Week1

Exercise 1

Data type Name	Class definition
Car	`class Car {` `private:  // private data members` `  string make;` `  string model;` `  int year;` `  int price;` `  int mileage;` `  string colour;` `public: // public member functions` `  // constructor` `  Car(string make, string model, int year, int price, int mileage, string colour) {` `  this->make = make;` `  this->model = model;` `  this->year = year;` `  this->price = price;` `  this->mileage = mileage;` `  this->colour = colour;` `  }` `  // print function` `  void print() {` `  cout << "Make: " << make << endl;` `  cout << "Model: " << model << endl;` `  cout << "Year: " << year << endl;` `  cout << "Price: " << price << endl;` `  cout << "Mileage: " << mileage << endl;` `  cout << "Colour: " << colour << endl;` `  }` `};`
Bird	`class Bird {` `  private:  // private data members` `  string name;` `  string color;` `  int age;` `  int weight;` `  string type;` `  public: // public member functions` `  // constructor` `  Bird(string name, string color, int age, int weight, string type) {` `  this->name = name;` `  this->color = color;` `  this->age = age;`

	`` this->weight = weight; this->type = type; } // print function void print() { cout << "Name: " << name << endl; cout << "Color: " << color << endl; cout << "Age: " << age << endl; cout << "Weight: " << weight << endl; cout << "Type: " << type << endl; } }; ```
MobilePhone	``` class MobilePhone{ private: // private data members string brand; string model; int year; int price; int memory; string color; public: // public member functions // constructor MobilePhone(string brand, string model, int year, int price, int memory, string color) { this->brand = brand; this->model = model; this->year = year; this->price = price; this->memory = memory; this->color = color; } // print function void print() { cout << "Brand: " << brand << endl; cout << "Model: " << model << endl; cout << "Year: " << year << endl; cout << "Price: " << price << endl; cout << "Memory: " << memory << endl; cout << "Color: " << color << endl; } }; ```
TV	``` class TV{ private: // private data members string brand; string model; int year; int price; int size; ```

	`` string color; public: // public member functions // constructor TV(string brand, string model, int year, int price, int size, string color) { this->brand = brand; this->model = model; this->year = year; this->price = price; this->size = size; this->color = color; } // print function void print() { cout << "Brand: " << brand << endl; cout << "Model: " << model << endl; cout << "Year: " << year << endl; cout << "Price: " << price << endl; cout << "Size: " << size << endl; cout << "Color: " << color << endl; } }; ```
Chair	``` class Chair{ private: // private data members int size; int price; string color; string material; public: // constructor Chair(int size, int price, string color, string material) { this->size = size; this->price = price; this->color = color; this->material = material; } // print function void print() { cout << "Size: " << size << endl; cout << "Price: " << price << endl; cout << "Color: " << color << endl; cout << "Material: " << material << endl; } }; ```
Train	``` class Train{ private: // private data members string name; ```

	`` string type; int year; int price; int speed; string color; public: // public member functions // constructor Train(string name, string type, int year, int price, int speed, string color) { this->name = name; this->type = type; this->year = year; this->price = price; this->speed = speed; this->color = color; } // print function void print() { cout << "Name: " << name << endl; cout << "Type: " << type << endl; cout << "Year: " << year << endl; cout << "Price: " << price << endl; cout << "Speed: " << speed << endl; cout << "Color: " << color << endl; } }; ```
Boat	``` class Boat{ private: // private data members string name; string type; int year; int price; int speed; string color; public: // public member functions // constructor Boat(string name, string type, int year, int price, int speed, string color) { this->name = name; this->type = type; this->year = year; this->price = price; this->speed = speed; this->color = color; } // print function void print() { cout << "Name: " << name << endl; ```

	`` cout << "Type: " << type << endl; cout << "Year: " << year << endl; cout << "Price: " << price << endl; cout << "Speed: " << speed << endl; cout << "Color: " << color << endl; } }; ```
Bear	``` class Bear{ private: // private data members string name; string type; int age; int weight; string color; public: // public member functions // constructor Bear(string name, string type, int age, int weight, string color) { this->name = name; this->type = type; this->age = age; this->weight = weight; this->color = color; } // print function void print() { cout << "Name: " << name << endl; cout << "Type: " << type << endl; cout << "Age: " << age << endl; cout << "Weight: " << weight << endl; cout << "Color: " << color << endl; } }; ```
Pen	``` class Pen{ private: // private data members string brand; string type; int price; int size; string color; public: // public member functions // constructor Pen(string brand, string type, int price, int size, string color) { this->brand = brand; this->type = type; this->price = price; this->size = size; ```

	`` this->color = color; } // print function void print() { cout << "Brand: " << brand << endl; cout << "Type: " << type << endl; cout << "Price: " << price << endl; cout << "Size: " << size << endl; cout << "Color: " << color << endl; } }; ```
ClassRoom	``` class ClassRoom{ private: // private data members int number; int capacity; string color; string type; public: // constructor ClassRoom(int number, int capacity, string color, string type) { this->number = number; this->capacity = capacity; this->color = color; this->type = type; } // print function void print() { cout << "Number: " << number << endl; cout << "Capacity: " << capacity << endl; cout << "Color: " << color << endl; cout << "Type: " << type << endl; } }; ```
ComputerRoom	``` class ComputerRoom{ private: // private data members int number; int capacity; string color; string type; public: // constructor ComputerRoom(int number, int capacity, string color, string type) { this->number = number; this->capacity = capacity; this->color = color; this->type = type; ```

```
        }
        // print function
        void print() {
            cout << "Number: " << number << endl;
            cout << "Capacity: " << capacity << endl;
            cout << "Color: " << color << endl;
            cout << "Type: " << type << endl;
        }
};
```

Car testing:

Main code:

```
int main()
{
    //create a car object
    Car car1("Toyota", "Camry", 2015, 20000, 4, "Black");
    //print the car object
    car1.print();
}
```

Output:

```
Make: Toyota
Model: Camry
Year: 2015
Price: 20000
Mileage: 4
Color: Black
```

Bird testing:

```
int main()
{
    //create a bird object
    Bird bird1("Eagle", "Bird", 2010, 100, "White");
    //print bird object
    bird1.print();
}
```

Output:

```
Name: Eagle
Color: Bird
Age: 2010
Weight: 100
Type: White
```

The rest were programmed in the exact same way so I believe that testing will suffice.

Week2

Exercise 1

Code:

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

//clas Time
class Time{
    private:
        int hour;
        int minute;
        int second;
    public:
        Time(){    // Default constructor
            hour = 0;
            minute = 0;
            second = 0;
        }
        //set method
        void setTime(int h, int m, int s){
            hour = h;
            minute = m;
            second = s;
        }
        //print military time
        void printMilitary(){
            cout << (hour < 10 ? "0" : "") << hour << ":";
            cout << (minute < 10 ? "0" : "") << minute << ":";
        }
        //print standard time
        void printStandard(){
            cout << ((hour == 0 || hour == 12) ? 12 : hour % 12);
            cout << ":" << (minute < 10 ? "0" : "") << minute;
            cout << ":" << (second < 10 ? "0" : "") << second;
            cout << (hour < 12 ? " AM" : " PM");
        }
        //tick method
        void tick(){
            second++;
            if(second == 60){
                second = 0;
                minute++;
                if(minute == 60){
                    minute = 0;
                    hour++;
                    if(hour == 24){
                        hour = 0;
                    }
                }
            }
        }
```

```
};
```

Testing:
Main code:

```
int main(){
    bool loop = true;
    Time t1;
    while(loop){
        t1.tick();
        t1.printStandard();
        cout << endl;
    }
}
```

Outputs:

```
12:04:55 AM
12:04:56 AM
12:04:57 AM
12:04:58 AM
12:04:59 AM
12:05:00 AM
12:05:01 AM
12:05:02 AM
12:05:03 AM
12:05:04 AM
12:05:05 AM
12:05:06 AM
```

These test for second and minute increments and they work fine.

```
12:59:54 AM
12:59:55 AM
12:59:56 AM
12:59:57 AM
12:59:58 AM
12:59:59 AM
1:00:00 AM
1:00:01 AM
1:00:02 AM
1:00:03 AM
1:00:04 AM
```

This shows that the minute and hour increments are working fine.

Exercise 2

Code:

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

class Rectangle {
    private:    // Private members
        float width;
        float height;
    public:    // Public members
```

```
        Rectangle(){    // Default constructor
            width = 1;
            height = 1;
        }
        //set and get methods
        void setWidth(float w) {
            if (w > 0 || w < 20) {  // Check if width is between 0 and 20
                width = w;
            }
            else{
                cout << "that number is invalid" << endl;
            }
        }
        void setHeight(float h) {
            if (h > 0 || h < 20) {  // Check if height is between 0 and 20
                height = h;
            }
            else{
                cout << "that number is invalid" << endl;
            }
        }
        int getWidth() {    // Get width
            return width;
        }
        int getHeight() {   // Get height
            return height;
        }
        // method to calculate area
        float area() {
            return width * height;
        }
        // method to calculate perimeter
        float perimeter() {
            return 2 * (width + height);
        }
};
```

Testing:

Main code:

```
int main()
{
    // Create a rectangle object
    Rectangle r1;
    // Set width and height
    r1.setWidth(4);
    r1.setHeight(40);
    // Display width, height, area, and perimeter
    cout << "Width: " << r1.getWidth() << endl;
```

```
    cout << "Height: " << r1.getHeight() << endl;
    cout << "Area: " << r1.area() << endl;
    cout << "Perimeter: " << r1.perimeter() << endl;
}
```

Output:

```
Width: 4
Height: 40
Area: 160
Perimeter: 88
```

Exercise 3

```
class SophisticatedRectangle{
    private:
        float cornerOne[2];
        float cornerTwo[2];
    public:
        //set functions
        void setCornerOne(float x, float y) {
            //if both x and y are between 0 and 20
            if (x > 0 || x < 20 && y > 0 || y < 20) {
                cornerOne[0] = x;
                cornerOne[1] = y;
            }
            else {
                cout << "that number is invalid" << endl;
            }
        }
        void setCornerTwo(float x, float y) {
            //if both x and y are between 0 and 20
            if (x > 0 || x < 20 && y > 0 || y < 20) {
                cornerTwo[0] = x;
                cornerTwo[1] = y;
            }
            else {
                cout << "that number is invalid" << endl;
            }
        }
        //constructor that uses the set functions
        SophisticatedRectangle(float x1, float y1, float x2, float y2) {
            setCornerOne(x1, y1);
            setCornerTwo(x2, y2);
        }
        //method to calculate length of the rectangle
        float length(){
            //find x length
            float xLength = cornerTwo[0] - cornerOne[0];
            //find y length
```

```
            float yLength = cornerTwo[1] - cornerOne[1];
            //return the larger length
            if (xLength > yLength) {
                return xLength;
            }
            else {
                return yLength;
            }
        }
        //method to calculate width of the rectangle
        float width(){
            //find x length
            float xLength = cornerTwo[0] - cornerOne[0];
            //find y length
            float yLength = cornerTwo[1] - cornerOne[1];
            //return the smaller length
            if (xLength < yLength) {
                return xLength;
            }
            else {
                return yLength;
            }
        }
        //function to check if coordinates form a rectangle using length and
width
        bool isRectangle() {
            if (length() == width()) {
                return false;
            }
            else {
                return true;
            }
        }
        //method to calculate perimeter of the rectangle using the length and
width methods
        float perimeter() {
            return 2 * (length() + width());
        }
        //method to calculate area of the rectangle using the length and width
methods
        float area() {
            return length() * width();
        }
        //method to check if the rectangle is a square
        bool isSquare() {
            if (length() == width()) {
                return true;
            }
```

```
            else {
                return false;
            }
        }
};
```

Testing:

Main code:

```
int main(){
    //create a rectangle object
    SophisticatedRectangle r1(1, 1, 4, 6);
    //display the length, width, perimeter, and area
    cout << "Length: " << r1.length() << endl;
    cout << "Width: " << r1.width() << endl;
    cout << "Perimeter: " << r1.perimeter() << endl;
    cout << "Area: " << r1.area() << endl;
    //check if the rectangle is a square
    if (r1.isSquare()) {
        cout << "This is a square" << endl;
    }
    else {
        cout << "This is not a square" << endl;
    }
    //check if the rectangle is a rectangle
    if (r1.isRectangle()) {
        cout << "This is a rectangle" << endl;
    }
    else {
        cout << "This is not a rectangle" << endl;
    }
}
```

Output:

```
Length: 5
Width: 3
Perimeter: 16
Area: 15
This is not a square
This is a rectangle
```

Exercise 4

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

class ComplexNumbers
{
    private:    //private variables
```

```
        double real;
        double imaginary;
    public:    //public methods
        //constructors
        ComplexNumbers(double r, double i){   //constructor with parameters
            real = r;
            imaginary = i;
        }
        ComplexNumbers(){   //default constructor
            real = 0;
            imaginary = 0;
        }
        //ComplexNumbers();
        //addition
        ComplexNumbers operator+(ComplexNumbers &c2);
        //subtraction
        ComplexNumbers operator-(ComplexNumbers &c2);
        //multiplication
        ComplexNumbers operator*(ComplexNumbers &c2);
        //division
        ComplexNumbers operator/(ComplexNumbers &c2);
        //print in the form (a, b) where a is the real part and b is the
imaginary part
        void print();

};



//addition
ComplexNumbers ComplexNumbers::operator+(ComplexNumbers &c2)
{
    ComplexNumbers temp;
    temp.real = real + c2.real;
    temp.imaginary = imaginary + c2.imaginary;
    return temp;
}

//subtraction
ComplexNumbers ComplexNumbers::operator-(ComplexNumbers &c2)
{
    ComplexNumbers temp;
    temp.real = real - c2.real;
    temp.imaginary = imaginary - c2.imaginary;
    return temp;
}

//multiplication
ComplexNumbers ComplexNumbers::operator*(ComplexNumbers &c2)
```

```
{
    ComplexNumbers temp;
    temp.real = (real * c2.real) - (imaginary * c2.imaginary);
    temp.imaginary = (real * c2.imaginary) + (imaginary * c2.real);
    return temp;
}

//division
ComplexNumbers ComplexNumbers::operator/(ComplexNumbers &c2)
{
    ComplexNumbers temp;
    temp.real = ((real * c2.real) + (imaginary * c2.imaginary)) / ((c2.real *
c2.real) + (c2.imaginary * c2.imaginary));
    temp.imaginary = ((imaginary * c2.real) - (real * c2.imaginary)) /
((c2.real * c2.real) + (c2.imaginary * c2.imaginary));
    return temp;
}

//print in the form (a, b) where a is the real part and b is the imaginary
part
void ComplexNumbers::print()
{
    cout << "(" << real << ", " << imaginary << ")" << endl;
}
```

Testing:

Main code:

```
int main()
{
    bool loop = true;   //main loop
    while(loop == true){
        //display menu
        cout << "1. Add two complex numbers" << endl;
        cout << "2. Subtract two complex numbers" << endl;
        cout << "3. Multiply two complex numbers" << endl;
        cout << "4. Divide two complex numbers" << endl;
        cout << "5. Exit" << endl;
        int choice; //user input
        cin >> choice;  //get user input
        if(choice == 1){    //addition
            double real1, imaginary1, real2, imaginary2; //variables for user
input
            cout << "Enter the real part of the first complex number: ";
            cin >> real1;
            cout << "Enter the imaginary part of the first complex number: ";
            cin >> imaginary1;
            cout << "Enter the real part of the second complex number: ";
```

```
            cin >> real2;
            cout << "Enter the imaginary part of the second complex number: ";
            cin >> imaginary2;
            ComplexNumbers c1(real1, imaginary1);    //create first complex
number
            ComplexNumbers c2(real2, imaginary2);    //create second complex
number
            ComplexNumbers c3 = c1 + c2;    //add the two complex numbers
            cout << "The sum of the two complex numbers is: ";
            c3.print(); //print the sum
        }
        else if(choice == 2){   //subtraction
            double real1, imaginary1, real2, imaginary2; //variables for user
input
            cout << "Enter the real part of the first complex number: ";
            cin >> real1;
            cout << "Enter the imaginary part of the first complex number: ";
            cin >> imaginary1;
            cout << "Enter the real part of the second complex number: ";
            cin >> real2;
            cout << "Enter the imaginary part of the second complex number: ";
            cin >> imaginary2;
            ComplexNumbers c1(real1, imaginary1);    //create first complex
number
            ComplexNumbers c2(real2, imaginary2);    //create second complex
number
            ComplexNumbers c3 = c1 - c2;    //subtract the two complex numbers
            cout << "The difference of the two complex numbers is: ";
            c3.print(); //print the difference
        }
        else if(choice == 3){   //multiplication
            double real1, imaginary1, real2, imaginary2; //variables for user
input
            cout << "Enter the real part of the first complex number: ";
            cin >> real1;
            cout << "Enter the imaginary part of the first complex number: ";
            cin >> imaginary1;
            cout << "Enter the real part of the second complex number: ";
            cin >> real2;
            cout << "Enter the imaginary part of the second complex number: ";
            cin >> imaginary2;
            ComplexNumbers c1(real1, imaginary1);    //create first complex
number
            ComplexNumbers c2(real2, imaginary2);    //create second complex
number
            ComplexNumbers c3 = c1 * c2;    //multiply the two complex numbers
            cout << "The product of the two complex numbers is: ";
            c3.print(); //print the product
```

```
        }
        else if(choice == 4){   //division
            double real1, imaginary1, real2, imaginary2; //variables for user
input
            cout << "Enter the real part of the first complex number: ";
            cin >> real1;
            cout << "Enter the imaginary part of the first complex number: ";
            cin >> imaginary1;
            cout << "Enter the real part of the second complex number: ";
            cin >> real2;
            cout << "Enter the imaginary part of the second complex number: ";
            cin >> imaginary2;
            ComplexNumbers c1(real1, imaginary1);    //create first complex
number
            ComplexNumbers c2(real2, imaginary2);    //create second complex
number
            ComplexNumbers c3 = c1 / c2;    //divide the two complex numbers
            cout << "The quotient of the two complex numbers is: ";
            c3.print(); //print the quotient
        }
        else if(choice == 5){   //exit
            loop = false;
        }
        else{   //invalid input
            cout << "Please input a valid number" << endl;
        }

    }
}
```

Output for addition:

```
1. Add two complex numbers
2. Subtract two complex numbers
3. Multiply two complex numbers
4. Divide two complex numbers
5. Exit
1
Enter the real part of the first complex number: 3
Enter the imaginary part of the first complex number: 4
Enter the real part of the second complex number: 3
Enter the imaginary part of the second complex number: 6
The sum of the two complex numbers is: (6, 10)
```

Output for subtraction:

```
1. Add two complex numbers
2. Subtract two complex numbers
3. Multiply two complex numbers
4. Divide two complex numbers
5. Exit
2
Enter the real part of the first complex number: 4
Enter the imaginary part of the first complex number: 7
Enter the real part of the second complex number: 2
Enter the imaginary part of the second complex number: 3
The difference of the two complex numbers is: (2, 4)
```

Output for multiplication:

```
2. Subtract two complex numbers
3. Multiply two complex numbers
4. Divide two complex numbers
5. Exit
3
Enter the real part of the first complex number: 4
Enter the imaginary part of the first complex number: 5
Enter the real part of the second complex number: 2
Enter the imaginary part of the second complex number: 6
The product of the two complex numbers is: (-22, 34)
```

Output for division:

```
  1. Add two complex numbers
  2. Subtract two complex numbers
  3. Multiply two complex numbers
  4. Divide two complex numbers
  5. Exit
  4
  Enter the real part of the first complex number: 4
  Enter the imaginary part of the first complex number: 5
  Enter the real part of the second complex number: 2
  Enter the imaginary part of the second complex number: 2
  The quotient of the two complex numbers is: (2.25, 0.25)
```

Week3

Exercise 1

Code:

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

class RationalNumbers
{
    private:    //private data members
        int numerator;
        int denominator;
        //find reduced rational number
```

```
        void reduceRational(int num, int den);
    public: //public methods
        //default constructor
        RationalNumbers()
        {
            numerator = 0;
            denominator = 1;
        }
        //constructor with parameters
        RationalNumbers(int num, int den);
        //set numerator
        void setNumerator(int num);
        //set denominator
        void setDenominator(int den);
        //get numerator
        int getNumerator();
        //get denominator
        int getDenominator();
        //set rational number
        void setRational(int num, int den);
        //add rational numbers
        void addRational(RationalNumbers r1, RationalNumbers r2);
        //subtract rational numbers
        void subtractRational(RationalNumbers r1, RationalNumbers r2);
        //multiply rational numbers
        void multiplyRational(RationalNumbers r1, RationalNumbers r2);
        //divide rational numbers
        void divideRational(RationalNumbers r1, RationalNumbers r2);
        //is greater than
        bool isGreater(RationalNumbers r1, RationalNumbers r2);
        //is equal to
        bool isEqual(RationalNumbers r1, RationalNumbers r2);
        //is between
        bool isBetween(RationalNumbers r1, RationalNumbers r2, RationalNumbers
r3);
        //print rational number in the form of a/b
        void printRational();
};

RationalNumbers::RationalNumbers(int num, int den)
{
    //find the reduced form of the fraction using the parameterized
constructor
    int gcd = 1;
    int j = 2;
    while (j <= num && j <= den){
        if (num % j == 0 && den % j == 0)
            gcd = j;
```

```
        j++;
    }
    numerator = num / gcd;
    denominator = den / gcd;
}

//set methods
void RationalNumbers::setNumerator(int num)
{
    numerator = num;
}
void RationalNumbers::setDenominator(int den)
{
    denominator = den;
}

//get methods
int RationalNumbers::getNumerator()
{
    return numerator;
}
int RationalNumbers::getDenominator()
{
    return denominator;
}

void RationalNumbers::reduceRational(int num, int den)
{
    //find the reduced form of the fraction
    int gcd = 1;
    int j = 2;
    while (j <= num && j <= den){
        if (num % j == 0 && den % j == 0)
            gcd = j;
        j++;
    }
    numerator = num / gcd;
    denominator = den / gcd;
}
//set rational number
void RationalNumbers::setRational(int num, int den)
{
    //set the rational number
    numerator = num;
    denominator = den;
}

//add rational numbers
```

```
void RationalNumbers::addRational(RationalNumbers r1, RationalNumbers r2)
{
    //add two rational numbers
    int num = r1.numerator * r2.denominator + r2.numerator * r1.denominator;
    int den = r1.denominator * r2.denominator;
    reduceRational(num, den);      //reduce the rational number
}
//subtract rational numbers
void RationalNumbers::subtractRational(RationalNumbers r1, RationalNumbers r2)
{
    //subtract two rational numbers
    int num = r1.numerator * r2.denominator - r2.numerator * r1.denominator;
    int den = r1.denominator * r2.denominator;
    reduceRational(num, den);      //reduce the rational number
}
//multiply rational numbers
void RationalNumbers::multiplyRational(RationalNumbers r1, RationalNumbers r2)
{
    //multiply two rational numbers
    int num = r1.numerator * r2.numerator;
    int den = r1.denominator * r2.denominator;
    reduceRational(num, den);      //reduce the rational number
}
//divide rational numbers
void RationalNumbers::divideRational(RationalNumbers r1, RationalNumbers r2)
{
    //divide two rational numbers
    int num = r1.numerator * r2.denominator;
    int den = r1.denominator * r2.numerator;
    reduceRational(num, den);      //reduce the rational number
}

//is greater than
bool RationalNumbers::isGreater(RationalNumbers r1, RationalNumbers r2)
{
    //check if r1 is greater than r2
    if (r1.numerator * r2.denominator > r2.numerator * r1.denominator)
        return true;
    else
        return false;
}
//is equal to
bool RationalNumbers::isEqual(RationalNumbers r1, RationalNumbers r2)
{
    //check if r1 is equal to r2
    if (r1.numerator * r2.denominator == r2.numerator * r1.denominator)
        return true;
    else
```

```
        return false;
}
//is between
bool RationalNumbers::isBetween(RationalNumbers r1, RationalNumbers r2,
RationalNumbers r3)
{
    //check if r1 is between r2 and r3
    if (r1.numerator * r2.denominator > r2.numerator * r1.denominator &&
r1.numerator * r3.denominator < r3.numerator * r1.denominator)
        return true;
    else
        return false;
}

//print rational number in the form of a/b
void RationalNumbers::printRational()
{
    //print the rational number
    cout << numerator << "/" << denominator;
}
```

Testing:

Main code:

```
int main(){
    bool loop = true;   //main loop
    while(loop == true){
        //main menu
        cout << "1. Add two rational numbers" << endl;
        cout << "2. Subtract two rational numbers" << endl;
        cout << "3. Multiply two rational numbers" << endl;
        cout << "4. Divide two rational numbers" << endl;
        cout << "5. Check if one rational number is greater than another" <<
endl;
        cout << "6. Check if one rational number is equal to another" << endl;
        cout << "7. Check if one rational number is between two other rational
numbers" << endl;
        cout << "8. Exit" << endl;
        //get user input
        int option;
        cout << "Enter your choice: ";
        cin >> option;
        if(option == 1)//add two rational numbers
        {
            //get user input
            int num1, den1, num2, den2;
            cout << "Enter the numerator of the first rational number: ";
            cin >> num1;
```

```
            cout << "Enter the denominator of the first rational number: ";
            cin >> den1;
            cout << "Enter the numerator of the second rational number: ";
            cin >> num2;
            cout << "Enter the denominator of the second rational number: ";
            cin >> den2;
            //create rational numbers
            RationalNumbers r1(num1, den1);
            RationalNumbers r2(num2, den2);
            RationalNumbers r3(0, 0);
            //add the rational numbers
            r3.addRational(r1, r2);
            //print the result
            cout << "The sum of the rational numbers is: ";
            r3.printRational();
            cout << endl;
        }
        else if(option == 2)//subtract two rational numbers
        {
            //get user input
            int num1, den1, num2, den2;
            cout << "Enter the numerator of the first rational number: ";
            cin >> num1;
            cout << "Enter the denominator of the first rational number: ";
            cin >> den1;
            cout << "Enter the numerator of the second rational number: ";
            cin >> num2;
            cout << "Enter the denominator of the second rational number: ";
            cin >> den2;
            //create rational numbers
            RationalNumbers r1(num1, den1);
            RationalNumbers r2(num2, den2);
            RationalNumbers r3(0, 0);
            //subtract the rational numbers
            r3.subtractRational(r1, r2);
            //print the result
            cout << "The difference of the rational numbers is: ";
            r3.printRational();
            cout << endl;
        }
        else if(option == 3)//multiply two rational numbers
        {
            //get user input
            int num1, den1, num2, den2;
            cout << "Enter the numerator of the first rational number: ";
            cin >> num1;
            cout << "Enter the denominator of the first rational number: ";
            cin >> den1;
```

```
            cout << "Enter the numerator of the second rational number: ";
            cin >> num2;
            cout << "Enter the denominator of the second rational number: ";
            cin >> den2;
            //create rational numbers
            RationalNumbers r1(num1, den1);
            RationalNumbers r2(num2, den2);
            RationalNumbers r3(0, 0);
            //multiply the rational numbers
            r3.multiplyRational(r1, r2);
            //print the result
            cout << "The product of the rational numbers is: ";
            r3.printRational();
            cout << endl;
        }
        else if(option == 4)//divide two rational numbers
        {
            //get user input
            int num1, den1, num2, den2;
            cout << "Enter the numerator of the first rational number: ";
            cin >> num1;
            cout << "Enter the denominator of the first rational number: ";
            cin >> den1;
            cout << "Enter the numerator of the second rational number: ";
            cin >> num2;
            cout << "Enter the denominator of the second rational number: ";
            cin >> den2;
            //create rational numbers
            RationalNumbers r1(num1, den1);
            RationalNumbers r2(num2, den2);
            RationalNumbers r3(0, 0);
            //divide the rational numbers
            r3.divideRational(r1, r2);
            //print the result
            cout << "The quotient of the rational numbers is: ";
            r3.printRational();
            cout << endl;
        }
        else if(option == 5)//check if one rational number is greater than
another
        {
            //get user input
            int num1, den1, num2, den2;
            cout << "Enter the numerator of the first rational number: ";
            cin >> num1;
            cout << "Enter the denominator of the first rational number: ";
            cin >> den1;
            cout << "Enter the numerator of the second rational number: ";
```

```
            cin >> num2;
            cout << "Enter the denominator of the second rational number: ";
            cin >> den2;
            //create rational numbers
            RationalNumbers r1(num1, den1);
            RationalNumbers r2(num2, den2);
            //check if r1 is greater than r2
            if(r1.isGreater(r1, r2) == true)
                cout << "The first rational number is greater than the second
rational number" << endl;
            else
                cout << "The first rational number is not greater than the
second rational number" << endl;
        }
        else if(option == 6)//check if one rational number is equal to another
        {
            //get user input
            int num1, den1, num2, den2;
            cout << "Enter the numerator of the first rational number: ";
            cin >> num1;
            cout << "Enter the denominator of the first rational number: ";
            cin >> den1;
            cout << "Enter the numerator of the second rational number: ";
            cin >> num2;
            cout << "Enter the denominator of the second rational number: ";
            cin >> den2;
            //create rational numbers
            RationalNumbers r1(num1, den1);
            RationalNumbers r2(num2, den2);
            //check if r1 is equal to r2
            if(r1.isEqual(r1, r2) == true)
                cout << "The first rational number is equal to the second
rational number" << endl;
            else
                cout << "The first rational number is not equal to the second
rational number" << endl;
        }
        else if(option == 7) //check if the rational number is between two
others
        {
            //get user input
            int num1, den1, num2, den2, num3, den3;
            cout << "Enter the numerator of the first rational number: ";
            cin >> num1;
            cout << "Enter the denominator of the first rational number: ";
            cin >> den1;
            cout << "Enter the numerator of the second rational number: ";
            cin >> num2;
```

```
            cout << "Enter the denominator of the second rational number: ";
            cin >> den2;
            cout << "Enter the numerator of the third rational number: ";
            cin >> num3;
            cout << "Enter the denominator of the third rational number: ";
            cin >> den3;
            //create rational numbers
            RationalNumbers r1(num1, den1);
            RationalNumbers r2(num2, den2);
            RationalNumbers r3(num3, den3);
            //check if r1 is between r2 and r3
            if(r1.isBetween(r1, r2, r3) == true)
                cout << "The first rational number is between the second and
third rational numbers" << endl;
            else
                cout << "The first rational number is not between the second
and third rational numbers" << endl;
        }
        else if(option == 8)//print the rational number in floating point
format
        {
            loop = false;
        }
    }
}
```

Output:

Addition method:

```
1. Add two rational numbers
2. Subtract two rational numbers
3. Multiply two rational numbers
4. Divide two rational numbers
5. Check if one rational number is greater than another
6. Check if one rational number is equal to another
7. Check if one rational number is between two other rational numbers
8. Exit
Enter your choice: 1
Enter the numerator of the first rational number: 1
Enter the denominator of the first rational number: 4
Enter the numerator of the second rational number: 1
Enter the denominator of the second rational number: 4
The sum of the rational numbers is: 1/2
```

Subtraction method:

```
1. Add two rational numbers
2. Subtract two rational numbers
3. Multiply two rational numbers
4. Divide two rational numbers
5. Check if one rational number is greater than another
6. Check if one rational number is equal to another
7. Check if one rational number is between two other rational numbers
8. Exit
Enter your choice: 2
Enter the numerator of the first rational number: 3
Enter the denominator of the first rational number: 4
Enter the numerator of the second rational number: 1
Enter the denominator of the second rational number: 4
The difference of the rational numbers is: 1/2
```

Multiplication method:

```
1. Add two rational numbers
2. Subtract two rational numbers
3. Multiply two rational numbers
4. Divide two rational numbers
5. Check if one rational number is greater than another
6. Check if one rational number is equal to another
7. Check if one rational number is between two other rational numbers
8. Exit
Enter your choice: 3
Enter the numerator of the first rational number: 2
Enter the denominator of the first rational number: 5
Enter the numerator of the second rational number: 1
Enter the denominator of the second rational number: 3
The product of the rational numbers is: 2/15
```

Division method:

```
1. Add two rational numbers
2. Subtract two rational numbers
3. Multiply two rational numbers
4. Divide two rational numbers
5. Check if one rational number is greater than another
6. Check if one rational number is equal to another
7. Check if one rational number is between two other rational numbers
8. Exit
Enter your choice: 4
Enter the numerator of the first rational number: 3
Enter the denominator of the first rational number: 4
Enter the numerator of the second rational number: 1
Enter the denominator of the second rational number: 2
The quotient of the rational numbers is: 3/2
```

Greater than method:

```
1. Add two rational numbers
2. Subtract two rational numbers
3. Multiply two rational numbers
4. Divide two rational numbers
5. Check if one rational number is greater than another
6. Check if one rational number is equal to another
7. Check if one rational number is between two other rational numbers
8. Exit
Enter your choice: 5
Enter the numerator of the first rational number: 3
Enter the denominator of the first rational number: 4
Enter the numerator of the second rational number: 1
Enter the denominator of the second rational number: 2
The first rational number is greater than the second rational number
```

Equal method:

```
1. Add two rational numbers
2. Subtract two rational numbers
3. Multiply two rational numbers
4. Divide two rational numbers
5. Check if one rational number is greater than another
6. Check if one rational number is equal to another
7. Check if one rational number is between two other rational numbers
8. Exit
Enter your choice: 6
Enter the numerator of the first rational number: 1
Enter the denominator of the first rational number: 2
Enter the numerator of the second rational number: 2
Enter the denominator of the second rational number: 4
The first rational number is equal to the second rational number
```

Between method:

```
3. Multiply two rational numbers
4. Divide two rational numbers
5. Check if one rational number is greater than another
6. Check if one rational number is equal to another
7. Check if one rational number is between two other rational numbers
8. Exit
Enter your choice: 7
Enter the numerator of the first rational number: 1
Enter the denominator of the first rational number: 2
Enter the numerator of the second rational number: 1
Enter the denominator of the second rational number: 4
Enter the numerator of the third rational number: 3
Enter the denominator of the third rational number: 4
The first rational number is between the second and third rational numbers
```

Week4

Exercise 1

Point.h file:

```
#include <iostream>
```

```
using namespace std;
#pragma once

//class point
class Point {
private:    //private data members
    int x;
    int y;
public:    //public methods
    //constructor
    Point(int x, int y);
    //default constructor
    Point();
    //destructor that prints the point
    ~Point();
};
```

Point.cpp file:

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

Point::Point(int x, int y) {
    this->x = x;
    this->y = y;
}

Point::Point()
{
    this->x = 0;
    this->y = 0;
}

Point::~Point() {
    cout << "Point(" << x << ", " << y << ")" << endl;
}
```

Circle.h file:

```
#include <iostream>
#include "Point.h"
using namespace std;
#pragma once

//class Circle
class Circle {
private:    //private data members
    Point center;
    double radius;
```

```
public:    //public methods
    //constructor
    Circle(Point center, double radius);
    //default constructor
    Circle();
    //destructor that prints the circle
    ~Circle();
};
```

Circle.cpp file:

```
#include <iostream>
using namespace std;
#include "Circle.h"
#include "Point.h"

Circle::Circle(Point center, double radius) {
    this->center = center;
    this->radius = radius;
}

Circle::Circle()
{
    this->center = Point();
    this->radius = 0;
}

Circle::~Circle() {
    cout << "Radius: " << radius << endl;
}
```

Cylinder.h file:

```
#pragma once
#include <iostream>
#include "Circle.h"
using namespace std;

//class Cylinder
class Cylinder {
private:    //private data members
    Circle base;
    double height;
public:    //public methods
    //constructor
    Cylinder(Circle base, double height);
    //default constructor
    Cylinder();
    //destructor that prints the cylinder
    ~Cylinder();
```

```
};
```

Cylinder.cpp:

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

Cylinder::Cylinder(Circle base, double height) {
    this->base = base;
    this->height = height;
}

Cylinder::Cylinder()
{
    this->base = Circle();
    this->height = 0;
}

Cylinder::~Cylinder() {
    cout << "Height: " << height << endl;
}
```

Testing:

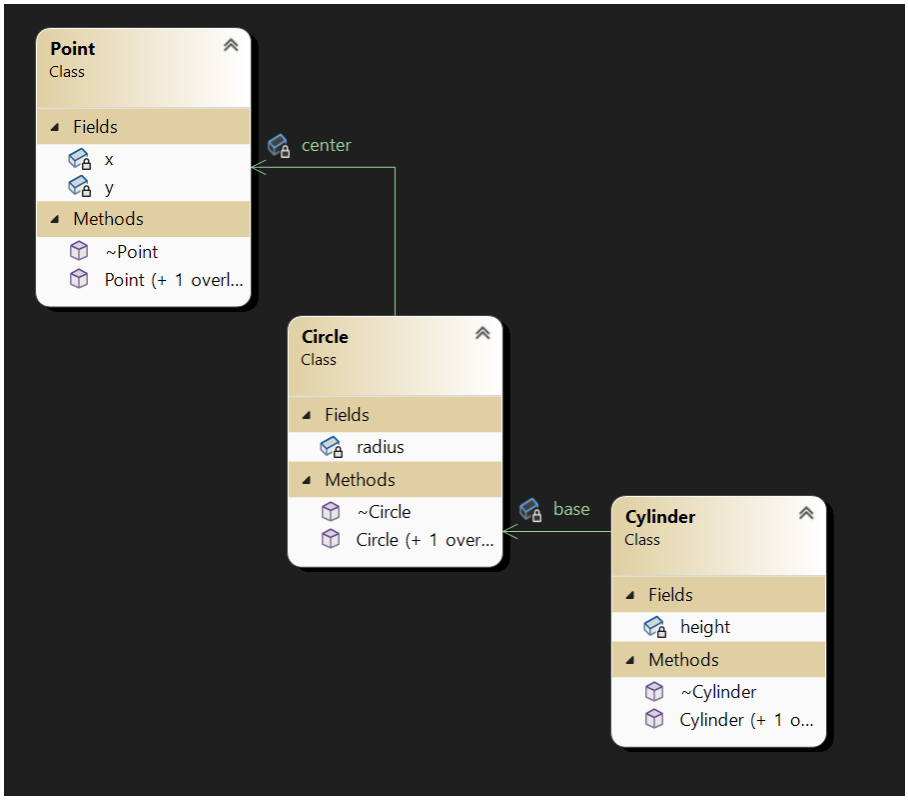
Main code:

```
#include <iostream>
using namespace std;
#include "Point.h"
#include "Circle.h"
#include "Cylinder.h"

int main() {
    //create a cylinder
    Cylinder c = Cylinder(Circle(Point(1, 2), 3), 4);
    return 0;
}
```

Output:

```
Microsoft Visual Studio Debug Consc
Point(1, 2)
Point(0, 0)
Radius: 3
Point(1, 2)
Radius: 3
Point(1, 2)
Point(1, 2)
Height: 4
Radius: 3
Point(1, 2)
```

UML:



Exercise 2

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

//Point class
class Point {
    protected:    //private data members
        int x;
        int y;
    public:    //public methods
        //constructor
        Point(int x, int y) {
            this->x = x;
            this->y = y;
```

```
        }
        //default constructor
        Point() {
            x = 0;
            y = 0;
        }
};

//Circle class that inherits from Point
class Circle: public Point {
    protected:    //private data members
        double radius;
    public:    //public methods
        //constructor
        Circle(int x, int y, double radius): Point(x, y) {
            this->radius = radius;
        }
        //default constructor
        Circle(): Point() {
            this->radius = 1;
        }
};

//Cylinder class that inherits from Circle
class Cylinder: public Circle {
    protected:    //private data members
        double height;
    public:    //public methods
        //constructor
        Cylinder(int x, int y, double radius, double height): Circle(x, y,
radius) {
            this->height = height;
        }
        //default constructor
        Cylinder(): Circle() {
            this->height = 1;
        }
        //print the cylinder
        void print() {
            cout << "Point: " << x << ", " << y << endl;
            cout << "Radius: " << radius << endl;
            cout << "Height: " << height << endl;
        }
};
```

Testing:

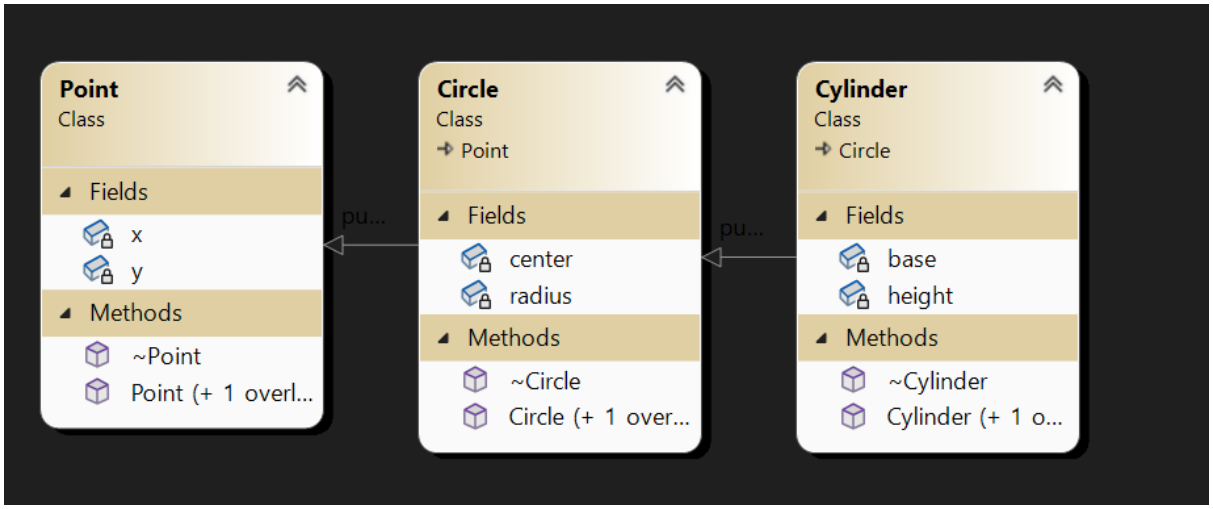
Main code:

```
int main() {
    //create a cylinder
    Cylinder c(1, 2, 3, 4);
    //print the cylinder
    c.print();
}
```

Output:

```
int main() {
    //create a cylinder
    Cylinder c(1, 2, 3, 4);
    //print the cylinder
    c.print();
}
```

UML:



Exercise 3

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

//class Point
class Point {
    protected:    //private data members
        int x;
        int y;
    public:    //public methods
        //constructor
        Point(int x, int y) {
            this->x = x;
            this->y = y;
        }
        //default constructor
        Point() {
            x = 0;
```

```
            y = 0;
        }
};

//class Line that inherits from Point
class Line: public Point {
    protected:    //private data members
        int length;
    public:    //public methods
        //constructor
        Line(int x, int y, int length): Point(x, y) {
            this->length = length;
        }
        //default constructor
        Line(): Point() {
            this->length = 1;
        }
};

//class square that inherits from Line
class Square: public Line {
    protected:    //private data members
        int width;
        int height;
        int baseArea;
    public:    //public methods
        //constructor
        Square(int x, int y, int length, int width, int height, int baseArea):
Line(x, y, length) {
            this->width = width;
            this->height = height;
            this->baseArea = baseArea;
        }
        //default constructor
        Square(): Line() {
            this->width = 1;
            this->height = 1;
            this->baseArea = 1;
        }
};

//class cube that inherits from Square
class Cube: public Square {
    protected:    //private data members
        int volume;
    public:    //public methods
        //constructor
```

```
        Cube(int x, int y, int length, int width, int height, int baseArea,
int volume): Square(x, y, length, width, height, baseArea) {
            this->volume = volume;
        }
        //default constructor
        Cube(): Square() {
            this->volume = 1;
        }
        //print the cube
        void print() {
            cout << "Point: " << x << ", " << y << endl;
            cout << "Length: " << length << endl;
            cout << "Width: " << width << endl;
            cout << "Height: " << height << endl;
            cout << "Base Area: " << baseArea << endl;
            cout << "Volume: " << volume << endl;
        }
};
```

Testing:

Main code:

```
int main() {
    //create a cube
    Cube c(1, 2, 3, 4, 5, 6, 7);
    //print the cube
    c.print();
    return 0;
}
```

Output:

```
Point: 1, 2
Length: 3
Width: 4
Height: 5
Base Area: 6
Volume: 7
PS C:\Users\Owner\Des
```

Exercise 4

Section 1:

Time.h file:

```
#pragma once
class Time
{
private:
    int hour;
    int minute;
```

```
    int second;
public:
    Time();
    void setHour(int h);
    void setMinute(int m);
    void setSecond(int s);
    void setTime(int s, int m, int h);
    ~Time();
};
```

Time.cpp file:

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

//constructor
Time::Time()
{
    hour = 0;
    minute = 0;
    second = 0;
}

//set the hour
void Time::setHour(int h)
{
    hour = h;
}

//set the minute
void Time::setMinute(int m)
{
    minute = m;
}

//set the second
void Time::setSecond(int s)
{
    second = s;
}

//set the time
void Time::setTime(int s, int m, int h)
{
    second = s;
    minute = m;
    hour = h;
}
```

```cpp
//destructor
Time::~Time()
{
    std::cout << "Time object is destroyed" << std::endl;
}
```

Date.h file:

```cpp
#pragma once
class Date {
private:
    int day;
    int month;
    int year;
public:
    Date();
    void setDay(int d);
    void setMonth(int m);
    void setYear(int y);
    void setDate(int d, int m, int y);
};
```

Date.cpp file:

```cpp
#include "Date.h"
#include <iostream>
using namespace std;

//constructor
Date::Date()
{
    day = 0;
    month = 0;
    year = 0;
}

//set the day
void Date::setDay(int d)
{
    day = d;
}
//set the month
void Date::setMonth(int m)
{
    month = m;
}
//set the year
void Date::setYear(int y)
```

```
{
    year = y;
}
//set the date
void Date::setDate(int d, int m, int y)
{
    day = d;
    month = m;
    year = y;
}

//destructor
Date::~Date()
{
    cout << "Date object is destroyed" << endl;
}
```

Testing:

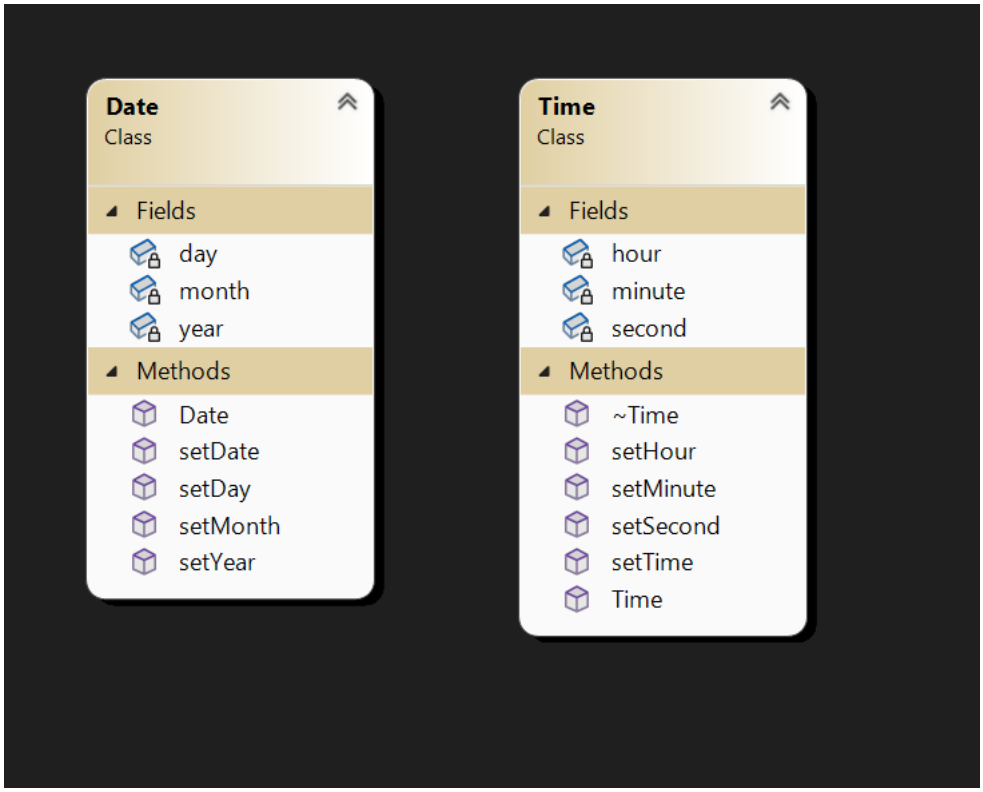
Main code:

```
#include <iostream>
#include "Time.h"
#include "Date.h"
using namespace std;


int main()
{
    //create a Date object
    Date date1;
    //set the date
    date1.setDate(12, 25, 2012);
    //create a Time object
    Time time1;
    //set the time
    time1.setTime(11, 59, 59);
    //display the date and time
}
```

Output:

I ran this code and no errors were thrown

UML:



Section 2:

Supervisor.h file:

```
#pragma once
#include <string>
class Supervisor
{
    private:
        std::string name;
    public:
        Supervisor();
        void setSupervisor(std::string name);
        ~Supervisor();
};
```

Supervisor.cpp file:

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

Supervisor::Supervisor()
{
    name = "";
}

void Supervisor::setSupervisor(string name)
{
```

```
    this->name = name;
}

Supervisor::~Supervisor()
{
    cout << "supervisor object destroyed";
}
```

Project.h file:

```
#pragma once
#include "Date.h"
#include "Supervisor.h"
#include <string>
class Project
{
private:
    Date StartDate;
    Supervisor supervisor;
    std::string pname;
public:
    Project();
    void setProject(string name, Date date, Supervisor supervisor);
    ~Project();
};
```

Project.cpp file:

```
#include "Project.h"
#include "Date.h"
#include "Supervisor.h"
#include <iostream>
using namespace std;

Project::Project()
{
    pname = " ";
    StartDate = Date();
    supervisor = Supervisor();
}

void Project::setProject(string name, Date date, Supervisor supervisor)
{
    pname = name;
    StartDate = date;
    supervisor = supervisor;
}

Project::~Project()
```

```
{
    cout << "project object destroyed";
}
```

Student.h file:

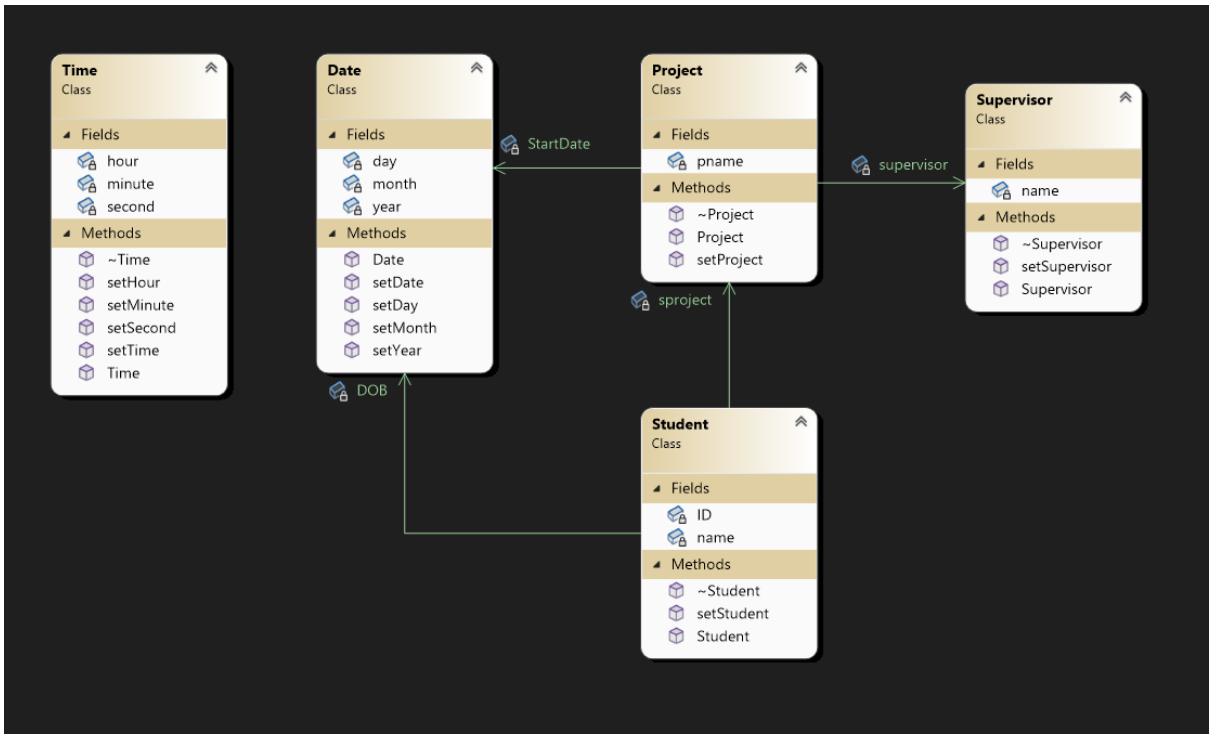
```
#pragma once
#include <string>
#include "Date.h"
#include "Project.h"
using namespace std;
class Student
{
    private:
        Date DOB;
        Project sproject;
        string ID;
        string name;
    public:
        Student();
        void setStudent(Date DOB, Project sproject, string ID, string name);
        ~Student();
};
```

Student.cpp file:

```
#include "Student.h"
#include "Date.h"
#include "Project.h"
#include <iostream>
using namespace std;

Student::Student()
{
    DOB = Date();
    sproject = Project();
    ID = "";
    name = "";
}

void Student::setStudent(Date DOB, Project sproject, string ID, string name)
{
    this->DOB = DOB;
    this->sproject = sproject;
    this->ID = ID;
    this->name = name;
}

Student::~Student()
{
    cout << "student object destroyed";
```

```
}
```

UML:



Section 3

Deliverable.h file:

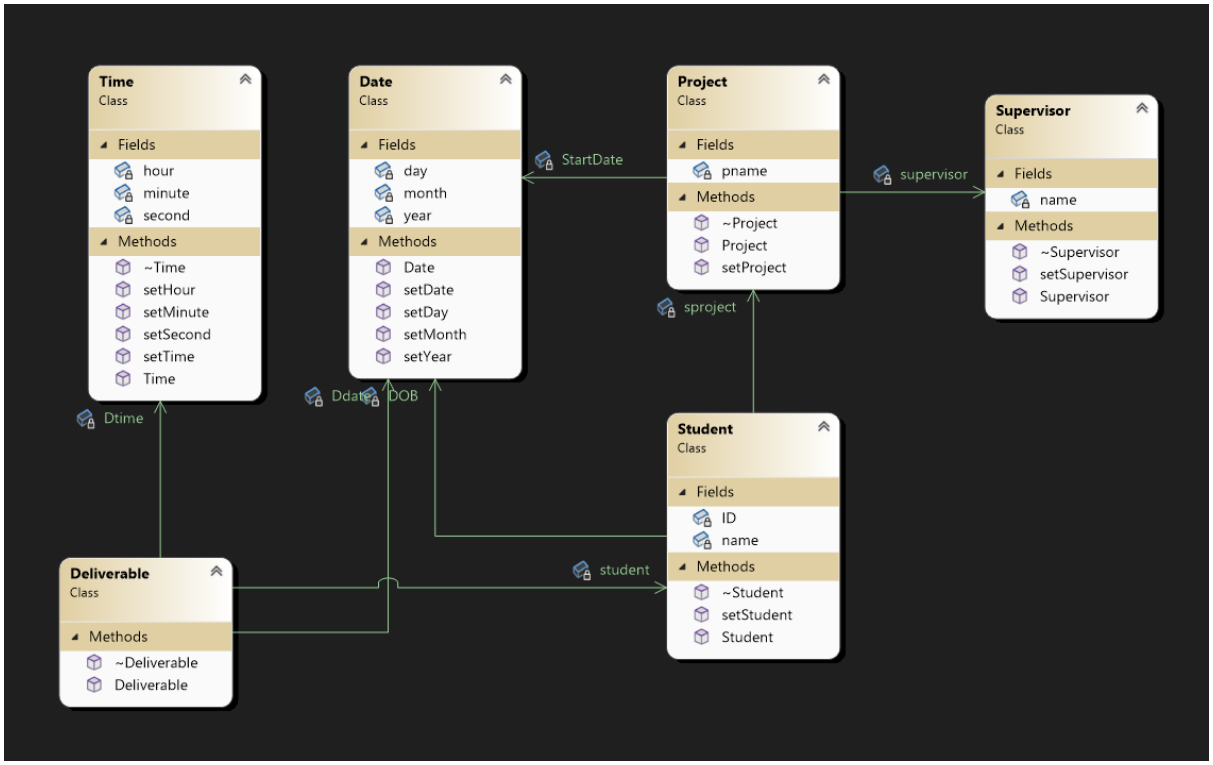
```
#pragma once
#include "Date.h"
#include "Time.h"
#include "Student.h"

class Deliverable
{
    private:
        Time Dtime;
        Date Ddate;
        Student student;
    public:
        Deliverable();
        ~Deliverable();
};
```

Deliverable.cpp file:

```
#include "Deliverable.h"
#include "Date.h"
#include "Time.h"
#include "Student.h"
```

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

Deliverable::Deliverable()
{
    Dtime = Time();
    Ddate = Date();
    student = Student();
}

Deliverable::~Deliverable()
{
    cout << "Deliverable object destroyed";
}
```

UML:



Testing:

Main code:

```cpp
#include <iostream>
#include "Time.h"
#include "Date.h"
#include "Student.h"
#include "Project.h"
#include "Supervisor.h"
using namespace std;
```

```
int main()
{
    //create a student
    Student stu1();
}
```
Output:

The code ran without crashing.

Week5

Exercise 1

Code:

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

class DataStructure {
    private:
        int size;
    public:
        DataStructure(int size) {
            this->size = size;
        }
        virtual void initialize();
        virtual bool isFull();
        virtual bool isEmpty();
        virtual void insert(int value);
        virtual void remove(int value);
        virtual void push(int value);
        virtual void pop();
        virtual void ShowElements(int index);
        virtual void ShowAllElements();
        virtual void clear();
        virtual void deleteStructure();
        virtual void sort();
        virtual void print();
};
```
The use of the virtual function is that it would be inheritable by another class but then can be over-ridden in the case of something being different. This would be useful as there are many different types of data structures with lots of different methods that are similar but not the same.

Exercise 2

Code:

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

```
//Write down all the shapes you can think of--both two-dimensional and three-dimensional--and form those shapes into a shape hierarchy. Your hierarchy should have base class Shape from which class TwoDimensionalShape and class ThreeDimensionalShape are derived. Once you have developed the hierarchy, define each of the classes in the hierarchy. We will use this hierarchy in the coming exercises to process all shapes as objects of base-class Shape.

class Shape{
    protected:  //protected so that derived classes can access
        string name;
    public: //public methods
        //constructor
        Shape(string name){
            this->name = name;
        }
        //virtual print method
        virtual void print(){
            cout << "Shape: " << name << endl;
        }
};


class TwoDimensionalShape: public Shape{
    private:    //private variables
        double area;
    public: //public methods
        //constructor
        TwoDimensionalShape(string name): Shape(name){
            area = 0;
        }
        //virtual print method
        virtual void print(){
            cout << "Two Dimensional Shape: " << name << endl;
        }
        //virtual getArea method
        virtual double getArea(){
            return area;
        }
        //virtual setArea method
        virtual void setArea(double area){
            this->area = area;
        }
};

class ThreeDimensionalShape: public Shape{
    private:    //private variables
        double volume;
    public: //public methods
```

```
        //constructor
        ThreeDimensionalShape(string name): Shape(name){
            volume = 0;
        }
        //virtual print method
        virtual void print(){
            cout << "Three Dimensional Shape: " << name << endl;
        }
        //virtual getVolume method
        virtual double getVolume(){
            return volume;
        }
        //virtual setVolume method
        virtual void setVolume(double volume){
            this->volume = volume;
        }
};
```

Testing:

Main code:

```
int main() {
    //list of shapes
    TwoDimensionalShape *shapes[3];
    //create shapes
    shapes[0] = new TwoDimensionalShape("Square");
    shapes[1] = new TwoDimensionalShape("Triangle");
    shapes[2] = new TwoDimensionalShape("Circle");
    //print shapes
    for(int i = 0; i < 3; i++){
        shapes[i]->print();
    }
    //list of 3D shapes
    ThreeDimensionalShape *shapes3D[3];
    //create 3D shapes
    shapes3D[0] = new ThreeDimensionalShape("Cube");
    shapes3D[1] = new ThreeDimensionalShape("Sphere");
    shapes3D[2] = new ThreeDimensionalShape("Cylinder");
    //print 3D shapes
    for(int i = 0; i < 3; i++){
        shapes3D[i]->print();
    }
}
```

Output:

```
Two Dimensional Shape: Square
Two Dimensional Shape: Triangle
Two Dimensional Shape: Circle
Three Dimensional Shape: Cube
Three Dimensional Shape: Sphere
Three Dimensional Shape: Cylinder
```

Week6

Exercise 1

As I already used operator overloading for rational number arithmetic in week 3, please refer to week 3 exercise 1 for my code to this solution.

Exercise 2

1 code:

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

template <typename T>
// Function to swap two values
void Swap(T& a, T& b)
{
    T temp = a;
    a = b;
    b = temp;
}
```

Testing:

Main:

```cpp
int main(){
    // Declare variables
    int a = 1;
    int b = 2;
    cout << "Before swap: " << a << " " << b << endl;
    Swap(a, b); // Swap the values
    cout << "After swap: " << a << " " << b << endl;
}
```

Output:

```
Before swap: 1 2
After swap: 2 1
```

2 code:

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

template <typename T>
```

```
class Array{
    private:    //private members
        T *arr;
        int size;
        int capacity;
    public: //public methods
        //constructor
        Array(int capacity){
            this->capacity = capacity;
            arr = new T[capacity];
            size = 0;
        }
        //add new element
        void add(T element){
            if(size == capacity){   //if the array is full
                cout << "Array is full" << endl;
                return;
            }
            arr[size] = element;    //add the element
            size++; //increase the size
        }
        void remove(int index){ //remove an element at a certain index
            if(index < 0 || index >= size){ //if the index is out of bounds
                cout << "Index out of range" << endl;
                return;
            }
            for(int i = index; i < size - 1; i++){  //shift the elements to
the left
                arr[i] = arr[i + 1];
            }
            size--;
        }
        void show(){    //show the array elements
            for(int i = 0; i < size; i++){  //loop through the array
                cout << arr[i] << " ";  //print the element
            }
            cout << endl;
        }
};
```

Testing:

Main code:

```
    Array<int> arr(5);  //create an array of integers
    // add elements
    arr.add(1);
    arr.add(2);
    arr.add(3);
    arr.add(4);
```

```
    arr.add(5);
    arr.show(); //show the array
    arr.remove(2);  //remove the element at index 2
    arr.show(); //show the array
}
```

Output:

```
1 2 3 4 5
1 2 4 5
PS C:\Users\Ow
```

Exercise 3

Code:

1 code:

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

class BugsRobot{
    protected: // protected data members
        int speed;
        int weight;
        int size;
        string name;
    public: // public member functions
        BugsRobot(int s, int w, int si, string n){
            speed = s;
            weight = w;
            size = si;
            name = n;
        }
        virtual void move() = 0; // pure virtual function
        virtual void eat() = 0; // pure virtual function
        virtual void sleep() = 0; // pure virtual function
        virtual void display() = 0; // pure virtual function

};

class AntRobot: public BugsRobot{   // derived class
    public: // public member functions
        AntRobot(int s, int w, int si, string n): BugsRobot(s, w, si, n){}
        void move(){
            cout << "Ant Robot " << name << " is moving" << endl;
        }
        void eat(){
            cout << "Ant Robot " << name << " is eating" << endl;
```

```
        }
        void sleep(){
            cout << "Ant Robot " << name << " is sleeping" << endl;
        }
        void display(){
            cout << "Ant Robot " << name << " is displaying" << endl;
        }
};

class BeeRobot: public BugsRobot{   // derived class
    public: // public member functions
        BeeRobot(int s, int w, int si, string n): BugsRobot(s, w, si, n){}
        void move(){
            cout << "Bee Robot " << name << " is moving" << endl;
        }
        void eat(){
            cout << "Bee Robot " << name << " is eating" << endl;
        }
        void sleep(){
            cout << "Bee Robot " << name << " is sleeping" << endl;
        }
        void display(){
            cout << "Bee Robot " << name << " is displaying" << endl;
        }
};

class ButterflyRobot: public BugsRobot{   // derived class
    public: // public member functions
        ButterflyRobot(int s, int w, int si, string n): BugsRobot(s, w, si,
n){}
        void move(){
            cout << "Butterfly Robot " << name << " is moving" << endl;
        }
        void eat(){
            cout << "Butterfly Robot " << name << " is eating" << endl;
        }
        void sleep(){
            cout << "Butterfly Robot " << name << " is sleeping" << endl;
        }
        void display(){
            cout << "Butterfly Robot " << name << " is displaying" << endl;
        }
};
```

3 code and testing:

```
//template function to check size of array
template <class T>
```

```
int arraySize(T &array){
    return sizeof(array)/sizeof(array[0]);
}

int main(){
    //create an array of AntRobot objects
    AntRobot ant[3] = {AntRobot(10, 20, 30, "Ant1"), AntRobot(40, 50, 60,
"Ant2"), AntRobot(70, 80, 90, "Ant3")};
    //create an array of BeeRobot objects
    BeeRobot bee[3] = {BeeRobot(10, 20, 30, "Bee1"), BeeRobot(40, 50, 60,
"Bee2"), BeeRobot(70, 80, 90, "Bee3")};
    //create an array of ButterflyRobot objects
    ButterflyRobot butterfly[3] = {ButterflyRobot(10, 20, 30, "Butterfly1"),
ButterflyRobot(40, 50, 60, "Butterfly2"), ButterflyRobot(70, 80, 90,
"Butterfly3")};
}
```

Output:

The program ran without issue.

Week7

Exercise 1

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

//Write a function template palindrome that takes as a parameter a const
vector and returns true or false depending upon whether the vector does or
does not read the same forwards as backwards
template <typename T>   //template function
bool palindrome(const vector<T> &v)
{
    for (int i = 0; i < v.size(); i++)  //loop through vector
    {
        if (v[i] != v[v.size() - 1 - i])    //if the first element isnt equal
to the last element
        {
            return false;
        }
    }
    return true;
}
```

Testing:

Main code:

```
int main(){
    //Test the function template with the following vectors
```

```
    vector<int> v1 = {1, 2, 3, 4, 5, 4, 3, 2, 1};
    vector<int> v2 = {1, 2, 3, 4, 5, 6, 7, 8, 9};
    //Test if the vectors are palindromes
    cout << palindrome(v1) << endl;
    cout << palindrome(v2) << endl;
}
```

Output:

```
1
0
PS C:\User
```

Exercise 2

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

//student class that has four lists, one for first name, one for last name,
one for year1 marks, and one for year2 marks
class Student
{
    private:
        list<char> firstName;
        list<char> lastName;
        list<int> year1;
        list<int> year2;
    public:
        //setters
        void setFirstName(string s)
        {
            for (int i = 0; i < s.length(); i++)
            {
                firstName.push_back(s[i]);
            }
        }
        void setLastName(string s)
        {
            for (int i = 0; i < s.length(); i++)
            {
                lastName.push_back(s[i]);
            }
        }
        void setYear1(int n)
        {
            year1.push_back(n);
        }
        void setYear2(int n)
        {
            year2.push_back(n);
```

```
        }
        //getters
        list<char> getFirstName()
        {
            return firstName;
        }
        list<char> getLastName()
        {
            return lastName;
        }
        list<int> getYear1()
        {
            return year1;
        }
        list<int> getYear2()
        {
            return year2;
        }
        //space filter function that removes spaces from a list
        list<char> spaceFilter(list<char> l)
        {
            list<char> result;  //create a new list
            for (list<char>::iterator i = l.begin(); i != l.end(); i++) //loop
through the list
            {
                if (*i != ' ')  //if the element isnt a space
                {
                    result.push_back(*i);   //add it to the new list
                }
            }
            return result;
        }
        //function that adds both names together and returns a list of
characters
        list<char> spliceNames()
        {
            list<char> result;  //create a new list
            list<char> first = spaceFilter(getFirstName()); //filter the first
name
            list<char> last = spaceFilter(getLastName());   //filter the last
name
            for (list<char>::iterator i = first.begin(); i != first.end();
i++) //loop through the first name
            {
                result.push_back(*i);
            }
            for (list<char>::iterator i = last.begin(); i != last.end();
i++)   //loop through the last name
```

```
            {
                result.push_back(*i);
            }
            return result;
        }
        //function that sorts the marks in ascending order
        list<int> sortMarks(list<int> l)
        {
            list<int> result;
            for (list<int>::iterator i = l.begin(); i != l.end(); i++)  //loop
through the list
            {
                result.push_back(*i);   //add the element to the new list
            }
            result.sort();
            return result;  //return the sorted list
        }
        //function that merges the two marks lists together and returns a list
of integers
        list<int> finalMarksList()
        {
            list<int> result;
            list<int> year1 = sortMarks(getYear1());
            list<int> year2 = sortMarks(getYear2());
            for (list<int>::iterator i = year1.begin(); i != year1.end(); i++)
            {
                result.push_back(*i);
            }
            for (list<int>::iterator i = year2.begin(); i != year2.end(); i++)
            {
                result.push_back(*i);
            }
            return result;
        }
        //function that finds the average from the final marks list
        double average()
        {
            list<int> final = finalMarksList();
            int sum = 0;
            for (list<int>::iterator i = final.begin(); i != final.end(); i++)
            {
                sum += *i;
            }
            return (double)sum / final.size();
        }
        //function that prints the student's name and average
        void showFinal()
        {
```

```
            list<char> name = spliceNames();
            cout << "Name: ";
            for (list<char>::iterator i = name.begin(); i != name.end(); i++)
            {
                cout << *i;
            }
            cout << endl;
            cout << "Average: " << average() << endl;
        }
};
```

Testing:

Main code:

```
int main()
{
    bool loop = true;   //loop variable
    while(loop == true){
        //main menu
        cout << "1. Enter student data" << endl;
        cout << "2. Show student data" << endl;
        cout << "3. Sort marks" << endl;
        cout << "4. Merge marks and print average" << endl;
        cout << "5. Exit" << endl;
        //get user input
        int choice;
        cout << "Enter your choice: ";
        cin >> choice;
        if(choice == 1){
            //create a new student object
            Student s;
            //get user input for first name
            string firstName;
            cout << "Enter first name: ";
            cin >> firstName;
            s.setFirstName(firstName);
            //get user input for last name
            string lastName;
            cout << "Enter last name: ";
            cin >> lastName;
            s.setLastName(lastName);
            //get user input for year1 marks
            int year1;
            cout << "Enter year 1 marks: ";
            cin >> year1;
            s.setYear1(year1);
            //get user input for year2 marks
            int year2;
            cout << "Enter year 2 marks: ";
```

```
            cin >> year2;
            s.setYear2(year2);
            //print the student's name and average
            s.showFinal();
        }
        else if(choice == 2){
            //create a new student object
            Student s;
            //get user input for first name
            string firstName;
            cout << "Enter first name: ";
            cin >> firstName;
            s.setFirstName(firstName);
            //get user input for last name
            string lastName;
            cout << "Enter last name: ";
            cin >> lastName;
            s.setLastName(lastName);
            //get user input for year1 marks
            int year1;
            cout << "Enter year 1 marks: ";
            cin >> year1;
            s.setYear1(year1);
            //get user input for year2 marks
            int year2;
            cout << "Enter year 2 marks: ";
            cin >> year2;
            s.setYear2(year2);
            //print the student's name and average
            s.showFinal();
        }
        else if(choice == 3){
            //create a new student object
            Student s;
            //get user input for year1 marks
            int year1;
            cout << "Enter year 1 marks: ";
            cin >> year1;
            s.setYear1(year1);
            //get user input for year2 marks
            int year2;
            cout << "Enter year 2 marks: ";
            cin >> year2;
            s.setYear2(year2);
            //print the sorted marks
            list<int> sorted = s.sortMarks(s.finalMarksList());
            for (list<int>::iterator i = sorted.begin(); i != sorted.end();
i++)
```

```
            {
                cout << *i << " ";
            }
            cout << endl;
        }
        else if(choice == 4){
            //create a new student object
            Student s;
            //get user input for first name
            string firstName;
            cout << "Enter first name: ";
            cin >> firstName;
            s.setFirstName(firstName);
            //get user input for last name
            string lastName;
            cout << "Enter last name: ";
            cin >> lastName;
            s.setLastName(lastName);
            //get user input for year1 marks
            int year1;
            cout << "Enter year 1 marks: ";
            cin >> year1;
            s.setYear1(year1);
            //get user input for year2 marks
            int year2;
            cout << "Enter year 2 marks: ";
            cin >> year2;
            s.setYear2(year2);
            //print the student's name and average
            s.showFinal();
        }
        else if(choice == 5){
            loop = false;
        }
        else{
            cout << "Invalid choice" << endl;
        }
    }
}
```

Enter student Output:

```
 1. Enter student data
 2. Show student data
 3. Sort marks
 4. Merge marks and print average
 5. Exit
 Enter your choice: 1
 Enter first name: Giles
 Enter last name: Turnbull
 Enter year 1 marks: 4 5 7 3
 Enter year 2 marks: Name: GilesTurnbull
 Average: 4.5
```

Show student data output:

```
1. Enter student data
2. Show student data
3. Sort marks
4. Merge marks and print average
5. Exit
Enter your choice: 2
Enter first name: Giles
Enter last name: Turnbull
Enter year 1 marks: 4
Enter year 2 marks: 7
Name: GilesTurnbull
Average: 5.5
```

Sort marks output:

```
 2. Show student data
 3. Sort marks
 4. Merge marks and print average
 5. Exit
 Enter your choice: 3
 Enter year 1 marks: 4
 Enter year 2 marks: 6
 4 6
```

Week8

Exercise 1

1 and 2:

```cpp
#include <iostream>
#include <set>
#include <map>
using namespace std;
//implement a multi set to hold car prices
//implement a set to hold more car prices

int main(){
    std::multiset<double> carPrices;
    std::set<double> carPrices2;
```

```
    carPrices.insert(12342.00);
    carPrices.insert(223942.00);
    carPrices.insert(234234.00);
    carPrices.insert(9856985.00);
    carPrices.insert(12342.00);
    //print out the prices
    for(auto it = carPrices.begin(); it != carPrices.end(); ++it){  //print
out the prices
        cout << *it << endl;
    }
    cout << "------------------" << endl;
    carPrices2.insert(12342.00);
    carPrices2.insert(223942.00);
    carPrices2.insert(234234.00);
    carPrices2.insert(9856985.00);
    carPrices2.insert(12342.00);
    //print out the prices
    for(auto it = carPrices2.begin(); it != carPrices2.end(); ++it){  //print
out the prices
        cout << *it << endl;
    }

}
```

Output:

```
12342
12342
223942
234234
9.85698e+06
------------------
12342
223942
234234
9.85698e+06
```

This shows that the “set” library removes the duplicate value whereas multiset does not.

3 and 4:

Next I added this to the code:

```
    std::map<std::string, double> Cars;
    Cars["Ford"] = 39845.00;
    Cars["Toyota"] = 56907.00;
    Cars["Honda"] = 34095.00;
    Cars["Chevy"] = 23423.00;
    Cars["BMW"] = 234234.00;
    Cars["Ford"] = 39845.00;
    //print out the map
```

```
    for(auto it = Cars.begin(); it != Cars.end(); ++it){
        cout << it->first << " " << it->second << endl;
    }
    cout << "------------------" << endl;
    std::multimap<std::string, double> Cars2;
    Cars2.insert(std::pair<std::string, double>("Ford", 39845.00));
    Cars2.insert(std::pair<std::string, double>("Toyota", 56907.00));
    Cars2.insert(std::pair<std::string, double>("Honda", 34095.00));
    Cars2.insert(std::pair<std::string, double>("Chevy", 23423.00));
    Cars2.insert(std::pair<std::string, double>("BMW", 234234.00));
    Cars2.insert(std::pair<std::string, double>("Ford", 39845.00));
    //print out the map
    for(auto it = Cars2.begin(); it != Cars2.end(); ++it){
        cout << it->first << " " << it->second << endl;
    }
```

Output:

```
BMW 234234
Chevy 23423
Ford 39845
Honda 34095
Toyota 56907
------------------
BMW 234234
Chevy 23423
Ford 39845
Ford 39845
Honda 34095
Toyota 56907
```

The same can be seen above where map removes one of the duplicate values but multimap does not.

Exercise 2

1 Code:

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

//Implement a class for each of the following, use a deque structure to
represent:
//1.    MyStack
//2.    MyQueue

class MyStack{
    private:    //private data members
        deque<int> stack;   //stack using deque
    public:
        void push(int x){   //push function
            stack.push_back(x);
```

```
        }
        void pop(){ //pop function
            stack.pop_back();
        }
        int top(){  //top function
            return stack.back();
        }
        bool empty(){   //empty function
            return stack.empty();
        }
        //print function
        void print(){
            for(auto it = stack.begin(); it != stack.end(); ++it){
                cout << *it << endl;
            }
        }
};
```

2 code:

```
//Queue class
class MyQueue{
    private:    //private data members
        deque<int> queue;   //queue using deque
    public:
        void push(int x){   //push function
            queue.push_back(x);
        }
        void pop(){ //pop function
            queue.pop_front();
        }
        int front(){  //front function
            return queue.front();
        }
        bool empty(){   //empty function
            return queue.empty();
        }
        //print function
        void print(){
            for(auto it = queue.begin(); it != queue.end(); ++it){
                cout << *it << endl;
            }
        }
};
```

Testing:

Main code:

```cpp
int main(){
    //take integers from the user and create a stack and a queue
    int x, y, z;
    cout << "Enter three integers: ";
    cin >> x >> y >> z;
    MyStack stack;
    MyQueue queue;
    stack.push(x);
    stack.push(y);
    stack.push(z);
    queue.push(x);
    queue.push(y);
    queue.push(z);
    cout << "Stack: " << endl;
    stack.print();
    cout << "Queue: " << endl;
    queue.print();
    //pop the stack and queue
    stack.pop();
    queue.pop();
    cout << "an element has been popped from the stack and queue" << endl;
    cout << "Stack: " << endl;
    stack.print();
    cout << "Queue: " << endl;
    queue.print();
    //push another element to the stack and queue
    stack.push(25);
    queue.push(28);
    cout << "an element has been pushed to the stack and queue" << endl;
    cout << "Stack: " << endl;
    stack.print();
    cout << "Queue: " << endl;
    queue.print();
}
```

Output:

```
Enter three integers: 4 7 3
Stack:
4
7
3
Queue:
4
7
3
an element has been popped from the stack and queue
Stack:
4
7
Queue:
7
3
an element has been pushed to the stack and queue
Stack:
4
7
25
Queue:
7
3
28
```