M.SC. Computer Applications
Sem-I
Lab Book: Object Oriented Programming with C++

1.Write a C++ program to prompt the user to input her/his name and print this name on the screen, as shown below. The text from keyboard can be read by using cin>> and to display the text on the screen you can use cout<<.

Source Code:

```
#include <iostream>
#include <string>
using namespace std;

int main() {
    string name;

    // Prompt the user to input their name
    cout << "Please enter your name: ";

    // Read the input name from the user
    getline(cin, name);  // Use getline to read the entire line including spaces

    // Print the name on the screen
    cout << "Hello, " << name << "!" << endl;

    return 0;
}
```

Output:

Please enter your name: ABC

Hello, ABC!

Explanation:

1. **Include Headers:**
    - #include <iostream>: Required for input and output operations.
    - #include <string>: Required for using the string class.
2. **Main Function:**
    - **Declare a String Variable:**
        - string name;: This variable will store the user's name.
    - **Prompt User for Input:**

        - cout << "Please enter your name: ";: Outputs a message asking the user to enter their name.
    - **Read User Input:**
        - getline(cin, name);: Reads a line of text from the user, including spaces, and stores it in the name variable. Using getline ensures that if the user enters a full name with spaces, it will be read correctly.
    - **Display the Name:**
        - cout << "Hello, " << name << "!" << endl;: Outputs a greeting message with the user's name.
3. **Return Statement:**
    - return 0;: Indicates that the program ended successfully.

2. Write a program in C++ to display various type or arithmetic operation (Menu driven)

To create a C++ program that displays a menu for performing various arithmetic operations, follow these steps:

1. **Display a Menu** with options for different arithmetic operations (addition, subtraction, multiplication, division).
2. **Read User Choice** and perform the corresponding arithmetic operation.
3. **Handle Division by Zero** gracefully.
4. **Allow Multiple Operations** until the user chooses to exit.

Source Code:

```
#include <iostream>
using namespace std;

// Function prototypes for arithmetic operations
void add(double a, double b);
void subtract(double a, double b);
void multiply(double a, double b);
void divide(double a, double b);

int main() {
    int choice;
    double num1, num2;

    do {
        // Display menu
        cout << "Arithmetic Operations Menu" << endl;
        cout << "1. Addition" << endl;
        cout << "2. Subtraction" << endl;
        cout << "3. Multiplication" << endl;
        cout << "4. Division" << endl;
        cout << "5. Exit" << endl;
        cout << "Enter your choice (1-5): ";
        cin >> choice;

        // Check if the choice is to exit
        if (choice == 5) {
            cout << "Exiting the program." << endl;
            break;
        }

        // Input two numbers
        cout << "Enter first number: ";
        cin >> num1;
        cout << "Enter second number: ";
        cin >> num2;

        // Perform the chosen operation
        switch (choice) {
        case 1:
            add(num1, num2);
            break;
```

```
            case 2:
                subtract(num1, num2);
                break;
            case 3:
                multiply(num1, num2);
                break;
            case 4:
                divide(num1, num2);
                break;
            default:
                cout << "Invalid choice! Please choose between 1 and 5." << endl;
            }

            cout << endl; // Blank line for readability
        } while (choice != 5);

        return 0;
}

// Function definitions
void add(double a, double b) {
    cout << "Result: " << a + b << endl;
}

void subtract(double a, double b) {
    cout << "Result: " << a - b << endl;
}

void multiply(double a, double b) {
    cout << "Result: " << a * b << endl;
}

void divide(double a, double b) {
    if (b != 0) {
        cout << "Result: " << a / b << endl;
    }
    else {
        cout << "Error: Division by zero is not allowed." << endl;
    }
}
```

Output:

Arithmetic Operations Menu
1. Addition
2. Subtraction
3. Multiplication
4. Division
5. Exit
Enter your choice (1-5): 1
Enter first number: 15
Enter second number: 15
Result: 30

Arithmetic Operations Menu
1. Addition
2. Subtraction
3. Multiplication
4. Division
5. Exit
Enter your choice (1-5):

Explanation:

1. **Function Prototypes:**
    - void add(double a, double b);: Function to perform addition.
    - void subtract(double a, double b);: Function to perform subtraction.
    - void multiply(double a, double b);: Function to perform multiplication.
    - void divide(double a, double b);: Function to perform division, with a check for division by zero.
2. **Main Function:**
    - **Menu Display:**
        - Presents options for addition, subtraction, multiplication, division, and exit.
    - **User Choice:**
        - Reads user input to select an operation.
    - **Input Validation:**
        - Checks if the user wants to exit (choice == 5). If so, it breaks the loop.
    - **Input Two Numbers:**
        - Reads two numbers from the user.
    - **Switch Statement:**
        - Calls the appropriate function based on user choice.
        - Handles invalid choices by displaying a message.
3. **Arithmetic Operations Functions:**
    - **add():** Outputs the sum of two numbers.
    - **subtract():** Outputs the difference between two numbers.
    - **multiply():** Outputs the product of two numbers.
    - **divide():** Outputs the quotient of two numbers with a check for division by zero.

3.Write a C++ program to prompt the user to input 3 integer values and print these values in forward and reversed order.

To write a C++ program that prompts the user to input three integer values and then prints these values in both forward and reversed order, follow these steps:

1. **Prompt the User** to input three integers.
2. **Store the Input** values in variables.
3. **Print the Values** in the order they were entered.
4. **Print the Values** in reverse order.

Source Code:

```
#include <iostream>
using namespace std;

int main() {
    int num1, num2, num3;

    // Prompt the user to input three integer values
    cout << "Enter three integer values:" << endl;
    cout << "Enter the first value: ";
    cin >> num1;
    cout << "Enter the second value: ";
    cin >> num2;
    cout << "Enter the third value: ";
    cin >> num3;

    // Print the values in forward order
    cout << "Values in forward order: " << num1 << ", " << num2 << ", " << num3 <<
endl;

    // Print the values in reversed order
    cout << "Values in reversed order: " << num3 << ", " << num2 << ", " << num1 <<
endl;

    return 0;
}
```

Output:

Enter three integer values:

Enter the first value: 15

Enter the second value: 20

Enter the third value: 5

Values in forward order: 15, 20, 5

Values in reversed order: 5, 20, 15

Explanation:

1. Include the Required Header:
    - #include <iostream>: This header is necessary for input and output operations using cin and cout.
2. Main Function:
    - Declare Variables:
        - int num1, num2, num3;: These variables will store the three integer values entered by the user.
    - Prompt User for Input:
        - cout << "Enter three integer values:" << endl;: Display a message asking the user to enter three integers.
        - cin >> num1;, cin >> num2;, cin >> num3;: Read the integers entered by the user and store them in the respective variables.
    - Display Values in Forward Order:
        - cout << "Values in forward order: " << num1 << ", " << num2 << ", " << num3 << endl;: Print the values in the order they were entered.
    - Display Values in Reverse Order:
        - cout << "Values in reversed order: " << num3 << ", " << num2 << ", " << num1 << endl;: Print the values in reverse order.
3. Return Statement:
    - return 0;: Indicates that the program completed successfully.

4.Write a program to find greatest from three numbers.

To write a C++ program that finds the greatest of three numbers, follow these steps:

1. **Prompt the User** to input three numbers.
2. **Compare** the numbers to determine the greatest one.
3. **Display** the greatest number.

Source Code:

```cpp
#include <iostream>
using namespace std;

int main() {
    // Declare three variables to store the numbers
    double num1, num2, num3;

    // Prompt the user to enter three numbers
    cout << "Enter the first number: ";
    cin >> num1;
    cout << "Enter the second number: ";
    cin >> num2;
    cout << "Enter the third number: ";
    cin >> num3;

    // Determine the greatest number
    double greatest;

    if (num1 >= num2 && num1 >= num3) {
        greatest = num1;
    }
    else if (num2 >= num1 && num2 >= num3) {
        greatest = num2;
    }
    else {
        greatest = num3;
    }

    // Display the greatest number
    cout << "The greatest number is: " << greatest << endl;

    return 0;
}
```

Output:

Enter the first number: 15

Enter the second number: 50

Enter the third number: 2

The greatest number is: 50

Explanation:

1. **Include the Required Header:**
    - #include <iostream>: This header is necessary for input and output operations using cin and cout.
2. **Main Function:**
    - **Declare Variables:**
        - double num1, num2, num3;: These variables store the three numbers entered by the user. We use double to handle both integer and floating-point numbers.
    - **Prompt User for Input:**
        - cout << "Enter the first number: ";, cin >> num1;: Prompt the user to enter the first number and read it.
        - Similarly, prompt and read the second and third numbers.
    - **Determine the Greatest Number:**
        - Use a series of if-else statements to compare the numbers:
            - if (num1 >= num2 && num1 >= num3): Checks if num1 is greater than or equal to both num2 and num3.
            - else if (num2 >= num1 && num2 >= num3): Checks if num2 is greater than or equal to both num1 and num3.
            - else: If neither of the above conditions is true, num3 is the greatest.
        - Assign the greatest number to the variable greatest.
    - **Display the Greatest Number:**
        - cout << "The greatest number is: " << greatest << endl;: Output the greatest number.
3. **Return Statement:**
    - return 0;: Indicates successful completion of the program.

5.Write a program in C++ to swap two numbers

To write a C++ program that swaps two numbers, you can use various methods. Here, I'll provide a simple example using a temporary variable, which is a common and straightforward approach. Additionally, I'll show how to swap numbers without using a temporary variable by using arithmetic operations.

Using a Temporary Variable

```
#include <iostream>
using namespace std;

int main() {
    // Declare variables to store the numbers
    int a, b, temp;

    // Prompt the user to enter two numbers
    cout << "Enter the first number (a): ";
    cin >> a;
    cout << "Enter the second number (b): ";
    cin >> b;

    // Display the numbers before swapping
    cout << "Before swapping: a = " << a << ", b = " << b << endl;

    // Swap the numbers using a temporary variable
    temp = a;
    a = b;
    b = temp;

    // Display the numbers after swapping
    cout << "After swapping: a = " << a << ", b = " << b << endl;

    return 0;
}
```

Output:

Enter the first number (a): 10

Enter the second number (b): 20

Before swapping: a = 10, b = 20

After swapping: a = 20, b = 10

Swapping Without a Temporary Variable

Here's how you can swap two numbers without using a temporary variable, using arithmetic operations:

```
#include <iostream>
using namespace std;

int main() {
    // Declare variables to store the numbers
    int a, b;

    // Prompt the user to enter two numbers
    cout << "Enter the first number (a): ";
    cin >> a;
    cout << "Enter the second number (b): ";
    cin >> b;

    // Display the numbers before swapping
    cout << "Before swapping: a = " << a << ", b = " << b << endl;

    // Swap the numbers without using a temporary variable
    a = a + b;
    b = a - b;
    a = a - b;

    // Display the numbers after swapping
    cout << "After swapping: a = " << a << ", b = " << b << endl;

    return 0;
}
```

Output:

Enter the first number (a): 15

Enter the second number (b): 20

Before swapping: a = 15, b = 20

After swapping: a = 20, b = 15

Explanation:

1. **Using a Temporary Variable:**
    - **Declare Variables:**
        - int a, b, temp;: Variables to store the two numbers and a temporary variable for swapping.
    - **Input Numbers:**
        - cin >> a;, cin >> b;: Read two numbers from the user.
    - **Swap Operation:**
        - Store the value of a in temp.

        - Assign the value of b to a.
        - Assign the value of temp (original a) to b.
    - **Output Results:**
        - Print the values of a and b before and after swapping.
2. **Swapping Without a Temporary Variable:**
    - **Declare Variables:**
        - int a, b;: Variables to store the two numbers.
    - **Input Numbers:**
        - cin >> a;, cin >> b;: Read two numbers from the user.
    - **Swap Operation:**
        - a = a + b;: Store the sum of a and b in a.
        - b = a - b;: Subtract b from the new value of a to get the original a in b.
        - a = a - b;: Subtract the new value of b from the new value of a to get the original b in a.
    - **Output Results:**
        - Print the values of `a` and `b` before and after swapping.

6.To calculate the area of circle, rectangle and triangle using function overloading.

Function overloading allows you to define multiple functions with the same name but different parameter lists. This is particularly useful when you want to perform similar operations on different types or numbers of inputs. Below is a C++ program that calculates the area of a circle, rectangle, and triangle using function overloading.

Source Code:

```
#include <iostream>
#include<conio.h>
using namespace std;

int area(int, int);
float area(float);
float area(float, float);


int main()
{

    int l, b;
    float r, bs, ht;
    cout << "Enter length and breadth of rectangle:";
    cin >> l >> b;
    cout << "Enter radius of circle:";
    cin >> r;
    cout << "Enter base and height of triangle:";
    cin >> bs >> ht;
    cout << "\nArea of rectangle is " << area(l, b);
    cout << "\nArea of circle is " << area(r);
    cout << "\nArea of triangle is " << area(bs, ht);

}

int area(int l, int b)
{
    return(l * b);
}
float area(float r)
{
    return(3.14 * r * r);
}
float area(float bs, float ht)
{
    return((bs * ht) / 2);
}
```

Output:

Enter length and breadth of rectangle:15

Enter radius of circle:5

Enter base and height of triangle:15

20

Area of rectangle is 300

Area of circle is 78.5

Area of triangle is 150

1. Include Directives and Namespace

- #include <iostream>: Includes the standard input/output stream library which allows for reading from the keyboard and writing to the console.
- #include <conio.h>: This header is used for console input/output functions like getch(), but it is non-standard and not available in all compilers. You might not need this header unless using functions like getch().
- using namespace std;: This directive allows you to use standard library names without qualifying them with std::.

2. Function Declarations

- These are the declarations for three overloaded functions named area:
    - int area(int, int);: Calculates the area of a rectangle, given two integers (length and breadth).
    - float area(float);: Calculates the area of a circle, given one float (radius).
    - float area(float, float);: Calculates the area of a triangle, given two floats (base and height).

3. Main Function

- **Variable Declarations:**
    - int l, b; for length and breadth of the rectangle.
    - float r, bs, ht; for radius of the circle, and base and height of the triangle.
- **User Input:**
    - Prompts the user to input dimensions for the rectangle, circle, and triangle using cin.
- **Function Calls:**

- Calls the appropriate area function based on the shape:
  - area(l, b) for the rectangle.
  - area(r) for the circle.
  - area(bs, ht) for the triangle.
- Outputs the results to the console.

4. Function Definitions

- **Rectangle Area:**
  - int area(int l, int b): Computes the area of a rectangle using the formula length * breadth. The result is an integer.
- **Circle Area:**
  - float area(float r): Computes the area of a circle using the formula π * radius^2. Here, π is approximated as 3.14. The result is a float.
- **Triangle Area:**
  - float area(float bs, float ht): Computes the area of a triangle using the formula 0.5 * base * height. The result is a float.

7. Write a C++ Program to sort an array of numbers in descending order.

Source Code:

```
#include <iostream>
#include<conio.h>
using namespace std;



int main()
{

	int arr[10] = { 34,86,7,90,63,72,14,81,74,18 };
	int n = 10, i, j;
	int temp;

	cout << "Unsorted Array elements:" << endl;
	for (i = 0; i < n; i++)
		cout << arr[i] << "\t";
	cout << endl;

	for (i = 0; i < n; i++)
	{
		for (j = i + 1; j < n; j++)
		{
			if (arr[i] < arr[j])
			{
				temp = arr[i];
				arr[i] = arr[j];
				arr[j] = temp;
			}
		}
	}

	cout << "Sorted (Descending Order) Array elements:" << endl;
	for (i = 0; i < n; i++)
		cout << arr[i] << "\t";
	cout << endl;


}
```

Output:

Unsorted Array elements:
34 86 7 90 63 72 14 81 74 18
Sorted (Descending Order) Array elements:
90 86 81 74 72 63 34 18 14 7

Explanation:

This C++ program sorts an array of integers in descending order using a simple sorting algorithm. It demonstrates how to manipulate and sort arrays and display results using basic C++ constructs.

Key Points

1. **Array Initialization:**

   ```
   int arr[10] = {34, 86, 7, 90, 63, 72, 14, 81, 74, 18};
   ```

   - An array arr of 10 integers is created with specified values.
2. **Display Unsorted Array:**

   ```
   cout << "Unsorted Array elements:" << endl;
   for (i = 0; i < n; i++)
      cout << arr[i] << "\t";
   cout << endl;
   ```

   - This section prints the elements of the array before sorting. The \t adds tab spaces between elements for better readability.
3. **Sorting Algorithm (Descending Order):**

   ```
   for (i = 0; i < n; i++) {
      for (j = i + 1; j < n; j++) {
         if (arr[i] < arr[j]) {
            temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
         }
      }
   }
   ```

   - Uses a nested loop to sort the array in descending order.
   - The outer loop iterates through each element, while the inner loop compares it with subsequent elements.
   - If an element is smaller than a subsequent one, they are swapped. This process places the largest unsorted element at the correct position after each pass.
4. **Display Sorted Array:**

   ```
   cout << "Sorted (Descending Order) Array elements:" << endl;
   for (i = 0; i < n; i++)
      cout << arr[i] << "\t";
   cout << endl;
   ```

   - Prints the elements of the array after sorting, showing them in descending order.

Summary

- **Initialization:** An array of integers is defined and initialized.
- **Unsorted Display:** The array is printed in its original order.
- **Sorting:** A nested loop sorts the array in descending order using comparisons and swaps.
- **Sorted Display:** The array is printed after sorting to show the new order.

8.Write a C++ program to accept and display employee (e_no,e_name,e_designation) details using this pointer

Source Code:

```
#include <iostream>
#include<conio.h>
#include <string.h>
using namespace std;

class Employee {
public:
	int id;
	char name[50];
	char designation[50];
	Employee(int id, char name[50], char designation[50])
	{
		this->id = id;
		strcpy_s(this->name, name);
		strcpy_s(this->designation, designation);
	}
	void display()
	{
		cout << this->id << "  " << this->name << "  " << this->designation <<
endl;
	}
};


int main()
{

	int id;
	char name[50];
	char designation[50];

	cout << "\n Enter employee Id/no : ";
	cin >> id;
	cout << "\n Enter employee name : ";
	cin >> name;
	cout << "\n Enter employee designation : ";
	cin >> designation;

	Employee e1 = Employee(id, name, designation);

	e1.display();


}
```

Output:

Enter employee Id/no : 1

Enter employee name : abc

Enter employee designation : manager
1 abc manager

Explanations:

This C++ program demonstrates the use of classes and basic object-oriented programming principles. It defines an Employee class, creates an object of this class, initializes it with user input, and then displays its details.

1. **Header Inclusions:**

   ```
   #include <iostream>
   #include <conio.h>
   #include <string.h>
   ```

   - #include <iostream> is used for input and output operations (cin and cout).
   - #include <conio.h> is included but not used in this program. It provides console input/output functions, which are not needed here.
   - #include <string.h> is used for string manipulation functions (strcpy_s).

2. **Employee Class Definition:**

   ```
   class Employee {
   public:
      int id;
      char name[50];
      char designation[50];

      Employee(int id, char name[50], char designation[50]) {
         this->id = id;
         strcpy_s(this->name, name);
         strcpy_s(this->designation, designation);
      }

      void display() {
         cout << this->id << "  " << this->name << "  " << this->designation << endl;
      }
   };
   ```

   - **Data Members:**
     - id: An integer to store the employee's ID.
     - name: A character array (C-string) to store the employee's name.
     - designation: A character array (C-string) to store the employee's designation.
   - **Constructor:**

        - The constructor initializes the id, name, and designation data members. It uses strcpy_s to copy the strings into the member variables. strcpy_s is a safer version of strcpy that helps prevent buffer overflow.
    - **Member Function display:**
        - This function prints the id, name, and designation of the employee to the console.

3. **Main Function:**

```
int main() {
   int id;
   char name[50];
   char designation[50];

   cout << "\n Enter employee Id/no : ";
   cin >> id;
   cout << "\n Enter employee name : ";
   cin >> name;
   cout << "\n Enter employee designation : ";
   cin >> designation;

   Employee e1 = Employee(id, name, designation);
   e1.display();
}
```

    - **Input Collection:**
        - The program prompts the user to enter an employee's id, name, and designation. These are read from the standard input using cin.
    - **Object Creation and Initialization:**
        - An Employee object e1 is created and initialized with the provided values using the constructor.
    - **Displaying Object Data:**
        - The display method of the Employee object e1 is called to print the employee details to the console.

Additional Notes

- **strcpy_s Usage:** This function is used to safely copy strings, helping prevent buffer overflow errors. It is a safer alternative to strcpy.
- **conio.h Inclusion:** The conio.h header is not utilized in this program and can be removed.

9.Write a C++ program to print area of circle, square and rectangle using inline function.

Source Code:

```
#include <iostream>
#include<conio.h>
#include <string.h>
using namespace std;

inline float areaOfCircle(float r) {

	return 3.14 * r * r;
}

inline float areaOfRectangle(float l, float b) {

	return l * b;
}

inline float areaOfSquare(float s) {

	return s * s;
}


int main()
{
	cout << "\nArea of circle is : " << areaOfCircle(15);
		cout << "\nArea of rectangle is : " << areaOfRectangle(12, 15);
		cout << "\nArea of square is : " << areaOfSquare(30);
}
```

Output:

Area of circle is : 706.5

Area of rectangle is : 180

Area of square is : 900

Program Overview

This C++ program calculates and displays the area of different shapes (circle, rectangle, and square) using inline functions for each calculation.

Key Points

1. **Header Inclusions:**

#include <iostream>

#include <conio.h>

#include <string.h>

- #include <iostream> is used for input and output operations (e.g., cout).
- #include <conio.h> is included but not used in this program. It is typically used for console input/output functions.
- #include <string.h> is also not used in this program. It provides functions for string manipulation.

2. **Inline Functions:** Inline functions are used to suggest to the compiler to insert the function's body at the point of each function call, which can optimize performance for small, frequently called functions.

```
inline float areaOfCircle(float r) {
   return 3.14 * r * r;
}
```

- **Function:** areaOfCircle calculates the area of a circle using the formula Area=π×r2\text{Area} = \pi \times r^2Area=π×r2.
- **Parameter:** r is the radius of the circle.
- **Return Type:** float, representing the area.

```
inline float areaOfRectangle(float l, float b) {
   return l * b;
}
```

- **Function:** areaOfRectangle calculates the area of a rectangle using the formula Area=l×b\text{Area} = l \times bArea=l×b.
- **Parameters:** l is the length and b is the breadth of the rectangle.
- **Return Type:** float, representing the area.

```
inline float areaOfSquare(float s) {
   return s * s;
}
```

- **Function:** areaOfSquare calculates the area of a square using the formula Area=s2\text{Area} = s^2Area=s2.
- **Parameter:** s is the side length of the square.
- **Return Type:** float, representing the area.

3. **Main Function:**

```
int main() {
   cout << "\nArea of circle is : " << areaOfCircle(15);
   cout << "\nArea of rectangle is : " << areaOfRectangle(12, 15);
   cout << "\nArea of square is : " << areaOfSquare(30);
}
```

- **Area Calculations and Display:**
  - Calls areaOfCircle(15) to compute the area of a circle with radius 15.
  - Calls areaOfRectangle(12, 15) to compute the area of a rectangle with length 12 and breadth 15.
  - Calls areaOfSquare(30) to compute the area of a square with side length 30.
- **Output:** The results of these calculations are printed to the console using cout.

Additional Notes

- **Precision of Pi:** The value of pi is hard-coded as 3.14, which is a rough approximation. For more precision, you could use M_PI from the <cmath> library.
- **Unused Headers:** The conio.h and string.h headers are not used in this program and can be removed to clean up the code.

10.Write a C++ program to read an integer n and prints the factorial of n.(using recursive function)

Source Code:

```
#include <iostream>
#include<conio.h>
#include <string.h>
using namespace std;

int factorial(int n);

int main()
{
    int n;
    cout << "Enter a positive integer: ";
    cin >> n;
    cout << "Factorial of " << n << " = " << factorial(n);

}

int factorial(int n) {
    if (n > 1)
        return n * factorial(n - 1);
    else
        return 1;
}
```

Output:

Enter a positive integer: 5

Factorial of 5 = 120

Explanation:

This program prompts the user for a positive integer and then computes and displays its factorial using a recursive function.

1. **Header Inclusions:**

#include <iostream>

#include <conio.h>

#include <string.h>

- #include <iostream>: Includes the standard input/output stream library, which provides the cin and cout objects for input and output operations.

- #include <conio.h> and #include <string.h>: These headers are not used in this program and can be omitted. conio.h is typically used for console input/output functions, while string.h provides functions for string manipulation.

2. **Function Declaration:**

```
int factorial(int n);
```

- Declares a function factorial that takes an integer n as a parameter and returns an integer. The function will compute the factorial of n.

3. **Main Function:**

```
int main()
{
   int n;
   cout << "Enter a positive integer: ";
   cin >> n;
   cout << "Factorial of " << n << " = " << factorial(n);
}
```

- Prompts the user to enter a positive integer and stores this input in the variable n.
- Calls the factorial function with the input n and displays the result using cout.

4. **Factorial Function Definition:**

```
int factorial(int n) {
   if (n > 1)
      return n * factorial(n - 1);
   else
      return 1;
}
```

- **Recursive Definition:**
  - **Base Case:** If n is not greater than 1 (i.e., n is 1 or 0), the function returns 1. This is the terminating condition for the recursion.
  - **Recursive Case:** If n is greater than 1, the function returns n multiplied by the result of factorial(n - 1). This means the function calls itself with n - 1, thus reducing the problem size with each call.

How It Works

1. **User Input:**
    - The user is asked to input a positive integer.
    - The integer is read from the standard input using cin.
2. **Factorial Calculation:**
    - The factorial function calculates the factorial of the input number recursively.
    - For example, to calculate factorial(4), the function performs:
        - 4 * factorial(3)
        - 3 * factorial(2)
        - 2 * factorial(1)
        - factorial(1) returns 1, so the final result is 4 * 3 * 2 * 1 = 24.
3. **Output:**
    - The result of the factorial calculation is printed to the console.

Example

If the user inputs 5, the factorial calculation would proceed as follows:

- factorial(5) returns 5 * factorial(4)
- factorial(4) returns 4 * factorial(3)
- factorial(3) returns 3 * factorial(2)
- factorial(2) returns 2 * factorial(1)
- factorial(1) returns 1
- Combining these, 5 * 4 * 3 * 2 * 1 = 120, so the output will be Factorial of 5 = 120.

Summary

- **Functionality:** The program computes the factorial of a given positive integer using recursion.
- **User Interaction:** Prompts the user for input and displays the result.
- **Recursive Approach:** Utilizes recursion to break down the factorial calculation into simpler subproblems.

11.Write a C++ program to compute the sum of the specified number of Prime numbers.

Source Code:

```
#include<iostream>
#include<conio.h>
using namespace std;

int main() {

    int num, i, count, sum = 0, limit;

    cout << "\nEnter the maximum limit : ";
    cin >> limit;

    for (num = 1; num <= limit; num++) {

        count = 0;

        for (i = 2; i <= num / 2; i++) {
            if (num % i == 0) {
                count++;
                break;
            }
        }

        if (count == 0 && num != 1)
            sum = sum + num;
    }
    cout << "Sum of prime numbers is: " << sum;
}
```

Output:

Enter the maximum limit : 10

Sum of prime numbers is: 17

Explanation:

This C++ program prompts the user to enter a maximum limit and then calculates the sum of all prime numbers from 1 up to that limit. It uses basic loops and conditional statements to determine if a number is prime.

Key Components

Header Inclusions:

#include<iostream>

```
#include<conio.h>
using namespace std;
```

- #include<iostream>: Includes the input/output stream library for using cin and cout.
- #include<conio.h>: Included but not used in this program. It’s typically used for console input/output functions.
- using namespace std;: Allows usage of standard library features without prefixing them with std::.

Main Function:

```
int main() {
   int num, i, count, sum = 0, limit;

   cout << "\nEnter the maximum limit : ";
   cin >> limit;

   for (num = 1; num <= limit; num++) {
      count = 0;

      for (i = 2; i <= num / 2; i++) {
         if (num % i == 0) {
            count++;
            break;
         }
      }

      if (count == 0 && num != 1)
         sum = sum + num;
   }
   cout << "Sum of prime numbers is: " << sum;
```

}

Variable Initialization:

- num: Used to iterate through numbers from 1 to limit.
- i: Loop variable for checking factors of num.
- count: Used to count the number of factors of num. It helps to determine if num is prime.
- sum: Accumulates the sum of all prime numbers found.
- limit: Stores the user-specified maximum limit up to which prime numbers are to be considered.

Input from User:

```
cout << "\nEnter the maximum limit : ";
cin >> limit;
```

- Prompts the user to enter the maximum limit and reads it using cin.

Prime Number Detection:

```
for (num = 1; num <= limit; num++) {
   count = 0;

   for (i = 2; i <= num / 2; i++) {
      if (num % i == 0) {
         count++;
         break;
      }
   }

   if (count == 0 && num != 1)
      sum = sum + num;
}
```

- Outer loop iterates from 1 to limit to check each number (num).

- Inner loop checks if num is divisible by any number from 2 to num / 2. If num is divisible, it is not a prime number, and count is incremented.
- If count remains 0 and num is not 1 (since 1 is not a prime number), then num is prime, and it is added to sum.

Output Result:

cout << "Sum of prime numbers is: " << sum;

- Prints the sum of all prime numbers up to the specified limit.

12.Write a program in C++ to demonstrate the manipulators: endl, setw and setfill

Source Code:

```
#include <iostream>
#include <iomanip> // For manipulators like setw, setfill
using namespace std;

int main() {
    // Displaying integers with formatting
    cout << "Displaying integers with formatting:" << endl;

    int num1 = 123;
    int num2 = 4567;
    int num3 = 89;

    // Without manipulators
    cout << num1 << " " << num2 << " " << num3 << endl;

    // Using setw and setfill to format the output
    cout << "\nUsing setw and setfill:" << endl;

    // setw sets the width of the next output field
    // setfill sets the character used to fill the width if the data is shorter than
the width
    cout << setfill('-') << setw(10) << num1 << setw(10) << num2 << setw(10) << num3
<< endl;

    // Reset setfill to space and use setw for alignment
    cout << setfill(' ') << setw(10) << num1 << setw(10) << num2 << setw(10) << num3
<< endl;

    // Using endl to move to the next line
    cout << "\nUsing endl to end lines:" << endl;

    cout << "Line 1" << endl;
    cout << "Line 2" << endl;
    cout << "Line 3" << endl;

    // Combining manipulators
    cout << "\nCombining manipulators:" << endl;
    cout << setw(15) << setfill('*') << "Formatted Text" << endl;
    cout << setw(10) << setfill('#') << 42 << endl;

    return 0;
}
```

Output:

Displaying integers with formatting:

123 4567 89

Using setw and setfill:

-------123------4567--------89

123 4567 89

Using endl to end lines:

Line 1

Line 2

Line 3

Combining manipulators:

*Formatted Text

########42

Explanation

1. **Header Inclusions:**

   #include <iostream>
   #include <iomanip> // For manipulators like setw, setfill

   - #include <iostream>: Provides functionalities for input and output.
   - #include <iomanip>: Includes manipulators like setw, setfill, etc.
2. **Manipulators Demonstration:**
   - **endl:**

     cout << "Line 1" << endl;

     - Inserts a newline character and flushes the output buffer.
   - **setw:**

     cout << setw(10) << num1;

     - Sets the width of the next output field. If the value is shorter than the specified width, the field is padded (by default with spaces).
   - **setfill:**

     cout << setfill('-') << setw(10) << num1;

        - Sets the fill character for the width of the field when the data is shorter than the specified width. In this case, it uses '-' to fill the extra space.
3. **Program Output:**
    - **Without Manipulators:** Shows integers without any formatting.
    - **With setw and setfill:** Demonstrates how to align output fields and fill extra space with a specified character.
    - **Using endl:** Moves to a new line and flushes the stream.
    - **Combining Manipulators:** Shows how to use setw and setfill together for formatted output.

13. Write a C++ program to demonstrate the use of Friend function in class.

A friend function in C++ is a special function that is not a member of a class but is allowed to access the private and protected members of that class. This is useful when you want to provide certain functions with access to the internals of a class without making them member functions.

Here's a C++ program that demonstrates the use of a friend function:

Source Code:

```
#include <iostream>
using namespace std;

class Rectangle {
private:
    int width;
    int height;

public:
    // Constructor
    Rectangle(int w, int h) : width(w), height(h) {}

    // Declare friend function
    friend void displayArea(Rectangle rect);

    // Declare another friend function
    friend void setDimensions(Rectangle& rect, int w, int h);
};

// Friend function definition
void displayArea(Rectangle rect) {
    int area = rect.width * rect.height;
    cout << "Area of the rectangle: " << area << endl;
}

// Another friend function definition
void setDimensions(Rectangle& rect, int w, int h) {
    rect.width = w;
    rect.height = h;
}

int main() {
    // Create a Rectangle object
    Rectangle rect(10, 5);

    // Display area of the rectangle
    displayArea(rect);

    // Change dimensions using the friend function
    setDimensions(rect, 15, 7);

    // Display area again after changing dimensions
    displayArea(rect);

    return 0;
```

}

Output:

Area of the rectangle: 50

Area of the rectangle: 105

Explanation

1. **Class Definition:**

```
class Rectangle {
private:
   int width;
   int height;

public:
   // Constructor
   Rectangle(int w, int h) : width(w), height(h) {}

   // Declare friend function
   friend void displayArea(Rectangle rect);
   friend void setDimensions(Rectangle &rect, int w, int h);
};
```

   - Rectangle class has private members width and height.
   - The constructor initializes these members.
   - The friend keyword is used to declare displayArea and setDimensions as friend functions. This allows these functions to access the private members of Rectangle.

2. **Friend Function Definitions:**

```
void displayArea(Rectangle rect) {
   int area = rect.width * rect.height;
   cout << "Area of the rectangle: " << area << endl;
}

void setDimensions(Rectangle &rect, int w, int h) {
   rect.width = w;
   rect.height = h;
}
```

- displayArea computes and prints the area of the rectangle using its private members.
- setDimensions modifies the private members of Rectangle directly.

3. **Main Function:**

```
int main() {
    Rectangle rect(10, 5);

    displayArea(rect);

    setDimensions(rect, 15, 7);

    displayArea(rect);

    return 0;
}
```

- Creates an instance of Rectangle with initial dimensions.
- Calls displayArea to print the area.
- Calls setDimensions to change the dimensions.
- Calls displayArea again to print the updated area.

14.Write a C++ program to sort an Array in Ascending order.

C++ Program Using Selection Sort

Here's a C++ program that demonstrates how to sort an array in ascending order using the Selection Sort algorithm

Source Code:

```
#include <iostream>
using namespace std;

// Function to perform Selection Sort
void selectionSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        // Find the index of the smallest element in the unsorted part
        int minIndex = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }
        // Swap the found minimum element with the first element of the unsorted
part
        if (minIndex != i) {
            int temp = arr[i];
            arr[i] = arr[minIndex];
            arr[minIndex] = temp;
        }
    }
}

// Function to print an array
void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}

int main() {
    int arr[] = { 64, 25, 12, 22, 11 };
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Original array: ";
    printArray(arr, n);

    selectionSort(arr, n);

    cout << "Sorted array: ";
    printArray(arr, n);

    return 0;
}
```

Output:

Original array: 64 25 12 22 11

Sorted array: 11 12 22 25 64

Explanation

1. **Selection Sort Function:**

```
void selectionSort(int arr[], int n) {
   for (int i = 0; i < n - 1; i++) {
      int minIndex = i; // Assume the current position is the smallest
      for (int j = i + 1; j < n; j++) {
         if (arr[j] < arr[minIndex]) {
            minIndex = j; // Update the index of the smallest element
         }
      }
      // Swap the smallest element found with the element at the current position
      if (minIndex != i) {
         int temp = arr[i];
         arr[i] = arr[minIndex];
         arr[minIndex] = temp;
      }
   }
}
```

   - **Outer Loop:** Iterates over each element of the array.
   - **Inner Loop:** Finds the smallest element in the unsorted part of the array.
   - **Swapping:** Once the smallest element is found, it is swapped with the element at the current position of the outer loop.

2. **Print Array Function:**

```
void printArray(int arr[], int size) {
   for (int i = 0; i < size; i++) {
      cout << arr[i] << " ";
   }
   cout << endl;
}
```

   - Prints each element of the array, separated by spaces.

3. **Main Function:**

```
int main() {
    int arr[] = {64, 25, 12, 22, 11};
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Original array: ";
    printArray(arr, n);

    selectionSort(arr, n);

    cout << "Sorted array: ";
    printArray(arr, n);

    return 0;
}
```

   - Initializes an array with unsorted values.
   - Prints the original array.
   - Calls selectionSort to sort the array.
   - Prints the sorted array.

15.Write a C++ program to create a class Person which contains data members as P_Name, P_City, P_Contact_Number. Write member functions to accept and display five Persons information.

Source Code:

```
#include <iostream>
#include <string>
using namespace std;

class Person {
private:
    string P_Name;
    string P_City;
    string P_Contact_Number;

public:
    // Function to accept person details
    void acceptDetails() {
        cout << "Enter Name: ";
        getline(cin, P_Name);
        cout << "Enter City: ";
        getline(cin, P_City);
        cout << "Enter Contact Number: ";
        getline(cin, P_Contact_Number);
    }

    // Function to display person details
    void displayDetails() const {
        cout << "Name: " << P_Name << endl;
        cout << "City: " << P_City << endl;
        cout << "Contact Number: " << P_Contact_Number << endl;
    }
};

int main() {
    const int numPersons = 5;
    Person persons[numPersons];

    // Accept details for each person
    for (int i = 0; i < numPersons; ++i) {
        cout << "Enter details for Person " << (i + 1) << ":\n";
        persons[i].acceptDetails();
        cout << endl;
    }

    // Display details for each person
    cout << "\nDisplaying details of all persons:\n";
    for (int i = 0; i < numPersons; ++i) {
        cout << "Person " << (i + 1) << ":\n";
        persons[i].displayDetails();
        cout << endl;
    }

    return 0;
}
```

Output:

Enter details for Person 1:

Enter Name: abc

Enter City: pune

Enter Contact Number: 7894561230

Enter details for Person 2:

Enter Name: pqr

Enter City: mumbai

Enter Contact Number: 7891234560

Enter details for Person 3:

Enter Name: xyz

Enter City: nagpur

Enter Contact Number: 9638527410

Enter details for Person 4:

Enter Name: lmn

Enter City: pune

Enter Contact Number: 45612378590

Enter details for Person 5:

Enter Name: acd

Enter City: mumbai

Enter Contact Number: 8745219630

Displaying details of all persons:

Person 1:

Name: abc

City: pune

Contact Number: 7894561230

Person 2:

Name: pqr

City: mumbai

Contact Number: 7891234560

Person 3:

Name: xyz

City: nagpur

Contact Number: 9638527410

Person 4:

Name: lmn

City: pune

Contact Number: 45612378590

Person 5:

Name: acd

City: mumbai

Contact Number: 8745219630

Explanation

Class Definition:

- **Private Members:** P_Name, P_City, and P_Contact_Number store the name, city, and contact number of a person.
- **Public Methods:**
  - acceptDetails(): Reads details from the user using getline() to allow spaces in the input.
  - displayDetails(): Displays the details of the person.

Main Function:

- **Creating an Array of Person Objects:** The array persons holds 5 instances of the Person class.
- **Accepting Details:** The program prompts the user to enter details for each person.
- **Displaying Details:** The program prints out the details for all persons after data entry.

16.Write a C++ program to create a class Student which contains data members as Roll_Number, Stud_Name, Marks in five subjects. Write member functions to accept Student information. Display all details of student along with a percentage and class obtained depending on percentage. (Use array of objects)

C++ program that defines a Student class with data members for Roll_Number, Stud_Name, and Marks in five subjects. The program includes member functions to accept student information, display all details, and calculate the percentage and class obtained based on the percentage.

Source Code:

```
#include <iostream>
#include <string>
using namespace std;

class Student {
private:
    int Roll_Number;
    string Stud_Name;
    float Marks[5];

public:
    // Function to accept student details
    void acceptDetails() {
        cout << "Enter Roll Number: ";
        cin >> Roll_Number;
        cin.ignore(); // To ignore the newline character left in the input buffer

        cout << "Enter Student Name: ";
        getline(cin, Stud_Name);

        cout << "Enter marks in 5 subjects:\n";
        for (int i = 0; i < 5; ++i) {
            cout << "Subject " << (i + 1) << ": ";
            cin >> Marks[i];
        }
    }

    // Function to calculate and return the total marks
    float calculateTotalMarks() const {
        float total = 0;
        for (int i = 0; i < 5; ++i) {
            total += Marks[i];
        }
        return total;
    }

    // Function to calculate and return the percentage
    float calculatePercentage() const {
        float total = calculateTotalMarks();
        return (total / 500) * 100; // Assuming each subject is out of 100 marks
    }
```

```
    // Function to determine the class based on percentage
    string determineClass() const {
        float percentage = calculatePercentage();
        if (percentage >= 60) {
            return "First Class";
        }
        else if (percentage >= 50) {
            return "Second Class";
        }
        else if (percentage >= 40) {
            return "Third Class";
        }
        else {
            return "Fail";
        }
    }

    // Function to display student details
    void displayDetails() const {
        cout << "\nRoll Number: " << Roll_Number << endl;
        cout << "Student Name: " << Stud_Name << endl;

        cout << "Marks:\n";
        for (int i = 0; i < 5; ++i) {
            cout << "Subject " << (i + 1) << ": " << Marks[i] << endl;
        }

        float percentage = calculatePercentage();
        cout << "Percentage: " << percentage << "%" << endl;
        cout << "Class Obtained: " << determineClass() << endl;
    }
};

int main() {
    const int numStudents = 3;
    Student students[numStudents];

    // Accept details for each student
    for (int i = 0; i < numStudents; ++i) {
        cout << "Enter details for Student " << (i + 1) << ":\n";
        students[i].acceptDetails();
        cout << endl;
    }

    // Display details for each student
    cout << "\nDisplaying details of all students:\n";
    for (int i = 0; i < numStudents; ++i) {
        cout << "Details of Student " << (i + 1) << ":\n";
        students[i].displayDetails();
        cout << endl;
    }

    return 0;
}
```

Output:

Enter details for Student 1:

Enter Roll Number: 1

Enter Student Name: kiran

Enter marks in 5 subjects:

Subject 1: 55

Subject 2: 60

Subject 3: 70

Subject 4: 46

Subject 5: 77

Enter details for Student 2:

Enter Roll Number: 2

Enter Student Name: akash

Enter marks in 5 subjects:

Subject 1: 77

Subject 2: 66

Subject 3: 88

Subject 4: 75

Subject 5: 56

Enter details for Student 3:

Enter Roll Number: 89

Enter Student Name: 78

Enter marks in 5 subjects:

Subject 1: 86

Subject 2: 82

Subject 3: 84

Subject 4: 81

Subject 5: 79

Displaying details of all students:

Details of Student 1:

Roll Number: 1

Student Name: kiran

Marks:

Subject 1: 55

Subject 2: 60

Subject 3: 70

Subject 4: 46

Subject 5: 77

Percentage: 61.6%

Class Obtained: First Class

Details of Student 2:

Roll Number: 2

Student Name: akash

Marks:

Subject 1: 77

Subject 2: 66

Subject 3: 88

Subject 4: 75

Subject 5: 56

Percentage: 72.4%

Class Obtained: First Class

Details of Student 3:

Roll Number: 89

Student Name: 78

Marks:

Subject 1: 86

Subject 2: 82

Subject 3: 84

Subject 4: 81

Subject 5: 79

Percentage: 82.4%

Class Obtained: First Class

Explanation

1. **Class Definition:**
    - **Private Members:**
        - Roll_Number: Student's roll number.
        - Stud_Name: Student's name.
        - Marks[5]: Array to store marks in five subjects.
    - **Public Methods:**
        - acceptDetails(): Reads student details including roll number, name, and marks in five subjects.
        - calculateTotalMarks(): Computes the total marks.
        - calculatePercentage(): Computes the percentage based on total marks.
        - determineClass(): Determines the class based on the percentage.
        - displayDetails(): Displays all the student details, percentage, and class obtained.
2. **Main Function:**
    - **Array of Student Objects:** The array students holds the details of multiple Student objects.

- **Accept Details:** Iterates through the array to accept details for each student.
- **Display Details:** Iterates through the array to display details for each student.

17.Create a C++ class for a student object with the following attributes—roll no, name, number of subjects, marks of subjects. Write member function for accepting marks and display all information of student along with total and Percentage. Display marklist with Use of manipulators.

create a C++ class Student with attributes for roll number, name, number of subjects, and marks of subjects. We will include member functions to accept the marks and display all the student's information, including the total and percentage. We'll also use manipulators to format the display of the marklist.

Source Code:

```
#include <iostream>
#include <iomanip>
#include <string>
using namespace std;

class Student {
private:
    int rollNo;
    string name;
    int numSubjects;
    float* marks; // Pointer to dynamically allocate marks array

public:
    // Constructor
    Student() : rollNo(0), name(""), numSubjects(0), marks(nullptr) {}

    // Destructor
    ~Student() {
        delete[] marks; // Free allocated memory
    }

    // Function to accept student details
    void acceptDetails() {
        cout << "Enter Roll Number: ";
        cin >> rollNo;
        cin.ignore(); // Ignore newline left in the buffer

        cout << "Enter Name: ";
        getline(cin, name);

        cout << "Enter Number of Subjects: ";
        cin >> numSubjects;

        // Allocate memory for marks
        marks = new float[numSubjects];

        cout << "Enter marks for " << numSubjects << " subjects:\n";
        for (int i = 0; i < numSubjects; ++i) {
            cout << "Subject " << (i + 1) << ": ";
            cin >> marks[i];
        }
    }
```

```
    // Function to calculate total marks
    float calculateTotal() const {
        float total = 0;
        for (int i = 0; i < numSubjects; ++i) {
            total += marks[i];
        }
        return total;
    }

    // Function to calculate percentage
    float calculatePercentage() const {
        return (calculateTotal() / (numSubjects * 100)) * 100;
    }

    // Function to display student details
    void displayDetails() const {
        cout << "\nRoll Number: " << rollNo << endl;
        cout << "Name: " << name << endl;
        cout << "Number of Subjects: " << numSubjects << endl;

        // Display marks in a tabular format
        cout << left << setw(20) << "Subject" << setw(15) << "Marks" << endl;
        cout << "-----------------------------" << endl;
        for (int i = 0; i < numSubjects; ++i) {
            cout << left << setw(20) << ("Subject " + to_string(i + 1))
                 << setw(15) << marks[i] << endl;
        }

        // Display total and percentage
        cout << "\nTotal Marks: " << calculateTotal() << endl;
        cout << "Percentage: " << fixed << setprecision(2) << calculatePercentage()
<< "%" << endl;
    }
};

int main() {
    Student student;

    student.acceptDetails();
    student.displayDetails();

    return 0;
}
```

Output:

Enter Roll Number: 1

Enter Name: Suraj

Enter Number of Subjects: 5

Enter marks for 5 subjects:

Subject 1: 74

Subject 2: 70

Subject 3: 80

Subject 4: 85

Subject 5: 78

Roll Number: 1

Name: Suraj

Number of Subjects: 5

Subject Marks

Subject 1 74

Subject 2 70

Subject 3 80

Subject 4 85

Subject 5 78

Total Marks: 387

Percentage: 77.40%

Explanation

1. **Class Definition:**

```
class Student {
private:
    int rollNo;
    string name;
    int numSubjects;
    float* marks; // Pointer to dynamically allocate marks array

public:
    Student();
```

```
    ~Student();
    void acceptDetails();
    float calculateTotal() const;
    float calculatePercentage() const;
    void displayDetails() const;
};
```

- **Private Members:**
  - rollNo: Student's roll number.
  - name: Student's name.
  - numSubjects: Number of subjects.
  - marks: Pointer to dynamically allocate an array of marks.
- **Public Methods:**
  - acceptDetails(): Reads student details including roll number, name, number of subjects, and marks.
  - calculateTotal(): Computes the total marks.
  - calculatePercentage(): Computes the percentage based on total marks.
  - displayDetails(): Displays all student details, including marks in a formatted table, total marks, and percentage.

2. **Constructor and Destructor:**

```
Student::Student() : rollNo(0), name(""), numSubjects(0), marks(nullptr) {}

Student::~Student() {
    delete[] marks; // Free allocated memory
}
```

- **Constructor:** Initializes member variables.
- **Destructor:** Frees the dynamically allocated memory for the marks array.

3. **Accept Details Function:**

```
void Student::acceptDetails() {
    cout << "Enter Roll Number: ";
    cin >> rollNo;
    cin.ignore(); // Ignore newline left in the buffer

    cout << "Enter Name: ";
    getline(cin, name);

    cout << "Enter Number of Subjects: ";
```

```
    cin >> numSubjects;

    marks = new float[numSubjects]; // Allocate memory for marks

    cout << "Enter marks for " << numSubjects << " subjects:\n";
    for (int i = 0; i < numSubjects; ++i) {
        cout << "Subject " << (i + 1) << ": ";
        cin >> marks[i];
    }
}
```

4. **Display Details Function:**

```
void Student::displayDetails() const {
    cout << "\nRoll Number: " << rollNo << endl;
    cout << "Name: " << name << endl;
    cout << "Number of Subjects: " << numSubjects << endl;

    cout << left << setw(20) << "Subject" << setw(15) << "Marks" << endl;
    cout << "------------------------------" << endl;
    for (int i = 0; i < numSubjects; ++i) {
        cout << left << setw(20) << ("Subject " + to_string(i + 1))
             << setw(15) << marks[i] << endl;
    }

    cout << "\nTotal Marks: " << calculateTotal() << endl;
    cout << "Percentage: " << fixed << setprecision(2) << calculatePercentage() << "%" <<
endl;
}
```

   - **Formatting Output:** Uses setw from <iomanip> to format the table of marks, and fixed and setprecision to format the percentage.

5. **Main Function:**

```
int main() {
    Student student;

    student.acceptDetails();
    student.displayDetails();

    return 0;
}
```

- Creates a Student object, accepts details from the user, and then displays the details.

18.Write a class Complex (real, img) along with appropriate constructors. Also write appropriate functions to overload '+' and '-' operator.

To create a Complex class in C++ that supports operator overloading for + and -, you'll need to follow these steps:

1. **Define the Complex class** with private members for real and imaginary parts.
2. **Implement constructors** to initialize the complex numbers.
3. **Overload the + and - operators** to perform addition and subtraction of complex numbers.
4. **Implement a method** to display the complex number in a user-friendly format.

Source Code:

```
#include <iostream>
using namespace std;

class Complex {
private:
    float real;
    float img;

public:
    // Default constructor
    Complex() : real(0), img(0) {}

    // Parameterized constructor
    Complex(float r, float i) : real(r), img(i) {}

    // Overload the + operator
    Complex operator+(const Complex& other) const {
        return Complex(real + other.real, img + other.img);
    }

    // Overload the - operator
    Complex operator-(const Complex& other) const {
        return Complex(real - other.real, img - other.img);
    }

    // Function to display complex number
    void display() const {
        if (img >= 0)
            cout << real << " + " << img << "i" << endl;
        else
            cout << real << " - " << -img << "i" << endl;
    }
};

int main() {
    // Create two complex numbers
    Complex c1(3.5, 2.5);
```

```
    Complex c2(1.5, 4.5);

    // Display the complex numbers
    cout << "Complex number 1: ";
    c1.display();
    cout << "Complex number 2: ";
    c2.display();

    // Perform addition and subtraction
    Complex sum = c1 + c2;
    Complex diff = c1 - c2;

    // Display the results
    cout << "Sum: ";
    sum.display();
    cout << "Difference: ";
    diff.display();

    return 0;
}
```

Output:

Complex number 1: 3.5 + 2.5i

Complex number 2: 1.5 + 4.5i

Sum: 5 + 7i

Difference: 2 - 2i

Explanation:

1. **Class Definition:**
   - `Complex` class has private members `real` and `img` for the real and imaginary parts of the complex number.
   - There are two constructors: a default constructor that initializes both members to `0` and a parameterized constructor that allows initialization with specific values.
2. **Operator Overloading:**
   - The `+` operator is overloaded to add the real and imaginary parts of two complex numbers.
   - The `-` operator is overloaded to subtract the real and imaginary parts of two complex numbers.
3. **Display Method:**
   - The `display` method formats the output so that if the imaginary part is non-negative, it is shown as `+`, otherwise, it is shown as `-`.
4. **Main Function:**
   - Two `Complex` objects are created and initialized.
   - The `+` and `-` operators are used to add and subtract these objects.

- Results are displayed using the `display` method.

This code will create and manipulate complex numbers and display their results in a readable format.

19.Write a C++ program to find volume of cube, cylinder and rectangle using function overloading

To write a C++ program that calculates the volume of a cube, cylinder, and rectangle using function overloading, follow these steps:

1. **Define functions** with the same name but different parameters for calculating the volume of each shape.
2. **Implement the volume calculations** inside these functions.
3. **Use function overloading** to differentiate between different shapes based on their parameters.

Source Code:

```
#include <iostream>
#include <cmath>  // For M_PI if you prefer a more accurate value of π

using namespace std;

// Function to calculate the volume of a cube
float findVolume(float s) {
    return s * s * s;
}

// Function to calculate the volume of a cylinder
float findVolume(float r, float h) {
    return 3.14f * r * r * h;  // Use 3.14f for float literal
}

// Function to calculate the volume of a rectangular prism (cuboid)
float findVolume(float l, float w, float h) {
    return l * w * h;
}

// Main function
int main() {
    cout << "Volume of cube is: " << findVolume(10.0f) << endl;
    cout << "Volume of cylinder is: " << findVolume(5.0f, 10.0f) << endl;
    cout << "Volume of rectangular prism is: " << findVolume(20.0f, 10.0f, 10.0f) <<
endl;

    cout << "Press Enter to exit...";
    cin.ignore();  // Ignore any leftover input
    cin.get();     // Wait for Enter key press

    return 0;
}
```

Output:

Volume of cube is: 1000

Volume of cylinder is: 785

Volume of rectangular prism is: 2000

Press Enter to exit...

Explanation:

1. **Function Overloading:**
   - volume(float side): Calculates the volume of a cube. Here, side is the length of one side of the cube.
   - volume(float radius, float height): Calculates the volume of a cylinder. Here, radius is the radius of the base, and height is the height of the cylinder.
   - volume(float length, float width, float height): Calculates the volume of a rectangular prism (or cuboid). Here, length, width, and height are the dimensions of the rectangular prism.
2. **Volume Calculations:**
   - **Cube:** V=side3V = \text{side}^3V=side3
   - **Cylinder:** V=π×radius2×heightV = \pi \times \text{radius}^2 \times \text{height}V=π×radius2×height. M_PI from <cmath> provides the value of π.
   - **Rectangular Prism (Cuboid):** V=length×width×heightV = \text{length} \times \text{width} \times \text{height}V=length×width×height
3. **Main Function:**
   - Prompts the user for dimensions of each shape.
   - Calls the appropriate volume function based on the parameters provided.
   - Displays the calculated volumes.

20. Write a C++ program to find area of triangle, circle, and rectangle using function overloading.

Source Code:

```cpp
#include <iostream>
#include <cmath>  // For mathematical functions

using namespace std;

// Define a constant for π if M_PI is not available
const float PI = 3.14159265358979323846f;  // More accurate value of π

// Function to calculate the area of a rectangle
int area(int length, int breadth) {
    return length * breadth;
}

// Function to calculate the area of a circle
float area(float radius) {
    return PI * radius * radius;  // Use the defined PI constant
}

// Function to calculate the area of a triangle
float area(float base, float height) {
    return (base * height) / 2;
}

int main() {
    int length, breadth;
    float radius, base, height;

    // Input for rectangle
    cout << "Enter length and breadth of rectangle: ";
    cin >> length >> breadth;

    // Input for circle
    cout << "Enter radius of circle: ";
    cin >> radius;

    // Input for triangle
    cout << "Enter base and height of triangle: ";
    cin >> base >> height;

    // Output areas
    cout << "\nArea of rectangle is: " << area(length, breadth) << endl;
    cout << "Area of circle is: " << area(radius) << endl;
    cout << "Area of triangle is: " << area(base, height) << endl;

    return 0;
}
```

Output:

Enter length and breadth of rectangle: 5

5

Enter radius of circle: 5

Enter base and height of triangle: 5

5

Area of rectangle is: 25

Area of circle is: 78.5398

Area of triangle is: 12.5

Detailed Breakdown

1. **Header Files and Namespace:**

   ```
   #include <iostream>
   #include <cmath>  // For mathematical functions

   using namespace std;
   ```

   - **<iostream>:** Provides functionalities for input and output using cin and cout.
   - **<cmath>:** Includes mathematical functions; in this case, it's included but not used explicitly because we define our own constant for π (PI).

2. **Constant Definition:**

   ```
   const float PI = 3.14159265358979323846f;  // More accurate value of π
   ```

   - Defines a constant PI with a more accurate value for π. This value is used for calculating the area of a circle.

3. **Function Overloading:**
   - **Area of Rectangle:**

     ```
     int area(int length, int breadth) {
        return length * breadth;
     }
     ```

- This function calculates the area of a rectangle given the length and breadth. It uses integer parameters and returns an integer value representing the area.

- **Area of Circle:**

```
float area(float radius) {
   return PI * radius * radius;  // Use the defined PI constant
}
```

  - This function calculates the area of a circle using the formula π×radius2\pi \times \text{radius}^2π×radius2. It takes a floating-point parameter (radius) and returns the area as a floating-point number.

- **Area of Triangle:**

```
float area(float base, float height) {
   return (base * height) / 2;
}
```

  - This function calculates the area of a triangle using the formula base×height2\frac{\text{base} \times \text{height}}{2}2base×height. It also takes floating-point parameters and returns the area as a floating-point number.

4. **Main Function:**

```
int main() {
   int length, breadth;
   float radius, base, height;

   // Input for rectangle
   cout << "Enter length and breadth of rectangle: ";
   cin >> length >> breadth;

   // Input for circle
   cout << "Enter radius of circle: ";
   cin >> radius;

   // Input for triangle
   cout << "Enter base and height of triangle: ";
   cin >> base >> height;

   // Output areas
   cout << "\nArea of rectangle is: " << area(length, breadth) << endl;
   cout << "Area of circle is: " << area(radius) << endl;
   cout << "Area of triangle is: " << area(base, height) << endl;
```

```
    return 0;
}
```

- **Variables:** Declares variables to store dimensions for each shape.
- **Input:** Prompts the user to enter dimensions for a rectangle, circle, and triangle.
- **Function Calls:**
  - Calls the area function with different parameter lists to compute the area of each shape.
- **Output:** Displays the computed areas using cout.

21. Create a class student containing data members: a. Roll_no b. name c. marks1, marks2, marks3 Write necessary member functions:

1. **To accept details of all students**
2. **To display details of one student**

3. To display details of all students (Use Function overloading).

Source Code:

```
#include <iostream>
#include <vector>
#include <string>

using namespace std;

class Student {
private:
    int roll_no;
    string name;
    float marks1, marks2, marks3;

public:
    // Function to accept details of a single student
    void acceptDetails() {
        cout << "Enter Roll Number: ";
        cin >> roll_no;
        cin.ignore();  // Ignore leftover newline from input buffer
        cout << "Enter Name: ";
        getline(cin, name);
        cout << "Enter Marks 1: ";
        cin >> marks1;
        cout << "Enter Marks 2: ";
        cin >> marks2;
        cout << "Enter Marks 3: ";
        cin >> marks3;
    }

    // Function to display details of a single student
    void displayDetails() const {
        cout << "Roll Number: " << roll_no << endl;
        cout << "Name: " << name << endl;
        cout << "Marks 1: " << marks1 << endl;
        cout << "Marks 2: " << marks2 << endl;
        cout << "Marks 3: " << marks3 << endl;
    }

    // Overloaded function to display details of all students
    static void displayDetails(const vector<Student>& students) {
        for (const auto& student : students) {
            student.displayDetails();
            cout << "-----------------------" << endl;
        }
    }
};
```

```
int main() {
    vector<Student> students;
    int num_students;

    cout << "Enter number of students: ";
    cin >> num_students;
    cin.ignore();  // Ignore leftover newline from input buffer

    students.resize(num_students);

    // Accept details for each student
    for (int i = 0; i < num_students; ++i) {
        cout << "\nEntering details for student " << (i + 1) << ":" << endl;
        students[i].acceptDetails();
    }

    cout << "\nDisplaying details of all students:" << endl;
    Student::displayDetails(students);

    return 0;
}
```

Output:

Enter number of students: 2

Entering details for student 1:

Enter Roll Number: 1

Enter Name: ABC

Enter Marks 1: 55

Enter Marks 2: 60

Enter Marks 3: 50

Entering details for student 2:

Enter Roll Number: 2

Enter Name: PQR

Enter Marks 1: 60

Enter Marks 2: 60

Enter Marks 3: 52

Displaying details of all students:

Roll Number: 1

Name: ABC

Marks 1: 55

Marks 2: 60

Marks 3: 50

Roll Number: 2

Name: PQR

Marks 1: 60

Marks 2: 60

Marks 3: 52

Explanation:

1. **Header Files and Namespace:**

   ```
   #include <iostream>
   #include <vector>
   #include <string>

   using namespace std;
   ```

   - **<iostream>:** For input and output operations.
   - **<vector>:** For using the vector container which allows dynamic array management.
   - **<string>:** For using the string class for handling text.

2. **Class Definition (Student):**

   ```
   class Student {
   private:
      int roll_no;
      string name;
      float marks1, marks2, marks3;
   ```

- **Private Members:**
  - roll_no: Stores the roll number of the student.
  - name: Stores the name of the student.
  - marks1, marks2, marks3: Store marks for three subjects.

Member Functions:

- **acceptDetails():**

```
void acceptDetails() {
   cout << "Enter Roll Number: ";
   cin >> roll_no;
   cin.ignore();  // Ignore leftover newline from input buffer
   cout << "Enter Name: ";
   getline(cin, name);
   cout << "Enter Marks 1: ";
   cin >> marks1;
   cout << "Enter Marks 2: ";
   cin >> marks2;
   cout << "Enter Marks 3: ";
   cin >> marks3;
}
```

  - Prompts the user to enter the student's roll number, name, and marks for three subjects. Uses cin.ignore() to handle newline characters left in the input buffer after reading integer inputs.

- **displayDetails():**

```
void displayDetails() const {
   cout << "Roll Number: " << roll_no << endl;
   cout << "Name: " << name << endl;
   cout << "Marks 1: " << marks1 << endl;
   cout << "Marks 2: " << marks2 << endl;
   cout << "Marks 3: " << marks3 << endl;
}
```

  - Displays the details of a single student.
- **Static Function displayDetails(const vector<Student>& students):**

```
static void displayDetails(const vector<Student>& students) {
   for (const auto& student : students) {
      student.displayDetails();
      cout << "------------------------" << endl;
```

```
    }
}
```

- A static member function that accepts a vector of Student objects and displays the details of each student by calling the non-static displayDetails() method on each student. This function is static because it does not operate on a specific instance of the class but rather on a collection of instances.

3. **Main Function:**

```
int main() {
    vector<Student> students;
    int num_students;

    cout << "Enter number of students: ";
    cin >> num_students;
    cin.ignore();  // Ignore leftover newline from input buffer

    students.resize(num_students);

    // Accept details for each student
    for (int i = 0; i < num_students; ++i) {
        cout << "\nEntering details for student " << (i + 1) << ":" << endl;
        students[i].acceptDetails();
    }

    cout << "\nDisplaying details of all students:" << endl;
    Student::displayDetails(students);

    return 0;
}
```

- **Input Handling:**
  - Prompts the user to enter the number of students and then resizes the vector to hold that many Student objects.
  - Loops through the number of students and calls acceptDetails() for each student to input their data.
- **Output Handling:**
  - Calls the static displayDetails() method of the Student class to display the details of all students in the vector.

22.Create two classes' dist1 (meters, centimeters) and dist2 (feet, inches). Accept two distances from the user, one in meters and centimeters and the other in feet

and inches. Find the sum and difference of the two distances. Display the result in both (meters and centimeters) as well as feet and inches (use friend function).

Explanation:

1. **Classes Definition:**
   - **Dist1**: Manages distances in meters and centimeters.
   - **Dist2**: Manages distances in feet and inches.
2. **Operations:**
   - **Sum** and **Difference**: These operations will be performed by a friend function that can access the private members of both classes.
3. **Conversion Functions:**
   - Convert distances between meters/centimeters and feet/inches for calculations and display purposes.

Source Code:

Here is the complete C++ program that includes the `Dist1` and `Dist2` classes, as well as the necessary friend functions to calculate and display the results.

```
#include <iostream>

using namespace std;

class Dist2;  // Forward declaration for the friend function

class Dist1 {
private:
    float meters;
    float centimeters;

public:
    // Constructor
    Dist1(float m = 0, float cm = 0) : meters(m), centimeters(cm) {}

    // Function to accept distance in meters and centimeters
    void acceptDetails() {
        cout << "Enter distance in meters: ";
        cin >> meters;
        cout << "Enter additional centimeters: ";
        cin >> centimeters;
        normalize();  // Ensure correct meter and centimeter values
    }

    // Display distance in meters and centimeters
    void display() const {
        cout << meters << " meters and " << centimeters << " centimeters";
    }
```

```
    // Convert Dist1 to Dist2
    Dist2 toDist2() const;

    // Friend function to perform operations
    friend void calculateAndDisplay(const Dist1& d1, const Dist2& d2);

private:
    void normalize() {
        if (centimeters >= 100) {
            meters += centimeters / 100;
            centimeters = (int)centimeters % 100;
        }
    }
};

class Dist2 {
private:
    float feet;
    float inches;

public:
    // Constructor
    Dist2(float f = 0, float in = 0) : feet(f), inches(in) {}

    // Function to accept distance in feet and inches
    void acceptDetails() {
        cout << "Enter distance in feet: ";
        cin >> feet;
        cout << "Enter additional inches: ";
        cin >> inches;
        normalize();  // Ensure correct feet and inch values
    }

    // Display distance in feet and inches
    void display() const {
        cout << feet << " feet and " << inches << " inches";
    }

    // Convert Dist2 to Dist1
    Dist1 toDist1() const;

    // Friend function to perform operations
    friend void calculateAndDisplay(const Dist1& d1, const Dist2& d2);

private:
    void normalize() {
        if (inches >= 12) {
            feet += inches / 12;
            inches = (int)inches % 12;
        }
    }
};

// Convert Dist1 to Dist2
Dist2 Dist1::toDist2() const {
    float totalInches = meters * 39.3701f + centimeters * 0.393701f;
    float feet = totalInches / 12;
```

```
    float inches = totalInches - feet * 12;
    return Dist2(feet, inches);
}

// Convert Dist2 to Dist1
Dist1 Dist2::toDist1() const {
    float totalMeters = feet * 0.3048f + inches * 0.0254f;
    float meters = (int)totalMeters;
    float centimeters = (totalMeters - meters) * 100;
    return Dist1(meters, centimeters);
}

// Function to calculate and display the sum and difference
void calculateAndDisplay(const Dist1& d1, const Dist2& d2) {
    Dist2 d2InFeetInches = d1.toDist2();  // Convert d1 to feet and inches
    Dist1 d1FromD2 = d2.toDist1();        // Convert d2 to meters and centimeters

    // Calculate sum in meters and centimeters
    float totalMeters = d1.meters + d1FromD2.meters;
    float totalCentimeters = d1.centimeters + d1FromD2.centimeters;
    if (totalCentimeters >= 100) {
        totalMeters += totalCentimeters / 100;
        totalCentimeters = (int)totalCentimeters % 100;
    }

    // Calculate sum in feet and inches
    float totalFeet = d2.feet + d2InFeetInches.feet;
    float totalInches = d2.inches + d2InFeetInches.inches;
    if (totalInches >= 12) {
        totalFeet += totalInches / 12;
        totalInches = (int)totalInches % 12;
    }

    // Output results
    cout << "Sum in meters and centimeters: " << totalMeters << " meters and " <<
totalCentimeters << " centimeters" << endl;
    cout << "Sum in feet and inches: " << totalFeet << " feet and " << totalInches
<< " inches" << endl;

    // Calculate and display the difference
    float diffMeters = d1.meters - d1FromD2.meters;
    float diffCentimeters = d1.centimeters - d1FromD2.centimeters;
    if (diffCentimeters < 0) {
        diffMeters -= 1;
        diffCentimeters += 100;
    }

    float diffFeet = d2.feet - d2InFeetInches.feet;
    float diffInches = d2.inches - d2InFeetInches.inches;
    if (diffInches < 0) {
        diffFeet -= 1;
        diffInches += 12;
    }

    // Output results
    cout << "Difference in meters and centimeters: " << diffMeters << " meters and "
<< diffCentimeters << " centimeters" << endl;
```

```
    cout << "Difference in feet and inches: " << diffFeet << " feet and " <<
diffInches << " inches" << endl;
}

int main() {
    Dist1 d1;
    Dist2 d2;

    cout << "Enter details for distance 1 (meters and centimeters):" << endl;
    d1.acceptDetails();

    cout << "\nEnter details for distance 2 (feet and inches):" << endl;
    d2.acceptDetails();

    cout << "\nCalculating and displaying results:\n";
    calculateAndDisplay(d1, d2);

    return 0;
}
```

Output:

Enter details for distance 1 (meters and centimeters):

Enter distance in meters: 20

Enter additional centimeters: 21

Enter details for distance 2 (feet and inches):

Enter distance in feet: 25

Enter additional inches: 20

Calculating and displaying results:

Sum in meters and centimeters: 28 meters and 54.1201 centimeters

Sum in feet and inches: 92.9725 feet and 8 inches

Difference in meters and centimeters: 11 meters and 87.8799 centimeters

Difference in feet and inches: -39.6391 feet and 8 inches

Explanation:

1. **Class Dist1 and Dist2:**
    - **Dist1:** Manages distance in meters and centimeters. It includes methods for input, output, normalization (to handle cases where centimeters exceed 100), and conversion to Dist2.
    - **Dist2:** Manages distance in feet and inches. It includes methods for input, output, normalization (to handle cases where inches exceed 12), and conversion to Dist1.
2. **Conversion Functions:**
    - **Dist1::toDist2():** Converts a distance in meters and centimeters to feet and inches.
    - **Dist2::toDist1():** Converts a distance in feet and inches to meters and centimeters.
3. **Friend Function calculateAndDisplay():**
    - **Sum and Difference Calculation:** Computes the sum and difference of distances in both measurement systems (meters/centimeters and feet/inches). Converts between systems as needed and handles normalization.
4. **main() Function:**
    - Prompts the user for distances in both measurement systems.
    - Calls calculateAndDisplay() to perform calculations and output results.

23. Write a C++ program using class which contains two data members of type integer. Create and initialize the object using default constructor, parameterized

constructor and parameterized constructor with default value. Write a member function to display maximum from given two numbers for all objects

1. **Class Definition**:
    - **Data Members**: Two integer data members.
    - **Constructors**:
        - **Default Constructor**: Initializes data members to default values.
        - **Parameterized Constructor**: Initializes data members with given values.
        - **Parameterized Constructor with Default Values**: Initializes data members with given values, allowing default values to be used if not provided.
    - **Member Function**: Displays the maximum of the two numbers for a given object.

Here's a complete C++ program demonstrating this:

Source Code:

```
#include <iostream>
using namespace std;

class NumberPair {
private:
    int num1;
    int num2;

public:
    // Default constructor
    NumberPair() : num1(0), num2(0) {}

    // Parameterized constructor with default value
    NumberPair(int a, int b = 0) : num1(a), num2(b) {}

    // Member function to display the maximum of the two numbers
    void displayMax() const {
        int maxNum = (num1 > num2) ? num1 : num2;
        cout << "Maximum of " << num1 << " and " << num2 << " is: " << maxNum <<
endl;
    }
};

int main() {
    // Creating objects using different constructors

    // Using default constructor
    NumberPair np1;
    cout << "Object created using default constructor:" << endl;
    np1.displayMax();  // Should display 0, 0
```

```
    // Using parameterized constructor with two arguments
    NumberPair np2(10, 20);
    cout << "Object created using parameterized constructor with two arguments:" <<
endl;
    np2.displayMax();  // Should display 20

    // Using parameterized constructor with one argument (second argument defaults
to 0)
    NumberPair np3(15);
    cout << "Object created using parameterized constructor with default value:" <<
endl;
    np3.displayMax();  // Should display 15 (15, 0)

    return 0;
}
```

Output:

Object created using default constructor:

Maximum of 0 and 0 is: 0

Object created using parameterized constructor with two arguments:

Maximum of 10 and 20 is: 20

Object created using parameterized constructor with default value:

Maximum of 15 and 0 is: 15

Explanation:

1. **Constructors:**
    - **Default Constructor (NumberPair()):** Initializes num1 and num2 to 0.
    - **Parameterized Constructor with Default Value (NumberPair(int a, int b = 0)):** This constructor initializes num1 with a and num2 with b, where b defaults to 0 if not provided. This covers both cases where two values are provided or only one.
2. **main() Function:**
    - **Creating Objects:**
        - **np1**: Created using the default constructor.
        - **np2**: Created using the parameterized constructor with two arguments.

        - **np3**: Created using the parameterized constructor with one argument, with the second argument defaulting to 0.
    - **Calling displayMax():** For each object, the displayMax() function is called to show the maximum value of the two integers.

25.Create a class time that contains hours, minute and seconds as data members.
Write the member function to overload operator '+' to add two object of type time,(Use Parameterized constructor to accept values for time).

To create a C++ class that handles time and allows you to add two Time objects using the + operator, you'll need to follow these steps:

1. **Define the Time class** with data members for hours, minutes, and seconds.
2. **Overload the + operator** to add two Time objects.
3. **Implement parameterized constructors** to initialize the time values.

Source Code:

```
#include <iostream>
using namespace std;

class Time {
private:
    int hours;
    int minutes;
    int seconds;

    // Helper function to normalize the time values
    void normalize() {
        if (seconds >= 60) {
            minutes += seconds / 60;
            seconds %= 60;
        }
        if (minutes >= 60) {
            hours += minutes / 60;
            minutes %= 60;
        }
    }

public:
    // Parameterized constructor
    Time(int h = 0, int m = 0, int s = 0) : hours(h), minutes(m), seconds(s) {
        normalize();
    }

    // Overload the '+' operator
    Time operator+(const Time& other) const {
        // Add the time components
        int totalHours = hours + other.hours;
        int totalMinutes = minutes + other.minutes;
        int totalSeconds = seconds + other.seconds;

        // Create a new Time object and normalize the values
        Time result(totalHours, totalMinutes, totalSeconds);
        return result;
```

```
    }

    // Function to display the time
    void display() const {
        cout << hours << " hours, " << minutes << " minutes, " << seconds << "
seconds" << endl;
    }
};

int main() {
    // Create two Time objects using parameterized constructors
    Time time1(1, 50, 30); // 1 hour, 50 minutes, 30 seconds
    Time time2(2, 20, 45); // 2 hours, 20 minutes, 45 seconds

    // Add the two Time objects using the overloaded '+' operator
    Time result = time1 + time2;

    // Display the result
    cout << "Time 1: ";
    time1.display();
    cout << "Time 2: ";
    time2.display();
    cout << "Result of addition: ";
    result.display();

    return 0;
}
```

Output:

Time 1: 1 hours, 50 minutes, 30 seconds

Time 2: 2 hours, 20 minutes, 45 seconds

Result of addition: 4 hours, 11 minutes, 15 seconds

Explanation:

1. **Class Definition (Time):**
    - **Data Members:**
        - int hours: Number of hours.
        - int minutes: Number of minutes.
        - int seconds: Number of seconds.
    - **Parameterized Constructor:**
        - Initializes the data members and normalizes the time values.
    - **Normalization (normalize Function):**

        - Adjusts seconds and minutes to ensure they are within the valid ranges (0-59). For example, if seconds exceeds 59, it adjusts the minutes accordingly, and similarly for minutes and hours.
    - **Overloaded + Operator:**
        - Adds the time components of two Time objects. It then creates a new Time object with the summed values and normalizes them before returning.
    - **display Function:**
        - Prints the time in a readable format.
2. **main() Function:**
    - **Creating Objects:**
        - **time1**: Initialized with 1 hour, 50 minutes, and 30 seconds.
        - **time2**: Initialized with 2 hours, 20 minutes, and 45 seconds.
    - **Adding Time:**
        - Uses the overloaded + operator to add time1 and time2.
    - **Displaying Results:**
        - Displays the original times and the result of their addition.

26.Write a C++ program to create a class Person that contains data members as Person_Name, City, Mob_No. Write a C++ program to perform following functions:

a. To accept and display Person information
b. To search the mobile number of a given person
c. To search the Person details of a given mobile number (Use Function Overloading)

Certainly! Below is a C++ program that defines a Person class with data members for Person_Name, City, and Mob_No. The program includes functionalities to:

1. **Accept and Display Person Information**
2. **Search for a Person's Mobile Number by Name**
3. **Search for Person Details by Mobile Number (using function overloading)**

Source Code:

```
#include <iostream>
#include <vector>
#include <string>

using namespace std;

class Person {
private:
    string Person_Name;
    string City;
    string Mob_No;

public:
    // Constructor to initialize Person details
    Person(const string& name, const string& city, const string& mobNo)
        : Person_Name(name), City(city), Mob_No(mobNo) {}

    // Function to display Person information
    void display() const {
        cout << "Name: " << Person_Name << endl;
        cout << "City: " << City << endl;
        cout << "Mobile Number: " << Mob_No << endl;
    }

    // Function to get the mobile number
    string getMobileNumber() const {
        return Mob_No;
    }

    // Function to get the name
    string getName() const {
        return Person_Name;
    }

    // Static function to search for a mobile number by name
```

```
    static string searchMobileNumber(const vector<Person>& people, const string&
name) {
        for (const auto& person : people) {
            if (person.getName() == name) {
                return person.getMobileNumber();
            }
        }
        return "Not found";
    }

    // Overloaded static function to search for person details by mobile number
    static void searchPersonDetails(const vector<Person>& people, const string&
mobNo) {
        for (const auto& person : people) {
            if (person.getMobileNumber() == mobNo) {
                person.display();
                return;
            }
        }
        cout << "Person with mobile number " << mobNo << " not found." << endl;
    }

    // Overloaded static function to search for person details by mobile number
(when given as integer)
    static void searchPersonDetails(const vector<Person>& people, int mobNo) {
        // Convert integer to string for comparison
        string mobNoStr = to_string(mobNo);
        searchPersonDetails(people, mobNoStr);
    }
};

int main() {
    // Create a list of Person objects
    vector<Person> people = {
        Person("Alice", "New York", "1234567890"),
        Person("Bob", "Los Angeles", "9876543210"),
        Person("Charlie", "Chicago", "5555555555")
    };

    // Display all person information
    cout << "All person details:" << endl;
    for (const auto& person : people) {
        person.display();
        cout << endl;
    }

    // Search for a mobile number by name
    string searchName;
    cout << "Enter the name to search for mobile number: ";
    cin >> searchName;
    string mobileNumber = Person::searchMobileNumber(people, searchName);
    cout << "Mobile Number of " << searchName << " is: " << mobileNumber << endl;

    // Search for person details by mobile number
    string searchMobNo;
    cout << "Enter the mobile number to search for person details: ";
    cin >> searchMobNo;
    Person::searchPersonDetails(people, searchMobNo);
```

```
    // Overloaded search by integer mobile number
    int searchMobNoInt;
    cout << "Enter the mobile number (as integer) to search for person details: ";
    cin >> searchMobNoInt;
    Person::searchPersonDetails(people, searchMobNoInt);

    return 0;
}
```

Output:

All person details:

Name: Alice

City: New York

Mobile Number: 1234567890

Name: Bob

City: Los Angeles

Mobile Number: 9876543210

Name: Charlie

City: Chicago

Mobile Number: 5555555555

Enter the name to search for mobile number: john

Mobile Number of bob is: Not found

Enter the mobile number to search for person details: 1234567890

Name: Alice

City: New York

Mobile Number: 1234567890

Enter the mobile number (as integer) to search for person details: 55

Person with mobile number 55 not found.

Explanation:

1. **Class Definition (Person):**
    - **Data Members:**
        - string Person_Name: Name of the person.
        - string City: City where the person lives.
        - string Mob_No: Mobile number of the person.
    - **Constructor:**
        - Initializes the person's details.
    - **Member Functions:**
        - display(): Displays the details of the person.
        - getMobileNumber(): Returns the mobile number.
        - getName(): Returns the person's name.
    - **Static Functions:**
        - searchMobileNumber(const vector<Person>& people, const string& name): Searches for a person's mobile number by their name.
        - searchPersonDetails(const vector<Person>& people, const string& mobNo): Searches for a person's details by their mobile number. This function is overloaded to also handle integer inputs.
2. **main() Function:**
    - **Creates a List of Person Objects:** Initializes a vector of Person objects with sample data.
    - **Displays All Person Information:** Iterates through the list and displays each person's details.
    - **Searches by Name:** Prompts the user to enter a name and retrieves the corresponding mobile number.
    - **Searches by Mobile Number:** Prompts the user to enter a mobile number and displays the corresponding person's details.
    - **Search by Integer Mobile Number:** Demonstrates the overloaded function that accepts the mobile number as an integer.

26. **Create a base class Conversion. Derive three different classes Weight (Gram, Kilogram), Volume (Milliliter, Liter), Currency (Rupees, Paise) from Conversion class. Write a C++ program to perform read, convert and display operations. (Use Pure virtual function)**

Certainly! To design a C++ program with a base class Conversion and derived classes Weight, Volume, and Currency, we need to follow these steps:

1. **Create a base class Conversion** with pure virtual functions to enforce that derived classes implement their specific conversion and display logic.
2. **Derive three classes** (Weight, Volume, and Currency) from the Conversion class, implementing the required functionality.
3. **Implement conversion and display methods** in each derived class.
4. **Demonstrate the use** of these classes in the main() function.

Source Code:

```
#include <iostream>
#include <iomanip>
using namespace std;

// Base class with pure virtual functions
class Conversion {
public:
    virtual void read() = 0;       // Pure virtual function to read values
    virtual void convert() = 0;    // Pure virtual function to perform conversion
    virtual void display() const = 0; // Pure virtual function to display results
    virtual ~Conversion() {}       // Virtual destructor for proper cleanup
};

// Derived class for Weight conversion
class Weight : public Conversion {
private:
    double gram;
    double kilogram;

public:
    void read() override {
        cout << "Enter weight in grams: ";
        cin >> gram;
    }

    void convert() override {
        kilogram = gram / 1000.0;
    }

    void display() const override {
        cout << fixed << setprecision(2);
        cout << "Weight: " << gram << " grams = " << kilogram << " kilograms" <<
endl;
    }
```

```
};

// Derived class for Volume conversion
class Volume : public Conversion {
private:
    double milliliter;
    double liter;

public:
    void read() override {
        cout << "Enter volume in milliliters: ";
        cin >> milliliter;
    }

    void convert() override {
        liter = milliliter / 1000.0;
    }

    void display() const override {
        cout << fixed << setprecision(2);
        cout << "Volume: " << milliliter << " milliliters = " << liter << " liters"
<< endl;
    }
};

// Derived class for Currency conversion
class Currency : public Conversion {
private:
    double rupees;
    double paise;

public:
    void read() override {
        cout << "Enter amount in rupees: ";
        cin >> rupees;
    }

    void convert() override {
        paise = rupees * 100.0;
    }

    void display() const override {
        cout << fixed << setprecision(2);
        cout << "Currency: " << rupees << " rupees = " << paise << " paise" << endl;
    }
};

int main() {
    Conversion* conv;
    int choice;

    cout << "Select conversion type:\n";
    cout << "1. Weight\n";
    cout << "2. Volume\n";
    cout << "3. Currency\n";
    cout << "Enter your choice (1/2/3): ";
    cin >> choice;
```

```
    switch (choice) {
    case 1:
        conv = new Weight();
        break;
    case 2:
        conv = new Volume();
        break;
    case 3:
        conv = new Currency();
        break;
    default:
        cout << "Invalid choice!" << endl;
        return 1;
    }

    conv->read();
    conv->convert();
    conv->display();

    delete conv; // Clean up
    return 0;
}
```

Output:

Select conversion type:

1. Weight

2. Volume

3. Currency

Enter your choice (1/2/3): 1

Enter weight in grams: 1000

Weight: 1000.00 grams = 1.00 kilograms

Explanation:

1. **Base Class (Conversion):**
    - **Pure Virtual Functions:**
        - virtual void read() = 0; — To read the input value.
        - virtual void convert() = 0; — To perform the conversion.
        - virtual void display() const = 0; — To display the converted result.
    - **Virtual Destructor:** Ensures that the destructor of derived classes is called when an object is deleted via a base class pointer.

2. **Derived Classes:**
    - **Weight:** Converts grams to kilograms.
        - **read():** Reads weight in grams.
        - **convert():** Converts grams to kilograms.
        - **display():** Displays the weight in both grams and kilograms.
    - **Volume:** Converts milliliters to liters.
        - **read():** Reads volume in milliliters.
        - **convert():** Converts milliliters to liters.
        - **display():** Displays the volume in both milliliters and liters.
    - **Currency:** Converts rupees to paise.
        - **read():** Reads amount in rupees.
        - **convert():** Converts rupees to paise.
        - **display():** Displays the amount in both rupees and paise.
3. **main() Function:**
    - Prompts the user to select the type of conversion.
    - Creates an instance of the appropriate derived class based on user input.
    - Calls the read(), convert(), and display() methods on the created object.
    - Cleans up dynamically allocated memory.

28.Write a C++ program to create a class novel which contains data member as id, name andauthor. Write member function to accept and display novel information. Also display the count of novels (use static data member to maintain the count of novel).

To create a C++ program with a class Novel that contains data members for id, name, and author, and includes member functions to accept and display novel information, as well as to maintain and display the count of novels, follow these steps:

1. **Define the Novel class** with the necessary data members.
2. **Use a static data member** to keep track of the number of Novel objects created.
3. **Implement member functions** to set and get novel information.
4. **Implement a static member function** to display the count of novels.

Source Code:

```
#include <iostream>
#include <string>
using namespace std;

class Novel {
private:
    int id;
    string name;
    string author;
    static int count; // Static data member to maintain the count of novels

public:
    // Constructor
    Novel(int i, const string& n, const string& a) : id(i), name(n), author(a) {
        count++; // Increment count when a new object is created
    }

    // Destructor
    ~Novel() {
        count--; // Decrement count when an object is destroyed
    }

    // Member function to accept novel information
    void acceptInfo() {
        cout << "Enter novel ID: ";
        cin >> id;
        cin.ignore(); // Ignore newline character left in the buffer
        cout << "Enter novel name: ";
        getline(cin, name);
        cout << "Enter author name: ";
        getline(cin, author);
    }

    // Member function to display novel information
    void displayInfo() const {
        cout << "Novel ID: " << id << endl;
```

```
        cout << "Novel Name: " << name << endl;
        cout << "Author: " << author << endl;
    }

    // Static function to display the count of novels
    static void displayCount() {
        cout << "Total number of novels: " << count << endl;
    }
};

// Initialize the static member
int Novel::count = 0;

int main() {
    // Create some Novel objects
    Novel n1(1, "1984", "George Orwell");
    Novel n2(2, "To Kill a Mockingbird", "Harper Lee");

    // Display information of each novel
    cout << "Novel 1 Information:" << endl;
    n1.displayInfo();
    cout << endl;

    cout << "Novel 2 Information:" << endl;
    n2.displayInfo();
    cout << endl;

    // Display the count of novels
    Novel::displayCount();

    // Create a new novel using the acceptInfo function
    Novel n3(0, "", "");
    cout << "Enter details for a new novel:" << endl;
    n3.acceptInfo();

    // Display the new novel information and count
    cout << endl << "New Novel Information:" << endl;
    n3.displayInfo();
    cout << endl;
    Novel::displayCount();

    return 0;
}
```

Output:

Novel 1 Information:

Novel ID: 1

Novel Name: 1984

Author: George Orwell

Novel 2 Information:

Novel ID: 2

Novel Name: To Kill a Mockingbird

Author: Harper Lee

Total number of novels: 2

Enter details for a new novel:

Enter novel ID: 3

Enter novel name: 1996

Enter author name: abc

New Novel Information:

Novel ID: 3

Novel Name: 1996

Author: abc

Total number of novels: 3

Explanation:

1. **Class Definition (Novel):**
    - **Data Members:**
        - int id: ID of the novel.
        - string name: Name of the novel.
        - string author: Author of the novel.
        - static int count: Static member variable to keep track of the number of Novel objects created.
    - **Constructor:**

        - Initializes the data members and increments the count when a new Novel object is created.
    - **Destructor:**
        - Decrements the count when a Novel object is destroyed.
    - **Member Functions:**
        - acceptInfo(): Reads novel details from the user.
        - displayInfo(): Displays the details of the novel.
        - static void displayCount(): Displays the total number of novels. This function is static, so it can be called without an instance of the class.
2. **Static Data Member Initialization:**
    - The static member count must be initialized outside the class definition using int Novel::count = 0;.
3. **main() Function:**
    - Creates a few Novel objects and displays their information.
    - Uses the acceptInfo() function to create a new Novel object and then displays its information.
    - Calls Novel::displayCount() to show the total number of Novel objects.