1.What is an array? Explain declaration and initialization of one-dimensional array

An array in C is a collection of elements of the same data type stored in contiguous memory locations. Each element of an array is accessed using a common name and an index
(subscript). Arrays are useful when we need to store and process a large number of similar data items efficiently.

One-Dimensional Array

A one-dimensional array is a linear list of elements. The elements are stored one after another and are accessed using a single index.

Declaration of One-Dimensional Array

Declaration of an array tells the compiler:

- the data type of elements,
- the name of the array, and
- the number of elements it can store.

Syntax:

data_type array_name[size];

Example:

int marks[5];

This declares an array named marks that can store 5 integer values.

Initialization of One-Dimensional Array

Initialization means assigning values to the array elements. It can be done at the time of declaration or

later in the program. Initialization at the time of declaration:

int marks[5] = {70, 75, 80, 85, 90};

Initialization after declaration:
marks[0] = 70; marks[1] =
75; marks[2] = 80;

In a one-dimensional array, indexing starts from 0 and goes up to size-1.

Thus, arrays help in storing multiple values of the same type using a single variable name, and one-dimensional arrays are widely used in C programming for data storage and processing.

2.What is Function? Explain Function Prototype with example.

Function:

A function is a block of code that performs a specific task. It helps in code reusability, modular programming, and easier debugging. A function may take inputs (parameters) and return a value.

Syntax:

```
return_type function_name(parameters) {
   // body of the function }
```

Example:

```
int add(int a, int b) {  // Function definition
   return a + b; }
```

Function Prototype:

A function prototype is a declaration of a function that informs the compiler about the function name, return type, and parameters before its actual definition. It allows a function to be called before it is defined.

Syntax:

```
return_type function_name(parameter_type1, parameter_type2, ...);
```

Example:

```
#include <stdio.h> int add(int, int);  // Function prototype

int main() {
   int sum = add(5, 10);  // Function call     printf("Sum = %d", sum);
   return 0;
}

int add(int a, int b) {  // Function definition
   return a + b; }
```

Explanation:

- int add(int, int); tells the compiler that a function add exists.
- add(5, 10) calls the function.
- The actual work is done in the function definition.

3.explain strcpy() and strcmp() functions with example

1. strcpy() Function

Purpose: Copies one string into another.

Syntax:

strcpy(destination, source);

Example:

```
#include <stdio.h>
#include <string.h>
int main() {
    char str1[20], str2[20];
    strcpy(str1, "Hello");  // Copy "Hello" into str1      strcpy(str2, str1);      // Copy str1 into str2
printf("str1 = %s\nstr2 = %s", str1, str2);
    return 0; }
```

Output:

```
str1 = Hello
str2 = Hello
```

2. strcmp() Function

Purpose: Compares two strings.

Syntax:

strcmp(string1, string2);

Return Values:

- 0 → strings are equal
- <0 → string1 < string2
- >0 → string1 > string2

Example:

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

```
int main() {
    char str1[] = "Hello";     char str2[] = "World";
    int result = strcmp(str1, str2);

    if(result == 0)
        printf("Strings are equal");
    else if(result < 0)
        printf("str1 is less than str2");     else
        printf("str1 is greater than str2");

    return 0; }
```

Output:

```
str1 is less than str2
```

4.Explain types of array with example.

Arrays in C

An array is a collection of elements of the same data type stored at contiguous memory locations. Arrays are used to store multiple values in a single variable.

Syntax:

data_type array_name[size];

Types of Arrays

1. One-Dimensional Array (1D Array)
    - Stores elements in a single row.
    - Accessed using a single index.

Example:

```
#include <stdio.h>
int main() {
   int marks[5] = {90, 85, 78, 92, 88};

   printf("Marks of student 1: %d\n", marks[0]);     printf("Marks of student 3: %d\n", marks[2]);

   return 0; }
```

Output:

Marks of student 1: 90
Marks of student 3: 78

2. Two-Dimensional Array (2D Array)
    - Stores elements in rows and columns.
    - Accessed using two indices: [row][column].

Example:

```
#include <stdio.h>

int main() {
   int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}};
```

```
    printf("Element at row 1, column 2: %d\n", matrix[0]
[1]);
    printf("Element at row 2, column 3: %d\n", matrix[1]
[2]);
    return 0; }
```

Output:

```
Element at row 1, column 2: 2
Element at row 2, column 3: 6
```

3. Multi-Dimensional Array (3D or more)

- Used to store data in more than 2 dimensions.
- Accessed using multiple indices: [i][j][k]….

Example (3D Array):

```
#include <stdio.h>

int main() {
    int arr[2][2][2] = {        {{1, 2}, {3, 4}},
        {{5, 6}, {7, 8}}
    };
    printf("Element at [1][1][0]: %d\n", arr[1][1][0]);

    return 0; }
```

Output:

```
Element at [1][1][0]: 7
```

Summary Table:

Type	Dimension	Example Access
1D Array	1	arr[i]
2D Array	2	arr[i][j]
Multi-D Array	3+	arr[i][j][k]

5.Explain strlen() and strcat() with example.

Here’s a concise and exam-friendly explanation of strlen() and strcat() functions with examples:

1. strlen() Function
Purpose:
strlen() is used to find the length of a string (number of characters excluding the null character \0).

Syntax:

```
int strlen(char str[]);
```

Example:

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

int main() {
    char str[] = "Hello";     int len = strlen(str);

    printf("Length of the string: %d", len);
    return 0; }
```

Output:

```
Length of the string: 5
```

2. strcat() Function

Purpose:
strcat() is used to concatenate (join) two strings.

Syntax:

```
strcat(destination, source);
```

Example:

```
#include <stdio.h>
#include <string.h>
int main() {
    char str1[20] = "Hello, ";     char str2[] = "World!";
```

```
    strcat(str1, str2);  // Join str2 at the end of str1

    printf("Concatenated string: %s", str1);
    return 0; }
```

Output:

Concatenated string: Hello, World!

Notes for Exam:

- strlen() → returns length of string (excluding \0).
- strcat() → joins two strings; ensure destination array is large enough.
- Both functions are in string.h header file.

6.Explain Recursion with example.

Here’s a clear exam-friendly explanation of recursion with an example:

Recursion

Definition:

Recursion is a programming technique where a function calls itself to solve a smaller instance of a problem until a base condition is met.

Key Points:

1. Base Case → Condition where recursion stops.
2. Recursive Case → Part where function calls itself.
3. Useful in problems like factorial, Fibonacci series, tree traversal.

Example: Factorial of a Number

```
#include <stdio.h>
// Recursive function to calculate factorial
int factorial(int n) {     if(n == 0)  // Base
case         return 1;
    else
        return n * factorial(n - 1);  // Recursive call
}

int main() {     int num =
5;
    printf("Factorial of %d is %d", num, factorial(num));
    return 0; }
```

Output:

Factorial of 5 is 120

Explanation:

- factorial(5) calls factorial(4)
- factorial(4) calls factorial(3) … until factorial(0) returns 1
- Then the results are multiplied step by step: 5*4*3*2*1 = 120.

Unit 4

1.Explain in detail give syntax and suitable example of structure initialisation

Structure

Initialization in C

Definition:
A structure is a user-defined data type that groups different types of variables.

Syntax to define:

```
struct Student {
    int rollNo;     char name[20];
float marks; };
```

1. Initialization at Declarationstruct Student s1 = {101, "Krrish", 88.5};

Example:

```
printf("%d %s %.2f", s1.rollNo, s1.name, s1.marks);
```

2. Member-wise Assignment

```
struct Student s2; s2.rollNo = 102;
strcpy(s2.name, "Aarav");
s2.marks = 92.0;
```

Example:

```
printf("%d %s %.2f", s2.rollNo, s2.name, s2.marks);
```

2.Differentiate between structure and union

Structure vs Union in C

A structure and a union are both user-defined data types in C that can store different types of data under a single name. However, they differ in memory allocation, access, and use cases.

1. Memory Allocation
    - Structure: Allocates memory separately for each member. The total memory used is the sum of all member sizes.
    - Union: Allocates shared memory for all members. The memory used is equal to the size of the largest member.
2. Member Access
    - Structure: All members can be accessed and used simultaneously.
    - Union: Only one member can be accessed at a time; modifying one member affects others because they share the same memory.

3. Size
    - Structure: Total size = sum of sizes of all members (may include padding).
    - Union: Total size = size of largest member.

4. Initialization
    - Structure: Multiple members can be initialized at declaration.
    - Union: Only one member can be initialized at a time.

5. Use Case
    - Structure: Useful when you need to store all data together. Example: storing a student's roll number, name, and marks.
    - Union: Useful when only one value is needed at a time, saving memory.

    Example: storing a value that could be int, float, or char, but never simultaneously.

3.Explain pointer arithmetic with suitable example

Pointer Arithmetic (Short Version)

Definition:
Pointer arithmetic is performing operations on pointers to navigate memory locations.

Operations:

- ptr++ → moves to next element
- ptr-- → moves to previous element
- ptr + n → moves forward by n elements
- ptr - n → moves backward by n elements
- ptr1 - ptr2 → gives number of elements between two pointers

Example:

```
#include <stdio.h> int main() {
    int arr[3] = {10, 20, 30};      int *ptr = arr;

    printf("%d ", *ptr);    // 10      ptr++;
printf("%d", *ptr);     // 20

    return 0; }
```

Output:

10 20

4.Explain nesting of structure with suitable example

Nesting of Structures in C

Definition:Nesting of structures means one structure contains another structure as a member.

It allows grouping related information together in a more organized way.

Syntax:

```
struct Structure1 {
   // members of Structure1
};

struct Structure2 {
   struct Structure1 member;  // nested structure
   // other members };
```

Example:

```
#include <stdio.h>
#include <string.h>
// Nested structure example

struct Date {     int day;     int
month;     int year;
};

struct Student {     int rollNo;     char
name[20];
   struct Date dob;  // nested structure
};

int main() {     struct Student s1;

   // Assign values     s1.rollNo = 101;     strcpy(s1.name,
"Krrish");
   s1.dob×day = 15;     s1.dob×month = 12;     s1.dob×year = 2005;

   // Print student info
   printf("Roll No: %d\n", s1.rollNo);     printf("Name: %s\n", s1.name);
   printf("DOB: %d-%d-%d\n", s1.dob.day, s1.dob.month, s1.dob.year);

   return 0; }
```

Output:

```
Roll No: 101
Name: Krrish
DOB: 15-12-2005
```

5.Explain union with suitable example.

Union in C

Definition:
A union is a user-defined data type that allows storing different data types in the same memory location.
Unlike structures, a union can hold only one value at a time, and all members share the same memory.

Syntax:

union UnionName { data_type member1;
data_type member2;
... };

Key Points:

1. Memory allocated = size of largest member.
2. Only one member can contain a value at a time.
3. Useful when you need to store different types of data but only one at a time.
4. Declared like a structure but with the keyword union

Example:

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

union Data { int i; float f;
char str[20];
};

int main() { union Data d;

d.i = 10;

printf("d×i = %d\n", d.i);

d.f = 3.14; // Previous value overwritten printf("d×f = %.2f\n", d.f);

strcpy(d.str, "Hello"); printf("d×str = %s\n", d.str);

return 0; }
```

Output:

d.i = 10
d.f = 3.14
d.str = Hello

Explanation:

- d.i stores an integer.
- Assigning d.f overwrites the memory of d.i.
- Assigning d.str overwrites previous data.
- Memory is shared, so only one value can be correctly stored at a time.

Use Case:

- When memory efficiency is important, and only one value is needed at a time, e.g., storing a value that could be int, float, or char.

6.What is pointer? Explain with example

Pointer in C
Definition:A pointer is a variable that stores the memory address of another variable.

Pointers are used for dynamic memory allocation, efficient array handling, and function arguments.

Syntax:

data_type *pointer_name;

- * indicates that the variable is a pointer.
- The pointer stores the address of a variable of the same data type.

Example:

```
#include <stdio.h>

int main() {
   int num = 50;       // Normal variable      int *ptr;          // Pointer variable      ptr = &num;         // Store address of
num in ptr

   printf("Value of num = %d\n", num);      printf("Address of num = %p\n", &num);      printf("Value stored in ptr =
%p\n", ptr);      printf("Value pointed by ptr = %d\n", *ptr);  // Dereferencing

   return 0; }
```

Output:
Value of num = 50
Address of num = 0x7ffee3b8a4ac (example)
Value stored in ptr = 0x7ffee3b8a4ac
Value pointed by ptr = 50

Explanation:

1. ptr = # → ptr stores the address of num.
2. *ptr → Dereferencing, gives the value stored at that address.
3. Pointers can be used to access and modify variables indirectly.

Key Points for Exam:

- Declared with *.
- Stores the address of a variable.
- Use & to get the address.
- Use * to get the value at the address (dereferencing).