Lab 6

1. **Write a program that can convert the Distance (meter, centimeter) to meters measurement in float and vice versa. Make a class distance with two data members, meter and centimeter. You can add function members as per your requirement.**

```
#include <iostream>

class Distance {
private:
    float meter;
    float centimeter;

public:
    // Constructor to initialize meter and centimeter
    Distance(float m, float cm) : meter(m), centimeter(cm) {}

    // Function to convert Distance to meters
    float toMeters() {
        return meter + centimeter / 100.0; // 1 meter = 100 centimeters
    }

    // Function to convert Distance to centimeters
    float toCentimeters() {
        return meter * 100.0 + centimeter;
    }
};

int main() {
    float m, cm;

    std::cout << "Enter distance in meters: ";
    std::cin >> m;

    std::cout << "Enter distance in centimeters: ";
    std::cin >> cm;

    // Create a Distance object
    Distance d(m, cm);

    std::cout << "Distance in meters: " << d.toMeters() << " meters" << std::endl;
    std::cout << "Distance in centimeters: " << d.toCentimeters() << " centimeters"
<< std::endl;

    return 0;
}
```

Output:

Enter distance in meters: 2
Enter distance in centimeters: 4
Distance in meters: 2.04 meters
Distance in centimeters: 204 centimeters

2. Write two classes to store distances in meter-centimeter and feet-inch systems respectively. Write conversions functions so that the program can convert objects of both types.

```
#include <iostream>

class DistanceFeetInch;

class DistanceMeterCentimeter {
private:
    float meter;
    float centimeter;

public:
    DistanceMeterCentimeter(float m, float cm) : meter(m), centimeter(cm) {}

    // Conversion function to convert to DistanceFeetInch
    DistanceFeetInch toFeetInch();

    void display() {
        std::cout << "Distance in meter-centimeter: " << meter << " meters " <<
centimeter << " centimeters" << std::endl;
    }
};

class DistanceFeetInch {
private:
    float feet;
    float inch;

public:
    DistanceFeetInch(float ft, float in) : feet(ft), inch(in) {}

    // Conversion function to convert to DistanceMeterCentimeter
    DistanceMeterCentimeter toMeterCentimeter();

    void display() {
        std::cout << "Distance in feet-inch: " << feet << " feet " << inch << " inches"
<< std::endl;
    }
};

// Conversion function from DistanceMeterCentimeter to DistanceFeetInch
DistanceFeetInch DistanceMeterCentimeter::toFeetInch() {
```

```
    float totalInches = (meter * 100 + centimeter) / 2.54; // 1 inch = 2.54 centimeters
    float feet = totalInches / 12;
    float inch = totalInches - feet * 12;
    return DistanceFeetInch(feet, inch);
}

// Conversion function from DistanceFeetInch to DistanceMeterCentimeter
DistanceMeterCentimeter DistanceFeetInch::toMeterCentimeter() {
    float totalInches = feet * 12 + inch;
    float cm = totalInches * 2.54; // 1 inch = 2.54 centimeters
    float meter = cm / 100;
    cm = cm - (static_cast<int>(meter) * 100); // Remaining centimeters
    return DistanceMeterCentimeter(meter, cm);
}

int main() {
    // Example using DistanceMeterCentimeter
    DistanceMeterCentimeter distanceMC(5, 30);
    distanceMC.display();
    DistanceFeetInch convertedFI = distanceMC.toFeetInch();
    convertedFI.display();

    // Example using DistanceFeetInch
    DistanceFeetInch distanceFI(2, 6);
    distanceFI.display();
    DistanceMeterCentimeter convertedMC = distanceFI.toMeterCentimeter();
    convertedMC.display();

    return 0;
}
```

Output

```
Distance in meter-centimeter: 5 meters 30 centimeters
Distance in feet-inch: 17.3885 feet -7.62939e-06 inches
Distance in feet-inch: 2 feet 6 inches
Distance in meter-centimeter: 0.762 meters 76.2 centimeters
```

3. Create a class called Musicians to contain three methods `string()`, `wind()`, and `perc()`. Each of these methods should initialize a string array to contain the following instruments

- **veena, guitar, sitar, sarod and mandolin under `string()`**
- **flute, clarinet saxophone, nadhaswaram, and piccolo under `wind()`**
- **tabla, mridangam, bangos, drums and tambour under `perc()`**

It should also display the contents of the arrays that are initialized.

Create a derived class called `TypeIns` to contain a method

called `get()` and `show()`. The `get()` method must display a menu as follows

Type of instruments to be displayed

a. String instruments

b. Wind instruments

c. Percussion instruments

The `show()` method should display the relevant detail according to our choice. The base class variables must be accessible only to their derived classes.

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

class Musicians {
protected:
    std::string stringInstruments[5];
    std::string windInstruments[5];
    std::string percInstruments[5];

public:
    Musicians() {
        // Initialize the string array with string instruments
        stringInstruments[0] = "veena";
        stringInstruments[1] = "guitar";
        stringInstruments[2] = "sitar";
        stringInstruments[3] = "sarod";
        stringInstruments[4] = "mandolin";

        // Initialize the wind array with wind instruments
        windInstruments[0] = "flute";
        windInstruments[1] = "clarinet";
        windInstruments[2] = "saxophone";
        windInstruments[3] = "nadhaswaram";
        windInstruments[4] = "piccolo";

        // Initialize the perc array with percussion instruments
        percInstruments[0] = "tabla";
        percInstruments[1] = "mridangam";
        percInstruments[2] = "bongos";
        percInstruments[3] = "drums";
        percInstruments[4] = "tambour";
    }
```

```
    void displayStringInstruments() {
        std::cout << "String Instruments:" << std::endl;
        for (int i = 0; i < 5; i++) {
            std::cout << stringInstruments[i] << std::endl;
        }
    }

    void displayWindInstruments() {
        std::cout << "Wind Instruments:" << std::endl;
        for (int i = 0; i < 5; i++) {
            std::cout << windInstruments[i] << std::endl;
        }
    }

    void displayPercInstruments() {
        std::cout << "Percussion Instruments:" << std::endl;
        for (int i = 0; i < 5; i++) {
            std::cout << percInstruments[i] << std::endl;
        }
    }
};

class TypeIns : public Musicians {
public:
    void get() {
        char choice;
        std::cout << "Type of instruments to be displayed:" << std::endl;
        std::cout << "a. String instruments" << std::endl;
        std::cout << "b. Wind instruments" << std::endl;
        std::cout << "c. Percussion instruments" << std::endl;
        std::cout << "Enter your choice: ";
        std::cin >> choice;

        switch (choice) {
            case 'a':
                displayStringInstruments();
                break;
            case 'b':
                displayWindInstruments();
                break;
            case 'c':
```

```
            displayPercInstruments();
            break;
        default:
            std::cout << "Invalid choice" << std::endl;
    }
  }

  void show() {
    get();
  }
};

int main() {
  TypeIns typeIns;
  typeIns.show();
  return 0;
}
```

Output:

```
Type of instruments to be displayed:
a. String instruments
b. Wind instruments
c. Percussion instruments
Enter your choice: a
String Instruments:
veena
guitar
sitar
sarod
mandolin
Enter your choice: b
Wind Instruments:
flute
clarinet
saxophone
nadhaswaram
piccolo
Enter your choice: c
Percussion Instruments:
tabla
mridangam
bongos
```

drums
tambour

4. Write three derived classes inheriting functionality of base class `person` (should have a member function that asks to enter name and age) and with added unique features of `student`, and `employee`, and functionality to assign, change and delete records of student and employee. And make one member function for printing the address of the objects of classes (base and derived) using this pointer. Create two objects of the base class and derived classes each and print the addresses of individual objects. Using a calculator, calculate the address space occupied by each object and verify this with address spaces printed by the program.

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

class Person {
protected:
    std::string name;
    int age;

public:
    Person() : name(""), age(0) {}

    void enterData() {
        std::cout << "Enter name: ";
        std::cin >> name;
        std::cout << "Enter age: ";
        std::cin >> age;
    }

    void printAddress() {
        std::cout << "Address of Person object: " << this << std::endl;
    }
};

class Student : public Person {
private:
    int studentID;

public:
```

```
    Student() : studentID(0) {}

    void assignStudentRecord(int id) {
        studentID = id;
    }

    void changeStudentRecord(int id) {
        studentID = id;
    }

    void deleteStudentRecord() {
        studentID = 0;
    }

    void printAddress() {
        std::cout << "Address of Student object: " << this << std::endl;
    }
};

class Employee : public Person {
private:
    int employeeID;

public:
    Employee() : employeeID(0) {}

    void assignEmployeeRecord(int id) {
        employeeID = id;
    }

    void changeEmployeeRecord(int id) {
        employeeID = id;
    }

    void deleteEmployeeRecord() {
        employeeID = 0;
    }

    void printAddress() {
        std::cout << "Address of Employee object: " << this << std::endl;
    }
};
```

```
int main() {
    Person person1;
    Person person2;
    Student student1;
    Student student2;
    Employee employee1;
    Employee employee2;

    person1.enterData();
    person2.enterData();
    student1.enterData();
    student2.enterData();
    employee1.enterData();
    employee2.enterData();

    person1.printAddress();
    person2.printAddress();
    student1.printAddress();
    student2.printAddress();
    employee1.printAddress();
    employee2.printAddress();

    // Calculate address space occupied by objects
    std::cout << "Size of Person object: " << sizeof(Person) << " bytes" <<
        std::endl;
    std::cout << "Size of Student object: " << sizeof(Student) << " bytes" <<
        std::endl;
    std::cout << "Size of Employee object: " << sizeof(Employee) << "
        bytes" << std::endl;

    return 0;
}
```

Output

```
Enter name: Ramesh
Enter age: 21
Enter name: Hari
Enter age: 23
Enter name: Santosh
Enter age: 21
Enter name: Srinivash
Enter age: 22
```

```
Enter name: Swami
Enter age: 21
Enter name: Satyam
Enter age: 21
Address of Person object: 0x16b1e2fe8
Address of Person object: 0x16b1e2fc8
Address of Student object: 0x16b1e2f98
Address of Student object: 0x16b1e2f78
Address of Employee object: 0x16b1e2f58
Address of Employee object: 0x16b1e2f38
Size of Person object: 32 bytes
Size of Student object: 32 bytes
Size of Employee object: 32 bytes
```

5. Write a base class that asks the user to enter a complex number and make a derived class that adds the complex number of its own with the base. Finally, make a third class that is a friend of derived and calculate the difference of the base complex number and its own complex number.

```
#include <iostream>

class Complex {
protected:
    double real;
    double imag;

public:
    Complex() : real(0.0), imag(0.0) {}

    Complex(double r, double i) : real(r), imag(i) {}

    Complex operator+(const Complex& other) const {
        return Complex(real + other.real, imag + other.imag);
    }

    Complex operator-(const Complex& other) const {
        return Complex(real - other.real, imag - other.imag);
    }

    void enterComplexNumber() {
        std::cout << "Enter real part: ";
```

```
        std::cin >> real;
        std::cout << "Enter imaginary part: ";
        std::cin >> imag;
    }

    void displayComplexNumber() const {
        std::cout << real << " + " << imag << "i" << std::endl;
    }
};

int main() {
    Complex complex1;
    Complex complex2;

    std::cout << "Enter the first complex number:" << std::endl;
    complex1.enterComplexNumber();

    std::cout << "Enter the second complex number:" << std::endl;
    complex2.enterComplexNumber();

    Complex resultAddition = complex1 + complex2;
    Complex resultSubtraction = complex1 - complex2;

    std::cout << "Result of addition: ";
    resultAddition.displayComplexNumber();

    std::cout << "Result of subtraction: ";
    resultSubtraction.displayComplexNumber();

    return 0;
}
```

Output:

```
Enter the first complex number:
Enter real part: 1
Enter imaginary part: 2
Enter the second complex number:
Enter real part: 1
Enter imaginary part: 2
Result of addition: 2 + 4i
Result of subtraction: 0 + 0iß
```