PVP 23

Code: 23ES1102

I B.Tech - I Semester – Regular Examinations - JANUARY 2024

INTRODUCTION TO PROGRAMMING

(Common for ALL BRANCHES)

Duration: 3 hours Max. Marks: 70

Note: 1. This question paper contains two Parts A and B.
2. Part-A contains 10 short answer questions. Each Question carries 2 Marks.
3. Part-B contains 5 essay questions with an internal choice from each unit. Each Question carries 10 marks.
4. All parts of Question paper must be answered in one place.

BL – Blooms Level CO – Course Outcome

PART – A

		BL	CO
1.a)	Differentiate between the top-down and bottom-up problem-solving approaches.	L2	CO1
1.b)	Differentiate between algorithm and flow chart.	L2	CO1
1.c)	Differentiate between a "while" loop and a "do-while" loop, and provide an example of when you would choose one over the other.	L2	CO1
1.d)	Explain the difference between the "if" statement and the "if-else" statement in terms of their execution.	L2	CO1
1.e)	In programming, what is a string, and how is it typically represented in memory?	L1	CO1
1.f)	What is the purpose of declaring the size of an array when you create it in a programming language like C?	L1	CO1
1.g)	Differentiate between a pointer variable and a regular variable in terms of how they store data.	L2	CO1
1.h)	Explain the role of functions like 'malloc()' and 'free()' in dynamic memory allocation in C.	L2	CO1
1.i)	What is a function in programming, and why is it used?	L1	CO1
1.j)	Compare call-by-value with call-by-reference.	L2	CO1

PART – B

			BL	CO	Max. Marks
		UNIT-I			
2	a)	Explain various operators in C.	L2	CO1	5 M
	b)	Compare and contrast high-level programming languages and low-level programming languages. Give examples of each and discuss their respective advantages and disadvantages.	L2	CO1	5 M
		OR			
3	a)	Discuss the concept of data types and their importance in programming. Provide examples of situations where choosing the right data type is crucial for program efficiency.	L2	CO1	5 M
	b)	Write an algorithm and draw a flow chart to calculate the sum of first 10 natural numbers.	L2	CO1	5 M
		UNIT-II			
4	a)	Create a C program that employs a "while" loop to print all even numbers between 1 and 50, but skips any numbers that are divisible by 6 using the "continue" statement. Provide the code and a detailed explanation.	L3	CO2	5 M
	b)	Write a C program that uses a "for" loop to find the first prime number between 100 and 200. Implement the "break" statement to exit the loop once the prime number is found.	L3	CO2	5 M
		OR			
5	a)	Discuss the advantages of using a "switch" statement over a series of "if" statements in certain scenarios. Provide an example to illustrate your point.	L2	CO1	5 M
	b)	Create a C program that continuously prompts the user to enter a positive integer until a negative number is entered. Calculate and display the sum of all the positive integers entered by the user. Utilize a "while" loop,	L3	CO2	5 M

		conditional statements, and the "break" statement to terminate the loop when a negative number is provided.			
		UNIT-III			
6	a)	Discuss the importance of string manipulation in programming, including tasks like comparison, concatenation, and substring extraction. Provide a code example in C that demonstrates these string operations.	L3	CO3	5 M
	b)	Explain the advantages of using a two-dimensional array over a one-dimensional array when working with tabular data or grids. Provide real-world examples where two-dimensional arrays are useful.	L2	CO2	5 M
		OR			
7	a)	Imagine you need to manage a list of customer names in a business application. Discuss the advantages and disadvantages of using an array for this purpose.	L3	CO2	5 M
	b)	You have an array of integers representing the daily temperatures for a week (index0: Sunday, index1: Monday and so on). Write a C program that finds and prints the day with the highest temperature and the temperature itself.	L3	CO3	5 M
		UNIT-IV			
8	a)	Design a C program that reverses the elements of an integer array using pointers. Provide the code and a step-by-step explanation of the algorithm.	L3	CO3	5 M
	b)	Explain the concept of pointer arithmetic. Illustrate with an example program.	L3	CO3	5 M
		OR			
9	a)	You are developing a program to manage a library's book collection. Design a C program that uses a structure to represent book information, such as title, author, and	L4	CO4	5 M

		publication year. Implement functionalities to add and search for books in the collection. Include the code and explain how structures are used for this purpose.			
	b)	Discuss the significance of null pointers and the potential issues associated with using uninitialized pointers.	L2	CO3	5 M
		UNIT-V			
10	a)	Explain the concepts of variable scope and lifetime in a programming language and provide examples of local and global variables in C.	L2	CO3	5 M
	b)	You are designing a program to manage a library's catalog. Create a C program that defines a function to add books to the catalog. The function should take book details as parameters and append to a file.	L4	CO4	5 M
		OR			
11	a)	Define recursion. Develop a program to find factorial of a given number using recursion.	L3	CO3	5 M
	b)	Discuss the significance of file modes (e.g., "r," "w," "a") when opening and manipulating files, and provide an example for each mode.	L3	CO3	5 M

PVP 23

Code: 23ES1102

I B.Tech - I Semester – Regular Examinations - JANUARY 2024

INTRODUCTION TO PROGRAMMING

(Common for ALL BRANCHES)

Duration: 3 hours Max. Marks: 70

Note: 1. This question paper contains two Parts A and B.
2. Part-A contains 10 short answer questions. Each Question carries 2 Marks.
3. Part-B contains 5 essay questions with an internal choice from each unit. Each Question carries 10 marks.
4. All parts of Question paper must be answered in one place.

SHORT SCHEME

PART – A

1.a)	Differentiate between the top-down and bottom-up problem-solving approaches.	L2	CO1

Scheme: Any two differences – 2M

1.b)	Differentiate between algorithm and flow chart.	L2	CO1

Scheme: Any two differences – 2M

1.c)	Differentiate between a "while" loop and a "do-while" loop, and provide an example of when you would choose one over the other.	L2	CO1

Scheme: Any two differences - 1M

Example – 1M

1.d)	Explain the difference between the "if" statement and the "if-else" statement in terms of their execution.	L2	CO1

Scheme: Any two differences - 1M

Explanation – 1M

1.e)	In programming, what is a string, and how is it typically represented in memory?	L1	CO1

Scheme: Definition – 1M

Memory Model (diagram)– 1M

1.f)	What is the purpose of declaring the size of an array when you create it in a programming language like C?	L1	CO1

Scheme: Declaration with Explanation - 2M

1.g)	Differentiate between a pointer variable and a regular variable in terms of how they store data.	L2	CO1

Scheme: Any two differences – 1M

Memory model (diagram) – 1M

1.h)	Explain the role of functions like 'malloc()' and 'free()' in dynamic memory allocation in C.	L2	CO1

Scheme: Explanation malloc() – 1M

Explanation free() – 1M

1.i)	What is a function in programming, and why is it used?	L1	CO1

Scheme: Definition – 1M

Usage – 1M

1.j)	Compare call-by-value with call-by-reference.	L2	CO1

Scheme: Any two differences – 2M

PART – B

			BL	CO	Max. Marks
		UNIT-I			
2	a)	Explain various operators in C.	L2	CO1	5 M
	b)	Compare and contrast high-level programming languages and low-level programming languages. Give examples of each and discuss their respective advantages and disadvantages.	L2	CO1	5 M

Scheme: 2 a) Explanation of any 5 types of Operators - 5M

2 b) any 3 differences for each – 3M

Advantages and Disadvantages – 2M

		OR			
3	a)	Discuss the concept of data types and their importance in programming. Provide examples of situations where choosing the right data type is crucial for program efficiency.	L2	CO1	5 M
	b)	Write an algorithm and draw a flow chart to calculate the sum of first 10 natural numbers.	L2	CO1	5 M

Scheme: 3 a) Data types importance – 3M

Situation examples - 2M

3 b) Algorithm to find sum of 10 natural numbers – 2.5M

Flowchart to find sum of 10 natural numbers – 2.5M

		UNIT-II			
4	a)	Create a C program that employs a "while" loop to print all even numbers between 1 and 50, but skips any numbers that are divisible by 6 using the "continue" statement. Provide the code and a detailed explanation.	L3	CO2	5 M
	b)	Write a C program that uses a "for" loop to find the first prime number between 100 and 200. Implement the "break" statement to exit the loop once the prime number is found.	L3	CO2	5 M

Scheme: 4 a) Program using while loop, if and continue – 3+1+1 = 5M

4 b) Program using nested for loop, if and break – 3+1+1 = 5M

		OR			
5	a)	Discuss the advantages of using a "switch" statement over a series of "if" statements in certain scenarios. Provide an example to illustrate your point.	L2	CO1	5 M
	b)	Create a C program that continuously prompts the user to enter a positive integer until a negative number is entered. Calculate and display the sum of all the positive integers entered by the user. Utilize a "while" loop, conditional statements, and the "break" statement to terminate the loop when a negative number is provided.	L3	CO2	5 M

Scheme: 5 a) Advantages of switch statement – 2.5M

Demonstration with example program – 2.5M

5 b) Demonstrate of Program with while loop, if and break – 5M

		UNIT-III			
6	a)	Discuss the importance of string manipulation in programming, including tasks like comparison, concatenation, and substring extraction. Provide a code example in C that demonstrates these string operations.	L3	CO3	5 M
	b)	Explain the advantages of using a two-dimensional array over a one-dimensional array when working with tabular data or grids. Provide real-world examples where two-dimensional arrays are useful.	L2	CO2	5 M

Scheme: 6 a) Importance of string manipulation – 2M

Example program for 3 operations – 3M

6 b) Advantages of 2D arrays – 3M

Real-time Examples - 2M

		OR			
7	a)	Imagine you need to manage a list of customer names in a business application. Discuss the advantages and disadvantages of using an array for this purpose.	L3	CO2	5 M
	b)	You have an array of integers representing the daily temperatures for a week (index0: Sunday, index1: Monday and so on). Write a C program that finds and prints the day with the highest temperature and the temperature itself.	L3	CO3	5 M

Scheme: 7 a) Discussion of 2-D Character Arrays to store customer names – 2M

2-D Character array Advantages – 3M

7 b) Demonstration with example program – 5M

		UNIT-IV			
8	a)	Design a C program that reverses the elements of an integer array using pointers. Provide the code and a step-by-step explanation of the algorithm.	L3	CO3	5 M
	b)	Explain the concept of pointer arithmetic. Illustrate with an example program.	L3	CO3	5 M

Scheme: 8 a) Program to reverse array elements – 3M

Step-by-step Explanation – 2M

8 b) Explanation of pointer arithmetic – 2M

Example program – 3M

		OR			
9	a)	You are developing a program to manage a library's book collection. Design a C program that uses a structure to represent book information, such as title, author, and publication year. Implement functionalities to add and search for books in the collection. Include the code and explain how structures are used for this purpose.	L4	CO4	5 M
	b)	Discuss the significance of null pointers and the potential issues associated with using uninitialized pointers.	L2	CO3	5 M

Scheme: 9 a) Implementation library system using structure and functions – 5M

9 b) Significance of null pointer – 2.5M

Issues with wild/dangling pointer – 2.5M

		UNIT-V			
10	a)	Explain the concepts of variable scope and lifetime in a programming language and provide examples of local and global variables in C.	L2	CO3	5 M
	b)	You are designing a program to manage a library's catalog. Create a C program that defines a function to add books to the catalog. The function should take book details as parameters and append to a file. .	L4	CO4	5 M

Scheme: 10 a) Explanation of scope and life time of variables – 3M

Example of local and global variables – 2M

10 b) Implementation library system using structures, functions and files – 5M

		OR			
11	a)	Define recursion. Develop a program to find factorial of a given number using recursion.	L3	CO3	5 M
	b)	Discuss the significance of file modes (e.g., "r," "w," "a") when opening and manipulating files, and provide an example for each mode.	L3	CO3	5 M

Scheme: 11 a) Definition – 1M

Implementation of the program – 4M

11 b) Description of File opening modes – 2M

Example (Syntax) for each mode – 3M

PVP 23

Code: 23ES1102

I B.Tech - I Semester – Regular Examinations - JANUARY 2024

INTRODUCTION TO PROGRAMMING

(Common for ALL BRANCHES)

Duration: 3 hours Max. Marks: 70

Note: 1. This question paper contains two Parts A and B.
2. Part-A contains 10 short answer questions. Each Question carries 2 Marks.
3. Part-B contains 5 essay questions with an internal choice from each unit. Each Question carries 10 marks.
4. All parts of Question paper must be answered in one place.

SCHEME OF EVALUATION

PART – A

1.a)	Differentiate between the top-down and bottom-up problem-solving approaches.	L2	CO1

Ans: The main difference between the top-down and bottom-up approaches is the process's starting point and focus.

Any two differences – 2M

S.No.	Top-Down Approach	Bottom-Up Approach
1.	In this approach, the problem is broken down into smaller parts.	In this approach, the smaller problems are solved.
2.	It is generally used by structured programming languages such as C, COBOL, FORTRAN, etc.	It is generally used with object oriented programming paradigm such as C++, Java, Python, etc.
3.	It is generally used with documentation of module and debugging code.	It is generally used in testing modules.
5.	It contains redundant information.	It does not contain redundant information.
6.	Decomposition approach is used here.	Composition approach is used here.

7.	The implementation depends on the programming language and platform.	Data encapsulation and data hiding is implemented in this approach.

1.b) Differentiate between algorithm and flow chart. L2 CO1

Ans:

Any two differences – 2M

S. No	Algorithm	Flowchart
1.	An algorithm is a step-by-step procedure to solve a problem.	A flowchart is a diagram created with different shapes to show the flow of data.
2.	The algorithm is complex to understand.	A flowchart is easy to understand.
3.	In the algorithm, plain text is used.	In the flowchart, symbols/shapes are used.
4.	The algorithm is easy to debug.	A flowchart is hard to debug.
5.	The algorithm is difficult to construct.	A flowchart is simple to construct.
6.	The algorithm does not follow any rules.	The flowchart follows rules to be constructed.
7.	The algorithm is the pseudo-code for the program.	A flowchart is just a graphical representation of that logic.

1.c)	Differentiate between a "while" loop and a "do-while" loop, and provide an example of when you would choose one over the other.	L2	CO1

Ans:

Any two differences - 1M

Example – 1M

while	do-while
Condition is checked first then statement(s) is executed.	Statement(s) is executed atleast once, thereafter condition is checked.
It might occur statement(s) is executed zero times, If condition is false.	At least once the statement(s) is executed.
If there is a single statement, brackets are not required.	Brackets are always required.

while	do-while
Variable in condition is initialized before the execution of loop.	variable may be initialized before or within the loop.
while loop is entry controlled loop.	do-while loop is exit controlled loop.

while(condition)
{
statement(s);
}
	do {
	statement(s);
	}while(condition);

Choose between **while** and **do-while** based on your specific requirements. If you want to ensure the loop body is executed at least once, use **do-while**. If you want the loop to execute only if the condition is true initially, you can use **while**.

```
// Using while loop
while (count <= 5) {
    printf("While loop iteration %d\n", count);
    count++;
}
// Using do-while loop
do {
    printf("Do-while loop iteration %d\n", count);
    count++;
} while (count <= 5);
```

1.d)	Explain the difference between the "if" statement and the "if-else" statement in terms of their execution.	L2	CO1

Ans:

Any two differences - 1M

Explanation – 1M

if statement:

The if statement is a basic conditional statement that allows you to execute a block of code only if a certain condition is true. The general syntax of an if statement is as follows:

Copy code

```
if (condition) {
    // Code to be executed if the condition is true
}
```

if-else statement:

The if-else statement extends the if statement by providing an alternative block of code to be executed when the condition is false. The general syntax is as follows:

```
if (condition) {
   // Code to be executed if the condition is true
} else {
   // Code to be executed if the condition is false
}
```

Type of Decision Control Statements in C	Syntax	Description
If	The Syntax is: if (condition x) { Statements; }	The Description is: In the case of such a statement, when the available condition is true, there occurs an execution of a certain block of code.

if…else	The Syntax is:
	if (condition x)
	{ Statement x; Statement y; }
	else
	{ Statement a; Statement b; }
		The Description is:
		In the case of such a statement, when the available condition is true, then there occurs an execution of a certain group of statements. In case this available condition turns out to be false, there occurs an execution of the statement specified in the else part.

1.e)	In programming, what is a string, and how is it typically represented in memory?	L1	CO1

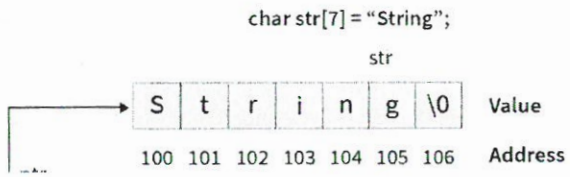
Ans:

Definition – 1M

Memory Model (diagram)– 1M

In computer programming, a string is traditionally a sequence of characters, either as a literal constant or as some kind of variable.

A string is represented by one dimensional character array in memory.



1.f)	What is the purpose of declaring the size of an array when you create it in a programming language like C?	L1	CO1

Ans:

Declaration with Explanation - 2M

int myArray[5]; // Declaring an array of integers with size 5

In programming languages like C, declaring the size of an array when it is created serves several important purposes:

Memory Allocation: When you declare an array, the compiler needs to allocate a specific amount of memory for that array.

Indexing: Arrays in C are zero-indexed, meaning the first element has an index of 0, the second has an index of 1, and so on.

Bounds Checking: While C itself doesn't perform bounds checking, specifying the size allows programmers to understand the valid range of indices for the array.

Compile-Time Validation: The size specified during array declaration is used for compile-time checks.

Performance: Knowing the size of an array at compile time allows the compiler to optimize code more effectively.

1.g)	Differentiate between a pointer variable and a regular variable in terms of how they store data.	L2	CO1

Ans:

Any two differences – 1M

Memory model (diagram) – 1M

The main difference between a pointer variable and a regular (non-pointer) variable lies in how they store and represents data.

Regular Variable: Stores the actual data it represents.

Pointer Variable: Stores the memory address of another variable.

Regular Variable: Directly holds a value of a specified data type.

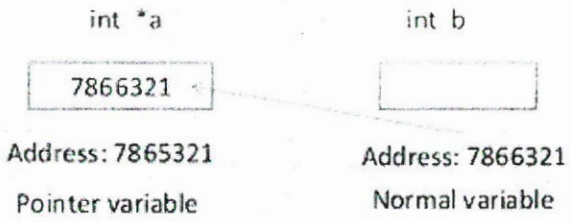
int regularVar = 42;

Pointer Variable: Holds the memory address of a variable of a specified data type.

int *pointerVar;

int anotherVar = 10;

pointerVar = &anotherVar;



1.h)	Explain the role of functions like 'malloc()' and 'free()' in dynamic memory allocation in C.	L2	CO1

Ans:

Explanation malloc() – 1M

Explanation free() – 1M

In the C programming language, malloc() and free() are functions used for dynamic memory allocation and deallocation, respectively. They play a crucial role in managing memory at runtime.

malloc() Function:

Purpose: Stands for "memory allocation." It is used to allocate a specified number of bytes of memory during the program's execution.

Syntax:

void* malloc(size_t size);

size: The number of bytes to allocate.

Returns a void pointer (void*) to the beginning of the allocated memory block.

Usage:

int *ptr = (int*)malloc(5 * sizeof(int));

free() Function:

Purpose: Used to deallocate memory that was previously allocated using malloc() or a related function.

Syntax:

void free(void *ptr);

ptr: A pointer to the memory block to be deallocated.

Usage:

int *dynamicArray = (int*)malloc(10 * sizeof(int));

// Use dynamicArray as needed

free(dynamicArray);

Deallocates the memory block previously allocated for dynamicArray.

1.i)	What is a function in programming, and why is it used?	L1	CO1

Ans:

Definition – 1M

Usage – 1M

In C programming, a function is a self-contained block of code that performs a specific task or set of tasks.

Functions provide a way to modularize code, making it more organized, readable, and reusable.

Why Functions are Used:

Modularity: Functions allow breaking down a program into smaller, manageable modules, making the code more organized and easier to understand.

Reuse: Once a function is defined, it can be called from different parts of the program, promoting code reuse and reducing redundancy.

Readability: Functions improve code readability by encapsulating complex logic into well-named and self-contained units.

Maintenance: Changes or updates can be made to individual functions without affecting the rest of the program, simplifying maintenance.

1.j)	Compare call-by-value with call-by-reference.	L2	CO1

Ans:

Ant two differences – 2M

In programming languages like C, there are two common methods for passing arguments to functions: call-by-value and call-by-reference. These methods determine how changes made to the parameters within the function affect the original values passed from the calling function.

Call By Value	Call By Reference
While calling a function, we pass the values of variables to it. Such functions are known as "Call By Values".	While calling a function, instead of passing the values of variables, we pass the address of variables (location of variables) to the function known as "Call By References.
In this method, the value of each variable in the calling function is copied into corresponding dummy variables of the called function.	In this method, the address of actual variables in the calling function is copied into the dummy variables of the called function.
With this method, the changes made to the dummy variables in the called function have no effect on the values of actual variables in the calling function.	With this method, using addresses we would have access to the actual variables and hence we would be able to manipulate them.

Call By Value	Call By Reference
In call-by-values, we cannot alter the values of actual variables through function calls.	In call by reference, we can alter the values of variables through function calls.
Values of variables are passed by the Simple technique.	Pointer variables are necessary to define to store the address values of variables.
This method is preferred when we have to pass some small values that should not change.	This method is preferred when we have to pass a large amount of data to the function.
Call by value is considered safer as original data is preserved	Call by reference is risky as it allows direct modification in original data

		PART – B	BL	CO	Max. Marks
		UNIT-I			
2	a)	Explain various operators in C.	L2	CO1	5 M
	b)	Compare and contrast high-level programming languages and low-level programming languages. Give examples of each and discuss their respective advantages and disadvantages.	L2	CO1	5 M

Ans:

2 a) Explanation of any 5 types of Operators - 5M

In the C programming language, operators are symbols used to perform operations on variables and values.

An arithmetic operator performs mathematical operations such as addition, subtraction, multiplication, division etc on numerical values (constants and variables).

Operator	Meaning of Operator	Example
+	addition or unary plus	5 + 3 is evaluated to 8
-	subtraction or unary minus	5 - 3 is evaluated to 2
*	multiplication	5 * 3 is evaluated to 15
/	division	5 / 3 is evaluated to 1.0
%	remainder after division (modulo division)	5 % 3 is evaluated to 2

C Increment and Decrement Operators

C programming has two operators increment ++ and decrement -- to change the value of an operand (constant or variable) by 1.

Increment ++ increases the value by 1 whereas decrement -- decreases the value by 1. These two operators are unary operators, meaning they only operate on a single operand.

C Assignment Operators

An assignment operator is used for assigning a value to a variable. The most common assignment operator is =

Operator	Example	Same as
=	a = b	a = b
+=	a += b	a = a+b
-=	a -= b	a = a-b
*=	a *= b	a = a*b
/=	a /= b	a = a/b
%=	a %= b	a = a%b

Relational Operators:

A relational operator checks the relationship between two operands. If the relation is true, it returns 1; if the relation is false, it returns value 0.

Relational operators are used in decision making and loops.

Operator	Meaning of Operator	Example
==	Equal to	5 == 3 is evaluated to 0
>	Greater than	5 > 3 is evaluated to 1
<	Less than	5 < 3 is evaluated to 0
!=	Not equal to	5 != 3 is evaluated to 1
>=	Greater than or equal to	5 >= 3 is evaluated to 1
<=	Less than or equal to	5 <= 3 is evaluated to 0

C Logical Operators

An expression containing logical operator returns either 0 or 1 depending upon whether expression results true or false. Logical operators are commonly used in decision making in C programming.

Operator	Meaning	Example
&&	Logical AND. True only if all operands are true	If c = 5 and d = 2 then, expression ((c==5) && (d>5)) equals to 0.
||	Logical OR. True only if either one operand is true	If c = 5 and d = 2 then, expression ((c==5) || (d>5)) equals to 1.
!	Logical NOT. True only if the operand is 0	If c = 5 then, expression !(c==5) equals to 0.

C Bitwise Operators

During computation, mathematical operations like: addition, subtraction, multiplication, division, etc are converted to bit-level which makes processing faster and saves power.

Bitwise operators are used in C programming to perform bit-level operations.

Operators	Meaning of operators
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement
<<	Shift left
>>	Shift right

Other Operators

Comma Operator

Comma operators are used to link related expressions together. For example:

int a, c = 5, d;

The sizeof operator

The sizeof is a unary operator that returns the size of data (constants, variables, array, structure, etc).

2 b) any 3 differences for each – 3M

Advantages and Disadvantages – 2M

Both of these are types of programming languages that provide a set of instructions to a system for performing certain tasks. Though both of these have specific purposes.

	It is programmer friendly language.	It is a machine friendly language.
1.		
2.	High level language is less memory efficient.	Low level language is high memory efficient.
3.	It is easy to understand.	It is tough to understand.
4.	Debugging is easy.	Debugging is complex comparatively.
5.	It is simple to maintain.	It is complex to maintain comparatively.
6.	It is portable.	It is non-portable.
7.	It can run on any platform.	It is machine-dependent.
8.	It needs compiler or interpreter for translation.	It needs assembler for translation.
9.	It is used widely for programming.	It is not commonly used now-a-days in programming.

Advantages and Disadvantages

Advantages of Low-Level Language

- They are better at performance compared to high-level languages as they provide direct control over the computer's hardware.
- They are better at memory management.
- Debugging is comparatively easy in low-level language.

Disadvantages of Low-Level Languages

- The programmers must know deeply about computer hardware.
- They are sometimes time-consuming because we need to manage the memory and complexity of the instructions.
- They are comparatively less portable than high-level languages.

Advantages of High-Level Language

- High-level languages provide a higher level of abstraction, allowing programmers to focus on the logic and functionality of their programs rather than the intricate details of hardware or low-level operations.
- Code written in high-level languages is often more readable and understandable.
- High-level languages offer built-in functions, libraries, and frameworks.

Disadvantages of High-Level Language

- High-level languages are generally slower than low-level languages in view of execution.
- High-level languages require more memory than low-level languages.

		OR			
3	a)	Discuss the concept of data types and their importance in programming. Provide examples of situations where choosing the right data type is crucial for program efficiency.	L2	CO1	5 M
	b)	Write an algorithm and draw a flow chart to calculate the sum of first 10 natural numbers.	L2	CO1	5 M

3 a) Data types importance – 3M

Situation examples - 2M

Ans:

In programming, a data type is a classification that specifies which type of value a variable can hold and what operations can be performed on that value. Data types define the characteristics of data and provide a way for the computer to interpret, store, and manipulate that data.

Common data types include:

Integer:

Represents whole numbers without a fractional part.

Float/Double:

Represents real numbers with a fractional part.

Character:

Represents a single character, like a letter or a digit.

String:

Represents a sequence of characters.

Array:

Represents a collection of elements of the same data type.

Pointer:

Represents a memory address, used for low-level memory manipulation.

User-defined:

Represents data types created by the programmer.

Choosing the right data types is crucial for program efficiency in various situations, especially when it comes to memory usage, computational speed, and overall performance.

Scenario: Performing complex numerical calculations, such as simulations or scientific computations.

Example: Using a float or double data type for floating-point arithmetic operations rather than a less precise data type like int to maintain accuracy.

Scenario: Implementing collection of homogeneous data elements.

Example: Choosing an appropriate data type for elements in a collection, such as using a specific array instead of multiple variables can impact memory usage and retrieval times.

Scenario: Implementing collection of heterogeneous data elements.

Example: Choosing an appropriate data type for elements in a collection, such as using structure or union instead of multiple variables of different data types can impact memory usage and retrieval times.

Scenario: Reading from or writing to files.

Example: Choosing appropriate data types for reading and writing data from/to files, such as using binary formats for efficiency, can impact file I/O performance.

3 b) Algorithm to find sum of 10 natural numbers – 2.5M

Flowchart to find sum of 10 natural numbers – 2.5M

To find the sum of the first 10 natural numbers, you can use a simple algorithm known as the arithmetic series formula. The sum of the first

n natural numbers is given by the formula:

(n*(n-1))/2

Algorithm: Sum Of First 10 Natural Numbers

Step 1: Start

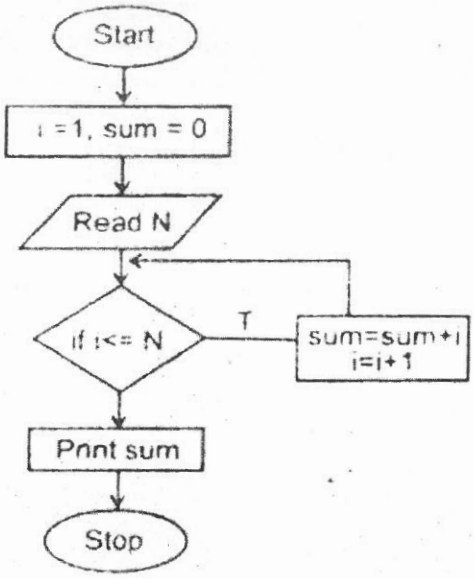
Step 2: Set n = 10

Step 3: Calculate S = (n * (n + 1)) / 2

Step 4: Display S

Step 5: Stop

Flowchart: Provided that N as 10



		UNIT-II			
4	a)	Create a C program that employs a "while" loop to print all even numbers between 1 and 50, but skips any numbers that are divisible by 6 using the "continue" statement. Provide the code and a detailed explanation.	L3	CO2	5 M
	b)	Write a C program that uses a "for" loop to find the first prime number between 100 and 200. Implement the "break" statement to exit the loop once the prime number is found.	L3	CO2	5 M

Ans: 4 a) Program using while loop, if and continue – 3+1+1 = 5M

```
//C Program to print all even numbers between 1 and 50 but not divisible by 6
#include <stdio.h>
int main() {
   int number = 0;  // Start with the zero
   while (number <= 50) {
      number += 1;  // Move to the next number
      if (number % 2 == 0){
         if (number % 6 == 0)
            continue;
         printf("%d\t", number);
      }
   }
```

```
    return 0;
}
```

Output:

2 4 8 10 14 16 20 22 26 28 32 34 38 40 44 46 50

Explanation:

In this C Program:

We start with number initialized to 0 (the first even number).

The while loop continues as long as number is less than or equal to 50.

Inside the loop, an if statement checks if the number is even (number % 2 == 0) and again in nested if we check the divisible by 6 (number % 6 == 0).

If both conditions are true, the number is not printed.

Otherwise the number will be printed using printf.

The loop then increments number by 1 in each iteration to move to the next number.

This code will print all even numbers between 1 and 50 (excluding multiples of 6) in C.

4 b) Program using nested for loop, if and break – 3+1+1 = 5M

Ans:

```
#include <stdio.h>
int main() {
    int start = 100;
    int end = 200;
    // Check each number in the range
    for (int num = start; num <= end; num++) {
        int isPrime = 1;
        // Check if the current number is prime
        for (int i = 2; i <= num / 2; i++) {
            if (num % i == 0) {
                isPrime = 0;
                break;  // No need to continue checking if it's not prime
            }
```

```
        }
        // If the number is prime, print it and break the loop
        if (isPrime == 1) {
            printf("The first prime number between %d and %d is: %d\n", start, end, num);
            break;  // Found the first prime, no need to continue
        }
    }
    return 0;
}
```

Output:

The first prime number between 100 and 200 is: 101

		OR			
5	a)	Discuss the advantages of using a "switch" statement over a series of "if" statements in certain scenarios. Provide an example to illustrate your point.	L2	CO1	5 M
	b)	Create a C program that continuously prompts the user to enter a positive integer until a negative number is entered. Calculate and display the sum of all the positive integers entered by the user. Utilize a "while" loop, conditional statements, and the "break" statement to terminate the loop when a negative number is provided.	L3	CO2	5 M

5 a) Advantages of switch statement – 2.5M

Demonstration with example program – 2.5M

Ans:

Using a switch statement over continuous if statements in C has several advantages, especially when dealing with multiple conditional branches. Here are some of the key advantages:

Readability: Switch statements can make your code more readable and concise, especially when dealing with multiple conditions. It provides a clear structure and is easier to understand than a series of nested if statements.

Efficiency: In some cases, a switch statement can be more efficient than a series of if statements. The compiler can optimize the switch statement, making it faster to execute.

Code Maintainability: Switch statements can be easier to maintain, especially when new cases need to be added. Modifying a switch statement typically involves adding or removing case labels, which is more straightforward than modifying a series of if statements.

Switch Fall-Through: Switch statements allow for fall-through behavior, where multiple case labels can share the same code block. This can be useful in certain scenarios where you want to execute the same code for multiple cases without duplicating it.

```
// Using switch statement
switch (variable) {
   case 1:
      // Code for case 1
      break;
   case 2:
      // Code for case 2
      break;
   default:
      // Code for default case
}

// Equivalent using if statements
if (variable == 1) {
   // Code for case 1
} else if (variable == 2) {
   // Code for case 2
} else {
   // Code for default case
}
#include <stdio.h>
int main() {
   int option = 2;
   switch (option) {
      case 1:
         printf("Option 1 selected.\n");
         break;
```

```
        case 2:
            printf("Option 2 selected.\n");
            break;
        default:
            printf("Invalid option.\n");
    }
    return 0;
}
```

Output:

Option 2 selected.

5 b) Demonstrate of Program with while loop, if and break – 5M

Ans:

```
#include <stdio.h>
int main() {
    int Sum = 0;
    int n;
    while(1)
    {
        printf("Enter any number:  ");
        scanf("%d", &n);
        if (n>=0)
            Sum += n;
        else
            break;
    }
    printf("The Sum is: %d", Sum);
    return 0;
}
```

Output:

Enter any number: 5

Enter any number: 2

Enter any number: 3

Enter any number: -5

The Sum is: 10

		UNIT-III			
6	a)	Discuss the importance of string manipulation in programming, including tasks like comparison, concatenation, and substring extraction. Provide a code example in C that demonstrates these string operations.	L3	CO3	5 M
	b)	Explain the advantages of using a two-dimensional array over a one-dimensional array when working with tabular data or grids. Provide real-world examples where two-dimensional arrays are useful.	L2	CO2	5 M

6 a) Ans:

String manipulation is a crucial aspect of programming with significant importance in various domains. Strings play a vital role in any programming language. Properly understanding string manipulation techniques can help developers easily handle tricky situations.

C strcmp()

The strcmp() compares two strings character by character. If the strings are equal, the function returns 0.

C strcmp() Prototype

The function prototype of strcmp() is:

int strcmp (const char* str1, const char* str2);

strcmp() Parameters

The function takes two parameters:

- str1 - a string
- str2 - a string

Return Value Remarks

0 if strings are equal

>0 if the first non-matching character in str1 is greater (in ASCII) than that of str2.

<0 if the first non-matching character in str1 is lower (in ASCII) than that of str2.

The strcmp() function is defined in the string.h header file.

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

```
  char str1[] = "abcd", str2[] = "abCd", str3[] = "abcd";
  int result;
  // comparing strings str1 and str2
  result = strcmp(str1, str2);
  printf("strcmp(str1, str2) = %d\n", result);
  // comparing strings str1 and str3
  result = strcmp(str1, str3);
  printf("strcmp(str1, str3) = %d\n", result);
  return 0;
}
```

Output

```
strcmp(str1, str2) = 1
strcmp(str1, str3) = 0
```

In the program,

- strings str1 and str2 are not equal. Hence, the result is a non-zero integer.
- strings str1 and str3 are equal. Hence, the result is 0.

C strcat()

The function definition of strcat() is:

```
char *strcat(char *destination, const char *source)
```

It is defined in the string.h header file.

strcat() arguments

As you can see, the strcat() function takes two arguments:

destination - destination string

source - source string

The strcat() function concatenates the destination string and the source string, and the result is stored in the destination string.

Example: C strcat() function

```
#include <stdio.h>
#include <string.h>
int main() {
  char str1[100] = "This is ", str2[] = "programiz.com";
  // concatenates str1 and str2
  // the resultant string is stored in str1.
```

```
  strcat(str1, str2);
  puts(str1);
  puts(str2);
  return 0;
}
```

Output

This is programiz.com

programiz.com

Note: When we use strcat(), the size of the destination string should be large enough to store the resultant string. If not, we will get the segmentation fault error.

strncpy()

The C library function char *strncpy(char *dest, const char *src, size_t n) copies up to n characters from the string pointed to, by src to dest. In a case where the length of src is less than that of n, the remainder of dest will be padded with null bytes.

Declaration

Following is the declaration for strncpy() function.

char *strncpy(char *dest, const char *src, size_t n)

Parameters

dest − This is the pointer to the destination array where the content is to be copied.

src − This is the string to be copied.

n − The number of characters to be copied from source.

Return Value

This function returns the pointer to the copied string.

Example

The following example shows the usage of strncpy() function. Here we have used function memset() to clear the memory location.

```
#include <stdio.h>
#include <string.h>
int main () {
  char src[40];
  char dest[12];
  memset(dest, '\0', sizeof(dest));
  strcpy(src, "This is tutorialspoint.com");
  strncpy(dest, src, 10);
```

```
    printf("Final copied string : %s\n", dest);
    return(0);
}
```

Output:

Final copied string: This is tu

6 b) Advantages of 2D arrays – 3M

Real-time Examples - 2M

Ans:

Two-dimensional arrays have several advantages over one-dimensional arrays in certain situations. Here are some of the key advantages:

Syntax:

data_type array_name[rows][columns];

Matrix Representation:

Two-dimensional arrays provide a natural and convenient way to represent matrices. In applications like graphics, image processing, and mathematical computations, matrices.

Tabular Data:

When dealing with tabular data, such as spreadsheets or databases, a two-dimensional array is often more suitable. Each row can represent a record, and each column can represent a different attribute or field.

Ease of Access: Accessing elements in a two-dimensional array is often more straightforward and readable, especially when dealing with data organized in rows and columns.

Simplified Code for Grids and Game Boards:

For applications involving grids, game boards, or maps, a two-dimensional array provides a natural representation.

Spatial Relationships:

Two-dimensional arrays are beneficial when dealing with spatial relationships or coordinates. For example, in graphics programming, each element in a 2D array could represent a pixel on the screen with x and y coordinates, simplifying operations like drawing shapes or images.

Real-world Examples:

Matrices in Mathematics

Sudoku Solvers

Crossword Puzzles and Word Grids

Image Processing

Graphics Programming

Spreadsheet Applications

Game Development

Database Systems

Geographic Information Systems (GIS)

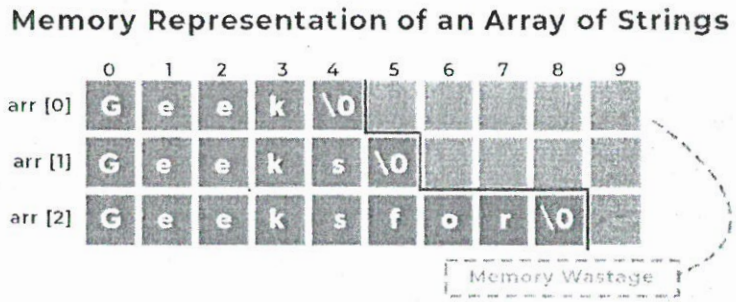
		OR			
7	a)	Imagine you need to manage a list of customer names in a business application. Discuss the advantages and disadvantages of using an array for this purpose.	L3	CO2	5 M
	b)	You have an array of integers representing the daily temperatures for a week (index0: Sunday, index1: Monday and so on). Write a C program that finds and prints the day with the highest temperature and the temperature itself.	L3	CO3	5 M

7 a) Ans:

When maintaining a list of customers in a business application, the choice of data structure, such as an array, comes with its own set of advantages and disadvantages.

Two-dimensional character arrays are best suitable for this application.

char variable_name[rows][columns] = {list of string separated by comma};



Advantages of using an array:

Sequential Access:

Arrays provide sequential access to elements, making it easy to iterate through the list of customers. This is beneficial when you need to perform operations that involve processing each customer in a specific order.

Simplicity and Efficiency:

Arrays are simple and efficient data structures for storing a fixed-size list of elements. They offer constant-time access to any element using the index, making retrieval and modification operations straightforward.

Memory Contiguity:

Elements in an array are stored in contiguous memory locations. This can lead to better cache locality, which may result in improved performance for certain operations compared to more scattered data structures.

Index-Based Access:

Array elements can be accessed directly using indices. This makes it easy to locate and manipulate specific customers based on their position in the array, facilitating quick retrieval and modification.

Fixed Size:

If the number of customers is known to be fixed or can be determined in advance, using an array allows you to allocate a fixed amount of memory. This can be advantageous in terms of memory management and resource allocation.

Disadvantages of using an array:

Fixed Size Limitation:

Arrays have a fixed size, and if the number of customers exceeds the allocated size, resizing the array becomes necessary. This process can be inefficient, especially if it involves copying elements to a larger array.

Inefficient Insertions and Deletions:

Inserting or deleting customers in the middle of the array can be inefficient, as it requires shifting elements to accommodate the change. This operation has a time complexity of O(n), where n is the number of elements in the array.

Wasted Memory:

If the array is allocated with a size larger than the actual number of customers, memory may be wasted. This is especially true if the size is chosen to accommodate potential future growth that may not occur.

Sparse Data Representation:

If there are gaps or empty slots in the array due to deletions, the array may not efficiently represent the actual number of customers. This can lead to inefficient memory usage.

Limited Dynamic Behavior:

Arrays do not dynamically resize themselves, and managing dynamic behavior, such as accommodating fluctuating numbers of customers, requires additional logic and potentially more complex data structures.

7 b) Demonstration with example program – 5M

Ans:

```
#include <stdio.h>

int main() {

  int temperatures[7] = {32, 28, 35, 30, 33, 29, 31};  // Replace with your actual temperatures

  int maxTemperature = temperatures[0]; // Initialize with the temperature of the first day

  int dayWithMaxTemperature = 0; // Initialize with the index of the first day

  for (int i = 1; i < 7; ++i) {

    if (temperatures[i] > maxTemperature) {
```

```
            maxTemperature = temperatures[i];
            dayWithMaxTemperature = i;
        }
    }
    // Adding 1 to the index to get the actual day (assuming days are numbered from 1 to 7)
    printf("The day of the week with the highest temperature is day %d.\n", dayWithMaxTemperature + 1);
    return 0;
}
```

Output:

The day of the week with the highest temperature is day 3.

		UNIT-IV			
8	a)	Design a C program that reverses the elements of an integer array using pointers. Provide the code and a step-by-step explanation of the algorithm.	L3	CO3	5 M
	b)	Explain the concept of pointer arithmetic. Illustrate with an example program.	L3	CO3	5 M

8 a) Program to reverse array elements – 3M

Step-by-step Explanation – 2M

Ans)

```
// C Program to reverse an array using pointers
#include <stdio.h>
int main() {
    // Initialize the array
    int arr[] = {1, 2, 3, 4, 5};
    int *start, *end, temp;
    // Calculate the size of the array
    int length = sizeof(arr)/sizeof(arr[0]);
    printf("Original Array: ");
    for(int i = 0; i < length; i++){
        printf("%d ", arr[i]);
    }
    start = arr;  // Points to first element of array
    end = arr + (length - 1);  // Points to last element of array
```

```
    // Reverse the array
    while(start < end){
        // Swap items stored at *start and *end
        temp = *start;
        *start = *end;
        *end = temp;
        start++;  // Move to next address
        end--;    // Move to previous address
    }
    printf("\nReversed Array: ");
    for(int i = 0; i < length; i++){
        printf("%d ", arr[i]);
    }
    return 0;
}
```

Output:

Original Array: 1 2 3 4 5

Reversed Array: 5 4 3 2 1

To reverse an array using pointers in C, we can use the following algorithm:

1. Initialize two pointers *start and *end of the same data type as of the array.
2. Initially, assign the array itself to the pointer *start so that it can point to the first element of the array.
3. Add the (size of the array – 1) to the array and assign it to the pointer *end so that it can point to the last element of the array.
4. Run a while loop until the pointers *start and *end point to the same address.
5. Inside the while loop, swap the elements pointed to by *start and *end.
6. Increment *start by 1 so that it can point to the next element of the array and decrement *end by 1 so that it can point to the previous element of the array in each iteration of the while loop.

8 b) Explanation of pointer arithmetic – 2M

Example program – 3M

Ans:

Pointer Arithmetic is the set of valid arithmetic operations that can be performed on pointers. The pointer variables store the memory address of another variable. It doesn't store any value.

Hence, there are only a few operations that are allowed to perform on Pointers in C language. The C pointer arithmetic operations are slightly different from the ones that we generally use for mathematical calculations. These operations are:

Increment/Decrement of a Pointer

Addition of integer to a pointer

Subtraction of integer to a pointer

Subtracting two pointers of the same type

Comparison of pointers

```
#include <stdio.h>

// pointer increment and decrement

//pointers are incremented and decremented by the size of the data type they point to

int main()

{

                    int a = 22;

                    int *p = &a;

                    printf("p = %u\n", p); // p = 6422288

                    p++;

                    printf("p++ = %u\n", p); //p++ = 6422292 +4 // 4 bytes

                    p--;

                    printf("p-- = %u\n", p); //p-- = 6422288  -4 // restored to original
value

  p = p + 3;

  printf("Pointer p after Addition: ");

  printf("%u \n", p);

  p = p - 3;

  printf("Pointer p after Subtraction: ");

  printf("%u \n", p);

  int *q, k = 5;

  int x = p - q;

  printf("Subtraction of p & q is %u\n", x);

  if (p == q) {

    printf("Pointers are Equal.");

  }
```

```
    else {
        printf("Pointers are not Equal.");
    }
                        return 0;
}
```

Output:

p = 3181500672

p++ = 3181500676

p-- = 3181500672

Pointer p after Addition: 3181500684

Pointer p after Subtraction: 3181500672

Subtraction of p & q is 4015551024

Pointers are not Equal.

		OR			
9	a)	You are developing a program to manage a library's book collection. Design a C program that uses a structure to represent book information, such as title, author, and publication year. Implement functionalities to add and search for books in the collection. Include the code and explain how structures are used for this purpose.	L4	CO4	5 M
	b)	Discuss the significance of null pointers and the potential issues associated with using uninitialized pointers.	L2	CO3	5 M

Ans: 9 a) Implementation library system using structure and functions – 5M

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

// Structure to represent book information
struct Book {
    char title[50];
    char author[50];
    int publicationYear;
```

```
};

// Function to add a book to the collection
void addBook(struct Book collection[], int *numBooks) {
    if (*numBooks < 100) {
        printf("Enter book title: ");
        scanf(" %[^\n]", collection[*numBooks].title);

        printf("Enter author name: ");
        scanf(" %[^\n]", collection[*numBooks].author);

        printf("Enter publication year: ");
        scanf("%d", &collection[*numBooks].publicationYear);

        (*numBooks)++;
        printf("Book added successfully!\n");
    } else {
        printf("Error: Collection is full. Cannot add more books.\n");
    }
}

// Function to search for a book by title
void searchBook(struct Book collection[], int numBooks, const char *searchTitle) {
    int found = 0;

    for (int i = 0; i < numBooks; ++i) {
        if (strcmp(collection[i].title, searchTitle) == 0) {
            printf("Book Found!\n");
            printf("Title: %s\n", collection[i].title);
            printf("Author: %s\n", collection[i].author);
            printf("Publication Year: %d\n", collection[i].publicationYear);
            found = 1;
            break;
```

```
        }
    }

    if (!found) {
        printf("Book not found in the collection.\n");
    }
}

int main() {
    struct Book bookCollection[100]; // Array to store books
    int numBooks = 0; // Number of books in the collection

    int choice;
    char searchTitle[50];

    do {
        // Display menu
        printf("\nMenu:\n");
        printf("1. Add a book to the collection\n");
        printf("2. Search for a book by title\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                addBook(bookCollection, &numBooks);
                break;
            case 2:
                printf("Enter the title of the book to search: ");
                scanf(" %[^\n]", searchTitle);
                searchBook(bookCollection, numBooks, searchTitle);
                break;
```

```
            case 3:
                printf("Exiting the program. Goodbye!\n");
                break;
            default:
                printf("Invalid choice. Please enter a valid option.\n");
        }
    } while (choice != 3);

    return 0;
```

Output:

Menu:

1. Add a book to the collection

2. Search for a book by title

3. Exit

Enter your choice: 1

Enter book title: Introduction to C Programming

Enter author name: Reema Thareja

Enter publication year: 2023

Book added successfully!

Menu:

1. Add a book to the collection

2. Search for a book by title

3. Exit

Enter your choice: 2

Enter the title of the book to search: Introduction to C Programming

Book Found!

Title: Introduction to C Programming

Author: Reema Thareja

Publication Year: 2023

9 b) Significance of null pointer – 2.5M

Issues with wild/dangling pointer – 2.5M

Ans:

Null Pointer in C:

A null pointer in C is a pointer that does not point to any memory location. It is represented by the constant NULL and is typically used to indicate that the pointer is intentionally not pointing to a valid memory location.

Syntax: type pointer_name = NULL;

The significance of null pointers includes:

Initialization and Indication of Absence:

Null pointers are commonly used to initialize pointers or indicate that a pointer does not currently point to any valid memory address. This is especially useful when the pointer's target is not known or not applicable.

Error Handling:

Functions that return pointers often use null pointers to indicate errors or exceptional conditions. For example, if a function fails to allocate memory, it might return a null pointer to signify the failure.

Pointer Comparison:

Null pointers are often used in comparisons to check whether a pointer points to a valid memory location. This is particularly useful for avoiding dereferencing invalid pointers and causing undefined behavior.

Potential Issues with Uninitialized Pointers:

Syntax/Example:

```
int *p;

*p = 12;
```

Undefined Behavior:

Using an uninitialized pointer (a pointer that has not been assigned a valid memory address) leads to undefined behavior. Dereferencing such a pointer can result in unpredictable consequences, including crashes, data corruption, or security vulnerabilities.

Memory Access Violations:

Dereferencing an uninitialized or null pointer can lead to memory access violations. This occurs when the program attempts to read or modify memory at an address that is not allocated or accessible.

Hard-to-Detect Bugs:

Bugs arising from uninitialized pointers can be challenging to detect and debug. The program's behavior might seem normal until it encounters a situation where the uninitialized pointer is accessed, causing unexpected issues.

Security Risks:

Uninitialized pointers can be exploited by attackers to manipulate the program's behavior, leading to security vulnerabilities. For example, they might use uninitialized pointers to overwrite memory, execute arbitrary code, or gain unauthorized access.

Resource Leaks:

If an uninitialized pointer is used to store the address of dynamically allocated memory, there is a risk of resource leaks. Without proper initialization or deallocation, the program may lose references to allocated memory, leading to memory leaks.

		UNIT-V			
10	a)	Explain the concepts of variable scope and lifetime in a programming language and provide examples of local and global variables in C.	L2	CO3	5 M
	b)	You are designing a program to manage a library's catalog. Create a C program that defines a function to add books to the catalog. The function should take book details as parameters and append to a file.	L4	CO4	5 M

10 a) Explanation of scope and life time of variables – 3M

Example of local and global variables – 2M

Ans:

In C, the concepts of scope and lifetime define when and where a variable is accessible and how long it exists during the execution of a program.

Scope of Variables:

Scope refers to the region of the program where a variable is visible and can be accessed. In C, there are three primary types of scope:

Block Scope (Local Scope):

Variables declared within a block of code, such as inside a function, have block scope.

They are only accessible within the block where they are declared and are not visible outside that block.

Block-scoped variables are typically used for temporary storage or as loop counters.

```
void exampleFunction() {
    int localVar = 10;  // Block-scoped variable
    // localVar is visible and usable only within this function
}
```

Function Scope:

In C, variables declared outside of any function (at the file level) have function scope.

They are accessible throughout the file after their declaration.

Function-scoped variables are often used as global variables.

```
// Function-scoped variable
int globalVar = 20;
void exampleFunction() {
  // globalVar is accessible within this function
}
```

File Scope (Global Scope):

Variables declared using the static keyword outside of any function have file scope.

They are accessible throughout the entire file in which they are declared but are not visible outside that file.

```
// File-scoped variable
static int fileVar = 30;
void exampleFunction() {
  // fileVar is accessible within this function
}
```

Lifetime of Variables:

Lifetime refers to the period during which a variable exists in memory, from its creation to its destruction. In C, there are four primary types of variable lifetime:

Automatic (Local) Variables:

Variables declared within a block without the static keyword have automatic storage duration.

They are created when the block is entered and destroyed when the block is exited.

Example:

```
void exampleFunction() {
  int localVar = 10;  // Automatic variable
  // localVar exists while this function is executing
}
```

Static Variables:

Variables declared with the static keyword have static storage duration.

They are created before the program starts and persist throughout the program's execution.

Example:

```
void exampleFunction() {
  static int staticVar = 20;  // Static variable
  // staticVar exists throughout the program's execution
```

```
}
#include <stdio.h>
// Global variable
int globalVar = 10;
// Function using both local and global variables
void exampleFunction() {
   // Local variable within the function
   int localVar = 5;
   // Accessing and modifying local and global variables
   printf("Local variable: %d\n", localVar);
   printf("Global variable: %d\n", globalVar);
   localVar += 2;
   globalVar += 5;
   printf("Modified local variable: %d\n", localVar);
   printf("Modified global variable: %d\n", globalVar);
}

int main() {
   // Accessing and modifying global variable from main function
   printf("Global variable in main: %d\n", globalVar);
   globalVar += 3;
   printf("Modified global variable in main: %d\n", globalVar);
   // Calling the function
   exampleFunction();

   return 0;
}
```

Output:

```
Global variable in main: 10
Modified global variable in main: 13
Local variable: 5
Global variable: 13
Modified local variable: 7
```

- Modified global variable: 18

10 b) Implementation library system using structures, functions and files – 5M

Ans:

```
#include <stdio.h>
// Structure definition for a book
struct Book {
    char title[100];
    char author[100];
    int year;
};
// Function to add a book to the file
void addBookToFile(struct Book book, const char *filename) {
    FILE *file = fopen("book_catalog.bin", "ab");  // Open the file in binary append mode
    if (file == NULL) {
        printf("Error opening file %s\n", filename);
        return;
    }
    // Write the book structure to the file
    fwrite(&book, sizeof(struct Book), 1, file);
    // Close the file
    fclose(file);
}
int main() {
    struct Book newBook;
    // Get input for the new book
    printf("Enter book title: ");
    scanf("%[^\n]",  newBook.title);
    printf("Enter author name: ");
    scanf("%[^\n]", newBook.author);
    printf("Enter publication year: ");
    scanf("%d", &newBook.year);
    // Add the book to the file
```

```
    addBookToFile(newBook, "books.dat");
    printf("Book added successfully!\n");
    return 0;
}
```

Output:

Enter book title: Introduction to C Programming

Enter author name: Reema Thareja

Enter publication year: 2023

Book added successfully!

		OR			
11	a)	Define recursion. Develop a program to find factorial of a given number using recursion.	L3	CO3	5 M
	b)	Discuss the significance of file modes (e.g., "r," "w," "a") when opening and manipulating files, and provide an example for each mode.	L3	CO3	5 M

11 a) Definition – 1M

Implementation of the program – 4M

Ans) Recursion is a programming concept where a function calls itself directly or indirectly to solve a problem. In recursive programs, a problem is divided into smaller subproblems, and each subproblem is solved using the same approach. Recursive functions have two main components: a base case and a recursive case.

```
#include <stdio.h>
// Function to calculate the factorial of a number using recursion
unsigned long long factorial(int n) {
    // Base case: factorial of 0 is 1
    if (n == 0) {
        return 1;
    } else {
        // Recursive case: factorial(n) = n * factorial(n-1)
        return n * factorial(n - 1);
    }
}
int main() {
    int number;
    // Input from the user
```

```
    printf("Enter a non-negative integer: ");
    scanf("%d", &number);

    // Check if the number is non-negative
    if (number < 0) {
        printf("Factorial is undefined for negative numbers.\n");
    } else {
        // Call the factorial function and display the result
        unsigned long long result = factorial(number);
        printf("Factorial of %d = %llu\n", number, result);
    }
    return 0;
}
```

Output:

Enter a non-negative integer: 6

Factorial of 6 = 720

11 b) Description of File opening modes – 2M

Example (Syntax) for each mode – 3M

Ans:

A File is a collection of data stored in the secondary memory. A file represents a sequence of bytes, regardless of it being a text file or a binary file.

File opening modes:

"r" (Read mode):

Opens the file for reading.

The file must exist; otherwise, the fopen function will return NULL.

The file pointer is positioned at the beginning of the file.

```
FILE *file = fopen("example.txt", "r");
```

"w" (Write mode):

Opens the file for writing.

If the file already exists, its contents are truncated (deleted).

If the file does not exist, a new file is created.

The file pointer is positioned at the beginning of the file.

```
FILE *file = fopen("example.txt", "w");
```

"a" (Append mode):

Opens the file for writing, but if the file exists, the file pointer is positioned at the end of the file.

If the file does not exist, a new file is created.

Existing content in the file is not truncated.

FILE *file = fopen("example.txt", "a");

"rb", "wb", "ab" (Binary modes):

These are similar to "r", "w", and "a" modes, respectively, but they open the file in binary mode.

Binary mode is used when working with binary data, and it ensures that the data is read or written as is without any newline character conversions.

FILE *file = fopen("example.bin", "rb");

FILE *file = fopen("example.bin", "wb");

FILE *file = fopen("example.bin", "ab");

"r+" (Read and Write mode):

Opens the file for both reading and writing.

The file must exist.

The file pointer is positioned at the beginning of the file.

FILE *file = fopen("example.txt", "r+");

"w+" (Read and Write mode):

Opens the file for both reading and writing.

If the file exists, its contents are truncated.

If the file does not exist, a new file is created.

The file pointer is positioned at the beginning of the file.

FILE *file = fopen("example.txt", "w+");

"a+" (Read and Append mode):

Opens the file for both reading and appending.

If the file exists, the file pointer is positioned at the end of the file.

If the file does not exist, a new file is created.

FILE *file = fopen("example.txt", "a+");

It's important to note that these modes can be combined, for example, "rb+", "w+", "a+", etc., to achieve different combinations of read and write operations. Additionally, always check whether the file was successfully opened by checking if the file pointer is not NULL after the **fopen** call.