

**Scheme & Solution of MSE: MSE-2**

|  |  |  |  |
| --- | --- | --- | --- |
| **Course Title :** | **problem solving through programming** | **Course Code:** | **cs1001-2** |
| **Prepared by:** | **Dr Vaikunth Pai** | **Signature:** |  |
| **Date:** | **29/10/2024** | **HOD Signature:** |  |

|  |  |  |
| --- | --- | --- |
| **Q.No.** | **Solution** | **Marks** |
| **1a)**  1b)  2a)  **2b)**  3a)  **3b)**  **4a)**  **4b)** | **PART – A**  **MCQ Answers**   |  |  | | --- | --- | | **1** | **A** | | **2** | **B** | | **3** | **B** | | **4** | **A** |   **PART – B**  **UNIT 1**  Define an array? Explain the different methods of initializing two-dimensional arrays with an example.  **Definition of Array (1 mark)** An array is a collection of elements of the same data type stored at contiguous memory locations. It allows us to store multiple values in a single variable, making data manipulation more efficient.  **Methods of Initializing Two-Dimensional Arrays (3 marks)**   1. **Compile-time Initialization** In this method, the values are assigned to the array at the time of declaration.   int arr[2][3] = { {1, 2, 3}, {4, 5, 6} };   1. **Run-time Initialization** Here, values are assigned to the array during program execution using loops.   int arr[2][3];  for (int i = 0; i < 2; i++) {  for (int j = 0; j < 3; j++) {  scanf(“%d”,&arr[i][j]);  }  }  Develop a C program to find a factorial of a given number using recursive function.  #include <stdio.h>  // Function to calculate factorial using recursion  int factorial(int n) {  // Base case: factorial of 0 or 1 is 1  if (n == 0 || n == 1)  return 1;  // Recursive case: n \* factorial of (n-1)  else  return n \* factorial(n - 1);  }  int main() {  int number;    // Get user input  printf("Enter a positive integer: ");  scanf("%d", &number);  // Check if the number is non-negative  if (number < 0) {  printf("Factorial is not defined for negative numbers.\n");  } else {  // Call the factorial function and display the result  printf("Factorial of %d is %d\n", number, factorial(number));  }  return 0;  }  What are the different storage classes in C? Describe each with an example.  **Storage Classes in C**   1. **Automatic (auto)**  * **Scope**: Local to the block/function. * **Lifetime**: Exists only during execution of the block.  1. **Register (register)**  * **Scope**: Local to the block. * **Lifetime**: Exists only during block execution. * **Storage**: stored in CPU register.  1. **Static (static)**  * **Scope**: Local to the function but retains value between calls. * **Lifetime**: Duration of the program  1. **External (extern)**  * **Scope**: Global, accessible across multiple files. * **Lifetime**: Duration of the program.   Develop a C program to transpose a matrix of order M x N and find the trace of the resultant matrix.  #include <stdio.h>  int main() {  int M, N;  int matrix[10][10], transpose[10][10];  // Input the dimensions of the matrix  printf("Enter the number of rows (M): ");  scanf("%d", &M);  printf("Enter the number of columns (N): ");  scanf("%d", &N);    // Input the elements of the matrix  printf("Enter elements of the matrix:\n");  for (int i = 0; i < M; i++) {  for (int j = 0; j < N; j++) {  printf("Element [%d][%d]: ", i, j);  scanf("%d", &matrix[i][j]);  }  }  // Transpose the matrix  for (int i = 0; i < M; i++) {  for (int j = 0; j < N; j++) {  transpose[j][i] = matrix[i][j];  }  }  // Display the transposed matrix  printf("\nTransposed Matrix:\n");  for (int i = 0; i < N; i++) {  for (int j = 0; j < M; j++) {  printf("%d ", transpose[i][j]);  }  printf("\n");  }  // Calculate the trace of the transposed matrix  int trace = 0;  for (int i = 0; i < M && i < N; i++) {  trace += transpose[i][i];  }  printf("\nTrace of the Transposed Matrix: %d\n", trace);  return 0;  }  **UNIT 1I**  Explain the elements of user-defined functions in C with an example.  **Elements of User-Defined Functions**   1. **Function Declaration (Prototype)**:    * It provides the function's name, return type, and parameters (if any) before the function is called.   Example:  int add(int a, int b); // Declaration of the function   1. **Function Definition**:  * It contains the actual code to be executed when the function is called.   Example:  int add(int a, int b) {  return a + b; // Function definition  }   1. **Function Call**:  * This is where the function is invoked in the main program or any other function.   Example:  int sum = add(5, 3); // Function call  Write a C program to reverse a given string without using built-in function.  #include <stdio.h>  void reverseString(char str[50], int length);  void main() {  char str[50];  int length;  // Input the string  printf("Enter a string: ");  gets(str)  for (length = 0; str[length] != '\0'; length++);  // Call the function to reverse the string  reverseString(str, length);  return 0;  }  void reverseString(char str[50], int length) {  int start = 0; // Start index  int end = length - 1; // End index  char temp; // Temporary variable for swapping  // Swap characters from start and end until the middle is reached  while (start < end) {  // Swap characters  temp = str[start];  str[start] = str[end];  str[end] = temp;  // Move towards the middle  start++;  end--;  }  // Output the reversed string  printf("Reversed string: %s\n", str);  }  Explain any four string manipulation functions with an example.   * **strlen**: Calculates the length of a string. * **strcpy**: Copies one string to another. * **strcat**: Concatenates two strings. * **strcmp**: Compares two strings.   Develop a C program to implement linear search using user-defined function.  #include <stdio.h>  int linearSearch(int arr[], int size, int target);  int main() {  int n, target;  // Input the size of the array  printf("Enter the number of elements in the array: ");  scanf("%d", &n);  int arr[n]; // Declare an array of size n  // Input the elements of the array  printf("Enter %d elements:\n", n);  for (int i = 0; i < n; i++) {  printf("Element [%d]: ", i);  scanf("%d", &arr[i]);  }  // Input the target value to search for  printf("Enter the value to search: ");  scanf("%d", &target);  // Call the linear search function  int result = linearSearch(arr, n, target);  // Output the result  if (result != -1) {  printf("Element %d found at index %d.\n", target, result);  } else {  printf("Element %d not found in the array.\n", target);  }  return 0;  }  // Function to perform linear search  int linearSearch(int arr[], int size, int target) {  for (int i = 0; i < size; i++) {  if (arr[i] == target) {  return i; // Return the index if the target is found  }  }  return -1; // Return -1 if the target is not found  } | **4**  **4**  **4**  **4**  **4**  **4**  **4**  **4**  **4** |