Q1) What is an array? Explain its advantages.

Answer:

An array is a **collection of elements of the same data type** stored in **contiguous (continuous)** memory locations and accessed using a common name and index.

Example:

int marks[5];

This represents an array of 5 integers:
marks[0], marks[1], marks[2], marks[3], marks[4]

⋆ Advantages of Arrays:

1. **Multiple values under one name**
   You can store many values using a single name instead of declaring many variables.
2. **Easy access using index**
   Any element can be accessed directly using index like a[3].
3. **Better code organization**
   Related data (like marks of students, prices, etc.) kept together.
4. **Easy to process with loops**
   All elements can be handled using for / while loops.
5. **Memory management**
   Fixed size memory reserved in advance.

Q2) Explain one-dimensional array with syntax, declaration, initialization and example program.

Answer:

Definition:

A one-dimensional array is a **linear list** of elements of same type.

Syntax of declaration:

datatype array_name[size];

Example:

```
int marks[5];
float price[10];
char grade[3];
```

Initialization:

1. At the time of declaration:

```
int marks[5] = {10, 20, 30, 40, 50};
```

2. Without size (compiler calculates size):

```
int marks[] = {10, 20, 30, 40, 50};
```

3. Partial initialization:

```
int a[5] = {1, 2};  // remaining become 0
```

4. Run-time initialization (using loop):

```
for(i = 0; i < 5; i++)
   scanf("%d", &marks[i]);
```

Example Program: Read and display 5 elements

```
#include<stdio.h>
void main()
{
   int a[5], i;
```

```
    printf("Enter 5 numbers:\n");
    for(i = 0; i < 5; i++) {
        scanf("%d", &a[i]);
    }
    printf("You entered:\n");
    for(i = 0; i < 5; i++) {
        printf("%d ", a[i]);
    }
}
```

Q3) Write a program to find maximum element from a 1-D array.

Answer:

```
#include<stdio.h>
void main()
{
    int n, i;
    printf("Enter size of array: ");
    scanf("%d", &n);
    int a[100], max;   // assuming max size 100
    printf("Enter %d elements:\n", n);
    for(i = 0; i < n; i++) {
        scanf("%d", &a[i]);
    }
```

```
    max = a[0];
    for(i = 1; i < n; i++) {
        if(a[i] > max)
            max = a[i];
    }
    printf("Maximum element = %d", max);
}
```

Q4) What is a two-dimensional array? Explain with example.

Answer:

A two-dimensional array is an array of arrays.
It is used to represent **tables or matrices** having rows and columns.

Syntax:

```
datatype array_name[rows][columns];
```

Example:

```
int a[3][3];
float b[2][4];
```

Here, a can store 9 integer values in 3 rows and 3 columns.

Initialization:

```
int a[2][3] = {
    {1, 2, 3},
    {4, 5, 6}
};
```

OR

```
int a[2][3] = {1, 2, 3, 4, 5, 6};
```

Example Program: Read and print a 3×3 matrix

```
#include<stdio.h>
void main()
{
   int a[3][3], i, j;
   printf("Enter 9 elements:\n");
   for(i = 0; i < 3; i++) {
      for(j = 0; j < 3; j++) {
         scanf("%d", &a[i][j]);
      }
   }
   printf("Matrix is:\n");
   for(i = 0; i < 3; i++) {
      for(j = 0; j < 3; j++) {
         printf("%d ", a[i][j]);
      }
      printf("\n");
   }
}
```

Q5) What is a string in C? How is it represented and stored?

Answer:

A string in C is a **sequence of characters** terminated by a special character called **null character '\0'**.

Strings are stored as **character arrays**.

Example:

char name[10] = "Atul";

Actually stored as:

'A' 't' 'u' 'l' '\0'

Important points:

- Last character is always '\0' (null terminator).
- Because of this, string length is number of characters before '\0'.

Q6) Explain different ways to read and display strings in C.

Answer:

Declaration:

char name[20];

Input Methods:

1. Using scanf("%s", name);
   → Reads a single word (stops at space).
2. Using gets(name);
   → Reads entire line including spaces (but unsafe in modern C).
3. Using fgets(name, size, stdin);
   → Safer alternative.

Output Methods:

1. Using printf("%s", name);
2. Using puts(name);

Example:

```
#include<stdio.h>

void main()

{

   char name[20];

   printf("Enter your name: ");

   scanf("%s", name);   // or use gets(name);

   printf("Hello %s", name);

}
```

Q7) Explain any four standard string handling functions in C.

Answer:

1. strlen(str) – Returns length of string (without '\0')

Example:

```
int len = strlen("Hello");   // len = 5
```

2. strcpy(dest, src) – Copies one string into another

```
char a[20], b[20] = "World";

strcpy(a, b);  // a now contains "World"
```

3. strcmp(s1, s2) – Compares two strings

- Returns 0 → if equal
- 0 → if s1 > s2

- <0 → if s1 < s2

```
if(strcmp("abc", "abc") == 0)
   printf("Equal");
```

4. strcat(s1, s2) – Concatenates s2 at the end of s1

```
char a[20] = "Hello ";
char b[20] = "World";
strcat(a, b);   // a = "Hello World"
```

Q8) What is an array of strings? Give example.

Answer:

An array of strings is a **two-dimensional character array** where:

- Each row represents **one string**.
- Each column represents **characters** of that string.

Example:

```
char names[3][10] = {"Ram", "Amit", "Atul"};
```

Here:

- There are 3 strings → "Ram", "Amit", "Atul"
- Each can store up to 9 characters + '\0'

Q9) What is a function? What are the advantages of using functions in C?

Answer:

A function is a **self-contained block of code** that performs a specific task and can be reused.

Example: printf(), scanf() are library functions.
We can also create **user defined functions**.

⋆ Advantages:

1. **Modularity**
   Program is divided into smaller parts (functions), making it easier to understand.
2. **Reusability**
   Once function is written, it can be called multiple times.
3. **Easy testing & debugging**
   Each function can be tested separately.
4. **Improves readability**
   Code becomes cleaner and structured.
5. **Team work friendly**
   Different team members can work on different functions.

Q10) Explain parts of a user-defined function: function declaration, definition, and call with example.

Answer:

1. Function Declaration (Prototype)

Tells the compiler about function name, return type, and parameters.

```
int add(int, int);
```

2. Function Definition

Contains actual body (logic) of function.

```
int add(int a, int b)
{
   int c;
   c = a + b;
```

```
    return c;
}
```

3. Function Call

Used to execute function from main() or another function.

```
sum = add(5, 10);
```

⋆ Complete Example:

```
#include<stdio.h>
int add(int, int);   // declaration
void main()
{
    int x = 5, y = 10, result;
    result = add(x, y);      // call
    printf("Sum = %d", result);
}
int add(int a, int b)       // definition
{
    return a + b;
}
```

Q11) Explain different categories of user-defined functions with examples.

Answer:

1. **No arguments, no return value**

```
void show();
```

```
void main()
{
   show();
}
void show()
{
   printf("Hello");
}
```

2. **Arguments, no return value**

```
void show(int x);
void main()
{
   show(10);
}
void show(int x)
{
   printf("x = %d", x);
}
```

3. **Arguments with return value**

```
int add(int a, int b);
void main()
{
```

```
    int s = add(2, 3);
}
int add(int a, int b)
{
    return a + b;
}
```

4. **No arguments, with return value**

```
int getNumber();
void main()
{
    int x = getNumber();
}
int getNumber()
{
    int n;
    scanf("%d", &n);
    return n;
}
```

Q12) What is call by value? Explain with example.

Answer:

In **call by value**, the **copy** of actual arguments is passed to the function.
Changes made to formal parameters **do not affect** the original values.

Example:

```
#include<stdio.h>

void change(int x)

{

   x = x + 10;

   printf("Inside function x = %d\n", x);

}

void main()

{

   int a = 5;

   change(a);

   printf("In main a = %d\n", a);
```

Output:

Inside function x = 15

In main a = 5

Original a remains unchanged.

Q13) What is the scope of a variable? Explain local and global variables.

Answer:

Scope of a variable represents the **region/program part where the variable can be accessed**.

Local Variable:

- Declared **inside a function or block**
- Accessible only within that function/block
- Created when function starts; destroyed when it ends

Example:

```
void fun()
{
   int x = 10;  // local variable
}
```

Global Variable:

- Declared **outside all functions**
- Accessible to all functions in that file

Example:

```
int x = 10;  // global
void fun()
{
   printf("%d", x);
}
```

Q14) Explain different storage classes in C.

Answer:

Storage class defines:

- Where the variable is stored
- Default initial value
- Scope (visibility)
- Lifetime (how long it exists)

C provides 4 storage classes:

1. auto

- Default for local variables
- Stored in RAM
- Scope: local
- Lifetime: within block only

```
void fun()
{
   auto int x = 10;
}
```

2. static

- Retains value **between function calls**
- Scope: local to function
- Lifetime: entire program run

```
void fun()
{
   static int count = 0;
   count++;
   printf("%d", count);
```

}

Each call increases count instead of resetting.

3. register

- Suggests compiler to store variable in CPU register
- Faster access, used in loops

```
void main()
{
   register int i;
}
```

4. extern

- Used to access a global variable defined in another file.
  (You can mention brief since in your sem 1 usually deep extern use nahi hota.)

Q15) What is recursion? Explain with an example.

Answer:

Recursion is a process in which a function **calls itself** directly or indirectly.

Every recursive function must have:

- A **base condition** to stop recursion
- A part which moves towards the base condition

Example: Factorial using recursion

Factorial definition:

- $n! = n \times (n-1) \times (n-2) \times \ldots \times 1$
- $0! = 1$

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

```
int fact(int n)
{
   if(n == 0)
      return 1;             // base case
   else
      return n * fact(n - 1); // recursive call
}
void main()
{
   int n;
   printf("Enter number: ");
   scanf("%d", &n);

   printf("Factorial = %d", fact(n));
}
```

Q16) Difference between recursion and iteration (looping).

Answer:

Recursion	Iteration (Loop)
Function calls itself	Uses for, while, do-while
Uses function call stack	Uses loop control variable
More readable for problems like factorial, Fibonacci, tree	More efficient in terms of memory
Must have base condition	Must have termination condition

Q17) Write a program to find sum of first n natural numbers using recursion.

```
#include<stdio.h>
int sum(int n)
{
   if(n == 1)
      return 1;
   else
      return n + sum(n - 1);
}
void main()
{
   int n;
   printf("Enter n: ");
   scanf("%d", &n);
   printf("Sum = %d", sum(n));
}
```