)

Example :

```
void main()

{   charvar1 ='a';
char var2 = 'E'

char var3 = '7', var4 = '*';
}
```

int

An integer datatype is used along with a variable to store a whole number.

signedint : +ve or –ve values

unsignedint : only +ve values

Example :

```
void main()
{
    int a = 10;
    int b, sum;
    b=60;

    sum=a+b;
}
```

float

A float datatype is used along with a variable to store a real number.

Example :

```
void main()
{
    float a, b, c;
    a=7.1;

    b=77.89;
    c=a-b;
}
```

double

A double datatype is used along with a variable to store a real number having a range higher than floating point number.

Example :

```
void main()
{
    double a, b, c;
    a=78907.1976;
    b=751237.89;

    c=a*b;
}
```

void

Stores no values, non-specific data type.

8. Write a C program which takes as input p, t, r, compute the simple interest and display the result.

```
#include<stdio.h>
void main()
{
float P, T, R, SI;

printf("Enter P,T,R\n");

scanf("%f %f %f", &P, &T, &R);
SI = (P*T*R)/100;

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

9. What is an operator? List and explain various types of operators

Operator: An **operator** is a symbol that tells the compiler to perform specific mathematical or logical functions using the operands.

Operator classification:

The operators in C language are classified based on:

- The number of operands
- The type of operation

Classification based on the number of operands:

The operators are classified into

Unary operators:

An operator that operates on one operand to produce a result is called unary operator.

Eg: a++, a--, sizeof(), casting

Contains single operator, single operand

Ex : + : 4, +6

- : -10, -(6+10)

Binary operators:

An operator that operates on two operands in an expression to produce a result is called a binary operator.

- Eg: a+b, 10/5, +, -, *, /, %

Ternary operators:

An operator that operates on three operands to produce a result is called ternary operator. Also called as the conditional operators.

Eg: a?b:c

Examples :

int x, y = 7, z = 10;

x = y >= z ? 70 : 100;

x = 7 >= 10 ? 70 : 100

Working : Since 7 >= 10 is false, 100 is evaluated to x;

x = 100

Special operators:

Ex: comma(,) sizeof() etc.

Classification based on the type of operation:

- **Arithmetic operators** + , - , * , / , %

An arithmetic operator is a mathematical function that takes two operands and performs a calculation on them

- **Relational operators** < , <= , > , >= , == , !=

- Used to compare the values of two expressions.
- If the result is true, return 1. If the result is false, returns 0.

- **Logical operators** ! , && , ||

- Used to combine two expressions. The result will be either true or false.

- **Assignment operators** =, +=, -=, *=, /=, %=

It is used to assign the value of a constant or variable or value of an expression to a variable on the left side of = operator.

Types

- Simple assignment statement Ex : int x; x = 40;
- Multiple assignment statement Ex : int x, y, z; x = y = z = 78;
- Shorthand expression += , -= , *= , /= , %=

- **Increment and decrement operators** ++ , --

Used to add/subtract 1 from the same variable.

Ex : a = a + 1 =====> a++ or ++a

Types

- Prefix expression
- Postfix expression

- **Conditional operators**

Decision making statement.

Syntax :

Expression ? True Expression : False expression;

If the value of the expression is true, the True expression is executed, otherwise the False expression is executed, not both.

- **Bitwise operators** ~ , << , >> , & , | , ^

- Used to manipulate bits of the given data
- Work only on integers

- **Special operators**

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

A token is a smallest or basic unit of a C program. One or more characters are grouped in sequence to form meaningful words. These meaningful words are called tokens.

Examples

$c = a + b * 10$; Here, c , $=$, a , $+$, b , $*$, 10 each one is called as a token.

Result = $(x * y) * (x - y)$;

The tokens in C language are broadly classified as shown below:

Types of Tokens:

Keywords

Identifiers

Constants

Operators

Keywords (Reserved words)

Reserved words which have a pre-defined meaning in C compiler.

Used for specific task in C language.

The keywords in C are in lowercase.

Some keywords

int	float	if	else
for	while	switch	break
long	void	char	double
continue	do	default	short
delete	static	const	unsigned

Identifiers

Names provided to the elements of the program such as variables, functions and arrays.

Rules for naming an identifier

Can consist of alphabets, digits and _ (underscore).

Must always begin with an alphabet or _ and can be followed by digits.

No special symbols allowed.

Case sensitive.

Cannot be a keyword.

Valid examples of Identifiers

Total	sum1
Voter_ID	roll_no

Constants

A fixed value assigned to an identifier whose value cannot be changed.

Types of constants

- | | |
|--|----------------------------|
| 1. Integer constant | Explain all of them |
| 2. Floating point constant | |
| 3. Character constant | |
| 4. String constant | |
| 5. Backslash constants (escape sequence) | |

Operator: An **operator** is a symbol that tells the compiler to perform specific mathematical or logical functions using the operands.

Operator classification:

The operators in C language are classified based on:

- The number of operands
- The type of operation

Explain all of them in brief

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

Variable

- An identifier which can store a value.
- The value may change later in the program.
- Refers to a specific memory location where the data can be stored.

}

}

Rules for naming a variable

- Can consist of alphabets, digits and _ (underscore).
- Must always begin with an alphabet or _ and can be followed by digits.
- No special symbols allowed.
- Case sensitive.
- Cannot be a keyword.

Examples for valid and invalid variable

Valid variable – Total, sum1, Voter ID, roll no

Invalid variable –for, 1stname, sum-of-digits, roll number, #USN NO, I Class

12. Explain scanf() and printf() functions in C language with syntax and example.

scanf()

Data read by a program can come from two places: the keyboard or a data file. In C programming language the data is read from the keyboard using formatted input statement ***scanf()***.

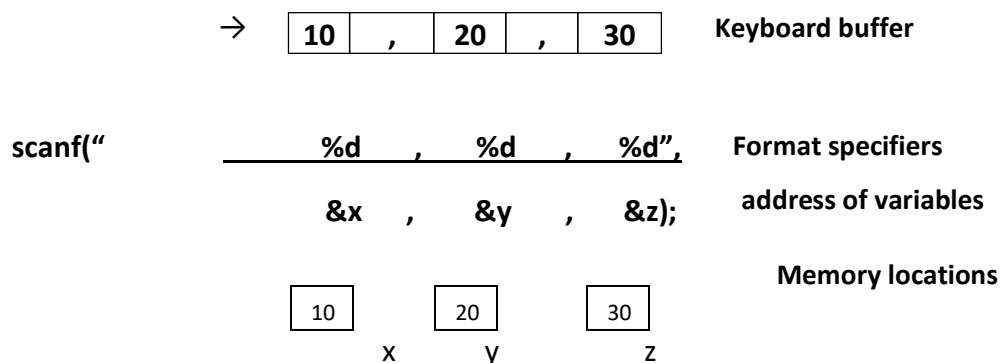
Syntax:

```
scanf("format string", address_list);  
address_list----->&variablename
```

Explain the instruction for the input data:

10, 20, 30

```
scanf ("%d, %d, %d", &x, &y, &z);
```



printf()

The statement which instructs the compiler to display or print the string, expression value or variable value

to the output device is called print statement. The printed results of running program are called output. In C programming language the output is printed using formatted output statement **printf()**.

There are two forms of **printf()**

1. That has a literal string – a sequence of characters within quotation marks.

Syntax:

```
printf("literal string");
```

Example:

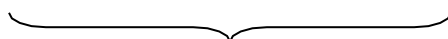
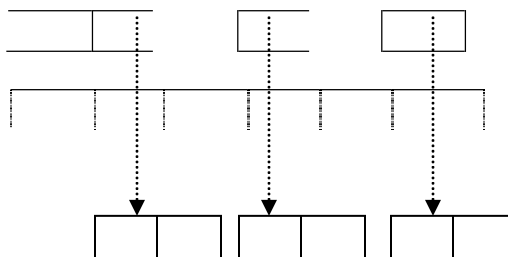
```
printf("welcome to C programming\n");
```

2. That has literal string, conversion specifier and any or all of the following: variables, constants and expressions values to be printed.

Syntax:

```
printf("Literal string/ conversion specifier" , list of variables);
```

"Format string"



Example:

```
printf("%d %d", number, square);  
printf(" the sum = %d", sum);
```

13. Write a 'c' program to find area of a circle.

```
#include <stdio.h>  
#include <conio.h>  
#include <math.h>  
#define PI 3.142  
void main()  
{  
clrscr();  
  
printf("Enter the radius of a circle\n");  
scanf ("%f", &radius);  
  
area = PI * pow (radius,2);  
printf ("Area of a circle = %5.2f\n", area);  
}
```

Output:

```
Enter the radius of a circle  
3.2  
Area of a circle = 22.17
```

14. What is an algorithm? Write an algorithm to find the largest of 3 numbers.

An algorithm is a finite sequence of instructions, a logic and explicit step-by-step procedure for solving a problem starting from a known beginning. – The number of instructions must be finite

Start

1. Input a, b, c
2. big = a
3. if b > big
big = b
4. if c > big
big = c
5. Print big

Stop

15. Explain the following operators in c language

- i. Relational ii. Logical iii. Conditional

i. Relational operators

The operators that are used to find the relationship between two operands are called relational operators. The relationship between two operand values results in true or false

The relational operators are

	Description	operator	priority	Associativity
Relational Operators	Less than	<	1	left to right
	Less than or equal to	<=	1	left to right
	Greater	>	1	left to right
	Greater than or equal to	>=	1	left to right
	Equal	==	2	left to right
	Not equal	!=	2	left to right
Equality operators				

Examples:
 $a + b > c$

Valid

 $a < = b$

Invalid – space between

< and = not allowed

ii. Logical operators

Used to combine two expressions. The result will be either true or false.

Types

- Logical NOT ! 1
- Logical AND && 2
- Logical OR || 3

Logical NOT!

It reverses the logical value of a variable or a constant or an expression.

Examples
 $!(3 > 4) \text{ ----> } !(false) = true = 1$

Logical AND &&

- Works on 2 operands.
- Result is true if values of both expressions are true.
- Result is false, if any one expression value is false.

Example

`78 > 3 && 4 <=3;`

`(78>3) && (4<=3) = 1 && 0 =0`

Logical OR | |

- Works on 2 operands.
- Result is false if values of both expressions are false.
- Result is true, if any one expression value is true.

Example

```
78 > 3 || 4 <= 3;
```

```
(78 > 3) || (4 <= 3) = 1 || 0 = 1
```

iii. Conditional operator / Ternary operator

Decision making statement.

Syntax:

Expression ? True Expression : False expression;

If the value of the expression is true, the True expression is executed, otherwise the False expression is executed, not both.

Examples:

```
int x, y = 7, z = 10;
```

```
x = y >= z ? 70 : 100;
```

```
x = 7 >= 10 ? 70 : 100
```

Working : Since 7 >= 10 is false, 100 is evaluated to x;

```
x = 100
```

16. What is an identifier? Write the rules to be followed while writing an identifier

Names provided to the elements of the program such as variables, functions and arrays.

Rules for naming an identifier

Can consist of alphabets, digits and _ (underscore).

Must always begin with an alphabet or _ and can be followed by digits.

No special symbols allowed.

Case sensitive.

Cannot be a keyword.

Valid examples of Identifiers

Total

sum1

Voter_ID

roll_no

Invalid Identifiers

for - keyword

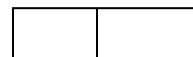
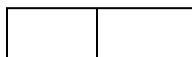
1stname - beginning with a digit

sum-of-digits - illegal use of special symbol -

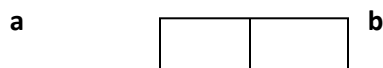
17. What is the value of 'x' in the following code segment? Justify your answer.

1. int a, b;
float x;
a=4;
b=5;
x=b/a;

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



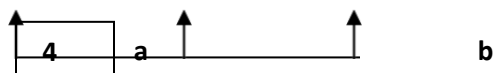
int a, b;



float x;



a=4;



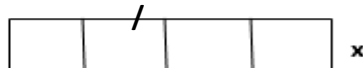
b=5;



int a, b;
x= b/a



float x;



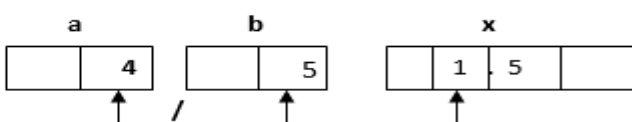
Working: b=5 and a=4

Divide $b/a = 5 / 4 = 1.25$

Truncated as 1 since both the data types are int

X= 1 but x is a floating value

Hence x= **1.000000**



Working: b=5 and a=4

(float) b/a

(float) 5/4

= 5.0/ 4 (explicit)

= 5.0/ 4.0 (implicit)

= 1.500000

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

19. Convert the mathematical expression into C expression

--	--

--	--

20. Evaluate the following expressions

1. $100\%20 \leq 20 - 5 + 100 \% 10 - 20 == 5 > 1 != 20$

0 $\leq 20 - 5 + 100 \% 10 - 20 == 5 > 1 != 20$

0 $\leq 20 - 5 + 0 - 20 == 5 > 1 != 20$

0 $\leq 15 + 0 - 20 == 5 > 1 != 20$

0 $\leq 15 - 20 == 5 > 1 != 20$

0 $\leq -5 == 5 > 1 != 20$

0 $== 5 > 1 != 20$

0 $== 1 != 20$

0 $!= 20$

1

2. $a += b * c -= 5$ where $a = 3$, $b = 5$ and $c = 8$

$a += b * c -= 5$

The given expression is:

$a += b * c -= 5$

$c = c - 5$

$$c = 8 - 5$$

$$\mathbf{c = 3}$$

$$a += b * 3$$

$$b = b * 3$$

$$b = 5 * 3$$

$$\mathbf{b = 15}$$

$$a = a + 15$$

$$a = 3 + 15$$

$$\mathbf{a = 18}$$

So a= 18, b= 15 and c = 3
