CPUC

Assignment-1

A> Define the following terms

ⓐ Algorithm :-

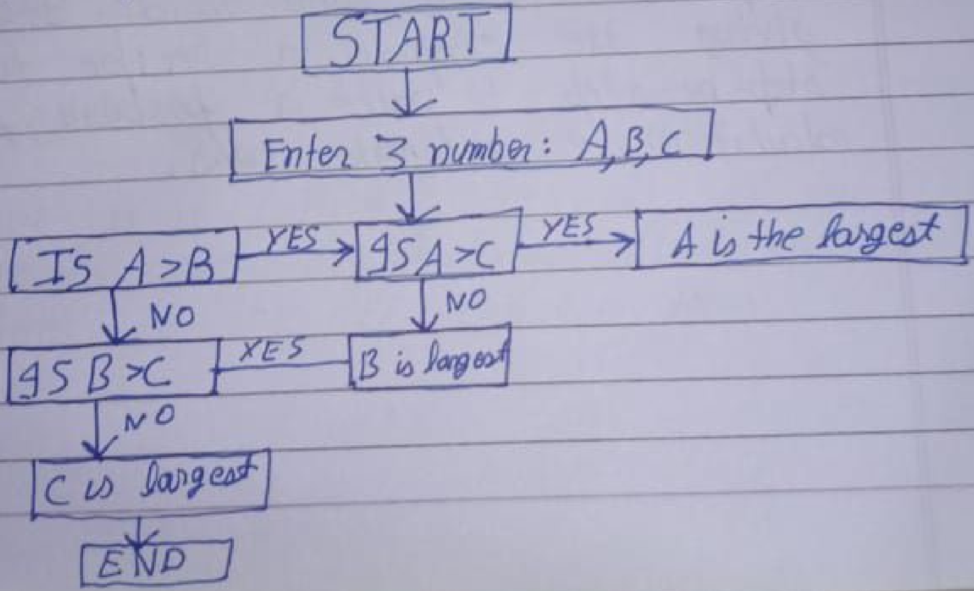
An algorithm is like a recibe, a step-by-step guide to solve a problem or do a task. It tells you exctly what to do, one instruction at a time, so you get the right result

e.g:- Check 1st Number → Compare to 2nd → Compare to 3rd → Largest No.

ⓑ Flow Chart :-

A flowchart is a visual diagram that uses shapes and arrows to show the step-by-step process of an algorithm or workflow. It helps simplify compler ideas into easy-to-follow steps

e.g:-



(c) Pseudo Code :-

It is a fake way to write code without real programming syntax. It uses plain English + basic logic to plan algorithms before actual coding.

e.g:-

```
BEGIN
  INPUT NUMBER
  IF number MOD 2 == 0 Then
  PRINT "even"
  ELSE
   PRINT "Odd"
   END IF
   END
```

(d) Program :-

It is a set of instructions written in a programming language that tells a computer exactly what to do. It's like giving the computer a recipe to follow step-by-step - to solve a problem, perform calculations, or automate tasks.

(e) Compiler

It is a special program that translates human-written code into machine code that a computer can directly execute.

(f) Interpreter:-

It is a program that directly executes human-written code line-by-line, without compiling it into machine code first, it reads, translates, and runs instructions on the fly.

(g) Links:-

It is a clickable reference in digital content that connects one piece of info. to another. It allows users to navigate between:

- Webpages [https://google.com/]
- Files (document, images, videos].

(h) Loder :-

It is a critical component of an OS that loads executable programs from storage into RAM so the CPU can run them.

(i) Source-Code:-

It is the human-readable version of a program, written by developers using a programming language. It contains the logic, instructions, and algorithms that define how the software should work.

(j) Object Code:-

It is a machine-readable output produced by a compilier or assembler after translating source code. It is a low level representation of the program, typically in binary or hexadecimal and is stored in files like .o[Linux] or .obj [windows].

(k) Type Casting:-

It is the process of converting a variable from one data type to another. It can be done implicity or explicity [automatically] or [By Programmer]

(l) Identifier:-

It is a user-defined name given to variables, functions, classes or other programming elements to uniquely identify them in code.

(M) Keywords :-

These are special words ~~that~~ predefined in a programming language that have fixed meaning and functionalities. They cannot be used as Identifier [They are part of the language's syntax].

(N) Variables :-

It is a named storage location in memory that holds which can be changed during program execution. It acts like a labeled container for values.

(O) Constant :-

It is a named value that cannot be changed during program execution. It holds fixed data like numbers, strings, or expressions and ensures code safety by preventing accidental modifications.

(1) Pre-Processor directive :-

~~It is~~ A preprocessor directive is a special command [Starts with #] that instructs the compilier to modify or prepare the code before actual compilation begins. It operates during the pre-processing phase [Before code is turned into machine language].

(2) Computer Language:-

It is a formal system of syntax and rules used to instruct a computer to perform specific tasks. It acts as a bridge between human logic and machine-executable commands.

(3) Bugs :-

A Bug is an error, flaw or unintended behavior in a program that causes it to produce incorrect results, crash, or behave unexpectedly. Bugs arise from mistakes in code, logic or system design.

{Explain} [Big]

Q1:- What do you mean by algorithm? Examples.

Ans:- Algorithm:-

It is a step-by-step method of solving a problem. It is a group of instructions written in a proper order so that if we follow them, we will always get the correct result. Just like a recipe tells us how to cook food step-by-step, An algorithm tells us how to solve a problem or complete a task in a clear way.

Example-1) In Daily Life:- Making tea by following steps:-

Boil Water → Add tea leaves → add sugar and milk → filter → serve.

2 → In Maths:- To add 2 numbers:-

Take 2 numbers → add them → Write the answer.

3 → In Computer:-

Arraging members from small to big using clear rules.

Q2:- What do you mean by flowchart? Explain its types.

Ans:- Flow Chart:-

A flowchart is a simple diagram that shows the steps of a process or problem in the form of different shapes connected by arrows. It is like a picture that explains how something works step by step. Each shape has a meaning: for example, an oval shows the start or end, a rectangle shows a process or action, a diamond shows a decision, and arrows show the direction of flow.

• Types of flowchart

1 → Process Flowchart:- Shows the steps of a process in order.

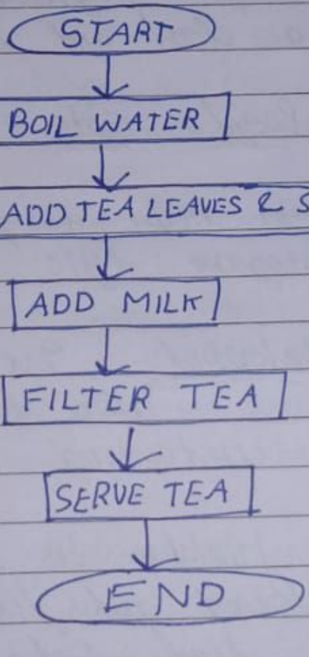
example:- Steps for making tea.

2 → System Flowchart:- Explains how data or information flows in a system, including input, processing, storage, and output.

3→ Data flowchart :- Focuses only on how data moves from one place to another in a system.

4→ Program FlowChart :- Used in computer programming to show the logical steps of a program, like loops, decisions, and calculations.

☆→ Diagram :-



◊ = Process ,

Q3 Why the PseudoCode is neccessary?
Uses of Pseudo Code.

Ans:- It is a way of writing the steps of a program in simple, plain language before writing it in an actual programming language. It does not follow strict rules of any coding language, but it looks a little like code. It is necessary because it helps programmers plan their program in a clear and simple way.

• Uses of Pseudo Code:-

1→ Planning:- It helps to plan the program logic step by step.

2→ Easy to Understand:- Even people who don't know coding can understand Pseudo Code.

3→ Debugging:- Mistakes in logic can be found and fixed easily before writing the final Code.

Q4 Short note on programming language evolution.

Ans Programming language did not appear suddenly, they developed step-by-step to make computers easier for humans to use. At the beginning, computers only understood machine language, which was very difficult for humans. Then, slowly new languages were created to make programming simpler and more powerful.

1st Gen:- Written in binary numbers (0 and 1).
. Very hard to understand and error-prone.

2nd Gen:- Used short codes or symbols [ADD, SUB].

3rd Gen:- Closer to human language, easier to write and understand.

4th Gen:- Reduced complexity & increase productivity.
. Examples: Java, Python, C++, PHP.

5th Gen:- Focus on problem solving and artificial intelligence
. examples:- AI based languages like Prolog, LISP etc.

Q5 How many types of error occur in the program.

Ans:- When writing a program, different types of errors (mistakes) can occur. These can stop a program from working correctly. There are 3 major types of errors that usally occur.

- Types:-

1→ Syntax Errors:-

- These happen when the rules of the programming language are not followed.
- example:- forgetting a semicolon ";", missing brackets "()" or using the wrong keyword.

2→ Logical errors:-

- The program runs, but the output is wrong because the logic of the program is incorrect.
- example:- Using "+" insted of "-" while calculating.

3→ Runtime errors:-

- These occur while a programe is running.
- example:- Dividing a no by 0, opening a file that dosent exist.

Q6 Explain Bugs / errors.

Ans:- In computer programming, a bug/error is a mistake in the program that cause it to work incorrectly or stop running. Bugs are very common because humans make mistakes while writing code. The main types are syntax, logical, runtime and sematic errors. Debugging is the process of fixing them.

Q7 explain the structure of C Program.

Ans:- A C program is written in a proper order so that the computer can understand and execute it correctly. The structure is made up of different sections:

1→ Documentation Section

- This part contain comments that explain what the program does
- It does not affect the program.
- example:- "// This program adds 2 numbers".

2→ Link Section

- It includes the header files that contains built-in functions
- example:- #include <stdio.h> [I/O function].

3→ Global Declaration Section

- Variables or functions that are declared here can be used anywhere in the program.
- example:- int a, b; //Global variables.

4→ main() Function Section

- Every C program must have a main() function because execution starts from here.
- Inside it, we write the main logic of the program.
- example:-
```
int main() {
        // Program logic
        return 0;
}
```

5→ Subprogram Section

- If the program is big, we can create extra functions for better understanding
- example:-
```
int add (int x, int y ) {
        return x +y ;
}
```

Q8 What is C language? features & characteristics.

Ans C language :-

The C language is often called the "mother of programming language" because many modern languages like C++, Java, and Python are based on its concept. It is a general-purpose, high level programming language developed by Dennis Ritchie at Bell labs in 1972. It is one of the most popular & powerful languages because it is simple, fast, and close to the machine.

• Features / Characteristics

1→ Simple & Easy to learn

• C has a small number of keywords and a clean structure, so beginners can learn it easily.

2→ Portable

• A program written in C can run on diff computers with little or no-change.

3→ Extensible

• New features and functions can be added easily to C programs.

4→ <u>Rich library of functions</u>

- C provides many built-in functions and also allows programmers to create their own functions.

5→ <u>Fast and efficient</u>

- Since C is close to a machine language, programs run very quickly and use less memory.

<u>Q9</u> Explain all the Characteristics used in C language.

Ans :- 1:- <u>Simple Language</u>:-

- C has a small set of keywords [only 32] and a clean structure.
- Programs are easy to write, read and understand.

2:- <u>Portable</u>:-

- C programs can run on different machines with little or no change.
- This is why C is called a machine-independent language.

3 → <u>Fast and efficient</u>

- C programs execute very quickly because C is close to machine ~~learning~~ language.
- It uses less memory, which makes it efficient for system programming.

4 → <u>Structured Programming Language</u>

- In C programs can be divided into small parts called functions.
- This makes debugging, testing, and reusing code easier.

5 → <u>Rich Library support</u>

- C provides many built-in functions for I/O, math, string handling, etc.
- example: printf() for output, scanf() for input.

6 → <u>Middle-Level Language</u>

- C combines the features of low-level language and high-level language.
- This makes it powerful for both system software and application software.

7→ Extensible

- New features and user-defined functions can be easily added to programs without affecting existing code.

8→ Memory Management

- C provides pointers, which allow direct access to memory.
- This gives programmers more control over memory usage.

9→ Case-Sensitive

- In C, uppercase and lowercase letters are treated differently.
- example : SUM and Sum are 2 diff var.

10→ Modularity

- A large program can be divided into multiple files and functions, making it easier to handle complex projects.

Q10 What are Identifiers?

Ans:- In C language, an identifier is the name given to different program elements such as variables, functions, arrays, classes etc.

- These are used so that we can refer to data or functions easily in a program. Without identifiers, we would not be able to store or reuse values.

example -

```
int age; // "age" is an identifier
float marks; // "marks" is an identifier.
```

- int and float are data types, but age and marks are Identifiers.

Q11 Define a variable.

Ans A variable in C language is a named storage location in memory that is used to store data. The value of a variable can change during the execution of a program, that's why it is called a variable.

- It works like a container or a box in which we can store value, and the name of that box is the identifier.

example

```
int age = 18; // 18 is the stored value
float marks = 75.5; //75.5 is the stored value.
```

Q12 Write down the rules which are used to declare the variables. [int, char, float, string].

Ans:- • General Rules for declaring variables are:-

1→ A variable name must start with a letter or underscore.
2→ It can only contain letters, digits, and underscores.
3→ No space or special characters are allowed.
4→ Variable names are case-sensitive [Age & age are diff].
5→ A variable name must not be a keyword [int, while].
6→ A variable must be declared before use in the program.

• Declaring Var with different Data Types:-

1→ Integer [int] → used for whole numbers.

ex:-
```
int age = 18;
int count, no1, no2;
```

2→ Character [char] → Used for storing single characters

ex:-
```
char grade = 'A';
char symbol = '$';
```

3→ Floating-point [float]→ Used for decimal numbers.

ex-
```
float marks = 75.5;
Float pi = 3.14;
```

4→ String → In C, there is no direct string type. Strings are stored using character arrays.

ex-
```
Char name[20] = "Aayan";
```

Q13 What is Pre-processor directive ?

Ans:- A pre-processor directive in C is an instruction given to the compiler before the actual compilation of the program begins. These directives tell the compiler to perform specific tasks such as including header files, defining constants, or conditionally compiling parts of the code. They always start with the # symbol.

• Types of Preprocessor directives :-

1→ Macro Definition [#define]

• Used to define Constants or macros.

example
```
#define PI 3.14
```

2→ File Inclusion [#include]

- Used to include header files.
- example :- #include <stdio.h> // standard header file
  # include "myfile.h> // User-defined file

3→ Conditional Compilation [#if, #else, #endif, #elif]

- Used to compile certain parts of code conditionally.

· example

```
# if DEBUG
        printf ("Debug mode on \n");
# else
        print f ("Debug mode off \n");
# end if
```

4→ Other directives

- #Undef → Undefines a macro.
- #pragma → special commands to the compiler.

Q14 What are Constants ?

Ans:- A constant is a value that cannot be changed during the execution of a program. Once a constant is defined, its value remains fixed. Constants make programs more readable, reliable, and error-free because they prevent accidental changes.

Types of Constants :

1→ Integer Constants

- Whole numbers without decimals.
- Can be +ive or -ive.

ex:- 10, -25, 0

2→ Floating - Point Constants

- Numbers with decimal points or in exponential form.

ex:- 3.14, -2.5, 1.2e3

3→ Character Constants

- A single character enclosed in single quotes

Ex:- Example: 'A', '9', '#'

4→ String Constants

- A sequence of characters enclosed in double quotes.

ex:- "Hello", "Hi".

5→ Symbolic Constants [#define]

- Constants defined using pre-processor directive.

ex:- #define PI 3.14

6→ Constant variables [using const keyword]

- Variables declared with const keyword cannot be modified.

ex
```
const int age = 18;
const float pi = 3.14159;
```

Q15 What do you mean by data types? explain in brief.

Ans. A data type in C defines the type of data that a variable can store. It tells the compiler what kind of value is stored and how much memory is needed.

- Without data types, the compilier will not know how to handle the data in a program.

- Types of data types in C

1→ Basic Data Types

- These are the fundamental data types used to store simple values.

ex:- Int → For whole number
Float → stores decimal no.
Char → Stores single character.
double → large decimal numbers with more precision.

2→ Derived Data Types

- Formed from basic data types.

ex- Arrays, functions, Pointers.

example.

```
# include <stdio.h>
int main() {
    auto int x = 10;  // auto keyword
    printf("auto variable x = %d ", x);
    return 0;
}
```

2 → Register Storage Class (register)

- Stores variable in CPU registers instead of RAM (if possible).
- Access become faster, but storage size is limited.
- Default value : Garbage.

example.

```
#include <stdio.h>
  int main() {
    register  int i; // stored in CPU register
    for (i=1; i<=5; i++) {
    printf("%d", i);
  }
    return 0;
}
```

3 → Static Storage Class [static]

- Retains its value throughout the program execution.
- Scope is local to the function, but value is preserved between function calls.
- Default value: ~~[illegible]~~ 0.

ex:

```
# include < stdio.h >
    void display () {
        static int count = 0; // retains value
        count++;
        printf ("count = %d\n", count);
    }
    int main () {
        display ();
        display ();
        display (); // value keeps increasing
        return 0;
    }
```

4 → External Storage Class [extern]

- Used to declare global variables that can be used across multiple files / functions.
- Lifetime: Entire program execution.
- Scope:- Global
- Default value: 0

Example:-
```
#include <stdio.h>
 int num = 10;   //global variable

  void display () {
      extern  int  num ;  // global variable
      printf ("Num = %d\n", num);
  }
    int  main () {
          display();
        return 0;
    }
```

☆→ Summary table of Storage Classes

	Class	Scope	lifetime	Default Value	Keyword
1	auto	Local	within Block	Garbage	auto
2	register	Local	within Block	Garbage	register
3	static	Local / Global	Entire program	0	static
4	extern	Global	Entire Program	0	extern

[illegible]
28/8/25