Problem Solving in Everyday Life

Problem solving in everyday life means finding a difficulty, thinking of different ways to fix it, choosing the best way, and doing it. We all solve small problems every day, even without noticing it.For example, deciding what to eat, choosing the right route to college, finishing assignments on time, or managing money are all problem-solving tasks. First, we must clearly understand the problem. Then we collect information and think of different possible solutions. After that, we choose the best and easiest solution and try it. In the end, we check if it worked. If it did not, we try another way. Good problem-solving needs clear thinking, creativity, and good decisions. These skills make students more confident. In computer programming, we follow the same steps—understand the task, plan the steps, test the program, and improve it. So, everyday problem solving helps us learn how to solve problems in programming too.

Problem solving means finding a solution to any situation or difficulty.
We do it daily, for example:

- Choosing the shortest route to college
- Planning monthly expenses
- Deciding what to cook with available ingredients

Everyday problem solving involves:

1. Identifying the problem
2. Understanding the situation
3. Finding possible solutions
4. Choosing the best solution
5. Applying it

How are problem solving and programming concepts useful in E-commerce?

E-commerce systems handle large amounts of data, online transactions, customer interactions, and security challenges. Problem solving helps in analysing business requirements and designing logical solutions, while programming concepts help in implementing those solutions efficiently. The importance of problem solving and programming concepts in E-commerce can be explained as follows:

1. Website and Application Development

Problem solving is essential during the development of E-commerce websites and mobile applications. Developers must understand user requirements such as product display, registration, login, shopping cart, and checkout process. Programming concepts like variables, loops, conditional statements, functions, and modular programming are used to implement these features. Without proper problem solving, applications may crash or behave incorrectly. Therefore, programming concepts transform ideas into functional E-commerce systems that are user-friendly and reliable.

2. Product Search and Recommendation System

E-commerce platforms contain thousands of products, making efficient search and recommendation systems very important. Problem solving helps in deciding how products should be searched and filtered quickly. Programming concepts such as searching algorithms, sorting techniques, arrays, lists, and database queries are used to implement search options like price, category, and brand. Recommendation systems analyse user behaviour and suggest relevant products. This improves customer satisfaction and increases sales. Thus, problem solving and programming together enhance the overall shopping experience.

3. Order Processing and Inventory Management

Order processing and inventory management require accurate logic and automation. Problem solving helps in designing systems that update stock levels after every order and prevent over-selling. Programming concepts like conditional statements, loops, and database management are used to track product availability and order status.

4. Secure Online Payment and Fraud Prevention

Security is a major concern in E-commerce applications. Problem solving is required to identify risks such as data theft, payment failure, and fraud. Programming concepts like encryption, authentication, authorization, and exception handling are used to protect sensitive customer information. Secure payment gateways are implemented using well-designed algorithms. Error handling ensures safe recovery from transaction failures. Hence, problem solving and programming help in building secure and trustworthy E-commerce systems.

5. User Experience and Personalization

User experience plays a key role in retaining customers. Problem solving helps developers understand customer behavior and improve website usability. Programming concepts like session management, cookies, event handling, and conditional logic are used to personalize content. Showing personalized offers, recommendations, and recently viewed products improves customer engagement.

6. Data Analysis and Business Decision Making

E-commerce platforms generate large volumes of data related to sales and customers. Problem solving helps in analysing this data to find patterns and trends. Programming concepts such as loops, data structures, and analytical algorithms are used to process data efficiently. Businesses use this analysis to make decisions related to marketing, pricing, and inventory planning.

Types of Problems

Problems can be divided in many ways based on how clear and how difficult they are. One way is into well-defined and ill-defined problems. A well-defined problem has clear information.

1) Well-defined problems
2) Simple problems
3) Complex problems

A well-defined problem has clear information, clear steps, and a clear goal, like solving a math sum or finding the average of numbers.Another way to classify problems is as simple or complex. Simple problems take less time and have easy steps, like checking if a number is even or odd. Complex problems need more steps, more data, and sometimes special tools, like creating a banking system or building an online shopping website. Problems can also be creative, logical, or technical.

Problem solving concepts for computers:

Problem solving for computers involves breaking a large problem into smaller tasks and arranging them in a logical sequence. The basic structure of computer problem solving includes input, process, and output. Input is the data provided by the user, processing refers to operations performed on the data, and output is the final result.To make a computer solve a problem, we must write clear and exact steps. These steps are written as an algorithm or shown in a flowchart. A computer follows the instructions exactly, so even a small mistake can give a wrong result. While solving a problem, we must choose the right variables, use correct operations, check conditions (like if/else), and repeat steps using loops when needed. Good problem solving makes the program fast, simple, and correct. This shows us why planning is important before writing code. These problem-solving ideas are the foundation for learning programming.

Algorithms:

An algorithm is a well-structured, step-by-step procedure used to solve a problem. It tells the computer exactly what to do, in what order, and how to handle the data. Algorithms must be clear, simple, and logically arranged so that both humans and computers can understand them. Algorithms can be written in simple English sentences or using pseudo code.

To write an algorithm, the following things are needed:

- A clear understanding of the problem we want to solve.
- The limits or rules (constraints) that must be followed while solving the problem.
- The input values that the algorithm will take.
- The output or result we expect after solving the problem.
- A step-by-step solution that works correctly within the given constraints.

Example1: Write an algorithm to add 3 numbers and print their sum:

1. START
2. Declare 3 integer variables num1, num2, and num3.
3. Take the three numbers, to be added, as inputs in variables num1, num2, and num3 respectively.
4. Declare an integer variable sum to store the resultant sum of the 3 numbers.
5. Add the 3 numbers and store the result in the variable sum.
6. Print the value of the variable sum
7. END

Example2: Write an algorithm to find the factorial of N number:

Step 1: Start

Step 2: Initialize variables:

i = 1, fact = 1

Step 3: Read a number n

Step 4: if i<= n go to step 5 otherwise go to step 8

Step 5: Calculate

fact = fact * i

Step 6: Increment the i by 1 (i=i+1) and go to step 4

Step 7: Print fact

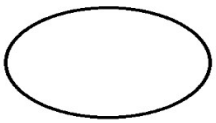
Step 8: Stop

Flowchart:

A flowchart is a visual representation of the steps involved in solving a problem. It uses different symbols like rectangles, diamonds, ovals, and arrows to show actions, decisions, start/end points, and the flow of logic. Flowcharts make it easier to understand the overall structure of a program. They show how data moves from one step to another and how decisions are made based on conditions.

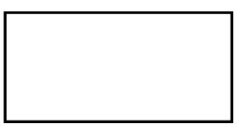
Symbols used in Flowchart

Oval: Start/Stop: The oval symbol indicates Start, Stopin a program's flow.

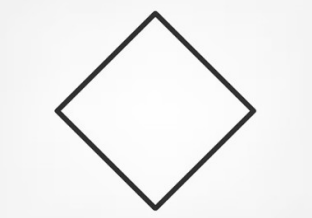


Parallelogram: Input/Output: A parallelogram in a flowchart is used for input and output. It shows steps where the program takes input from the user or displays output on the screen.

Rectangle: Process: A rectangle in a flowchart shows a process or action. It means something is being done, like calculating a sum or storing a value.

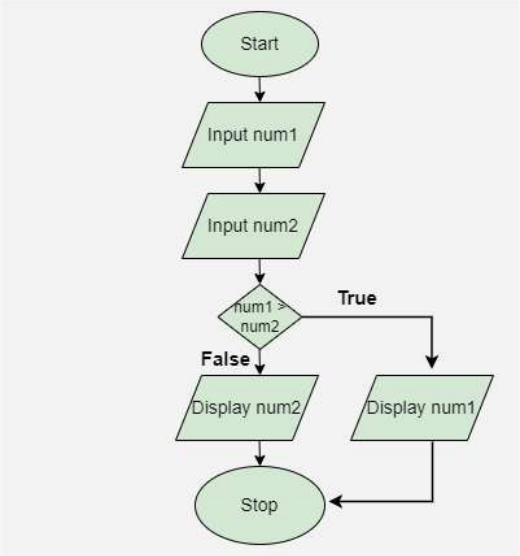


Diamond: Decision: A diamond shows a decision-making step. It asks a question like "Is X > Y?" and gives answers such as Yes/No or True/False.



Arrows: Direction of flow: Arrows show the order of steps in the flowchart. They tell us which step comes next and how the flow moves.

Example1: ***A flowchart to input two numbers from the user and display the largest of two numbers.***

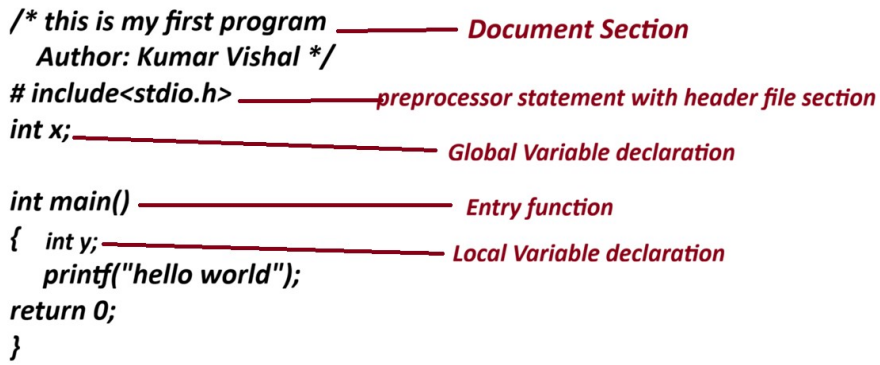


Introduction to Programming structure:

Programming structure refers to the basic layout or format. Means from where we have to start and where we have to finish.

Every C program follows a standard structure:

- Document Section
- Pre-processor statement with header file section
- Global variable declaration
- Main function
- Local Variable declaration



-

Document Section: This section is used to describe the program, such as program name, author, date, or purpose. It uses comments and does not affect program execution.

Single line comment: //

Multiline comment: /*_____ */

Pre-processor statement with header file section:

#include <stdio.h>

A pre-processor statement is a command that starts with # and is processed before compilation. The #include directive tells the compiler to include the contents of a header file. stdio.h header file is used for input/output functions.

Global variable declaration:

A variable which is defined out side of any scope is called global variable.

Main function:

It is called entry function in our program. Program execution always starts from main(). Every C program must have one main() function.

Local Variable declaration: It is used within the scope.

Basic Concepts:

Token in C:

Tokens are some of the most important elements used in the C language for creating a program. For instance, without words, you cannot create any sentence- similarly, you cannot create any program without using tokens in C language.

Here the token in C language:

- Identifiers in C
- Keywords in C
- Operators in C
- Strings in C

- Special Characters in C
- Constant in C

Identifier in C

An identifier is the name used to identify variables, functions, arrays, structures, etc. Suppose a contact list in a mobile phone contains names and phone numbers. Here, the name in the contact list acts as the identifier, and the phone number is the value.

Rules for Identifiers:

• Can contain letters (A–Z, a–z), digits (0–9), and underscore (_)

• Must begin with a letter or underscore

• Cannot be the same as a C keyword

• Case-sensitive (total, Total, and TOTAL are different)

• No special characters allowed (like @, $, %, etc.)

Example:

int age = 25;

float salary = 50000.50;

Here, age and salary are identifiers used for variables.

Keywords in C

Keywords are reserved words that have special meaning in the C language. You cannot use them as identifiers (variable names, function names, etc.). These keywords are part of the syntax and are used to perform specific operations.

List of Common C Keywords

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

Example:

int register;//Invalid

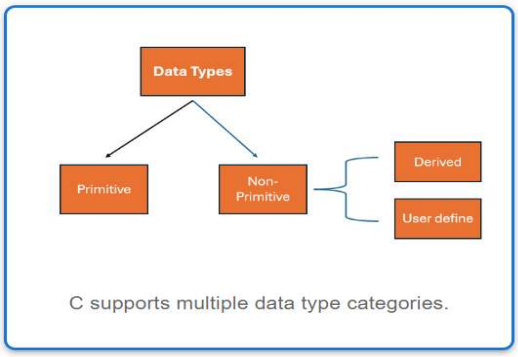
int reg; //valid

C Programming - Data Types

Data types define the type of data a variable can hold. It helps the compiler understand how much memory to allocate and what kind of operations can be performed on the data.

Suppose we have a mix of dry fruits. It is very difficult to separate individual dry fruits from the mix. So, it is better to arrange them in different containers to make the process easier. Data types work in a similar way—we define a data type according to the kind of data, just like choosing the right container based on the item. This helps store data efficiently according to its size and type.

C also supports different category of Data types:



Types	Description	Data Types
Primitive Data Types	Used for representing simple values	int, char, float, double, void
Derived Types	Derived from primitive types	array, pointers, function
User Defined Data Types	Defined by the user	structure, union, enum

Primitive Data Types: Integer (int)

- Any number including positive, negative
- It can be signed and unsigned
- Signed can store both negative and positive values
- Unsigned can store only non-negative values
- Size: 4 bytes
- Range: -2,147,483,648 to 2,147,483,647

Primitive Data Types: Character (char)

• Any single character defined in single quote
• It can be signed and unsigned
• Signed can store both negative and positive values
• Unsigned can store only non-negative values
• Size: 1 byte
• Range: (-128 to 127) or (0 to 255)

Primitive Data Types (float):

Float Data Type (float)
• Stores decimal (real) numbers with single precision
• **Size:** 4 bytes
• **Range:** 1.2E-38 to 3.4E+38
  ↳ 1.2E-38 means 1.2×10^{-38}
  ↳ 3.4E+38 means 3.4×10^{38}

It is especially used in cases where decimal values are required. For example, when we want to measure the human body

Primitive Data Types: Double (`double`)

The double data type is used to store decimal numbers (floating-point values) with double precision. It can accurately represent up to 16 to 17 digits before or after the decimal point.

Size: 8 bytes
Range: 1.7E-308 to 1.7E+308
Modifiers: long double – up to 16 bytes for even greater precision

Primitive Data Types (void)

In C, void is a special data type that represents nothing — no value or no type.

There are 3 common uses of void in C:

- **Void Return Type for Functions:** Used when a function does not return any value.
- **Void Pointers:** void *ptr; — A void pointer can store the address of any data type.
- **Void Function Parameters:** Used when a function takes no arguments.

Non-Primitive Data Types:

Category	Type	Description
Derived	Array	Collection of same type elements
Derived	Pointer	Stores address of another variable
Derived	Function	A block of code that performs a task
User-Defined	Struct	Combines different types into one unit
User-Defined	Union	Shares memory among members
User-Defined	Enum	Defines constants with integer values
User-Defined	Typedef	Creates a new name for an existing type

Constant in C

A constant is a fixed value that does not change during the execution of a program. It is also called a read-only value. Constants can be of types like integer, floating-point, string, or character. They must be initialized when declared; otherwise, they may hold garbage values.

Syntax:
constdata_typevar_name = value;
Example: const float pi = 3.14;
#define CONSTANT_NAME value
Example: #define PI 3.14

Variable in C

A **variable** in C is like a container that stores data. It can be:

- Empty
- Hold a single item
- Contain multiple items of same or different types

Variable Declaration:

Variable_TypeVariable_Name;
int number;

Variable Initialization:

Variable_TypeVariable_Name = value;
int number = 10;

Naming Conventions

- **Pascal Case:** UserName
- **Camel Case:** userName

- **Snake Case:** user_name
- **Kebab Case:** user-name (not used in C)

Types of variables:

1. Based on Size
2. Based on Scope

1. Based on Size:

Data Type	Keyword	Size	Range
char	char	1 byte	-128 to 127 / 0 to 255 (unsigned)
Integer	int	2 or 4 bytes	-32,768 to 32,767 / ~±2 billion
Short Int	short	2 bytes	-32,768 to 32,767
Long Int	long	4 or 8 bytes	Large ranges (platform-dependent)
Float	float	4 bytes	$\pm 3.4 \times 10^{38}$ (6-7 digit precision)
Double	double	8 bytes	$\pm 1.7 \times 10^{308}$ (15-16 digit precision)
Long Double	long double	10-16 bytes	Even higher precision

2. Based on Scope:

Variable Classification Based on Scope

Type	Declared In	Scope / Visibility	Lifetime
Local	Inside function/block	Only within that block	Until block ends
Global	Outside all functions	Entire file	Entire program run
Static	With static keyword	Limited, retains value	Until program ends
Extern	In another file	Used globally	Depends on origin
Register	With register keyword	CPU register (if possible)	Until block ends

Format Specifiers

A format specifier in C is a special symbol used in input and output functions like printf() and scanf() to tell the compiler the type of data being printed or read.

Specifier	Data Type	Example Usage	Output Example
%d	int	`printf("%d", 25);`	25
%f	float / double	`printf("%.2f", 5.6789);`	5.68
%lf	double	`printf("%.15lf", 3.1415926535);`	3.141592653500000
%c	char	`printf("%c", 'A');`	A
%s	string	`printf("%s", "Hello");`	Hello

Expression

A combination of operands and operators is called an expression in C.

Example: c = a + b;

a and b are operands
+ is an arithmetic operator
= is an assignment operator
The whole part a + b is an expression.

Modules and their functions in C:

What is a Module or How to define Module?

A module is a small, independent part of a program that performs a specific task.
A large program is divided into modules to make it easy to understand, develop, test, and maintain.

Each module generally contains one or more functions that perform a specific operation.

For example:

Input Module

Function: Takes input from the user

```
void input()
{
scanf("%d", &a);
}
```

Output Module

Function: Displays result

```
void output(int sum)
{
printf("Sum = %d", sum);
}
```

What is Function Prototype?

A function prototype in C is a declaration of a function that tells the compiler about the function's name, return type, and parameters.

Example: int add(int a, int b);

Operators in C

An **operator** is a symbol that tells the compiler to perform a specific operation on one or more operands (variables or values).

Example: int sum = a + b;
'+' is an operator; a and b are operands.

Types of Operators in C

1. Arithmetic Operators
2. Assignment Operators
3. Relational Operators
4. Logical Operators
5. Bitwise Operators
6. Increment/Decrement Operator
7. Unary Operator
8. Binary Operator
9. Ternary Operator or Conditional Operator
10. Other Operators

1. Arithmetic Operators

Arithmetic operators are used to perform mathematical operations such as addition, subtraction, multiplication, division, and modulus on numeric values.

Operator	Name	Description	Example (a = 10, b = 3)	Result
+	Addition	Adds two operands	a + b	13
-	Subtraction	Subtracts second operand from first	a - b	7
*	Multiplication	Multiplies two operands	a * b	30
/	Division	Divides first operand by second	a / b	3 (int)
%	Modulus	Returns remainder after division	a % b	1

2. Assignment Operators

Assignment operators are used to assign values to variables. The main operator used is = (equal to), which assigns the value from right to left.

Example: int a = 10;

The assignment operators can be combined with some other operators in C. in C, assignment operators can be combined with other arithmetic or bitwise operators to form shorthand assignment operators.

Operator	Description	Example	Equivalent To
+=	Add and assign	a += b;	a = a + b;
-=	Subtract and assign	a -= b;	a = a - b;
*=	Multiply and assign	a *= b;	a = a * b;
/=	Divide and assign	a /= b;	a = a / b;
%=	Modulus and assign	a %= b;	a = a % b;
<<=	Left shift and assign	a <<= 2;	a = a << 2;
>>=	Right shift and assign	a >>= 1;	a = a >> 1;
&=	Bitwise AND and assign	a &= b;	a = a & b;
^=	Bitwise XOR and assign	a ^= b;	a = a ^ b;

3. Relational Operators

Relational operators in C are used to compare two values or expressions. The result of a relational operation is either **true** or **false**. They are commonly used in:

- if, else if, and else conditions
- Loops (while, for)
- Decision-making statements

List of Relational Operators:

Operator	Description	Example (a = 10, b = 5)	Result
==	Equal to	a == b	0 (false)
!=	Not equal to	a != b	1 (true)
>	Greater than	a > b	1
<	Less than	a < b	0
>=	Greater than or equal to	a >= b	1
<=	Less than or equal to	a <= b	0

4. Logical Operators

Logical operators are used in C to combine multiple conditions. The result is either true or false. They are commonly used with if, while, for, and other control statements.

Operator	Name	Syntax Example	Meaning
&&	Logical AND	cond1 && cond2	True if both conditions are true
||	Logical OR	cond1 || cond2	True if any condition is true
!	Logical NOT	!cond	True if condition is false

5. Bitwise Operators

Operator	Name	Description
&	Bitwise AND	Sets bit to 1 if both bits are 1
|	Bitwise OR	Sets bit to 1 if any bits are 1
^	Bitwise XOR	Sets bit to 1 if bits are different
~	Bitwise NOT	Inverts all bits (1 → 0, 0 → 1)
<<	Left Shift	Shifts bits to the left
>>	Right Shift	Shifts bits to the right

6. Increment/Decrement Operators

Type	Syntax	Meaning
Pre-increment	++x	Increases value **before** using it
Post-increment	x++	Increases value **after** using it
Pre-decrement	--x	Decreases value before using it
Post-decrement	x--	Decreases value after using it

7. Unary Operator

A Unary Operator is an operator that works on only one operand (one variable or value).

Operator	Name	Purpose
++	Increment	Increases value by 1
--	Decrement	Decreases value by 1
+	Unary plus	Indicates positive value
-	Unary minus	Changes sign of value

8. Binary Operator

A Binary Operator is an operator that works on two operands to perform an operation.

Operator	Operation	Example
+	Addition	a + b
-	Subtraction	a - b
*	Multiplication	a * b
/	Division	a / b
%	Modulus	a % b

9. Ternary Operator or Conditional Operator

The ternary operator is the only operator in C that works on three operands.

Syntax:

condition ? expression1 : expression2;

If condition is true → expression1 is executed

If condition is false → expression2 is executed

Example:

```
int a = 10, b = 20;
int max = (a > b) ?a : b;
printf("%d", max);   // Output: 20
```

10. Other Operators:

sizeof Operator: Used to find the **size of data type or variable in bytes**.

Example: printf("%d", sizeof(int));

Address Operator (&): Used to get the address of a variable.

```
int x = 10;
printf("%p", &x);
```

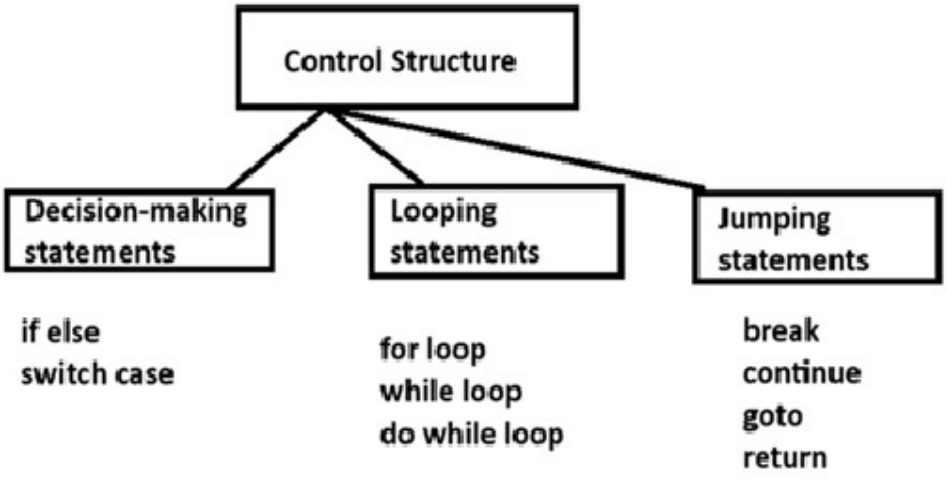
Control Structure in C (Four Logic structures)

Control structures help manage how a program runs its instructions. They allow the program to make choices, repeat certain steps, or follow different paths depending on conditions. For example, using if, a program can run a part of the code only when a condition is true. Loops let the program repeat tasks, and statements like break can stop the process early. These structures make programs more logical and flexible.

We can say that a control structure is a combination of decision-making statements, looping statements, and jumping statements.

Four logical structures are:

No.	Logic Structure	Example
1	Sequential	Simple statements(by default)
2	Decision	if, if-else, switch
3	Looping	for, while, do-while
4	Jumping	break, continue, goto, return



Decision-making statements:

Decision-making statements in C are used to control the flow of a program based on certain conditions. These statements check whether a condition is true and then perform a set of instructions accordingly. If the condition is not met, they may follow an alternative path. The decision-making statements are: **if else** and **switch case**. This is also known as selection statements.

1. If statement:

Syntax:

```
if (condition) {

  // Block of code if condition is true

}
```

Example:

```
#include <stdio.h>
int main()
{
   int age = 18;
   if (age >= 18)
   {
printf("Eligible for Vote\n");
printf("Welcome\n");
   }
 return 0;
}
```

2. if-else Statement:
A condition can be checked for either true or false. If the condition is true, the if block will be executed; if the condition is false, the else block will be executed.

Syntax:

```
if (condition) {
   // executes if condition is true
} else {
   // executes if condition is false
}
```

Example:

```
#include <stdio.h>
int main() {
   int num = 5;
   if (num % 2 == 0) {
printf("Even number");
   } else {
printf("Odd number");
   }
   return 0;
}
```

3. if-else if-else Ladder (Multiple Conditions):
To check multiple conditions, we can use an if–else if–else ladder. The first condition that evaluates to true is executed, and the rest are skipped. Once a condition is true, all remaining conditions are ignored.

Example:

```
#include <stdio.h>
int main() {
   int marks = 72;
   if (marks >= 90) {
printf("Grade A\n");
   } else if (marks >= 75) {
printf("Grade B\n");
   } else if (marks >= 50) {
printf("Grade C\n");
   } else {
printf("Fail\n");
   }
   return 0;
}
```

4. Nested if (if inside another if):

Nested if is useful when one check depends on another.

Example 1:

```
#include <stdio.h>
int main() {
   int age = 25;
   char citizen = 'Y'; // Y for Yes
   if (age >= 18) {
      if (citizen == 'Y') {
printf("You can vote in India.\n");
      }
   }
   return 0;
}
```

Switch-Case in C

When you want to choose one option from many possible options based on the value of a single expression go for switch-case. Especially used for menu-driven programs.

Syntax:

```
switch (expression) {
   case constant1:
      // code if expression == constant1
      break;
   case constant2:
      // code if expression == constant2
      break;
   ...
   default:
      // code if no case matches
}
```

- **Expression**: Can be int, char, enum (NOT float or double).
- **Case labels**: Must be constant or literal values (e.g., case 5:, case 'A':).
- **break**: Stops the switch from running the next cases.
- **default**: Runs when no case matches (optional).

Example 1:

```
#include <stdio.h>
int main() {
   int day = 3;
   switch (day) {
      case 1:
printf("Monday\n");
```

```
            break;
        case 2:
printf("Tuesday\n");
            break;
        case 3:
printf("Wednesday\n");
            break;
        case 4:
printf("Thursday\n");
            break;
        case 5:
printf("Friday\n");
            break;
        default:
printf("Weekend\n");
    }
    return 0;
}
```

Loops in C

A loop means repeating the same steps, which is called iteration. Loops are used to execute a block of code repeatedly until a certain condition is met.There are 3 main types of loops in C: while, do-while, and for loops. Loops are classified into Entry Control and Exit Control loops.

Entry Control Loop: Condition is checked **before** entering the loop. Example: for loop, while loop

Exit Control Loop: Condition is checked after executing the loop body. Example: do while loop

Difference between Entry Control Loop and Exit Control Loop

	Entry Control Loop	**Exit Control Loop**
Condition check	Condition is checked **before** executing the loop body	Condition is checked **after** executing the loop body
Minimum execution	May execute **zero times**	Executes **at least once**
Control type	Entry-controlled	Exit-controlled
Examples	while, for loop example	do–whileloop example
Use case	When loop should run only if condition is true	When loop must run at least once

While-Wend Structure: Used in some languages like old BASIC.

Syntax:
WHILE condition
  statements
WEND
It works same as while loop.
1. while Loop

Executes the code block **as long as** the condition is true. The condition is checked **before** the loop runs.

```
#include <stdio.h>
int main() {
   int i = 1;
   while (i<= 5) {
printf("%d\n", i);
i++;
   }
   return 0;
}
```

Repeat-Until loop executes statements at least once, then checks condition.
2. do-while Loop

Executes the code block **at least once** before checking the condition.

```
#include <stdio.h>
int main() {
   int i = 1;
   do {
printf("%d\n", i);
i++;
   } while (i<= 5);
   return 0;
}
```

3. for Loop (Automatic Counter Loop)

Used when you know **exactly how many times** to run the loop.

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

Jumping Statements in C

Jumping statements in C are used to transfer control from one part of the program to another. They also allow you to skip parts of code.There are different types of Jumping Statements in C:

1. break

The **break** statement is used to terminate loop or switch case immediately.

```
#include <stdio.h>
int main() {
for(int i = 1; i<= 5; i++) {
if(i == 3) break;
printf("%d\n", i);
   }
   return 0;
}
```

2. continue

The **continue** statement skips the remaining statements in the current iteration and moves to the next iteration.

```
#include <stdio.h>
int main() {
for(int i = 1; i<= 5; i++) {
if(i == 3) continue;
printf("%d\n", i);
   }
   return 0;
}
```

3. goto

The **goto** statement transfers control to a labeled statement.

```
#include <stdio.h>
int main() {
   int num = 2;
if(num == 2) goto label;
printf("This will be skipped\n");
label:
printf("Jumped here using goto\n");
   return 0;
}
```

4. return

The **return** statement exits from a function and optionally returns a value.

```
#include <stdio.h>
int add(int a, int b) {
   return a + b;
```

```
}
int main() {
printf("%d\n", add(3,4));
   return 0;
}
```

5. exit()

The **exit()** function terminates the program immediately. It Requires header file: #include <stdlib.h>.

```
#include <stdio.h>
#include <stdlib.h>
int main() {
printf("Start\n");
exit(0);
printf("This will not be executed\n");
   return 0;
}
```

Nested Loops and Recursion

A Nested Loop is a loop inside another loop. The loop that contains another loop is called the Outer Loop. The loop inside it is called the Inner Loop. The inner loop executes completely for each iteration of the outer loop.

Uses of Nested Loops:

- Pattern printing
- Matrix addition and multiplication
- 2D array processing
- Searching in tables

Syntax:

```
for(initialization1; condition1; increment1)
{
   for(initialization2; condition2; increment2)
   {
      // statements
   }
}
```

Example:

```
#include <stdio.h>
int main()
{
   for(int i = 1; i <= 3; i++)
   {
      for(int j = 1; j <= 3; j++)
      {
         printf("* ");
      }
      printf("\n");
   }
```

```
    return 0;
}
```

Recursion:

Recursion is a programming technique in which a function calls itself to solve a problem. Instead of using loops, the function repeats its task by calling itself with a smaller value.

Example: Factorial Using Recursion

```
#include <stdio.h>
int factorial(int n)
{
    if(n == 1)          // Base case
        return 1;
    else
        return n * factorial(n - 1);   // Recursive call
}
int main()
{
    int num = 4;
    printf("Factorial = %d", factorial(num));
    return 0;
}
```

Array Data Structure

An array in C is a collection of elements of the same data type stored at contiguous memory locations. Arrays allow us to store multiple values under a single variable name and access them using an index.

Important Features:

- Stores multiple elements of the same data type.
- Elements are stored in consecutive memory locations.
- Array index starts from 0.
- We traverse array elements using an index: array[index].

Syntax:

data_type array_name[size];

Example:

```
#include <stdio.h>
int main() {
    int marks[5] = {90, 85, 78, 92, 88};
    for (int i = 0; i < 5; i++) {
        printf("Marks[%d] = %d\n", i, marks[i]);
    }
    return 0;
}
```

Limitations of Arrays:

- Fixed size – cannot grow or shrink dynamically.
- All elements must be of the same data type.

Types of Arrays:

- One-Dimensional Array – Stores elements in a single row.
- Two-Dimensional Array – Stores elements in rows and columns (matrix form).
- Multi-Dimensional Array – More than two dimensions.

1. One-Dimensional Array (1D)

A one-dimensional array stores elements in a single row. It is the simplest form of an array.

Syntax:

```
data_type array_name[size];
```

Example:

```
#include <stdio.h>
int main() {
   int numbers[5] = {10, 20, 30, 40, 50};
   for (int i = 0; i < 5; i++) {
      printf("%d ", numbers[i]);
   }
   return 0;
}
```

2. Two-Dimensional Array (2D)

A two-dimensional array stores elements in rows and columns (like a matrix).

Syntax:

```
data_type array_name[rowsize][columnsize];
```

Example:

```
#include <stdio.h>
int main() {
   int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}};
   for (int i = 0; i < 2; i++) {
      for (int j = 0; j < 3; j++) {
         printf("%d ", matrix[i][j]);
      }
      printf("\n");
   }
   return 0;
}
```

3. Multi-Dimensional Array (n dimension)

A multi-dimensional array has more than two dimensions. In Multidimension Array, Array becomes Array of Arrays. They are mostly used for complex data structures like 3D matrices.

Syntax of 3D Array:

```
data_type array_name[no.of 2d arrays][rowsize][columnsize];
```

Example:

```
#include <stdio.h>
int main() {
   int cube[2][2][2] = {
      {{1, 2}, {3, 4}},
      {{5, 6}, {7, 8}}
```

```
    };
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            for (int k = 0; k < 2; k++) {
                printf("%d ", cube[i][j][k]);
            }
            printf("\n");
        }
        printf("\n");
    }
    return 0;
}
```

Searching and Sorting Techniques

SEARCHING TECHNIQUES

What is searching?

Searching is the process of finding the location of a specific element in an array. There are many algorithms to search an array element. One of the simplest algorithm is Linear Search algorithm. Linear search checks each element one by one until the element is found.

Algorithm:

1. Start
2. Read array and key element
3. Compare key with each element
4. If match found → Display position
5. If not found → Display "Not Found"
6. Stop

Example:

```
#include <stdio.h>
int main()
{
    int arr[5] = {10, 20, 30, 40, 50};
    int key = 30;
    int found = 0;
    for(int i = 0; i < 5; i++)
    {
        if(arr[i] == key)
        {
            printf("Element found at position %d", i+1);
            found = 1;
            break;
        }
    }
    if(found == 0)
        printf("Element not found");
    return 0;
}
```

SORTING TECHNIQUES

What is sorting?

Sorting is arranging elements in: Ascending order or Descending order. There are many algorithms for sorting but simplest one is Bubble Sort in which, Compares adjacent elements and swaps if needed. Largest element moves to the end each pass.

Example:

```
#include <stdio.h>
int main()
{
   int arr[5] = {50, 20, 40, 10, 30};
   int temp;
   for(int i = 0; i < 4; i++)
   {
      for(int j = 0; j < 4 - i; j++)
      {
         if(arr[j] > arr[j+1])
         {
            temp = arr[j];
            arr[j] = arr[j+1];
            arr[j+1] = temp;
         }
      }
   }
   printf("Sorted Array:\n");
   for(int i = 0; i < 5; i++)
      printf("%d ", arr[i]);
   return 0;
}
```

File handling Concepts

File handling in C allows storing and accessing data permanently on the hard disk. A file is a storage area on the hard disk where data remains saved even after the program ends.

Steps in File Handling

The steps involved in file handling are:

1. Declare a file pointer
2. Open the file
3. Perform read/write operations
4. Close the file

File Operations in C:

1. Writing data to a File

Steps:

1. Declare a file pointer
2. Open the file
3. Write operation
4. Close the file

```
#include <stdio.h>
int main()
{
   FILE *fp;
   // Open file in write mode
   fp = fopen("write.txt", "w");

   if (fp == NULL) {
      printf("Error opening file!");
      return 1;
   }
   // Writing text into the file
   fprintf(fp, "This is a text file.\n");
   fprintf(fp, "File handling in C.\n");
   // Closing the file
   fclose(fp);
   printf("Data written successfully!");
   return 0;
}
```

2. Reading data from a File

Steps:

1. Declare a file pointer
2. Open the file
3. Read operation
4. Close the file

```
#include <stdio.h>
int main() {
   FILE *fp;
   char line[100];
   // Open the existing file write.txt in read mode
   fp = fopen("write.txt", "r");
   if (fp == NULL) {
      printf("File not found!");
      return 1;
   }
   printf("Contents of File:\n");
   // Read the file line by line
   while (fgets(line, sizeof(line), fp)) {
      printf("%s", line);
   }
   // Close the file
   fclose(fp);
   return 0;
```

```
}
```

3. Add more data to a File

Steps:
1. Declare a file pointer
2. Open the file
3. Append operation
4. Close the file

```
#include <stdio.h>

int main()
{
   FILE *fp;
   // Open file in write mode
   fp = fopen("file.txt", "a");
   if (fp == NULL) {
      printf("Error opening file!");
      return 1;
   }
   // Writing more data into the file
   fprintf(fp, "Hello\n");
   fprintf(fp, "How are you?\n");
   // Closing the file
   fclose(fp);
   printf("Data written successfully!");
   return 0;
}
```

Sequential Access Files

In sequential access, data is read or written one by one in order, starting from the beginning of the file to the end. Common functions used are: fgetc(), fgets(), fprintf(), fscanf().

Example:

```
//Sequential Access File Example
#include <stdio.h>
#include <stdlib.h>
int main() {
   FILE *fp;
   char name[20], line[100];
   int age, choice;
   do {
      printf("\n*****SELECT CHOICE*****\n");
      printf("1. Write Data to File\n");
      printf("2. Read Data from File\n");
      printf("3. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);
```

```
        switch (choice) {
        case 1:
            fp = fopen("Student.txt", "a");
            if (fp == NULL) {
                printf("Error\n");
                return 1;
            }
            printf("Enter name and age: ");
            scanf("%s %d", name, &age);
            fprintf(fp, "Name: %s\nAge: %d\n", name, age);
            fclose(fp);
            printf("Data written successfully!\n");
            break;

        case 2:
            fp = fopen("student.txt", "r");
            if (fp == NULL) {
                printf("Error\n");
                return 1;
            }

            printf("\n----- File Content -----\n");
            while (fgets(line, sizeof(line), fp) != NULL) {
                printf("%s", line);
            }
            fclose(fp);
            printf("\n------------------------\n");
            break;

        case 3:
            printf("Exiting program\n");
            exit(0);

        default:
            printf("Please try again.\n");
        }

    } while (1);

    return 0;
}
```

Random Access Files

In random access, you can directly jump to any part of the file without reading previous data.
Important functions: fseek(), ftell(), rewind().
fseek() Syntax:
fseek(file_pointer, offset, position);
Position Options:
SEEK_SET → Beginning of file

SEEK_CUR → Current position
SEEK_END → End of file

Random Access – Full Example:

```
#include <stdio.h>
int main()
{
   FILE *fp;
   fp=fopen("emp1.dat","wb");
   if(fp==NULL)
   {
      printf("Error");
      return 1;
   }
   else{
      fseek(fp,10,SEEK_SET);
      fprintf(fp,"kumar");
      printf("Position %d",ftell(fp));
      fclose(fp);
      printf("\nData written Successfully");
   }
   return 0;
}
```

Application Domains

Bio-informatics and medical Applications:

What is Bioinformatics?

Bioinformatics is the use of computers and software to study biological data like:

- DNA
- RNA
- Proteins
- Genes
- Medical records

It combines:

- Biology
- Computer Science
- Mathematics
- Statistics

In simple words:
Bioinformatics helps scientists understand the human body using computers.

Medical Applications of Bioinformatics:

Bioinformatics is the use of computers and software to store, manage, and analyze biological and medical data. It plays an important role in modern healthcare by helping doctors understand diseases, study genes, and improve treatment methods.

One major application of bioinformatics is **disease diagnosis**. By analyzing a patient's DNA, doctors can detect genetic mutations that cause diseases such as cancer and inherited disorders. Research programs like The Cancer Genome Atlas help scientists study cancer-related genes and improve early detection and treatment.

Another important application is **personalized medicine**. In this approach, treatment is given according to a person's genetic makeup. Since each person's genes are different, bioinformatics helps doctors choose the right medicine and correct dosage. This reduces side effects and improves recovery.

Bioinformatics is also widely used in **drug discovery** and development. Scientists use computer models to study disease-causing proteins and design new drugs. This reduces the time and cost required for testing. For example, during the COVID-19 pandemic, genome analysis of SARS-CoV-2 helped in the fast development of medicines and vaccines.

In vaccine development, bioinformatics helps identify virus proteins that can be targeted to create vaccines. It also helps track virus mutations and improve vaccine effectiveness.

Another important medical application is the **Hospital Management System (HMS).** HMS is a computerized system used to manage hospital activities such as patient registration, electronic health records (EHR), appointment scheduling, laboratory reports, billing, and pharmacy management. It stores patient data digitally, reduces paperwork, improves efficiency, and helps doctors access medical records quickly. Bioinformatics tools also help analyze stored patient data for better diagnosis and treatment planning.

Main Functions of HMS:

Patient Registration

- Stores patient details (name, age, contact, medical history).
- Generates unique patient ID.

Electronic Health Records (EHR)

- Stores lab reports, prescriptions, diagnosis, and treatment history.
- Doctors can easily access patient data anytime.

Appointment Management

- Schedules doctor appointments.
- Reduces waiting time.

Billing and Accounts

- Generates bills automatically.
- Maintains payment records.

Laboratory Management

- Stores test results.
- Sends reports directly to doctors.

Pharmacy Management

- Tracks medicine stock.
- Manages prescriptions digitally.

Business Applications

Business applications of Information Technology and data analysis play a very important role in modern organizations. Businesses use computer systems and data analysis tools to improve decision-making, increase profits, and understand customer behavior.

One major application is **Customer Relationship Management (CRM)**. Businesses collect customer data such as purchase history, preferences, and feedback. This data is analyzed to understand customer needs and provide better services. For example, companies like Amazon recommend products based on previous purchases. This increases sales and customer satisfaction.

Another important application is **Market Analysis and Sales Forecasting**. Businesses analyze past sales data to predict future demand. This helps companies decide how much product to produce and when to launch new products. For example, companies like Walmart use data analysis to manage inventory and understand buying patterns.

Financial Management is also an important business application. Companies use software to manage accounts, track expenses, prepare financial reports, and detect fraud. Data analysis helps in identifying unusual transactions and reducing financial risks.

Supply Chain Management (SCM) is another major application. Businesses use IT systems to manage suppliers, transportation, and product delivery. It ensures that products reach customers on time and reduces operational costs.

In **Human Resource Management (HRM)**, businesses use information systems to manage employee records, payroll, attendance, and performance evaluation. Data analysis helps in selecting suitable candidates and improving workforce productivity.

Business Intelligence (BI) tools help managers make strategic decisions. These tools convert raw data into useful information through reports, charts, and dashboards. Managers use this information to plan business strategies and improve performance.

Law Enforcement

Law enforcement means government authorities make sure people follow the law by stopping crimes, finding rule-breakers, and taking action against them.
It includes:

- Maintaining public order
- Preventing an detecting crime
- Investigating legal violations
- Arresting or penalizing offenders
- Enforcing regulatory laws (such as tax, financial, and compliance laws)

Example of Law enforcement Applications

Cybercrime Investigation Software is specialized software used by law enforcement agencies to investigate crimes committed using computers, mobile devices, and the internet. It helps officers collect, preserve, and analyze digital evidence such as emails, chat messages, call records, deleted files, browsing history, and financial transactions.

Network Security and Monitoring Software is used to protect computer networks from hackers, viruses, and unauthorized access. It continuously watches network traffic to detect unusual or suspicious activities. If any threat is found, it alerts administrators so they can take quick action. This software helps prevent data theft, cyberattacks, and system damage. It also records logs of all activities for security analysis.

Records Management Systems (RMS) are software applications used by law enforcement agencies to store, organize, and manage official records digitally. They keep important information such as complaints (FIRs), arrest records, criminal history, incident reports, property details, and case updates in one secure system. RMS helps officers quickly search and retrieve data when needed. It reduces paperwork, improves accuracy, and ensures proper record keeping.

Income tax return–based software can be considered an example of a law enforcement application because it helps the government enforce tax laws. This software is used to collect, process, and verify income details submitted by individuals and businesses. It checks whether taxpayers have paid the correct amount of tax and identifies mismatches or suspicious transactions. If irregularities are found, the system helps generate notices, penalties, or further investigation.

Political Process Applications

Election Management Platforms are software systems used by election authorities to plan, organize, and manage the entire election process. They help in voter registration, candidate nomination, ballot preparation, polling station management, and result tabulation. These systems store large amounts of voter data securely and ensure accurate record keeping. They also help officials track polling staff, election materials, and voting machines.

Electronic Poll book Systems are digital versions of traditional paper voter lists used at polling stations. They help election officials verify voter identity quickly and check whether a person is eligible to vote. When a voter arrives, officials search their name in the system, confirm their details, and mark that they have voted.

Open Voting or Ballot Systems are digital or online systems used to conduct voting in a transparent and secure manner. These systems allow voters to cast their votes electronically, either at polling stations or remotely in some cases. They use encryption and security measures to protect voter privacy and prevent tampering. Votes are automatically recorded and counted, reducing manual errors.

Specialized Election Deployment Tools are software systems used to coordinate security forces, polling staff, and resources during elections. They help authorities manage the movement of police personnel, election officers, and materials such as voting machines. These tools track deployment locations, monitor sensitive areas, and ensure law and order at polling stations.

E-commerce

E-commerce (Electronic Commerce) refers to the buying and selling of goods and services over the internet. It allows businesses and customers to conduct transactions digitally using websites, mobile apps, and online payment systems. Instead of visiting physical stores, customers can browse products, compare prices, place orders, and make payments from anywhere using a computer or Smartphone.

E-commerce includes different types such as Business-to-Consumer (B2C), Business-to-Business (B2B), Consumer-to-Consumer (C2C), and Consumer-to-Business (C2B). Popular platforms like Amazon and Flipkart are examples of B2C e-commerce, where businesses sell directly to customers.

E-commerce offers many advantages, including 24/7 shopping, a wide variety of products, easy price comparison, and home delivery.

E-commerce Applications

Online Shopping Platforms (B2C)

Example: Amazon, Flipkart

These platforms allow businesses to sell products directly to customers. Users can search products, compare prices, read reviews, add items to cart, make online payments, and track deliveries. They provide 24/7 shopping convenience.

Online Payment Applications

Example: PayPal, PhonePe

These apps securely transfer money between buyers and sellers. They support debit cards, credit cards, UPI, and digital wallets.

Online Food Delivery Applications

Example: Zomato, Swiggy

Customers can order food from restaurants, make payments online, and track delivery in real time.

Online Ticket Booking Applications

Example: IRCTC

Users can book train tickets, check seat availability, and cancel tickets online.

Manufacturing Applications

Manufacturing applications are software systems used in factories and industries to plan, manage, control, and improve production processes. These applications help manufacturers increase efficiency, reduce costs, maintain quality, and manage resources properly.

Enterprise Resource Planning (ERP)

Example: SAP ERP

ERP systems integrate production, inventory, finance, HR, and sales into one system. They help track raw materials, manage orders, and plan production schedules.
Manufacturing Execution System (MES)
Example: Siemens Opcenter
MES monitors and controls production on the factory floor. It tracks work progress, machine performance, and product quality in real time.
Computer-Aided Design (CAD)
Example: AutoCAD
CAD software is used to design machines, tools, and products digitally before manufacturing begins.
Supply Chain Management (SCM)
Example: Oracle SCM Cloud
SCM software manages suppliers, transportation, and product delivery.

Education-Based Applications

Learning Management Systems (LMS) are online platforms used to manage teaching and learning activities. They allow teachers to upload study materials, assignments, quizzes, and videos in one place. Students can log in to access lessons, submit homework, take tests, and check results. LMS also helps teachers track student progress and attendance. It supports both online and blended learning. A popular example is Moodle, which is widely used by schools and colleges.

Online learning platforms provide courses and training programs through the internet. Students can learn new skills, attend lectures, watch recorded videos, and complete assignments from home. These platforms often offer certificates after course completion. They cover subjects like programming, business, language learning, and competitive exams. A well-known example is Coursera, which offers courses from universities worldwide.

Online Examination Systems are digital platforms used to conduct tests and assessments through computers or mobile devices. They allow institutions to create question papers, schedule exams, and evaluate answers automatically. Multiple-choice questions are graded instantly, saving time and effort. These systems also reduce paperwork and minimize cheating using security features like time limits and monitoring tools. An example is Google Forms, which is commonly used for quizzes and exams.

Student Information Systems (SIS) are software applications used to manage student data in schools and colleges. They store information such as admission details, attendance records, grades, fees, and personal data. Teachers and administrators can easily update and access student information when needed.

Entertainment based Applications

Video streaming applications allow users to watch movies, TV shows, web series, and documentaries online. Users can stream content anytime using smartphones, smart TVs, or computers. These apps provide on-demand entertainment without downloading files. They also offer personalized recommendations based on viewing history. A popular example is Netflix, Another example is Disney+ Hotstar. Video streaming applications have changed

entertainment by making content easily accessible, affordable, and available anywhere with an internet connection.

Music streaming applications allow users to listen to songs, albums, and podcasts online. Instead of downloading music files, users can stream unlimited songs using the internet. These apps provide playlists, personalized recommendations, and offline listening options. A well-known example is Spotify, which offers millions of songs in different languages.

Online gaming applications allow users to play games individually or with other players over the internet. These games can be played on mobile phones, computers, or gaming consoles. They include action games, sports games, puzzles, and multiplayer battle games. An example is PUBG: Battlegrounds, which allows players to compete in real-time matches. Another example is Free Fire.

Social media entertainment applications allow users to share photos, videos, and short clips for fun and creativity. Users can watch entertaining content, follow celebrities, and interact with friends. These platforms provide features like reels, live streaming, and messaging. A popular example is Instagram, known for photo and video sharing. Another example is YouTube, where users watch and upload videos.

Agriculture based applications

Farm Management Software helps farmers plan, monitor, and manage daily farming activities. It records information about crops, soil condition, irrigation schedules, fertilizer use, labor, and expenses. Farmers can track crop growth and predict yield using digital tools.An example is AgriWebb, which helps farmers manage livestock and farm records efficiently.

Weather forecasting applications provide farmers with accurate weather updates such as rainfall, temperature, humidity, and wind speed. This information helps farmers decide the best time for sowing, irrigation, and harvesting. By knowing weather conditions in advance, farmers can protect crops from storms or drought. An example is Skymet Weather, which provides agricultural weather updates.

Precision agriculture applications use advanced technologies like GPS, sensors, and drones to monitor crop health and soil quality. These tools collect data and provide accurate recommendations about fertilizer use, irrigation, and pest control. This reduces unnecessary use of chemicals and saves costs.

Agricultural market applications connect farmers directly with buyers, reducing middlemen. Farmers can check market prices, sell products online, and receive digital payments. These platforms help farmers get better prices and access larger markets. An example is eNAM, which is an online trading platform for agricultural produce in India.

****