QP CODE: 10033506

MUMBAI UNIVERSITY CBCGS SEM 2

C PROGRAMMING JUNE 2023 SOLUTIONS

Q1 Solve any three out of the following.

A. List and explain different components of Computer System. [5]

Ans:- A computer system consists of various components that work together to perform tasks efficiently. Here's a list of the main components along with brief explanations:

1. **Central Processing Unit (CPU**): The CPU is the brain of the computer responsible for executing instructions. It performs arithmetic, logic, and control operations.
2. **Memory:** Memory, also known as RAM (Random Access Memory), stores data and instructions that the CPU needs to access quickly.
3. **Storage Devices:** Storage devices, such as hard disk drives (HDDs) and solid-state drives (SSDs), store data persistently even when the computer is turned off.
4. **Input Devices**: Input devices allow users to provide data and instructions to the computer.
5. **Motherboard**: The motherboard is the main circuit board that connects and holds together all the essential components of the computer. It provides interfaces for communication between components, such as CPU, memory, storage devices, and peripherals.
6. **Operating System (OS):** The operating system is system software that manages computer hardware and provides a user interface for interacting with the computer.
7. **Networking Components:** Networking components enable communication between computers and other devices.
8. **Power Supply Unit (PSU)**: The PSU converts electricity from the outlet into usable power for the computer components. It provides different voltages required by various components and ensures stable power delivery.

B. Write a program to find largest of three numbers using nested if-else. [5]

Ans:-

```
#include <stdio.h>
 int main() {
// Declare three variables to store the numbers
int num1, num2, num3;

 // Input three numbers from the user
printf("Enter three numbers: ");
```

```
    scanf("%d %d %d", &num1, &num2, &num3);

    // Nested if-else statements to find the largest number
    if (num1 >= num2) {
        if (num1 >= num3) {
            printf("%d is the largest number.\n", num1);
        } else {
            printf("%d is the largest number.\n", num3);
        }
    } else {
        if (num2 >= num3) {
            printf("%d is the largest number.\n", num2);
        } else {
            printf("%d is the largest number.\n", num3);
        }
    }
 return 0;
}
```

Output:-

Enter Three Numbers: 4 5 1

5 Is the largest number.

C. Explain function prototype with proper example. [5]

Ans:- A function prototype in C is a declaration that provides the necessary information about a function before its actual implementation. It specifies the function's name, return type, and parameters, allowing the compiler to know how to call the function correctly when it's used before its definition. Function prototypes are typically declared at the beginning of a program or in a header file.

Here's an example to illustrate a function prototype:

```
#include <stdio.h>
// Function prototype
int add(int num1, int num2);

int main() {
    int result;
```

```
    // Calling the function before its definition
    result = add(3, 5);
    printf("Result: %d\n", result);
      return 0;
}

// Function definition
int add(int num1, int num2) {
    return num1 + num2;
}
```

In this example:

- The function add() is declared with a function prototype int add(int num1, int num2); before the main() function.
- The prototype specifies that add() is a function that returns an integer (int) and takes two integer parameters (int num1 and int num2).
- The add() function is defined after the main() function, which provides its implementation.
- Even though add() is called in main() before its definition, the program compiles successfully because the compiler knows about the function's signature through the prototype.

Function prototypes are especially useful when functions are defined in separate source files or when functions call each other. They allow the compiler to perform type checking and ensure that functions are used correctly throughout the program.

D. Explain the following functions with proper example. a) strlen( ) b) strcmp( ) c) strcat () [5]

Ans:-

a) strlen() Function:

The **strlen()** function is used to find the length of a null-terminated string, i.e., the number of characters in the string excluding the null terminator (**'\0'**). It is declared in the **<string.h>** header file.

Example:

```
#include <stdio.h>
#include <string.h>
int main() {
    char str[] = "Hello";
    int length = strlen(str);
    printf("Length of the string: %d\n", length);
```

```
    return 0;
}
```

Output: Length of the string: 13

b) **strcmp() Function:**

The **strcmp()** function is used to compare two strings lexicographically. It returns an integer value:

- 0 if the strings are equal,
- a negative value if the first string is less than the second string, and
- a positive value if the first string is greater than the second string.

Example:

```
#include <stdio.h>
#include <string.h>
int main() {
    char str1[] = "apple";
    char str2[] = "banana";
    int result = strcmp(str1, str2);
    printf("Comparison result: %d\n", result);
    return 0;
}
```

Output:- Comparison result: -1

c) **strcat() Function:**

The **strcat()** function is used to concatenate (append) one string to the end of another. It appends a copy of the source string to the destination string, overwriting the null terminator of the destination string, and then adds a new null terminator. It is declared in the <string.h> header file.

Example:

```
#include <stdio.h>
#include <string.h>
int main() {
    char dest[20] = "Hello, ";
    char src[] = "World!";
    strcat(dest, src);
    printf("Concatenated string: %s\n", dest);
    return 0;
}
```

Output:- Concatenated string: Hello, World!

E. Define Structure and explain the syntax of declaration of Structure with example [5]

Ans:- A structure in C is a user-defined data type that allows you to group together different variables under a single name. Each variable within a structure is called a member or field. Structures provide a way to represent complex data in a more organized and logical manner.

Here's the syntax for declaring a structure in C:

```
struct structure_name {
    // Member variables
    data_type1 member1;
    data_type2 member2;
    // ...
};
```

- struct: Keyword used to define a structure.
- structure_name: Name of the structure.
- { }: Encloses the member variables of the structure.
- data_type: Data type of each member variable.
- member: Name of each member variable.

Example:

```
#include <stdio.h>
// Define a structure called "Person"
struct Person {
    char name[50];
    int age;
};
int main() {
    struct Person person1;
    strcpy(person1.name, "John");
    person1.age = 30;
    printf("Name: %s\n", person1.name);
    printf("Age: %d\n", person1.age);
 return 0;
}
```

In this example:

- We define a structure called "Person" with two members: **name** and **age**.
- Inside the **main()** function, we declare a variable **person1** of type **struct Person**.
- We assign values to the members of **person1** using dot (**.**) notation.
- Finally, we print the details of **person1** using **printf()** statements.

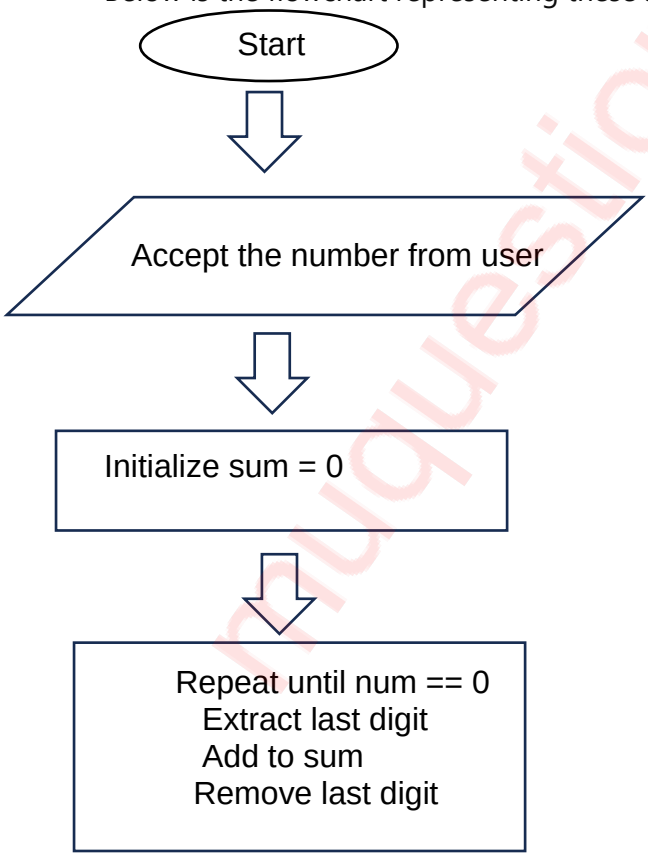
Q.2 A. Define Flowchart and Draw flowchart to find sum of digits of an accepted number. [5]

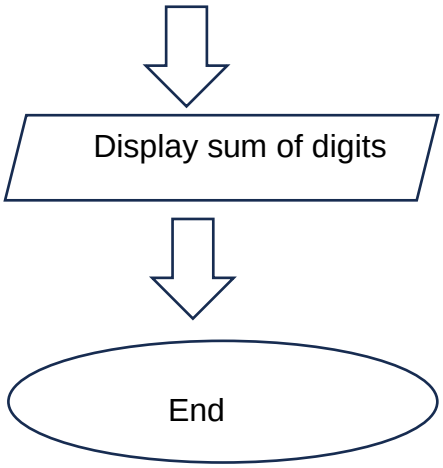
Ans:- A flowchart is a graphical representation of a process or algorithm, showing the steps as boxes of various kinds, and their order by connecting these with arrows. It provides a visual depiction of the sequence of steps or actions to be followed to solve a problem or execute a task.

To draw a flowchart to find the sum of digits of an accepted number, we can follow these steps:

1. Accept the number from the user.
2. Initialize a variable to store the sum of digits.
3. Repeat the following steps until the number becomes zero:
- Extract the last digit of the number.
- Add the extracted digit to the sum.
- Remove the last digit from the number.
4. Display the sum of digits

Below is the flowchart representing these steps:





In this flowchart:

- "Start" and "End" are terminal points indicating the beginning and end of the flowchart, respectively.
- "Accept the number from user" is a process for input.
- "Initialize sum = 0" initializes the variable to store the sum.
- The loop labeled "Repeat until num == 0" extracts digits, adds them to the sum, and removes them from the number until the number becomes zero.
- "Display sum of digits" is a process for output.

B.Write a program to sort the elements of one dimensional array in ascending order. [5]

Ans:- Here's a simple C program to sort the elements of a one-dimensional array in ascending order using the bubble sort algorithm:

```
#include <stdio.h>
void bubbleSort(int arr[], int n) {
    int i, j, temp;
    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                // Swap arr[j] and arr[j+1]
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}
int main() {
```

```
    int arr[10], i, n;

    printf("Enter the number of elements in the array (max 10): ");
    scanf("%d", &n);

    printf("Enter %d elements:\n", n);
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    bubbleSort(arr, n);
    printf("Sorted array in ascending order:\n");
    for (i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
 return 0;
}
```

This program:

1. Accepts the number of elements and the elements of the array from the user.
2. Uses the bubble sort algorithm to sort the elements in ascending order.
3. Prints the sorted array.

D.Explain conditional operator with proper example. [5]

Ans:- The conditional operator (also known as the ternary operator) in C is a ternary operator that takes three operands. It provides a compact way to express conditional statements. The syntax of the conditional operator is:

condition ? expression1 : expression2;

Here's how it works:

- If the condition is true, the value of **expression1** is returned.
- If the condition is false, the value of **expression2** is returned.

The conditional operator is often used as a replacement for simple if-else statements when assigning values based on a condition.

Example:

```
#include <stdio.h>

int main() {
    int num = 10;
    int result;

    // Using conditional operator to assign a value based on a condition
```

```
    result = (num % 2 == 0) ? 1 : 0;

    // Equivalent to:
    // if (num % 2 == 0)
    //     result = 1;
    // else
    //     result = 0;

    printf("Result: %d\n", result);

    return 0;
}
```

In this example:

- We have a variable **num** with a value of 10.
- We use the conditional operator **(num % 2 == 0) ? 1 : 0;** to check if **num** is even.
- If **num** is even, the value **1** is assigned to **result**.
- If **num** is odd, the value **0** is assigned to **result**.
- Finally, we print the value of **result**, which will be **1** since **num** is even.

The conditional operator is useful for writing concise and readable code, especially when performing simple conditional assignments.

Q.3 A. Explain control breaking statements available in C language [5]

Ans:-

In C language, control breaking statements are used to alter the flow of control in a program by breaking out of loops or skipping certain iterations. There are three main control breaking statements available in C:

1. **break**:
    - The **break** statement is used to terminate the execution of a loop immediately when a certain condition is met.
    - It is commonly used in **switch** statements and loop constructs such as **for**, **while**, and **do-while**.
    - When a **break** statement is encountered inside a loop, the loop is terminated, and the program continues with the statement immediately following the loop.

Example:

```
for (int i = 0; i < 10; i++) {
    if (i == 5) {
        break; // Terminate the loop when i equals 5
    }
    printf("%d ", i);

}
```

Output:- 0 1 2 3 4

2. **continue:**

The **continue** statement is used to skip the rest of the code inside a loop for the current iteration and proceed to the next iteration.

When a **continue** statement is encountered inside a loop, the remaining code inside the loop for that iteration is skipped, and the loop proceeds with the next iteration.

Example:

```
for (int i = 0; i < 5; i++) {
    if (i == 2) {
        continue; // Skip iteration when i equals 2
    }
    printf("%d ", i);
}
```

Output:- 0 1 3 4

3. **return**:

The return statement is used to terminate the execution of a function and return a value (if the function has a return type other than void) to the caller.

When a return statement is encountered inside a function, the function immediately exits, and control is returned to the calling function along with the specified return value (if any).

Example:-

```
int sum(int a, int b) {
    return a + b; // Return the sum of a and b
}
```

Usage:-

```
int result = sum(3, 5);
printf("Sum: %d\n", result);
```

These control breaking statements provide programmers with flexibility in controlling the flow of their programs, making it possible to write more efficient and structured code.

B. Write a program to calculate value of f(x), if x has different ranges of values as below

f(x) = x^2+2 0<=x<=10 **[5]**

= x^2+2x 10<x<=20

$=x^3+2x2$ $20<x<=30$

= 0 x>30

Ans:-

You can implement the program in C to calculate the value of $f(x)$ based on different ranges of x as follows:

```
#include <stdio.h>
double calculate_fx(double x) {
    double result;
 if (x >= 0 && x <= 10) {
        result = x * x + 2;
    } else if (x > 10 && x <= 20) {
        result = x * x + 2 * x;
    } else if (x > 20 && x <= 30) {
        result = x * x * x + 2 * x * x;
    } else {
        result = 0;
    }
 return result;
}

int main() {
    double x, result;
    printf("Enter the value of x: ");
    scanf("%lf", &x);
 result = calculate_fx(x);
    printf("f(x) = %.2lf\n", result);
   return 0;
}
```

In this program:

The **calculate_fx** function takes the value of x as input and calculates the corresponding value of $f(x)$ based on the given ranges using **if-else** statements.

- In the **main** function, the user is prompted to enter the value of x.
- The **calculate_fx** function is called with the input value of x, and the calculated result is printed.

C.Write a program to check whether the entered string is palindrome or not . **[5]**

Ans:-

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

// Function to check if a string is a palindrome
int isPalindrome(char *str) {
    int i, j;
    int len = strlen(str);

    // Convert the string to lowercase
    for (i = 0; i < len; i++) {
        str[i] = tolower(str[i]);
    }

    // Check for palindrome
    for (i = 0, j = len - 1; i < j; i++, j--) {
        // Skip non-alphanumeric characters
        while (!isalnum(str[i]) && i < j) {
            i++;
        }
        while (!isalnum(str[j]) && i < j) {
            j--;
        }
        if (str[i] != str[j]) {
            return 0; // Not a palindrome

        }

    }

    return 1; // Palindrome

}
```

```
int main() {
    char str[100];
    printf("Enter a string: ");
    fgets(str, sizeof(str), stdin);

    // Remove newline character from fgets
    if (str[strlen(str) - 1] == '\n') {
        str[strlen(str) - 1] = '\0';
    }
 if (isPalindrome(str)) {
        printf("The entered string is a palindrome.\n");
    } else {
        printf("The entered string is not a palindrome.\n");
    }

    return 0;
}
```

This program:

- Accepts a string from the user using fgets.
- Defines a function isPalindrome to check whether the given string is a palindrome.
- Converts the string to lowercase to make the comparison case-insensitive.
- Checks each character from the beginning and end of the string to determine whether it is a palindrome or not.
- Prints the appropriate message based on the result.

Q.4 A. Explain different data type modifiers available in C language. [5]

Ans:- In C language, data type modifiers are keywords used to modify the properties of data types, such as the storage size, range, and sign. These modifiers allow programmers to control how variables are stored in memory and how they behave during operations. Here are the different data type modifiers available in C:

1.signed and unsigned:

- These modifiers are used with integer data types (**char**, **int**, **short**, **long**) to specify whether the variable can represent both positive and negative values (signed) or only non-negative values (unsigned).
- By default, integer types are signed.

  Example:
- unsigned int x; // Only non-negative values
- signed char c; // Both positive and negative values

2.**short and long:**

- These modifiers are used with integer data types to specify the storage size of variables.
- short typically represents a smaller range of values compared to int, while long typically represents a larger range of values.

Example:

- short int s; // Short integer
- long int l; // Long integer

3. **long long**:

- This modifier is used with integer data types to specify an extended storage size, allowing for an even larger range of values.
- Introduced in C99 standard.

Example: long long int ll; // Long long integer

4**. float, double, and long double**:

- These modifiers are used with floating-point data types (float, double) to specify the precision and range of variables.
- float represents single-precision floating-point numbers, double represents double-precision floating-point numbers, and long double represents extended-precision floating-point numbers.

Example:

- float f; // Single-precision floating-point number
- double d; // Double-precision floating-point number
- long double ld; // Extended-precision floating-point number

5**. const:**

- The const modifier is used to declare constants, which are variables whose values cannot be modified once initialized.
- Attempting to modify a const variable results in a compiler error.

Example: const int MAX = 100; // Constant variable

B. Write a program to find square root of a accepted perfect square integer number without using standard sqrt( ) function [5]

Ans:- To find the square root of a perfect square integer without using the standard **sqrt()** function, you can implement a simple algorithm such as the binary search algorithm. Here's how you can do it in C:

#include <stdio.h>

```
int squareRoot(int num) {
    int low = 0, high = num, mid, result = -1;

    // Binary search for the square root
    while (low <= high) {
        mid = low + (high - low) / 2;
        int square = mid * mid;

        if (square == num) {
            result = mid;
            break;
        } else if (square < num) {
            low = mid + 1;
            result = mid; // Store potential square root
        } else {
            high = mid - 1;
        }
    }

    return result;
}

int main() {
    int num, result;

    printf("Enter a perfect square integer: ");
    scanf("%d", &num);

    result = squareRoot(num);

    if (result != -1) {
        printf("Square root of %d is: %d\n", num, result);
    } else {
        printf("%d is not a perfect square integer.\n", num);
    }

    return 0;
}
```

In this program:

- The **squareRoot()** function takes an integer **num** as input and performs a binary search to find the square root of the given number.
- The **main()** function prompts the user to enter a perfect square integer, calls the **squareRoot()** function to compute its square root, and then prints the result.
- If the input number is not a perfect square, the function returns -1, and the program indicates that the input is not a perfect square integer.

D.Explain the multi-way branching statement available in C language with example. **[5]**

Ans:- In C language, the multi-way branching statement available is the **switch** statement. The **switch** statement allows you to test the value of an expression against multiple possible constant values and execute different blocks of code based on the matching value. This provides a more efficient alternative to a series of nested **if-else** statements when you need to perform multiple comparisons.

Here's the syntax of the **switch** statement:

```
switch (expression) {
   case constant1:
      // code to be executed if expression matches constant1
      break;
   case constant2:
      // code to be executed if expression matches constant2
      break;
   // more case labels as needed
   default:
      // code to be executed if expression doesn't match any case
constant
}
```

The **switch** keyword is followed by the expression being tested.

Inside the **switch** block, you list multiple **case** labels, each followed by a constant value.

The **break** statement is used to terminate the **switch** block and exit to the end of the **switch** statement. Without **break**, execution will continue to the next **case** label regardless of whether its condition is met.

The **default** label is optional and is executed if no match is found for the expression.

Here's an example to demonstrate the **switch** statement:

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

   printf("Enter a number (1-3): ");
   scanf("%d", &choice);

   switch (choice) {
      case 1:
```

```
            printf("You entered 1.\n");
            break;
        case 2:
            printf("You entered 2.\n");
            break;
        case 3:
            printf("You entered 3.\n");
            break;
        default:
            printf("Invalid choice.\n");
    }
return 0;
}
```

In this example:

The user is prompted to enter a number (1-3).

The value entered by the user is compared to each **case** label inside the switch statement.

Q.5.A Write a program using function to check if the entered number is prime number or not [5]

Ans:-

```
#include <stdio.h>
// Function to check if a number is prime
int isPrime(int num) {
    // Handling special cases
    if (num <= 1) {
        return 0; // Not a prime number
    }
    // Check for factors from 2 to sqrt(num)
    for (int i = 2; i * i <= num; i++) {
        if (num % i == 0) {
            return 0; // Not a prime number
        }
    }

    return 1; // Prime number
}
```

```
int main() {
    int num;

    printf("Enter a positive integer: ");
    scanf("%d", &num);

    // Call the function isPrime and check the result
    if (isPrime(num)) {
        printf("%d is a prime number.\n", num);
    } else {
        printf("%d is not a prime number.\n", num);
    }

    return 0;
}
```

In this program:

The **isPrime** function takes an integer **num** as input and returns **1** if the number is prime and **0** otherwise.

The function checks whether the given number is less than or equal to **1** (not a prime number), and then iterates from **2** to the square root of the number to check for factors.

B. Define recursion and Write a program to calculate power of a given number using recursive function. [5]

Ans:- Recursion is a programming technique where a function calls itself in order to solve a problem. In a recursive solution, the problem is broken down into smaller, simpler instances of the same problem until the base case is reached, which is a problem that can be solved without further recursion. Recursion is often used in problems that can be solved by dividing them into smaller subproblems of the same type.

Now, let's write a program to calculate the power of a given number using a recursive function:

```
#include <stdio.h>

// Recursive function to calculate power
double power(double base, int exponent) {
    // Base case: exponent is 0
    if (exponent == 0) {
        return 1;
```

```
    }
    // Recursive case: exponent is positive
    else if (exponent > 0) {
        return base * power(base, exponent - 1);
    }

    // Recursive case: exponent is negative

    else {

        return 1 / power(base, -exponent)

}
}
int main() {
    double base;
    int exponent; // Input base and exponent from user
    printf("Enter base: ");
    scanf("%lf", &base);
    printf("Enter exponent: ");
    scanf("%d", &exponent);

 double result = power(base, exponent);
    printf("%.2lf raised to the power of %d = %.2lf\n", base, exponent, result);
 return 0;
}
```

We define a recursive function **power** that calculates the power of a given base raised to a given exponent.

The base case is when the exponent is 0, in which case the function returns 1.

For positive exponents, the function recursively calls itself with the exponent decremented by 1.

B. Write a program to accept number of rows from user and display following patterns for expected number of rows [5]

1234

123

12

1

Ans:- In this program:

- We accept the number of rows from the user.
- We then use nested loops to print each pattern separately.

- Each pattern iterates from the current row number down to 1, printing the numbers in ascending order on each row.

```
#include <stdio.h>
int main() {
    int rows, i, j;
 // Input the number of rows from the user
    printf("Enter the number of rows: ");
    scanf("%d", &rows);
 // Display the patterns
    printf("Pattern 1:\n");
    for (i = rows; i >= 1; i--) {
        for (j = 1; j <= i; j++) {
            printf("%d ", j);
        }
        printf("\n");
    }
 printf("\nPattern 2:\n");
    for (i = rows; i >= 1; i--) {
        for (j = 1; j <= i; j++) {
            printf("%d ", j);
        }
        printf("\n");
    }
  printf("\nPattern 3:\n");
    for (i = rows; i >= 1; i--) {
        for (j = 1; j <= i; j++) {
            printf("%d ", j);
        }
        printf("\n");
    }

    printf("\nPattern 4:\n");
    for (i = rows; i >= 1; i--) {
        for (j = 1; j <= i; j++) {
            printf("%d ", j);
```

```
        }
        printf("\n");
    }
return 0;
}
```

Q.5.A Write a program to find transpose of a square matrix using only one matrix [5]

Ans:-

```
#include <stdio.h>

#define N 3 // Define the size of the square matrix

void transpose(int mat[][N]) {
    int i, j, temp;

    // Swap elements across the main diagonal
    for (i = 0; i < N; i++) {
        for (j = i + 1; j < N; j++) {
            temp = mat[i][j];
            mat[i][j] = mat[j][i];
            mat[j][i] = temp;
        }
    }
}
int main() {
    int mat[N][N];
    int i, j;
  // Input elements of the square matrix
    printf("Enter elements of the square matrix (%d x %d):\n", N, N);
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            scanf("%d", &mat[i][j]);
        }
    }
   // Display the original matrix
    printf("\nOriginal Matrix:\n");
```

```
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            printf("%d\t", mat[i][j]);
        }
        printf("\n");
    }
  // Find and display the transpose of the matrix
    transpose(mat);
    printf("\nTranspose Matrix:\n");
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            printf("%d\t", mat[i][j]);
        }
        printf("\n");
    }
  return 0;

}
```

In this program:

- We define a square matrix **mat** of size **N x N**.
- The **transpose** function swaps elements across the main diagonal of the matrix.
- In the **main** function, we input elements of the square matrix from the user and display the original matrix.
- We then call the **transpose** function to find the transpose of the matrix and display it

B. Write a program to design a structure Employee with members Employee No, Employee Name, Experience and salary. Read the information of 100 employees and display employee information that is having 5 years or more experience and salary less than Rs. 10,000 [10]

Ans:-

```
#include <stdio.h>

#define MAX_EMPLOYEES 100

// Define structure Employee

struct Employee {

    int empNo;
```

```
    char empName[50];
    int experience;
    float salary;
};
int main() {
    struct Employee employees[MAX_EMPLOYEES];
    int numEmployees, i;
 // Input the number of employees
    printf("Enter the number of employees: ");
    scanf("%d", &numEmployees);

    // Input information for each employee
    for (i = 0; i < numEmployees; i++) {
        printf("\nEnter details for employee %d:\n", i + 1);
        printf("Employee No: ");
        scanf("%d", &employees[i].empNo);
        printf("Employee Name: ");
        scanf("%s", employees[i].empName); // Assuming single-word names
        printf("Experience (in years): ");
        scanf("%d", &employees[i].experience);
        printf("Salary: ");
        scanf("%f", &employees[i].salary);
    }

    // Display employees with 5 or more years of experience and salary less than Rs. 10,000
```

```
    printf("\nEmployees with 5 or more years of experience and salary less than Rs. 10,000:\n");
    for (i = 0; i < numEmployees; i++) {
        if (employees[i].experience >= 5 && employees[i].salary < 10000) {
            printf("Employee No: %d\n", employees[i].empNo);
            printf("Employee Name: %s\n", employees[i].empName);
            printf("Experience: %d years\n", employees[i].experience);
            printf("Salary: Rs. %.2f\n", employees[i].salary);
            printf("\n");
        }
    }
return 0;
}
```

This program defines a structure **Employee** with members **empNo**, **empName**, **experience**, and **salary**. It then reads the information of up to 100 employees, and finally, displays the information of employees who have 5 years or more experience and a salary less than Rs. 10,000.