|  |  |  |
| --- | --- | --- |
| **Q.No.** | **Solution** | **Marks** |
|  | **Part-A** |  |
| **1** | A | **1** |
| **2** | C | **1** |
| **3** | D | **1** |
| **4** | A | **1** |
| **Part-B** | | |
| **1 a)** | Illustrate the process of compiling and executing C program using a flowchart.    Compiling and Executing a ‘C’ Program:  Compiling a ‘C’ program means translating it into a computer understandable form known as machine language. Compilation is done by a ‘C’ compiler.  A compiler is a program which accepts the source code (actual ‘C’ program) as input and translates it to machine understandable form.  • ‘C’ compilers are available with or without editors. An editor is a program which allows the programmer to type in (key in) the program and modify (edit) it.  • Integrated Development Environment (IDE): It is an environment (a package) where we find the compiler, editor, debugging tools, linking facilities, tracing and testing tools. Example: Code Blocks, Turbo C (TC), Borland C, Microsoft C/C++, ANSI C, etc.  • The procedures used in compiling and executing a ‘C’ program differ from one operating system to other.  Executing a C program written in C involves a series of steps,  These are  • Creating the Program  • Compiling the Program  • Linking the program with the functions that are needed from the C library,  And  • Executing the program. | **5** |
| **b)** | Develop a C program to find the largest of three numbers using ternary operator.  **#include <stdio.h>**  **void main()**  **{**  **int a, b, c, big ;**  **printf("Enter three numbers : ") ;**  **scanf("%d %d %d", &a, &b, &c) ;**  **//Ternary operator code logic to find the biggest number**  **big = (a > b) ? ((a > c) ? a : c) : ((b > c) ? b : c) ;**  **printf("\nThe biggest number is : %d", big) ;**  **}**  **Output:**  **Enter three numbers : -1 5 -10**  **The biggest number is : 5** | **3**  **2** |
| **2. a)**  **b)**  **3 a)** | Demonstrate the functioning of Bitwise operator in C  BITWISE OPERATORS IN ‘C’:  All the data stored in the computer memory are in sequences of bits (0’s and 1’s).  • Some applications require the manipulation of these bits.  • Manipulation of individual bits is carried out in machine language or assembly language.  • ‘C’ provides six operators to perform bitwise operations.  • These operators work only with int and char data-types.  They cannot be used with floating point numbers.     1. **Bitwise AND:**   **Result of bitwise AND is 1 when both the bits are 1, otherwise**     1. **Bitwise OR (|):**   **Result of bitwise OR is 1 when one of the bits is 1, otherwise (when both bits are 0s) it is zero.**       1. **Exclusive OR (XOR ^) :**          1. **(e) Left Shift Operator (<<) and right shift operator (>>):**     Write a note on variables and data types in C, illustrate with examples.   * **Each variable in C has an associated data type.** * **It specifies the type of data that the variable can store like integer, character, floating, double, etc.** * **Each data type requires different amounts of memory and has some specific operations which can be performed over it.**      * **The Four fundamental data types:** * **'C' supports four basic data types:**     *A variable in C is a memory location with some name that helps store some form of data and retrieves it when required. We can store different types of data in the variable and reuse the same variable for storing some other data any number of times.*  C Variable Syntax  data\_type variable\_name = value; // defining single variable  or  data\_type variable\_name1, variable\_name2; // defining multiple variable  Here,   * data\_type: Type of data that a variable can store. * variable\_name: Name of the variable given by the user. * value: value assigned to the variable by the user.   Examples:  int var;  char a;  float fff;  #include <stdlib.h>  #include <math.h>  #include <stdio.h>  int main() {  float a[10], sum = 0, sumv = 0, mean, var, std;  int i, n;    printf("Enter the number of elements:\n");  scanf("%d", &n);    printf("Enter the numbers:\n");  for (i = 0; i < n; i++) {  scanf("%f", &a[i]);  sum = sum + a[i];  }    mean = sum / n;    for (i = 0; i < n; i++) {  sumv = sumv + pow(a[i] - mean, 2); // sumv = sumv + (a[i] - mean) \* (a[i] - mean);  }    printf("sumv = %f\n", sumv);    var = sumv / n;  std = sqrt(var);    printf("\nThe mean = %f\nThe variance = %f\nThe standard deviation = %f\n", mean, var, std);    return 0;  } | **4**  **4**  **2+2**  **4** |
| 3 b) | In programming, particularly in functions, parameters are used to pass values into functions. There are two types of parameters: formal parameters and actual parameters.  1. Formal Parameters:  These are the parameters defined in the function signature (declaration). They act as placeholders for the values that will be passed into the function when it is called. They specify what type of data the function will receive and can be thought of as local variables inside the function.  - Location: Defined in the function's definition.  - Purpose: Serve as placeholders for the actual values that will be passed when the function is called.  2. Actual Parameters:  These are the real values or arguments that are passed to the function when it is called. They can be constants, variables, or expressions.  - Location: Passed during the function call.  - Purpose: Provide the actual data that is used by the function.  Example in C:  #include <stdio.h>  // Function declaration with formal parameters  void add(int a, int b) {      int sum = a + b;  // Using formal parameters inside the function      printf("Sum: %d\n", sum);  }  int main() {      int x = 5, y = 3;      // Function call with actual parameters      add(x, y);  // x and y are the actual parameters      return 0;  }  Explanation:  - Formal Parameters: `int a, int b` are the formal parameters in the `add` function. They are used to receive the values when the function is called.  - Actual Parameters: `x` and `y` are the actual parameters in the function call `add(x, y)`. These are the actual values passed into the function. | **4** |
| 4a) | #include <stdio.h>  #include <stdlib.h>  void read(int x[10][10], int r, int c);  void display(int x[10][10], int row, int col);  void multiply(int a[10][10], int b[10][10], int c[10][10], int r, int col, int p);  int main() {  int a[10][10], b[10][10], c[10][10], m, n, p, q, i, j, k;  printf("Enter the order of matrix A\n");  scanf("%d%d", &m, &n);  printf("Enter the order of matrix B\n");  scanf("%d%d", &p, &q);  if (n != p) {  printf("Matrix multiplication not possible\n");  exit(0);  }  printf("Enter matrix A\n");  read(a, m, n);  printf("Enter matrix B\n");  read(b, p, q);  printf("Entered matrix A is:\n");  display(a, m, n);  printf("Entered matrix B is:\n");  display(b, p, q);  multiply(a, b, c, m, n, q);  printf("The resultant matrix is:\n");  display(c, m, q);  return 0;  }  void read(int x[10][10], int r, int c) {  int i, j;  for (i = 0; i < r; i++) {  for (j = 0; j < c; j++) {  scanf("%d", &x[i][j]);  }  }  }  void display(int x[10][10], int row, int col) {  int i, j;  for (i = 0; i < row; i++) {  for (j = 0; j < col; j++) {  printf("%d\t", x[i][j]);  }  printf("\n");  }  }  void multiply(int a[10][10], int b[10][10], int c[10][10], int m, int n, int q) {  int i, j, k;  for (i = 0; i < m; i++) {  for (j = 0; j < q; j++) {  c[i][j] = 0;  for (k = 0; k < n; k++) {  c[i][j] += a[i][k] \* b[k][j];  }  }  }  } | **4 marks** |
| 4 b) | Scope of a Variable in C  In C programming, the scope of a variable refers to the region of the program where the variable can be accessed or modified. The scope determines the visibility of the variable, i.e., which part of the program can refer to or change the value of that variable. There are several types of variable scopes in C:  1. Global Scope: A variable declared outside all functions has global scope. It is accessible throughout the entire program, from the point of declaration to the end of the program.  2. Local Scope: A variable declared inside a function or a block (such as within loops or conditionals) has local scope. It is accessible only within that function or block.  3. Block Scope: A variable declared within a block (a compound statement enclosed in `{}`) is accessible only within that block.  Example Program to Illustrate Variable Scope  #include <stdio.h>  int globalVar = 100; // Global variable with global scope  void display() {  int localVar = 50; // Local variable with local scope to the function 'display'    // Accessing global and local variables inside the function  printf("Inside display function:\n");  printf("globalVar (global): %d\n", globalVar);  printf("localVar (local): %d\n", localVar);  }  int main() {  int mainVar = 10; // Local variable with local scope to the main function    printf("Inside main function:\n");  printf("globalVar (global): %d\n", globalVar);  printf("mainVar (local): %d\n", mainVar);  // Calling the display function  display();  // Uncommenting the next line will cause an error because localVar is not visible here  // printf("localVar (local): %d\n", localVar); // Error: 'localVar' is not declared in this scope    return 0;  }  Explanation:  1. Global Variable (globalVar):  A variable declared outside all functions, such as `globalVar`, has global scope. It is accessible throughout the entire program, including within both the `main` and `display` functions.  2. Local Variables (mainVar, localVar):  - `mainVar` is declared inside the `main` function, making it local to that function. It is accessible only within the `main` function.  - `localVar` is declared inside the `display` function, making it local to `display`. It is not accessible outside the `display` function.  3. Access to Variables:  - Inside the `main` function, the global variable `globalVar` and the local variable `mainVar` can be accessed.  - Inside the `display` function, both the global variable `globalVar` and the local variable `localVar` can be accessed.  4. Scope Violation:  If an attempt is made to access `localVar` from outside the `display` function (such as in the `main` function), a compilation error occurs because `localVar` is not visible outside the `display` function due to its local scope. | **4 marks** |