1. Research and provide three real-world applications where C programming is extensively used, such as in embedded systems, operating systems, or game development.

1. Embedded Systems & Firmware

C is the dominant language for microcontrollers and embedded devices—think routers, thermostats, automotive ECUs, and medical gadgets. Its ability to manipulate hardware registers directly and operate in constrained environments makes it indispensable.

- Companies like Wind River Systems supply embedded platforms and RTOS solutions (e.g. VxWorks, Zephyr, µC/OS-II), all written in C (skillupwards.com, wired.com).
- Contiki and Zephyr—lightweight operating systems for IoT devices—are both implemented in C and used in sensor networks, smart lighting, and industrial control (en.wikipedia.org).

---

2. Operating Systems & System Software

C forms the backbone of almost every major OS and many core system tools.

- The Unix and Linux kernels are predominantly written in C (upgrad.com, reddit.com).
- Windows also relies heavily on C for system utilities and drivers (iies.in).

- Beyond kernels, foundational tools—compilers (e.g. GCC), interpreters (e.g. CPython, Lua), web servers (Apache, Nginx), database engines (MySQL, SQLite), and version control systems (e.g. Git)—are written in C for performance and portability (reddit.com).

---

3. Game Development & Graphics Engines

Although newer games often use C++ or higher-level engines, the performance-critical heart of games and graphics stacks is often written in C.

- Classic titles like *Doom* (1993), *Quake* (1996), and *Wolfenstein 3D* were built in C to squeeze out maximum performance (iies.in, codinginterviewpro.com).
- Graphics libraries like OpenGL, and middleware such as SDL or Allegro, rely on C and serve as the foundation for modern game engines (iies.in).

Summary Table

Application Area	Why C?	Examples
Embedded Systems & RTOS	Low-level hardware access, small footprint	Zephyr, µC/OS-II, Contiki; automotive ECUs, routers, medical devices

Application Area	Why C?	Examples
OS & System Tools	Performance, control, portability	Linux/Unix kernels, Windows components, Apache/Nginx, GCC, Git, SQLite
Games & Graphics Engines	Real-time speed, tight system control	Doom, Quake, OpenGL, SDL, Allegro

---

These examples highlight why C continues to be **the go-to language** when you need **speed**, **direct hardware control**, and **lean, efficient code**. It's foundational across modern operating systems, embedded devices, and high-performance gaming graphics—even after five decades since its creation.

2. Install a C compiler on your system and configure the IDE. Write your first program to print "Hello, World!" and run it.

Step 1: Install a C Compiler

Windows

1. **Install MinGW (Minimalist GNU for Windows):**
    - Download: https://sourceforge.net/projects/mingw/
    - Install mingw32-gcc-g++ under "Basic Setup"
    - Add C:\MinGW\bin to your system PATH

macOS

1. **Use Xcode Command Line Tools:**
    - Open Terminal and run:
    - xcode-select --install

Linux (Ubuntu/Debian)

1. Install gcc using:
2. sudo apt update
3. sudo apt install build-essential

Step 2: Choose and Set Up an IDE

Recommended IDEs:

- **VS Code** (Cross-platform):
    - Install: https://code.visualstudio.com/
    - Install extensions: "C/C++" by Microsoft
    - Configure tasks: create a tasks.json file to run the compiler
- **Code::Blocks** (Windows/Linux):
    - Download: http://www.codeblocks.org/downloads
    - Select version with mingw included

Step 3: Write Your First C Program

Create a new file named hello.c and paste the following code:

#include <stdio.h>

```
int main() {
   printf("Hello, World!\n");
   return 0;
}
```

Step 4: Compile and Run

Using Command Line (All Platforms):

1. **Compile:**
2. gcc hello.c -o hello
3. **Run:**
    - On Windows:
    - hello.exe
    - On macOS/Linux:
    - ./hello

You should see:

Hello, World!

---

3. Write a C program that includes variables, constants, and comments. Declare and use different data types (int, char, float) and display their values.

simple C program that demonstrates:

- **Variables**
- **Constants**
- **Comments**
- **Different data types**: int, char, and float
- **Printing values using printf**

Example C Program

```
#include <stdio.h>
// Define a constant
#define PI 3.14159
int main() {
   // Declare and initialize an integer variable
   int age = 25;
   // Declare and initialize a character variable
   char grade = 'A';
   // Declare and initialize a float variable
   float height = 5.9;
   // Output the values
   printf("Age: %d years\n", age);        // Integer output
   printf("Grade: %c\n", grade);          // Character output
   printf("Height: %.1f feet\n", height);  // Float output
   printf("Value of PI: %.5f\n", PI);     // Constant output
```

```
    return 0;
}
```

Explanation:

- #define PI 3.14159 creates a constant named PI.
- int, char, and float variables are declared and initialized.
- printf() is used to print their values.
- Comments (//) explain each part.

---

4. Write a C program that accepts two integers from the user and performs arithmetic, relational, and logical operations on them. Display the results.

Here is a C program that:

- Accepts **two integers** from the user
- Performs:
    - **Arithmetic operations** (+, -, *, /, %)
    - **Relational operations** (==, !=, <, >, <=, >=)
    - **Logical operations** (&&, ||, !)
- Displays the results using printf()

Complete C Program:

```
#include <stdio.h>
int main() {
   int a, b;

   // Input two integers
   printf("Enter the first integer: ");
   scanf("%d", &a);
   printf("Enter the second integer: ");
   scanf("%d", &b);

   // Arithmetic Operations
   printf("\n--- Arithmetic Operations ---\n");
   printf("a + b = %d\n", a + b);
   printf("a - b = %d\n", a - b);
   printf("a * b = %d\n", a * b);
   if (b != 0) {
      printf("a / b = %d\n", a / b);
      printf("a %% b = %d\n", a % b);
   } else {
      printf("Division and modulus by zero are undefined.\n");
   }
```

```
    // Relational Operations
    printf("\n--- Relational Operations ---\n");
    printf("a == b: %d\n", a == b);
    printf("a != b: %d\n", a != b);
    printf("a < b: %d\n", a < b);
    printf("a > b: %d\n", a > b);
    printf("a <= b: %d\n", a <= b);
    printf("a >= b: %d\n", a >= b);

    // Logical Operations
    printf("\n--- Logical Operations ---\n");
    printf("a && b: %d\n", a && b);
    printf("a || b: %d\n", a || b);
    printf("!a: %d\n", !a);
    printf("!b: %d\n", !b);

    return 0;
}
```

Notes:

- Logical operations treat non-zero as true and zero as false.
- Relational and logical results are displayed as 0 (false) or 1 (true).

- Division and modulus are checked to prevent divide-by-zero errors.

---

5. Write a C program to check if a number is even or odd using an if-else statement. Extend the program using a switch statement to display the month name based on the user's input (1 for January, 2 for February, etc.).

Complete C Program

```
#include <stdio.h>
int main() {
   int number, month;
   // Part 1: Check even or odd
   printf("Enter an integer: ");
   scanf("%d", &number);
   if (number % 2 == 0) {
      printf("%d is even.\n", number);
   } else {
      printf("%d is odd.\n", number);
   }

   // Part 2: Display month name
```

```
    printf("\nEnter a month number (1-12): ");
    scanf("%d", &month);

    printf("Month: ");
    switch (month) {
        case 1:  printf("January\n"); break;
        case 2:  printf("February\n"); break;
        case 3:  printf("March\n"); break;
        case 4:  printf("April\n"); break;
        case 5:  printf("May\n"); break;
        case 6:  printf("June\n"); break;
        case 7:  printf("July\n"); break;
        case 8:  printf("August\n"); break;
        case 9:  printf("September\n"); break;
        case 10: printf("October\n"); break;
        case 11: printf("November\n"); break;
        case 12: printf("December\n"); break;
        default: printf("Invalid month number!\n");
    }
    return 0;
}
```

How It Works:

- if (number % 2 == 0) determines if a number is even.
- The switch statement maps numbers 1–12 to month names.
- default handles invalid inputs (e.g., 13).

---

6. Write a C program to print numbers from 1 to 10 using all three types of loops (while, for, do-while).

Complete C Program

```
#include <stdio.h>
int main() {
   int i;
   // Using while loop
   printf("Using while loop:\n");
   i = 1;
   while (i <= 10) {
      printf("%d ", i);
      i++;
   }

   // Using for loop
   printf("\n\nUsing for loop:\n");
```

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

    // Using do-while loop
    printf("\n\nUsing do-while loop:\n");
    i = 1;
    do {
        printf("%d ", i);
        i++;
    } while (i <= 10);
    printf("\n");
    return 0;
}
```

Explanation:

- **while loop:** Checks the condition before executing the block.
- **for loop:** Compact syntax for initialization, condition, and increment.
- **do-while loop:** Always runs the block at least once (condition checked after).

---

7. Write a C program that uses the break statement to stop printing numbers when it reaches 5. Modify the program to skip printing the number 3 using the continue statement.

Complete C Program

```
#include <stdio.h>
int main() {
   int i;
   // Part 1: Using break to stop at 5
   printf("Using break statement (stop at 5):\n");
   for (i = 1; i <= 10; i++) {
      if (i == 5) {
         break; // Exit the loop when i is 5
      }
      printf("%d ", i);
   }

   // Part 2: Using continue to skip 3
   printf("\n\nUsing continue statement (skip 3):\n");
   for (i = 1; i <= 10; i++) {
      if (i == 3) {
         continue; // Skip this iteration when i is 3
```

```
        }
        printf("%d ", i);
    }
    printf("\n");
    return 0;
}
```

Output:

Using break statement (stop at 5):

1 2 3 4

Using continue statement (skip 3):

1 2 4 5 6 7 8 9 10

8. Write a C program that calculates the factorial of a number using a function. Include function declaration, definition, and call.

Complete C Program: Factorial Using a Function

```
#include <stdio.h>
// Function declaration (prototype)
int factorial(int n);
int main() {
    int number, result;
```

```
    // Input from user
    printf("Enter a positive integer: ");
    scanf("%d", &number);

    // Check for negative input
    if (number < 0) {
        printf("Factorial is not defined for negative numbers.\n");
    } else {
        // Function call
        result = factorial(number);
        printf("Factorial of %d is %d\n", number, result);
    }
    return 0;
}

// Function definition
int factorial(int n) {
    int i, fact = 1;
    for (i = 1; i <= n; i++) {
        fact *= i;
    }
    return fact;
```

```
}
```

Explanation:

- **factorial(int n)**: Computes n! using a loop.
- The **prototype** before main() tells the compiler that the function exists.
- Input is validated for non-negative integers.

---

9. Write a C program that stores 5 integers in a one-dimensional array and prints them. Extend this to handle a two-dimensional array (3x3 matrix) and calculate the sum of all elements.

C Program Code:

```
#include <stdio.h>

int main() {
   // Part 1: One-Dimensional Array
   int arr[5] = {10, 20, 30, 40, 50};
   printf("One-Dimensional Array Elements:\n");
   for(int i = 0; i < 5; i++) {
      printf("%d ", arr[i]);
   }
   printf("\n\n");
```

```
    // Part 2: Two-Dimensional Array (3x3 Matrix)
    int matrix[3][3] = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };
    int sum = 0;
    printf("Two-Dimensional Array (3x3 Matrix):\n");
    for(int i = 0; i < 3; i++) {
        for(int j = 0; j < 3; j++) {
            printf("%d ", matrix[i][j]);
            sum += matrix[i][j];
        }
        printf("\n");
    }
    printf("\nSum of all elements in the matrix = %d\n", sum);
    return 0;
}
```

Output:

One-Dimensional Array Elements:

10 20 30 40 50

Two-Dimensional Array (3x3 Matrix):

1 2 3

4 5 6

7 8 9

Sum of all elements in the matrix = 45

10. Write a C program to demonstrate pointer usage. Use a pointer to modify the value of a variable and print the result.

It modifies the value of a variable using a pointer and prints the result before and after modification:

```
#include <stdio.h>

int main() {
   int number = 10;     // Declare an integer variable
   int *ptr;           // Declare a pointer to an integer
   ptr = &number;       // Assign the address of 'number' to the pointer


   printf("Before modification:\n");
   printf("Value of number: %d\n", number);
   printf("Address of number: %p\n", (void*)&number);
   printf("Value stored in pointer ptr: %p\n", (void*)ptr);
```

```
    printf("Value pointed to by ptr: %d\n", *ptr);

    *ptr = 20;          // Modify the value of 'number' using the pointer

    printf("\nAfter modification:\n");
    printf("Value of number: %d\n", number);
    printf("Value pointed to by ptr: %d\n", *ptr);

    return 0;
}
```

Explanation:

- int *ptr; declares a pointer to an integer.
- ptr = &number; stores the address of number in ptr.
- *ptr = 20; changes the value at the memory location pointed to by ptr, effectively updating number.

Output:

Before modification:

Value of number: 10

Address of number: 0x7ffeeaad36fc

Value stored in pointer ptr: 0x7ffeeaad36fc

Value pointed to by ptr: 10

After modification:

Value of number: 20

Value pointed to by ptr: 20

Note: Actual memory addresses will vary depending on your system and runtime.

11. Write a C program that takes two strings from the user and concatenates them using strcat(). Display the concatenated string and its length using strlen().

```
#include <stdio.h>

#include <string.h>

int main() {

   char str1[100], str2[100];

   // Taking input from the user

   printf("Enter the first string: ");

   fgets(str1, sizeof(str1), stdin);

   str1[strcspn(str1, "\n")] = '\0';  // Remove trailing newline if present


   printf("Enter the second string: ");

   fgets(str2, sizeof(str2), stdin);

   str2[strcspn(str2, "\n")] = '\0';  // Remove trailing newline if present
```

```
    // Concatenating the strings
    strcat(str1, str2);

    // Displaying the result
    printf("\nConcatenated string: %s\n", str1);
    printf("Length of concatenated string: %lu\n", strlen(str1));

    return 0;
}
```

Notes:

- fgets() is used to safely read strings including spaces.
- strcspn(str, "\n") removes the newline character added by fgets() if present.
- strcat() appends the contents of str2 to str1.
- strlen() returns the length of the resulting string.

Example Output:

```
Enter the first string: Hello
Enter the second string: World

Concatenated string: HelloWorld
Length of concatenated string: 10
```

Make sure that str1 has enough space to hold both strings after concatenation.

12. Write a C program that defines a structure to store a student's details (name, roll number, and marks). Use an array of structures to store details of 3 students and print them.

```
#include <stdio.h>
// Define structure for student
struct Student {
    char name[50];
    int rollNumber;
    float marks;
};
int main() {
    struct Student students[3];
    // Input details for 3 students
    for (int i = 0; i < 3; i++) {
        printf("Enter details for student %d:\n", i + 1);

        printf("Name: ");
        scanf(" %[^\n]", students[i].name);  // Read string with spaces
```

```
        printf("Roll Number: ");
        scanf("%d", &students[i].rollNumber);

        printf("Marks: ");
        scanf("%f", &students[i].marks);
        printf("\n");
    }

    // Display student details
    printf("Student Details:\n");
    for (int i = 0; i < 3; i++) {
        printf("Student %d:\n", i + 1);
        printf("Name      : %s\n", students[i].name);
        printf("Roll Number: %d\n", students[i].rollNumber);
        printf("Marks     : %.2f\n", students[i].marks);
        printf("\n");
    }

    return 0;
}
```

Key Features:

- struct Student holds name, rollNumber, and marks.
- students[3] stores information for 3 students.

- " %[^\n]" format specifier in scanf allows input of full names with spaces.

13. Write a C program to create a file, write a string in to it, close the file, then open the file again to read and display its contents.

```
#include <stdio.h>
int main() {
   FILE *file;
   char str[] = "Hello, this is a test string written to the file.";
   char buffer[100];

   // Step 1: Create and write to the file
   file = fopen("example.txt", "w");  // Open file in write mode
   if (file == NULL) {
      printf("Error opening file for writing.\n");
      return 1;
   }
   fputs(str, file);  // Write string to file
   fclose(file);      // Close the file after writing

   // Step 2: Reopen the file to read
   file = fopen("example.txt", "r");  // Open file in read mode
```

```
    if (file == NULL) {
        printf("Error opening file for reading.\n");
        return 1;
    }

    // Read and display contents
    printf("Contents of the file:\n");
    while (fgets(buffer, sizeof(buffer), file) != NULL) {
        printf("%s", buffer);
    }
    fclose(file);  // Close the file after reading
    return 0;
}
```

Notes:

- "example.txt" is the name of the file created.
- fputs() writes the string to the file.
- fgets() reads the content line by line.
- fopen() is used in "w" mode to write and "r" mode to read.
- Always check if the file was opened successfully.