C-PROGRAMMING LECTURE NOTES

BITC-DAY/EVENING

DATA TYPES IN C PROGRAMMING

In C programming, data types specify the type of data that a variable can hold. The main categories of data types in C are:

1. Basic Data Types

These are the fundamental types used to declare variables.

- **int**: Used to store integer values (whole numbers).
    - Example:

```
int a = 5;  // Declaring an integer variable
```

- **float**: Used to store floating-point numbers (single precision).
    - Example:

```
float b = 3.14f;  // Declaring a float variable
```

- **double**: Used to store double-precision floating-point numbers.
    - Example:

```
double c = 3.1415926535;  // Declaring a double variable
```

- **char**: Used to store a single character.
    - Example:

```
char d = 'A';  // Declaring a char variable
```

2. Derived Data Types

These are types derived from the basic data types, including arrays, pointers, structures, and unions.

- **Arrays**: Used to store multiple values of the same type.
    - Example:

```
int arr[5] = {1, 2, 3, 4, 5};  // Array of integers
```

- **Pointers**: Used to store memory addresses.
    - Example:

```
int *ptr;

int a = 5;

ptr = &a;  // Pointer to an integer variable
```

- **Structures**: Used to group variables of different data types together.
    - Example:

```
struct Person {
   char name[50];
   int age;
};

struct Person person1 = {"Alice", 30};  // Declaring a structure variable
```

- **Unions**: Similar to structures, but all members share the same memory space.
    - Example:

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

union Data data;

data.i = 5;  // Using union to store an integer
```

3. Enumeration Data Type (enum)

Used to define a set of named integer constants.

- Example:

```
enum Day {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};

enum Day today = Wednesday;  // Using enum to define days
```

4. Void Data Type

Used for functions that do not return a value, or for pointers that are not associated with any specific data type.

- Example:

```
void functionName() {
   printf("This function returns no value.\n");
}
```

5. long, short Modifiers

These are modifiers that can be applied to int, long, and double to alter the size of the data type.

- **long int**: Typically used for larger integers.
    - Example:

```
long int e = 100000L;  // Declaring a long integer
```

- **short int**: Used for smaller integers.
    - Example:

```
short int f = 32767;  // Declaring a short integer
```

- **long double**: Used for extended precision floating-point numbers.
    - Example:

```
long double g = 3.14159265358979323846264338327950288419716939937510582097494 4L;
```

6. signed, unsigned Modifiers

These modifiers are used with integer types to specify whether the variable can store only positive values or both positive and negative values.

- **signed**: Can store both positive and negative integers.
    - Example:

signed int h = -100; // Declaring a signed integer

- **unsigned**: Can store only positive integers (zero and positive).
    - Example:

unsigned int i = 200; // Declaring an unsigned integer

Summary of Common Data Types in C:

- **int**: Integer numbers.
- **float**: Single-precision floating-point numbers.
- **double**: Double-precision floating-point numbers.
- **char**: Single character.
- **void**: No value.
- **long**: Longer integer type.
- **short**: Shorter integer type.
- **signed**: Can hold both positive and negative values (applies to int, char).
- **unsigned**: Can hold only non-negative values (applies to int, char).

Example Summary:

```
#include <stdio.h>

int main() {
    int a = 10;           // Integer
    float b = 5.5;        // Float
    double c = 3.14159;   // Double
```

```
    char d = 'X';          // Char
    long e = 100000L;      // Long
    short f = 32767;       // Short
    unsigned int g = 20;   // Unsigned integer
    // Output
    printf("Integer: %d\n", a);
    printf("Float: %.2f\n", b);
    printf("Double: %.5f\n", c);
    printf("Char: %c\n", d);
    printf("Long: %ld\n", e);
    printf("Short: %d\n", f);
    printf("Unsigned int: %u\n", g);
    return 0;
}
```

This program demonstrates various data types in C and their uses.

VARIABLES IN C PROGRAMMING

In **C programming**, a **variable** is a container used to store data that can be manipulated and used by the program. A variable has a name, a data type, and a value.

Key Points about Variables:

- **Declaration**: You must declare a variable before using it.
- **Initialization**: You can assign a value to a variable either at the time of declaration or later.
- **Scope**: The region of the program where the variable can be accessed (e.g., local or global).

- **Lifetime**: The duration for which the variable exists during the execution of the program.

Syntax for Declaring Variables:

data_type variable_name;

- **Example**:

int x; // Declares an integer variable 'x'

float y; // Declares a floating-point variable 'y'

Variables can be initialized at the time of declaration.

Initialization of Variables:

data_type variable_name = value;

- **Example**:

int x = 10; // Declares and initializes an integer 'x' with value 10

float y = 3.14; // Declares and initializes a float 'y' with value 3.14

Types of Variables in C:

1. **Local Variables**:
   These are declared within a function and can only be accessed inside that function.
   - **Example**:

```
void myFunction() {
   int localVar = 5;  // 'localVar' is a local variable
   printf("%d", localVar);
}
```

2. **Global Variables**:
   These are declared outside of all functions and can be accessed by any function in the program.
   - **Example**:

```
int globalVar = 100;  // Global variable

void myFunction() {

   printf("%d", globalVar);  // Accessing global variable

}
```

3. **Static Variables**:
   These variables retain their values between function calls. They are initialized only once and keep their values even after the function execution ends.
   - **Example**:

```
void counter() {

   static int count = 0;  // Static variable

   count++;

   printf("Count: %d\n", count);

}

int main() {

   counter();  // Count: 1

   counter();  // Count: 2

   counter();  // Count: 3

   return 0;

}
```

4. **External Variables**:
   These variables are declared in one file and can be accessed in other files using the extern keyword.
   - **Example**:

```
// In file1.c

int globalVar = 50;  // External variable
```

```
// In file2.c

extern int globalVar;  // Declaring external variable
```

5. **Const Variables**:
   These variables cannot be modified after initialization. They are used when the value should remain constant throughout the program.
   - **Example**:

```
const int MAX_VALUE = 100;  // A constant variable
```

Variable Naming Rules:

- A variable name must begin with a letter (A-Z or a-z) or an underscore (_).
- The remaining characters can be letters, digits (0-9), or underscores.
- Variable names are case-sensitive (age and Age are different).
- They cannot be C keywords (such as int, float, while, etc.).

Example Program Using Variables:

```
#include <stdio.h>

int globalVar = 10; // Global variable

void localFunction() {
    int localVar = 20;  // Local variable
    printf("Local variable: %d\n", localVar);
}

int main() {
    int num = 5; // Local variable
    const float pi = 3.14; // Constant variable
    printf("Global variable: %d\n", globalVar);
    localFunction();  // Call function with local variable
```

```
    printf("Local variable in main: %d\n", num);
    printf("Constant pi: %.2f\n", pi);
    return 0;
}
```

Output:

Global variable: 10

Local variable: 20

Local variable in main: 5

Constant pi: 3.14

Summary of Variable Types:

- **Local Variables**: Defined inside functions, only accessible within the function.
- **Global Variables**: Defined outside functions, accessible by all functions.
- **Static Variables**: Retain their value between function calls.
- **External Variables**: Declared in one file and used in another using the extern keyword.
- **Const Variables**: Their values cannot be modified after initialization.

INPUT AND OUTPUT FUNCTIONS IN C PROGRAMMING

In C, input and output (I/O) operations are used to interact with the user or external devices, such as reading data from the user or displaying information. These operations are performed using functions defined in the **stdio.h** library.

Definition of I/O Functions

1. **Input Functions**: Allow the program to accept data from the user.
    - Example: scanf(), getchar(), gets().
2. **Output Functions**: Display data or results to the user.
    - Example: printf(), putchar(), puts().

---

1. Input Functions

Purpose:
Input functions allow the program to accept data or input from the user. These inputs are stored in variables for processing and use within the program.

Examples and Purpose:

- scanf(): Reads formatted input from the user (e.g., integers, floats, strings).
    - Example: scanf("%d", &num); reads an integer input and stores it in num.
- getchar(): Reads a single character from the user.
    - Example: char ch = getchar(); reads one character and stores it in ch.
- gets(): Reads a string from the user, including spaces (deprecated due to buffer overflow issues).
    - Example: gets(str); reads a line of text into the string str.

---

2. Output Functions

Purpose:
Output functions are used to display data, messages, or results to the user. They help communicate the program's results or status.

Examples and Purpose:

- printf(): Displays formatted output with variable values.
    - Example: printf("The value is %d", num); prints the value of num.
- putchar(): Outputs a single character to the console.
    - Example: putchar('A'); prints the character A.
- puts(): Outputs a string followed by a newline character.
    - Example: puts("Hello, World!"); prints the string "Hello, World!" and moves to the next line.

Function	Purpose	Example
printf()	Outputs formatted data to the console.	printf("Value: %d", x);
scanf()	Accepts formatted input from the user.	scanf("%d", &x);
putchar()	Outputs a single character to the console.	putchar('A');
getchar()	Reads a single character from the user.	char c = getchar();
puts()	Outputs a string with a newline character.	puts("Hello, World!");
gets()	Reads a string from the user (deprecated).	gets(str);

Examples of Input and Output Functions

1. Basic Input and Output with scanf() and printf()

```
#include <stdio.h>

int main() {
    int age;
    float salary;
    char name[50];
    // Input
    printf("Enter your name: ");
    scanf("%s", name);
    printf("Enter your age: ");
    scanf("%d", &age);
    printf("Enter your salary: ");
    scanf("%f", &salary);
    // Output
```

```
    printf("\n--- User Details ---\n");
    printf("Name: %s\n", name);
    printf("Age: %d\n", age);
    printf("Salary: %.2f\n", salary);
    return 0;
}
```

---

2. Using getchar() and putchar()

```
#include <stdio.h>
int main() {
    char ch;
    // Input
    printf("Enter a character: ");
    ch = getchar();
    // Output
    printf("You entered: ");
    putchar(ch);
    printf("\n");
    return 0;
}
```

---

3. String Input with gets() and Output with puts()

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

```
char str[100];

// Input

printf("Enter a string: ");

gets(str); // Note: `gets` is deprecated; use `fgets` in modern programs.
```

---

1. **Define input and output functions in C programming.**
2. **Input Functions:** Allow the program to accept data from the user. Example: scanf(), getchar().

- **Output Functions:** Display data or results to the user. Example: printf(), putchar().

---

2. Write a program to read two integers from the user and display their sum.

Answer:

```
#include <stdio.h>

int main() {

    int num1, num2, sum;

    printf("Enter first number: ");

    scanf("%d", &num1);

    printf("Enter second number: ");

    scanf("%d", &num2);

    sum = num1 + num2;

    printf("The sum is: %d\n", sum);

    return 0;

}
```

---

3. Explain the difference between printf() and puts() in C.

Answer:

- **printf()**: Used to display formatted output, allowing variables to be inserted with format specifiers. Example: printf("Value: %d", x);.
- **puts()**: Used to display a string followed by a newline. Simpler but cannot format variables. Example: puts("Hello");.

---

4. What is the purpose of the & operator in scanf()?

Answer:
The & operator is used to provide the **address** of a variable where the input value will be stored. Without it, scanf() cannot modify the variable.

---

5. Write a program to read a character from the user and display its ASCII value.

Answer:

```
#include <stdio.h>

int main() {
    char ch;
    printf("Enter a character: ");
    ch = getchar();
    printf("The ASCII value of '%c' is: %d\n", ch, ch);
    return 0;
}
```

---

6. Why is gets() deprecated, and what can be used as a safer alternative?

Answer:

- **Reason:** gets() does not check for buffer overflow, leading to potential security issues.

- **Alternative:** Use fgets() instead, which allows you to specify the buffer size.

Example:

```
fgets(str, sizeof(str), stdin);
```

---

7. Write a program to display a floating-point number up to 2 decimal places.

Answer:

```
#include <stdio.h>
int main() {
   float num;
   printf("Enter a floating-point number: ");
   scanf("%f", &num);

   printf("Number up to 2 decimal places: %.2f\n", num);
   return 0;
}
```

---

8. What is the difference between getchar() and putchar()?

Answer:

- **getchar()**: Reads a single character from the input.
- **putchar()**: Outputs a single character to the console.

Example:

```
char ch = getchar();  // Input
putchar(ch);          // Output
```

---

9. Correct the errors in the following code snippet:

```
#include <stdio.h>
int main() {
    char name[20];
    printf("Enter your name: ");
    scanf("%s", name);  // Error?
    printf("Hello, %s\n", name);
    return 0;
}
```

Answer:

There is no error in this specific example, but note the following:

- scanf("%s", name); is correct for single-word input.
- For multi-word input, use fgets() instead of scanf():

```
fgets(name, sizeof(name), stdin);
```

---

10. Write a program to read a string and display it in uppercase using input and output functions.

Answer:

```
#include <stdio.h>
#include <ctype.h>
int main() {
    char str[100];
    int i;
    printf("Enter a string: ");
    fgets(str, sizeof(str), stdin);
```

```
    printf("String in uppercase: ");
    for (i = 0; str[i] != '\0'; i++) {
        putchar(toupper(str[i]));
    }
    return 0;
}
```

Additional 10 Examination Questions on Input and Output Functions in C Programming

---

11. Write a program to read an integer and check whether it is even or odd.

Answer:

```
#include <stdio.h>

int main() {
    int num;
    printf("Enter an integer: ");
    scanf("%d", &num);
    if (num % 2 == 0)
        printf("%d is even.\n", num);
    else
        printf("%d is odd.\n", num);
    return 0;
}
```

---

12. What is the purpose of the format specifiers in scanf() and printf()? Provide at least five examples.

Answer:

Format specifiers define the type of data being input or output. Examples:

- %d: Integer (e.g., scanf("%d", &x);)
- %f: Float (e.g., printf("Value: %f", num);)
- %c: Character (e.g., scanf("%c", &ch);)
- %s: String (e.g., printf("%s", name);)
- %lf: Double (e.g., scanf("%lf", &dbl);)

---

13. Write a program to read two floating-point numbers and display their division result rounded to two decimal places.

Answer:

```
#include <stdio.h>
int main() {
    float num1, num2;
    printf("Enter two numbers: ");
    scanf("%f %f", &num1, &num2);
    if (num2 != 0)
        printf("Result: %.2f\n", num1 / num2);
    else
        printf("Division by zero is not allowed.\n");
    return 0;
}
```

---

14. Explain the difference between scanf() and fgets() for string input.

Answer:

- **scanf()**: Reads a single word (stops at whitespace). Example: scanf("%s", str);.
- **fgets()**: Reads the entire line, including spaces, up to a specified size. Example: fgets(str, sizeof(str), stdin);.

---

15. Write a program to read a string and count the number of vowels in it.

Answer:

```
#include <stdio.h>
#include <ctype.h>
int main() {
    char str[100];
    int i, count = 0;

    printf("Enter a string: ");
    fgets(str, sizeof(str), stdin);
    for (i = 0; str[i] != '\0'; i++) {
        char ch = tolower(str[i]);
        if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u')
            count++;
    }
    printf("Number of vowels: %d\n", count);
    return 0;
}
```

---

16. What is the output of the following code snippet?

```
#include <stdio.h>
int main() {
    int x = 5, y = 10;
    printf("x: %d, y: %d, sum: %d\n", x, y, x + y);
    return 0;
}
```

Answer:

The output is:

x: 5, y: 10, sum: 15

---

17. Write a program to read a character, display it, and then display its uppercase form.

Answer:

```
#include <stdio.h>
#include <ctype.h>
int main() {
    char ch;
    printf("Enter a character: ");
    ch = getchar();
    printf("You entered: %c\n", ch);
    printf("Uppercase: %c\n", toupper(ch));
    return 0;
}
```

---

18. Explain how putchar() works with an example.

Answer:

putchar() is used to output a single character to the console.

Example:

```
#include <stdio.h>

int main() {
    char ch = 'A';
    printf("The character is: ");
    putchar(ch);
    putchar('\n'); // Adds a newline
    return 0;
}
```

---

19. Write a program to display a multiplication table for a given number using input and output functions.

Answer:

```
#include <stdio.h>

int main() {
    int num, i;
    printf("Enter a number: ");
    scanf("%d", &num);
    printf("Multiplication Table of %d:\n", num);
    for (i = 1; i <= 10; i++) {
        printf("%d x %d = %d\n", num, i, num * i);
    }
    return 0;
```

```
}
```

---

20. Write a program to read three integers and display the largest among them.

Answer:

```
#include <stdio.h>
int main() {
    int num1, num2, num3, largest;
    printf("Enter three numbers: ");
    scanf("%d %d %d", &num1, &num2, &num3);
    largest = (num1 > num2) ? (num1 > num3 ? num1 : num3) : (num2 > num3 ? num2 : num3);
    printf("The largest number is: %d\n", largest);

    return 0;
}
```

Control Flow Statements in C Programming

1. if-else Statement

The **if-else statement** allows the program to make decisions based on conditions. It executes one block of code if a condition is true and another block if the condition is false.

Syntax:

```
if (condition) {
    // Executes if the condition is true
} else {
    // Executes if the condition is false
}
```

Example:

```
#include <stdio.h>

int main() {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);
    if (num >= 0) {
        printf("The number is non-negative.\n");
    } else {
        printf("The number is negative.\n");
    }


    return 0;
}
```

---

2. Nested if Statement

The **nested if statement** is used when one condition depends on another. It enables multiple levels of decision-making.

Syntax:

```
if (condition1) {
    if (condition2) {
        // Executes if both condition1 and condition2 are true
    } else {
        // Executes if condition1 is true and condition2 is false
    }
```

```
} else {
    // Executes if condition1 is false
}
```

Example:

```
#include <stdio.h>
int main() {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);
    if (num > 0) {
        if (num % 2 == 0) {
            printf("The number is positive and even.\n");
        } else {
            printf("The number is positive and odd.\n");
        }
    } else if (num < 0) {
        printf("The number is negative.\n");
    } else {
        printf("The number is zero.\n");
    }
    return 0;
}
```

---

3. switch Statement

The **switch statement** is used when a variable is compared against multiple values. It provides an efficient alternative to multiple if-else statements.

Syntax:

```
switch (expression) {
   case value1:
      // Executes if expression == value1
      break;
   case value2:
      // Executes if expression == value2
      break;
   ...
   default:
      // Executes if no case matches
}
```

Example:

```
#include <stdio.h>
int main() {
   int choice;
   printf("Enter a number (1-3): ");
   scanf("%d", &choice);
   switch (choice) {
      case 1:
         printf("You selected option 1.\n");
         break;
      case 2:
```

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

Differences Between if-else, Nested if, and switch Statements in C Programming

---

Feature	**if-else Statement**	**Nested if Statement**	**switch Statement**
Purpose	Used to execute code based on a single condition.	Handles multiple interdependent conditions.	Compares a single variable or expression against multiple constant values.
Syntax	Simple and straightforward.	Adds additional levels of decision-making.	Designed for evaluating discrete cases.
Condition Type	Evaluates relational or logical expressions.	Combines multiple conditions, often hierarchically.	Checks equality against constant values only.
Complexity	Easy to use for simple conditions.	Can become complex with deep nesting.	Easier to read when many discrete cases are present.

Feature	if-else Statement	Nested if Statement	switch Statement
Performance	Slightly slower if many conditions are chained.	Performance depends on the number of nested conditions.	Generally faster for large numbers of discrete cases.
Readability	Readable for simple conditions; less so for many.	Readability decreases with more levels of nesting.	Highly readable for cases with many constant comparisons.
Fall-Through	No fall-through behavior.	No fall-through behavior.	Supports fall-through (without break).
Default Handling	Requires an explicit else block.	No specific default; logic must be explicitly coded.	Has a default case for unmatched values.

ASSIGNMENT: INPUT AND OUTPUT FUNCTIONS IN C PROGRAMMING:

1. ***Write a program to read an integer number from the user and print it on the screen.***
2. ***Write a program to read a floating-point number from the user and display the result with 2 decimal places.***
3. ***Write a program to read a string from the user and print the string in reverse order.***
4. ***Write a program to read multiple integers from the user until a sentinel value (e.g., -1) is entered, and then print the sum of all the integers.***
5. ***Write a program to read and display a character entered by the user.***
6. ***Write a program that uses scanf to read two integers and then prints their sum, difference, product, and quotient.***
7. ***Write a program that reads a name (a string) from the user and displays it with a greeting message, e.g., "Hello, [name]!".***
8. ***Write a program to read a series of space-separated integers and print their average.***

9. ***Write a program to read a matrix of integers (3x3) from the user and display it in a formatted way.***
10. ***Write a program to read an integer and print whether it is even or odd using the modulo operator.***

PRACTICAL: IF-ELSE, NESTED IF, AND SWITCH STATEMENTS

1. Check if a number is even or odd using an if-else statement.

Question:

Write a program to determine whether a given number is even or odd.

Answer:

```
#include <stdio.h>

int main() {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);
    if (num % 2 == 0) {
        printf("%d is even.\n", num);
    } else {
        printf("%d is odd.\n", num);
    }
    return 0;
}
```

---

2. Determine the largest of three numbers using nested if.

Question:

Write a program to find the largest number among three numbers.

Answer:

```
#include <stdio.h>
int main() {
    int a, b, c;
    printf("Enter three numbers: ");
    scanf("%d %d %d", &a, &b, &c);
    if (a > b) {
        if (a > c) {
            printf("The largest number is %d.\n", a);
        } else {
            printf("The largest number is %d.\n", c);
        }
    } else {
        if (b > c) {
            printf("The largest number is %d.\n", b);
        } else {
            printf("The largest number is %d.\n", c);
        }
    }
    return 0;
}
```

---

3. Display a day of the week using a switch statement.

Question:

Write a program to display the day of the week based on a number entered by the user (1 for Monday, 2 for Tuesday, etc.).

Answer:

```
#include <stdio.h>

int main() {
    int day;
    printf("Enter a number (1-7): ");
    scanf("%d", &day);
    switch (day) {
        case 1:
            printf("Monday\n");
            break;
        case 2:
            printf("Tuesday\n");
            break;
        case 3:
            printf("Wednesday\n");
            break;
        case 4:
            printf("Thursday\n");
            break;
        case 5:
            printf("Friday\n");
            break;
```

```
        case 6:
            printf("Saturday\n");
            break;
        case 7:
            printf("Sunday\n");
            break;
        default:
            printf("Invalid input! Please enter a number between 1 and 7.\n");
    }
    return 0;
}
```

---

4. Basic Calculator Using switch

Question:

Write a program to perform addition, subtraction, multiplication, or division based on user input.

Answer:

```
#include <stdio.h>
int main() {
    int num1, num2, choice;
    float result;
    printf("Enter two numbers: ");
    scanf("%d %d", &num1, &num2);
    printf("Choose an operation:\n");
    printf("1. Addition\n");
```

```
printf("2. Subtraction\n");
printf("3. Multiplication\n");
printf("4. Division\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
    case 1:
        result = num1 + num2;
        printf("Result: %.2f\n", result);
        break;
    case 2:
        result = num1 - num2;
        printf("Result: %.2f\n", result);
        break;
    case 3:
        result = num1 * num2;
        printf("Result: %.2f\n", result);
        break;
    case 4:
        if (num2 != 0) {
            result = (float)num1 / num2;
            printf("Result: %.2f\n", result);
        } else {
```

```
            printf("Division by zero is not allowed.\n");
        }
        break;
    default:
        printf("Invalid choice.\n");
  }
  return 0;
}
```

---

5. Determine a student's grade using nested if.

Question:

Write a program to determine a student's grade based on the following marks:

- Marks >= 90: Grade A
- Marks >= 75 and < 90: Grade B
- Marks >= 50 and < 75: Grade C
- Marks < 50: Grade F

Answer:

```
#include <stdio.h>
int main() {
  int marks;
  printf("Enter the marks: ");
  scanf("%d", &marks);
  if (marks >= 90) {
    printf("Grade: A\n");
  } else if (marks >= 75) {
```

```
        printf("Grade: B\n");
    } else if (marks >= 50) {
        printf("Grade: C\n");
    } else {
        printf("Grade: F\n");
    }
    return 0;
}
```

---

6. Check if a year is a leap year using if-else.

Question:
Write a program to check if a given year is a leap year.

Answer:

```
#include <stdio.h>
int main() {
    int year;
    printf("Enter a year: ");
    scanf("%d", &year);
    if ((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0)) {
        printf("%d is a leap year.\n", year);
    } else {
        printf("%d is not a leap year.\n", year);
    }
    return 0;
}
```

---

7. Find whether a character is a vowel or consonant using switch.

Question:

Write a program to check if a given character is a vowel or a consonant.

Answer:

```
#include <stdio.h>

int main() {
    char ch;
    printf("Enter a character: ");
    scanf(" %c", &ch);
    switch (ch) {
        case 'a': case 'e': case 'i': case 'o': case 'u':
        case 'A': case 'E': case 'I': case 'O': case 'U':
            printf("%c is a vowel.\n", ch);
            break;
        default:
            printf("%c is a consonant.\n", ch);
    }
    return 0;
}
```

Question:

Write a program to check if a number is divisible by both 3 and 5.

Answer:

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

```
int main() {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);
    if (num % 3 == 0) {
        if (num % 5 == 0) {
            printf("%d is divisible by both 3 and 5.\n", num);
        } else {
            printf("%d is divisible by 3 but not by 5.\n", num);
        }
    } else {
        printf("%d is not divisible by 3.\n", num);
    }
    return 0;
}
```

ASSIGNMENT-2: IF-ELSE, NESTED IF, AND SWITCH STATEMENTS FOR AN ASSIGNMENT:

If-Else:

1. Write a program to find whether a given number is positive, negative, or zero using an if-else statement.
2. Write a program to check whether a given year is a leap year or not using the if-else statement.
3. Write a program to determine the largest of three numbers using if-else statements.

4. Write a program to check if a number is divisible by both 5 and 10 using if-else statements.
5. Write a program to input a number and check if it is within the range of 1 to 100 using if-else statement.

Nested If:

6. Write a program to check if a student has passed or failed based on their marks using nested if statements. (Consider: Passed with distinction if the marks are above 75, passed if above 40, and failed otherwise.)
7. Write a program to find the roots of a quadratic equation using nested if statements. (Use the discriminant to check the type of roots: real and distinct, real and equal, or complex.)
8. Write a program to input a number and check if it is a prime number using nested if statements.

Switch Statement:

9. Write a program to accept a day number (1 to 7) and display the corresponding day of the week using a switch statement.
10. Write a program that accepts a grade (A, B, C, D, F) and displays the corresponding message. Use a switch statement to determine the message (e.g., "Excellent" for grade A, "Good" for grade B, etc.).

LECTURE 3-LOOPS

1. For Loop

Syntax:

```
for(initialization; condition; update)
{
   // Code block to be executed
}
```

Semantics:

- **Initialization**: It is executed once at the beginning of the loop. This is where you define and initialize the loop control variable.
- **Condition**: It is checked before each iteration of the loop. If the condition is true, the loop executes the code block. If it's false, the loop terminates.
- **Update**: After each iteration, the update expression is executed. It typically increments or decrements the loop control variable.

Example:

```
#include <stdio.h>

int main() {
    for (int i = 1; i <= 5; i++) {
        printf("%d\n", i);
    }
    return 0;
}
```

Output:

1

2

3

4

5

---

2. While Loop

Syntax:

```
while(condition)
{
   // Code block to be executed
}
```

Semantics:

- The condition is evaluated before entering the loop.
- If the condition is true, the loop executes the code block. After each iteration, the condition is checked again.
- If the condition becomes false, the loop terminates.

Example:

```
#include <stdio.h>
int main() {
   int i = 1;
   while (i <= 5) {
      printf("%d\n", i);
      i++;
   }
   return 0;
}
```

Output:

```
1
2
3
```

4

5

---

3. Do-While Loop

Syntax:

```
do
{
   // Code block to be executed
} while(condition);
```

Semantics:

- In a **do-while** loop, the condition is evaluated **after** the code block is executed.
- This means that the loop always executes at least once, even if the condition is false initially.
- If the condition is true, the loop continues executing. If the condition is false, the loop terminates.

Example:

```
#include <stdio.h>


int main() {
   int i = 1;
   do {
      printf("%d\n", i);
      i++;
   } while (i <= 5);
   return 0;
```

}

Output:

1

2

3

4

5

Key Differences:

- **For Loop**: Typically used when the number of iterations is known in advance.
- **While Loop**: Used when the number of iterations is not known, and the loop executes as long as the condition is true.
- **Do-While Loop**: Similar to a while, but the code block is guaranteed to execute at least once, regardless of the condition.

Key Differences Summary:

Feature	For Loop	While Loop	Do-While Loop
When to Use	Known number of iterations	Unknown number of iterations	Always at least one iteration
Condition Check	Before each iteration	Before each iteration	After each iteration

Feature	For Loop	While Loop	Do-While Loop
Guarantee of Execution	May not execute if the condition is false initially	May not execute if the condition is false initially	Always executes at least once
Common Use Cases	Fixed iteration count (e.g., loops for a range)	Continuously check a condition (e.g., user input)	Need to ensure action runs at least once
Flexibility	Least flexible (fixed structure)	More flexible (condition checked each iteration)	Flexible, always runs once

QUESTIONS RELATED TO LOOPS IN C PROGRAMMING, ALONG WITH THEIR ANSWERS:

1. Q: Write a program using a for loop to print numbers from 1 to 10.

Answer:

```
#include <stdio.h>

int main() {
    for (int i = 1; i <= 10; i++) {
        printf("%d ", i);
    }
    return 0;
}
```

Output:

1 2 3 4 5 6 7 8 9 10

---

2. Q: Write a program using a while loop to print even numbers from 2 to 20.

Answer:

```
#include <stdio.h>

int main() {
    int i = 2;
    while (i <= 20) {
        printf("%d ", i);
        i += 2;
    }
    return 0;
}
```

Output:

2 4 6 8 10 12 14 16 18 20

---

3. Q: Write a program to find the factorial of a number using a for loop.

Answer:

```
#include <stdio.h>

int main() {
    int num, factorial = 1;
    printf("Enter a number: ");
    scanf("%d", &num);
    for (int i = 1; i <= num; i++) {
        factorial *= i;
    }
```

```
    printf("Factorial of %d is %d", num, factorial);
    return 0;
}
```

Output (for input 5):

Factorial of 5 is 120

---

4. Q: Write a program to check whether a number is prime using a while loop.

Answer:

```
#include <stdio.h>
int main() {
    int num, i = 2;
    int isPrime = 1;
    printf("Enter a number: ");
    scanf("%d", &num);
    while (i <= num / 2) {
        if (num % i == 0) {
            isPrime = 0;
            break;
        }
        i++;
    }
    if (isPrime && num > 1)
        printf("%d is a prime number.", num);
    else
```

```
        printf("%d is not a prime number.", num);
    return 0;
}
```

Output (for input 7):

7 is a prime number.

---

5. Q: Write a program using a do-while loop to print numbers from 1 to 5.

Answer:

```
#include <stdio.h>
int main() {
    int i = 1;
    do {
        printf("%d ", i);
        i++;
    } while (i <= 5);
    return 0;
}
```

Output:

CopyEdit

1 2 3 4 5

---

6. Q: Write a program to print the Fibonacci series up to the 10th term using a for loop.

Answer:

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

```
int main() {
    int n = 10, t1 = 0, t2 = 1, nextTerm;
    printf("Fibonacci Series: ");
    for (int i = 1; i <= n; i++) {
        if (i == 1) {
            printf("%d, ", t1);
            continue;
        }
        if (i == 2) {
            printf("%d, ", t2);
            continue;
        }
        nextTerm = t1 + t2;
        t1 = t2;
        t2 = nextTerm;
        printf("%d, ", nextTerm);
    }
    return 0;
}
```

Output:

Fibonacci Series: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34,

---

7. Q: Write a program to find the sum of all odd numbers between 1 and 100 using a for loop.

Answer:

```
#include <stdio.h>
int main() {
    int sum = 0;
    for (int i = 1; i <= 100; i += 2) {
        sum += i;
    }
    printf("Sum of all odd numbers between 1 and 100 is %d", sum);
    return 0;
}
```

Output:

```
Sum of all odd numbers between 1 and 100 is 2500
```

---

8. Q: Write a program to reverse a number using a while loop.

Answer:

```
#include <stdio.h>
int main() {
    int num, reversed = 0;
    printf("Enter a number: ");
    scanf("%d", &num);
    while (num != 0) {
        reversed = reversed * 10 + num % 10;
        num /= 10;
    }
```

```
    printf("Reversed number is %d", reversed);
    return 0;
}
```

Output (for input 123):

Reversed number is 321

---

9. Q: Write a program to find the largest of three numbers using a nested if statement.

Answer:

```
#include <stdio.h>
int main() {
    int num1, num2, num3;
    printf("Enter three numbers: ");
    scanf("%d %d %d", &num1, &num2, &num3);
    if (num1 >= num2) {
        if (num1 >= num3) {
            printf("Largest number is %d", num1);
        } else {
            printf("Largest number is %d", num3);
        }
    } else {
        if (num2 >= num3) {
            printf("Largest number is %d", num2);
        } else {
            printf("Largest number is %d", num3);
```

```
    }
  }
  return 0;
}
```

Output (for input 5, 10, 3):

Largest number is 10

---

10. Q: Write a program using a switch statement to display the name of the day for a given number (1 for Sunday, 2 for Monday, etc.).

Answer:

```
#include <stdio.h>
int main() {
  int day;
  printf("Enter a number (1-7): ");
  scanf("%d", &day);
  switch (day) {
    case 1: printf("Sunday"); break;
    case 2: printf("Monday"); break;
    case 3: printf("Tuesday"); break;
    case 4: printf("Wednesday"); break;
    case 5: printf("Thursday"); break;
    case 6: printf("Friday"); break;
    case 7: printf("Saturday"); break;
    default: printf("Invalid input");
  }
```

```
    return 0;
}
```

Output (for input 3):

Tuesday

ASSIGNMENT 3

1. Write a program using a for loop to print the multiplication table of a given number.

2. Write a program to find the sum of all even numbers between 1 and 100 using a while loop.

3. Write a program to count how many times the digit '7' appears in the numbers from 1 to 1000 using a for loop.

4. Write a program using a do-while loop to check if a number entered by the user is positive or negative, and repeat the process until the user enters zero.

5. Write a program to display the Fibonacci series up to a given term using a while loop.

6. Write a program to calculate the sum of digits of a number entered by the user using a do-while loop.

7. Write a program using a for loop to print the prime numbers between 1 and 100.

8. Write a program using a while loop to reverse the digits of a given number.

9. Write a program to find the factorial of a number using a do-while loop.

10. Write a program using a for loop to print a pattern of stars (*), with each row having an increasing number of stars, starting from 1 up to a given number of rows.

Strings and Arrays in C Programming

In C programming, **strings** and **arrays** are fundamental concepts, but they are treated differently in certain ways. Below is a detailed discussion of both concepts, their characteristics, and their usage.

Arrays in C

An **array** in C is a collection of elements of the same data type stored in contiguous memory locations. It is used to store multiple values in a single variable, making it easier to work with large datasets.

Characteristics:

1. **Fixed Size**: The size of an array is fixed at the time of declaration and cannot be changed during runtime.
2. **Homogeneous**: All elements in an array must be of the same data type (e.g., int, char, float).
3. **Indexed Access**: Each element in an array can be accessed using its index (or position), starting from 0 up to n-1, where n is the number of elements in the array.
4. **Memory Allocation**: Arrays are allocated in contiguous memory locations, making them efficient for storing and accessing large datasets.

Syntax:

```
data_type array_name[array_size];
```

Example:

```
#include <stdio.h>

int main() {
   int arr[5] = {1, 2, 3, 4, 5};
   for (int i = 0; i < 5; i++) {
      printf("%d ", arr[i]);
   }


   return 0;
}
```

Output:

```
1 2 3 4 5
```

Types of Arrays:

1. **One-Dimensional Arrays**: A simple array with a single index.
2. **Two-Dimensional Arrays**: An array of arrays, often used to represent matrices.
3. **Multi-Dimensional Arrays**: Arrays with more than two dimensions, like 3D arrays.

Strings in C

In C, strings are arrays of characters terminated by a special character called the **null character ('\0')**. Unlike some other programming languages where strings are a built-in data type, in C, strings are treated as character arrays.

Characteristics:

1. **Character Array**: A string is a sequence of characters, including the null terminator, stored in contiguous memory locations.
2. **Null Terminated**: C strings must be null-terminated, meaning the last character of the string is always '\0' to indicate the end of the string.
3. **Fixed Size**: The size of a string is typically fixed at the time of declaration, but you can use dynamic memory allocation for more flexible storage.
4. **Manipulation**: Strings in C are manipulated using built-in functions from the string.h library.

Syntax:

```
char string_name[size];
```

Example:

```
#include <stdio.h>

int main() {
    char str[] = "Hello, World!";
    printf("%s", str);
    return 0;
```

}

Output:

Hello, World!

Common String Functions:

- **strlen()**: Returns the length of the string (not including the null terminator).
- **strcpy()**: Copies one string to another.
- **strcat()**: Concatenates two strings.
- **strcmp()**: Compares two strings.
- **strchr()**: Finds the first occurrence of a character in a string.

Difference Between Strings and Arrays

1. **Definition**:
    - An **array** is a collection of variables of the same data type.
    - A **string** in C is a special case of an array, specifically an array of characters terminated by '\0'.
2. **Memory**:
    - An **array** can store any type of data (e.g., integers, floats), whereas a **string** is specifically designed to hold character data (text).
    - In both cases, data is stored in contiguous memory, but strings must account for the null terminator to mark the end of the string.
3. **Manipulation**:
    - **Arrays** are manipulated by accessing each element using the index.
    - **Strings** are also arrays but require the null terminator to indicate the end. String manipulation often involves functions from the string.h library.

Example 1: Array of Integers

```
#include <stdio.h>

int main() {

   int arr[3] = {1, 2, 3};

   printf("Array elements: %d %d %d\n", arr[0], arr[1], arr[2]);

   return 0;

}
```

Output:

Array elements: 1 2 3

PRACTICAL

Example 2: String (Array of Characters)

```
#include <stdio.h>

int main() {

   char str[] = "Hello, C!";

   printf("String: %s\n", str);

   return 0;

}
```

Output:

String: Hello, C!

---

2. Write a program to initialize an array with the values 1, 2, 3, 4, 5 and print the array elements.

Answer:

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

```
int main() {
    int arr[] = {1, 2, 3, 4, 5};
    for (int i = 0; i < 5; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

---

3. How do you find the sum of all elements in an array?

Answer:

```
#include <stdio.h>
int main() {
    int arr[] = {1, 2, 3, 4, 5};
    int sum = 0;
    for (int i = 0; i < 5; i++) {
        sum += arr[i];
    }
    printf("Sum: %d\n", sum);
    return 0;
}
```

---

4. Write a C program to find the largest number in an array.

Answer:

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

```
int main() {
    int arr[] = {10, 20, 5, 45, 30};
    int largest = arr[0];
    for (int i = 1; i < 5; i++) {
        if (arr[i] > largest) {
            largest = arr[i];
        }
    }
    printf("Largest element is: %d\n", largest);
    return 0;
}
```

---

5. How do you reverse an array in C?

Answer:

```
#include <stdio.h>
void reverseArray(int arr[], int n) {
    int temp;
    for (int i = 0; i < n / 2; i++) {
        temp = arr[i];
        arr[i] = arr[n - i - 1];
        arr[n - i - 1] = temp;
    }
}
int main() {
```

```
    int arr[] = {1, 2, 3, 4, 5};
    int n = 5;
    reverseArray(arr, n);
        printf("Reversed array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

6. How do you find the first occurrence of a character in a string in C?

Answer:

```
#include <string.h>
int main() {
    char str[] = "Hello, World!";
    char ch = 'o';
    char *ptr = strchr(str, ch);  // Finds the first occurrence of 'o'
    if (ptr != NULL) {
        printf("Character found at position: %ld\n", ptr - str);
    }
    return 0;
}
```

---

7. How do you check if a character is present in a string?

Answer:

```
#include <string.h>
int main() {
    char str[] = "Hello";
    char ch = 'e';
    if (strchr(str, ch)) {
        printf("Character '%c' found in the string.\n", ch);
    } else {
        printf("Character '%c' not found.\n", ch);
    }
    return 0;
}
```

---

8. How do you concatenate two strings in C?

Answer:

```
#include <string.h>
int main() {
    char str1[50] = "Hello ";
    char str2[] = "World!";
    strcat(str1, str2);  // Concatenates str2 to str1
    printf("%s\n", str1);
    return 0;
}
```

---

9. How do you find the length of a string in C?

Answer:

```
#include <string.h>

int main() {
    char str[] = "Hello";
    int len = strlen(str);  // Finds the length of the string
    printf("Length of the string: %d\n", len);
    return 0;
}
```

---

10. How do you copy one string to another in C?

Answer:

```
#include <string.h>

int main() {
    char str1[] = "Hello";
    char str2[50];
    strcpy(str2, str1);  // Copies str1 to str2
    printf("Copied string: %s\n", str2);
    return 0;
}
```

ASSIGNMENT: ON ARRAYS AND STRINGS:

1. Find the largest and smallest elements in an array of integers.
2. Write a function to reverse a given string without using any built-in functions.
3. Given two strings, check if one is a permutation of the other.

4. Write a program to find the first non-repeating character in a string.
5. Find the common elements between two sorted arrays.

Functions in C Programming

Definition:

A function in C programming is a block of code that performs a specific task. Functions allow you to break down a program into smaller, manageable parts, each focusing on a single task. Functions enhance modularity, code reusability, and readability.

Syntax:

```
return_type function_name(parameter1, parameter2, ...) {
   // Function body
   // Code to perform the task
   return value; // (if the return type is not void)
}
```

Where:

- return_type: Specifies the type of value the function returns (e.g., int, char, float). Use void if the function doesn't return anything.
- function_name: The name of the function.
- parameter1, parameter2, ...: The parameters (optional) passed to the function for processing. These are values that the function uses.

Semantics:

- **Function Declaration (Prototype):** Before calling a function, you can declare it with the return type and parameters, specifying how the function should be used.

```
return_type function_name(parameter1, parameter2, ...);
```

- **Function Definition:** This is where the actual functionality of the function is implemented.

```
return_type function_name(parameter1, parameter2, ...) {
    // Function body
}
```

- **Function Call:** This is where the function is invoked, and the program jumps to the function definition to execute it.

```
function_name(argument1, argument2, ...);
```

Advantages of Using Functions:

1. **Modularity:** Functions allow the breaking down of a large program into smaller, manageable chunks.
2. **Code Reusability:** Once a function is defined, it can be reused in different parts of the program without writing the same code again.
3. **Easier Debugging and Maintenance:** Functions make it easier to debug and maintain code by isolating tasks.
4. **Improved Readability:** Functions make code more readable by abstracting complex tasks.
5. **Memory Management:** Functions allow better use of memory by limiting the scope of variables to the function.

Example 1: Function Without Arguments and Return Value

```
#include <stdio.h>
void greet() {
    printf("Hello, welcome to C programming!\n");
}
int main() {
    greet(); // Function call
    return 0;
```

```
}
```

Example 2: Function with Arguments and Return Value

```
#include <stdio.h>

// Function to add two numbers
int add(int a, int b) {
    return a + b;
}

int main() {
    int result = add(5, 7);  // Function call with arguments
    printf("The sum is: %d\n", result);  // Printing the result
    return 0;
}
```

Example 3: Function with Void Return Type (No Return Value)

```
#include <stdio.h>

// Function to print a message
void printMessage() {
    printf("This is a void function.\n");
}

int main() {
    printMessage();  // Function call
    return 0;
}
```

In this case, the function printMessage() has a void return type because it does not return any value. It simply prints a message.

Example: Arithmetic Operations Using Function Calls

```
#include <stdio.h>
// Function prototypes
int add(int a, int b);
int subtract(int a, int b);
int multiply(int a, int b);
float divide(int a, int b);
int main() {
    int num1, num2;
    int sum, diff, prod;
    float quot;
    // Input two numbers
    printf("Enter two numbers: ");
    scanf("%d %d", &num1, &num2);
    // Function calls
    sum = add(num1, num2);
    diff = subtract(num1, num2);
    prod = multiply(num1, num2);
    quot = divide(num1, num2);
    // Display results
    printf("Sum: %d\n", sum);
    printf("Difference: %d\n", diff);
    printf("Product: %d\n", prod);
    printf("Quotient: %.2f\n", quot);
    return 0;
```

```
}

// Function definitions

int add(int a, int b) {

   return a + b;

}

int subtract(int a, int b) {

   return a - b;

}

int multiply(int a, int b) {

   return a * b;

}


float divide(int a, int b) {

   if (b != 0) {

      return (float)a / b;  // Type cast to float to get a floating-point result

   } else {

      printf("Error! Division by zero.\n");

      return 0.0;  // Return 0 if division by zero

   }

}
```

Explanation:

1. **Function Prototypes**: We declare functions for addition (add), subtraction (subtract), multiplication (multiply), and division (divide). Each function takes two integers as parameters, except the divide function, which returns a floating-point value (float) to handle decimal results.

2. **User Input**: The program asks the user to input two numbers, which are stored in num1 and num2.
3. **Function Calls**:
    - sum = add(num1, num2); calls the add() function.
    - diff = subtract(num1, num2); calls the subtract() function.
    - prod = multiply(num1, num2); calls the multiply() function.
    - quot = divide(num1, num2); calls the divide() function.
4. **Output**: The results of the operations are displayed using printf.
5. **Error Handling in Division**: In the divide() function, we check if the divisor b is zero before performing the division to avoid a runtime error.

Sample Output:

Enter two numbers: 10 5

Sum: 15

Difference: 5

Product: 50

Quotient: 2.00

Example: Arithmetic Operations Using Function Calls with Three Integers

```
#include <stdio.h>

// Function prototypes

int add(int a, int b, int c);

int subtract(int a, int b, int c);

int multiply(int a, int b, int c);

float divide(int a, int b, int c);

int main() {

   int num1, num2, num3;
```

```
    int sum, diff, prod;
    float quot;
    // Input three numbers
    printf("Enter three numbers: ");
    scanf("%d %d %d", &num1, &num2, &num3);
    // Function calls
    sum = add(num1, num2, num3);
    diff = subtract(num1, num2, num3);
    prod = multiply(num1, num2, num3);
    quot = divide(num1, num2, num3);
    // Display results
    printf("Sum: %d\n", sum);
    printf("Difference: %d\n", diff);
    printf("Product: %d\n", prod);
    printf("Quotient: %.2f\n", quot);

    return 0;
}
// Function definitions
int add(int a, int b, int c) {
    return a + b + c;
}
int subtract(int a, int b, int c) {
    return a - b - c;
```

```
}

int multiply(int a, int b, int c) {
    return a * b * c;
}

float divide(int a, int b, int c) {
    // Check if either of the divisors is zero
    if (b != 0 && c != 0) {
        return (float)a / b / c;  // Type cast to float to get floating-point result
    } else {
        printf("Error! Division by zero.\n");
        return 0.0;  // Return 0 if division by zero
    }
}
```

Explanation:

1. **Function Prototypes**: The functions add, subtract, multiply, and divide now take three integers as parameters (a, b, c), instead of two.
2. **User Input**: The program asks the user to input three integers, which are stored in num1, num2, and num3.
3. **Function Calls**:
    - sum = add(num1, num2, num3); calls the add() function with three arguments.
    - diff = subtract(num1, num2, num3); calls the subtract() function with three arguments.
    - prod = multiply(num1, num2, num3); calls the multiply() function with three arguments.
    - quot = divide(num1, num2, num3); calls the divide() function with three arguments.

4. **Error Handling in Division**: The divide() function checks if any of the divisors b or c is zero to avoid a division by zero error.
5. **Output**: The program displays the results of the arithmetic operations on the three integers.

Sample Output:

Enter three numbers: 10 5 2

Sum: 17

Difference: 3

Product: 100

Quotient: 1.00

In this example, the program performs the arithmetic operations on three integers and demonstrates how function calls work with multiple arguments in C.

Example: Finding Roots of a Quadratic Equation

```
#include <stdio.h>

#include <math.h>

// Function prototypes

void findRoots(int a, int b, int c);

int main() {

   int a, b, c;

   // Input coefficients of the quadratic equation

   printf("Enter the coefficients of the quadratic equation (a, b, c): ");

   scanf("%d %d %d", &a, &b, &c);

   // Call the function to find the roots

   findRoots(a, b, c);

   return 0;
```

```
}

// Function to find the roots of the quadratic equation
void findRoots(int a, int b, int c) {
    int discriminant = b*b - 4*a*c;  // Calculate the discriminant
    if (discriminant > 0) {
        // Two real and distinct roots
        float root1 = (-b + sqrt(discriminant)) / (2 * a);
        float root2 = (-b - sqrt(discriminant)) / (2 * a);
        printf("The roots are real and distinct: %.2f and %.2f\n", root1, root2);
    }
    else if (discriminant == 0) {
        // One real and repeated root
        float root = -b / (2 * a);
        printf("The root is real and repeated: %.2f\n", root);
    }
    else {
        // Complex (imaginary) roots
        float realPart = -b / (2 * a);
        float imaginaryPart = sqrt(-discriminant) / (2 * a);
        printf("The roots are complex: %.2f + %.2fi and %.2f - %.2fi\n", realPart, imaginaryPart,
realPart, imaginaryPart);
    }
}
```

Explanation:

1. **Input**: The user inputs the values of aaa, bbb, and ccc, which represent the coefficients of the quadratic equation.
2. **Discriminant Calculation**: The discriminant D=b2−4acD = b^2 - 4acD=b2−4ac is calculated. Based on its value, the program determines the nature of the roots.
3. **Roots Calculation**:
    - If D>0D > 0D>0, it computes two real and distinct roots using the quadratic formula.
    - If D=0D = 0D=0, it computes one real repeated root.
    - If D<0D < 0D<0, it computes the real and imaginary parts of the complex roots.
4. **Output**: The program outputs the roots, whether they are real or complex, based on the discriminant.

Sample Outputs:

Case 1: Two Real and Distinct Roots

Enter the coefficients of the quadratic equation (a, b, c): 1 -7 10

The roots are real and distinct: 5.00 and 2.00

Case 2: One Real and Repeated Root

Enter the coefficients of the quadratic equation (a, b, c): 1 6 9

The root is real and repeated: -3.00

Case 3: Complex Roots

Enter the coefficients of the quadratic equation (a, b, c): 1 2 5

The roots are complex: -1.00 + 2.00i and -1.00 - 2.00i

Explanation of the Program:

- **Mathematical Functions**: We use the sqrt() function from the <math.h> library to compute the square root of the discriminant.
- **Conditions**: The program checks the value of the discriminant to determine whether the roots are real, repeated, or complex.

- **Output Format**: The results are formatted with two decimal places using %.2f for real numbers and complex roots.

1. Question:

Write a C function to find the factorial of a number.

Answer:

```
#include <stdio.h>
int factorial(int n) {
    if (n == 0) return 1;
    return n * factorial(n - 1);
}
int main() {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);
    printf("Factorial of %d is %d\n", num, factorial(num));
    return 0;
}
```

---

2. Question:

Write a C program to reverse a string using a function.

Answer:

```
#include <stdio.h>
#include <string.h>
void reverseString(char str[]) {
```

```
    int length = strlen(str);
    for (int i = 0; i < length / 2; i++) {
        char temp = str[i];
        str[i] = str[length - i - 1];
        str[length - i - 1] = temp;
    }
}
int main() {
    char str[100];
    printf("Enter a string: ");
    scanf("%s", str);
    reverseString(str);
    printf("Reversed string: %s\n", str);
    return 0;
}
```

---

3. Question:

Write a C function to find the largest element in an array of integers.

Answer:

```
#include <stdio.h>
int findLargest(int arr[], int n) {
    int largest = arr[0];
    for (int i = 1; i < n; i++) {
        if (arr[i] > largest) {
```

```
            largest = arr[i];
        }
    }
    return largest;
}
int main() {
    int arr[] = {2, 5, 8, 1, 3};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("Largest element is: %d\n", findLargest(arr, n));
    return 0;
}
```

---

4. Question:

Write a C program to find the roots of a quadratic equation using the quadratic formula.

Answer:

```
#include <stdio.h>
#include <math.h>
void findRoots(int a, int b, int c) {
    int discriminant = b*b - 4*a*c;
    if (discriminant > 0) {
        float root1 = (-b + sqrt(discriminant)) / (2 * a);
        float root2 = (-b - sqrt(discriminant)) / (2 * a);
        printf("The roots are real and distinct: %.2f and %.2f\n", root1, root2);
    }
```

```
    else if (discriminant == 0) {
        float root = -b / (2 * a);
        printf("The root is real and repeated: %.2f\n", root);
    }
    else {
        float realPart = -b / (2 * a);
        float imaginaryPart = sqrt(-discriminant) / (2 * a);
        printf("The roots are complex: %.2f + %.2fi and %.2f - %.2fi\n", realPart, imaginaryPart,
realPart, imaginaryPart);
    }
}
int main() {
    int a, b, c;
    printf("Enter the coefficients of the quadratic equation (a, b, c): ");
    scanf("%d %d %d", &a, &b, &c);
    findRoots(a, b, c);
    return 0;
}
```

---

5. Question:

Write a C function to find the sum of digits of a number.

Answer:

```
#include <stdio.h>
int sumOfDigits(int n) {
    int sum = 0;
```

```
    while (n != 0) {
        sum += n % 10;
        n /= 10;
    }
    return sum;
}
int main() {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);
    printf("Sum of digits: %d\n", sumOfDigits(num));
    return 0;
}
```

---

6. Question:

Write a C program to check whether a number is prime or not.

Answer:

```
#include <stdio.h>
int isPrime(int n) {
    if (n <= 1) return 0;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) return 0;
    }
    return 1;
```

```
}
int main() {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);
    if (isPrime(num)) {
        printf("%d is a prime number.\n", num);
    } else {
        printf("%d is not a prime number.\n", num);
    }
    return 0;
}
```

---

7. Question:

Write a C program to count the number of vowels in a string.

Answer:

```
#include <stdio.h>
#include <string.h>
int countVowels(char str[]) {
    int count = 0;
    for (int i = 0; i < strlen(str); i++) {
        char ch = str[i];
        if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u' ||
            ch == 'A' || ch == 'E' || ch == 'I' || ch == 'O' || ch == 'U') {
```

```
            count++;
        }
    }
    return count;
}

int main() {
    char str[100];
    printf("Enter a string: ");
    scanf("%s", str);
    printf("Number of vowels: %d\n", countVowels(str));
    return 0;
}
```

---

8. Question:

Write a C program to concatenate two strings using a function.

Answer:

```
#include <stdio.h>
#include <string.h>
void concatenateStrings(char str1[], char str2[]) {
    strcat(str1, str2);
    printf("Concatenated string: %s\n", str1);
}
int main() {
```

```
    char str1[100], str2[100];
    printf("Enter the first string: ");
    scanf("%s", str1);
    printf("Enter the second string: ");
    scanf("%s", str2);
    concatenateStrings(str1, str2);
    return 0;
}
```

---

9. Question:

Write a C program to calculate the power of a number using a function (x^y).

Answer:

```
#include <stdio.h>
int power(int x, int y) {
    int result = 1;
    for (int i = 1; i <= y; i++) {
        result *= x;
    }
    return result;
}
int main() {
    int x, y;
    printf("Enter the base number: ");
    scanf("%d", &x);
```

```
    printf("Enter the exponent: ");
    scanf("%d", &y);
    printf("%d raised to the power of %d is %d\n", x, y, power(x, y));
    return 0;
}
```

---

10. Question:

Write a C function to sort an array of integers in ascending order using bubble sort.

Answer:

```
#include <stdio.h>
void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n-1; i++) {
        for (int j = 0; j < n-i-1; j++) {
            if (arr[j] > arr[j+1]) {
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }
}
int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr) / sizeof(arr[0]);
```

```
    bubbleSort(arr, n);
    printf("Sorted array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```

ASSIGNMENT

1. Question:

Write a C function to check whether a given number is a prime number or not.

---

2. Question:

Write a C function to calculate the sum of all elements in an array.

---

3. Question:

Write a C function to find the length of a string without using the strlen() function.

---

4. Question:

Write a C function to find the maximum of three numbers.

---

5. Question:

Write a C function to reverse a string.

---

6. Question:

Write a C function to find the area of a rectangle. The function should take length and breadth as arguments.

---

7. Question:

Write a C function to find the roots of a quadratic equation in the form ax^2 + bx + c = 0.

---

8. Question:

Write a C function to calculate the Fibonacci sequence up to the nth term.

---

9. Question:

Write a C function to sort an array of integers in ascending order using the bubble sort algorithm.

---

10. Question:

Write a C function to swap two integers using call by reference.