Overloading Problem Statements

1. Chemical Mixture Analysis

Problem Statement:

You are tasked with building a `ChemicalMixture` class to analyze and manage various chemical solutions. Your class must demonstrate the use of constructor, method, and operator overloading.

- **Constructor Overloading:** Implement a constructor that accepts an array of strings for chemical compound names and a parallel array of doubles for their masses in grams. The constructor should then calculate the total mass of the mixture. Overload this constructor to also accept two arrays, one for chemical names and one for their molar masses. In this case, the constructor should calculate the total number of moles for the mixture.
- **Method Overloading:** Implement a method named `calculateConcentration`. One version of this method should accept a `double` representing the volume in liters and return the molar concentration of the mixture. Overload this method to also accept a `double` for volume and another `double` for the temperature in Celsius, calculating and returning the concentration while adjusting for thermal expansion (assume a simplified linear expansion model where volume increases by 0.001% per degree Celsius above 25°C).
- **Operator Overloading:** Overload the `+` operator to combine two `ChemicalMixture` objects. The new `ChemicalMixture` object should contain a consolidated list of all compounds from both mixtures. If a compound exists in both, their masses should be summed. The new object should also correctly update the total mass and moles.

Sample Input:

```
// Mixture 1: by mass
string m1_names[] = {"NaCl", "H2O"};
double m1_masses[] = {58.44, 18.02};
ChemicalMixture mixture1(m1_names, m1_masses, 2);

// Mixture 2: by molar mass
string m2_names[] = {"C12H22O11", "H2O"};
double m2_molarMasses[] = {342.3, 18.02};
ChemicalMixture mixture2(m2_names, m2_molarMasses, 2);
double m2_volume = 2.5;
double m2_temperature = 50.0;

// Combine mixtures
ChemicalMixture combinedMixture = mixture1 + mixture2;
```

Sample Output:

```
Mixture 1 Info:
Total Mass: 76.46 g
```

```
Mixture 2 Info:
Concentration at 2.5 L and 50.0 C: 0.165146 mol/L

Combined Mixture Info:
Total Mass: 418.78 g
Total Moles: 21.0
Combined Compounds:
NaCl: 58.44 g
H2O: 18.02 g
C12H22O11: 342.3 g
```

---

2. Financial Portfolio Simulation

Problem Statement:

Create a `FinancialPortfolio` class to simulate the management of an investment portfolio. Your class should demonstrate constructor, method, and operator overloading.

- **Constructor Overloading:** Implement a constructor that initializes a portfolio with a starting cash balance. Overload this constructor to accept two parallel arrays, one for stock tickers (e.g., "GOOG") and one for the number of shares. The constructor should then calculate the total value of the portfolio based on a fixed, pre-defined set of stock prices.
- **Method Overloading:** Create a method named `calculateValue`. One version should take no arguments and return the current total value of the portfolio. Overload this method to accept a `double` interest rate and an `int` number of years to project the future value of the portfolio using a simple compound interest formula.
- **Operator Overloading:** Overload the `+` operator to simulate adding a new investment to the portfolio. This could be a new stock with a certain number of shares, or a new cash deposit. The operator should return a new `FinancialPortfolio` object with the updated assets.

Sample Input:

```
// Portfolio 1: by cash
FinancialPortfolio portfolio1(10000.0);

// Portfolio 2: by stocks
string tickers[] = {"AAPL", "GOOG"};
int shares[] = {10, 5};
FinancialPortfolio portfolio2(tickers, shares, 2);

// Simulate a new investment
portfolio2 = portfolio2 + {"TSLA", 2};
double projectedValue = portfolio2.calculateValue(0.05, 10);
```

Sample Output:

```
Portfolio 1 Initial Value: $10,000.00
Portfolio 2 Initial Value: $38,500.00
Portfolio 2 Value after adding TSLA: $44,500.00
Projected value of Portfolio 2 in 10 years at 5% interest: $72,551.78
```

---

3. GPS Navigation and Route Planning

Problem Statement:

Develop a `GPSPoint` class to represent and manipulate geographical locations. Your class must utilize constructor, method, and operator overloading.

- **Constructor Overloading:** Implement a constructor that takes latitude and longitude as `double` values. Overload this constructor to also accept a city name as a `string` and look up its corresponding latitude and longitude from a pre-defined array of city data.
- **Method Overloading:** Create a method named `calculateDistance`. One version should take another `GPSPoint` object and return the straight-line distance in kilometers using the Haversine formula. Overload this method to also accept a `double` for average speed (in km/h) and a `bool` to account for a "traffic factor" (if true, multiply the distance by 1.2 to simulate traffic) to return the estimated travel time in hours.
- **Operator Overloading:** Overload the `==` operator to compare two `GPSPoint` objects. They should be considered equal if their latitude and longitude are within a very small tolerance (e.g., `1e-6`).

Sample Input:

```
// GPS points
GPSPoint startPoint(34.0522, -118.2437); // Los Angeles
GPSPoint endPoint("New York"); // New York is pre-defined in a data array

// Calculate distance and time
double distance_km = startPoint.calculateDistance(endPoint);
double travel_time_hours = startPoint.calculateDistance(endPoint, 80.0, true);

// Compare points
GPSPoint samePoint(34.05220000001, -118.24370000001);
bool is_same = (startPoint == samePoint);
```

Sample Output:

```
Distance between Los Angeles and New York: 3935.75 km
Estimated travel time with traffic at 80 km/h: 59.03 hours
Are the two points considered the same? Yes
```

---

4. Signal Processing and Audio Mixing

Problem Statement:

Create an `AudioSignal` class to represent and manipulate sound waves. Your class must demonstrate constructor, method, and operator overloading.

- **Constructor Overloading:** Implement a constructor that initializes a signal from a dynamically allocated array of `double` samples, representing the amplitude at each point in time. Overload this constructor to accept a single `double` for frequency and a `double` for amplitude, generating a simple sine wave with a fixed duration and sampling rate.
- **Method Overloading:** Create a method named `mixSignals`. One version should take another `AudioSignal` object and combine the two signals by averaging their sample values. Overload this method to accept another `AudioSignal` and a `double` scaling factor, mixing the signals with the specified weight.
- **Operator Overloading:** Overload the `*` operator to scale the volume of an `AudioSignal` object. Multiplying a signal by a `double` value should scale all its sample values by that factor.

Sample Input:

```
// Signal 1: from samples
double* samples = new double[5]{0.1, 0.5, 0.9, 0.5, 0.1};
AudioSignal signal1(samples, 5);

// Signal 2: generated sine wave
AudioSignal signal2(440.0, 0.7); // 440 Hz, 0.7 amplitude

// Mix signals
AudioSignal mixedSignal1 = signal1.mixSignals(signal2);
AudioSignal mixedSignal2 = signal1.mixSignals(signal2, 0.5);

// Scale a signal
AudioSignal scaledSignal = signal1 * 2.0;

delete[] samples;
```

Sample Output:

```
Signal 1 Info:
Number of samples: 5
Maximum amplitude: 0.9

Signal 2 Info:
Frequency: 440 Hz
Amplitude: 0.7

Mixed Signal 1 Info:
Mixed with equal weight. Max amplitude: 0.8
Mixed Signal 2 Info:
Mixed with a scaling factor. Max amplitude: 0.62
```

```
Scaled Signal Info:
Scaled by 2.0. Max amplitude: 1.8
```

---

5. 3D Architectural Rendering

Problem Statement:

Develop a `3DObject` class to represent and manipulate simple geometric shapes for a 3D rendering application. Your class must use constructor, method, and operator overloading.

- **Constructor Overloading:** Implement a constructor to create a sphere, taking a single `double` for its radius. Overload this constructor to accept a `double` for length, width, and height to create a rectangular prism. The constructor should calculate the volume and surface area of the object.
- **Method Overloading:** Create a method named `calculateVolume`. One version should take no arguments and return the object's stored volume. Overload this method to accept a `double` scaling factor, returning the new volume after the object has been scaled by that factor.
- **Operator Overloading:** Overload the `+` operator to "merge" two `3DObject` objects. The new object should be a new `3DObject` that has the combined volume of the two original objects. The surface area of the new object should be calculated as the sum of the surface areas of the two original objects minus a fixed overlap value (e.g., 10% of the smaller surface area).

Sample Input:

```
// Objects
3DObject sphere(5.0); // radius 5
3DObject cube(10.0, 10.0, 10.0); // 10x10x10 cube

// Calculate volume
double scaled_volume = sphere.calculateVolume(1.5);

// Merge objects
3DObject mergedObject = sphere + cube;
```

Sample Output:

```
Sphere Info:
Volume: 523.6 cubic units
Surface Area: 314.16 square units

Cube Info:
Volume: 1000.0 cubic units
Surface Area: 600.0 square units

Scaled Sphere Volume (factor 1.5): 1776.66 cubic units

Merged Object Info:
```

```
Combined Volume: 1523.6 cubic units
Combined Surface Area: 882.744 square units
```