SHEET 5 SOLUTIONS

SHEHAB MAHMOUD SALAH | 2100320

GENERAL NOTE: to **copy** code **from** this PDF document, **copy each block of code** separately to **not** lose the code's formatting, alternatively you can find all the **source code** to **PROGRAMMING EXERCISES** on my **GitHub**: https://bit.ly/CSE131Sheets **happy compiling** 😁

1. (Exercise(1) on GitHub)

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

using namespace std;

class Country {
public:
    string name;
    long long population;
    double area; // in square kilometers

    Country(const string &name, long long population, double area)
        : name(name), population(population), area(area) {}

    double populationDensity() {
        return population / area;
    }
};

int main() {
    vector<Country> countries;
    for(int i=0; i<4; i++){
        string name;
        long long population;
        double area;

        cout << "Enter details for country " << i+1 << "\n";
        cout << "Name: ";
        cin.ignore(); // this will ignore the '\n' from previous input
        getline(cin, name);
        cout << "Population: ";
        cin >> population;
        cout << "Area: ";
        cin >> area;

        countries.push_back(Country(name, population, area));
    }

    Country maxAreaCountry = countries[0];
    Country maxPopulationCountry = countries[0];
    Country maxDensityCountry = countries[0];

    for(int i=1; i<4; i++){
        if(countries[i].area > maxAreaCountry.area)
            maxAreaCountry = countries[i];
```

```
        if(countries[i].population > maxPopulationCountry.population)
            maxPopulationCountry = countries[i];

        if(countries[i].populationDensity() >
maxDensityCountry.populationDensity())
            maxDensityCountry = countries[i];
    }

    cout << "Country with largest area: " << maxAreaCountry.name <<
"\n";
    cout << "Country with largest population: " <<
maxPopulationCountry.name << "\n";
    cout << "Country with largest population density: " <<
maxDensityCountry.name << "\n";

    return 0;
}
```

A few bullet points on the previous code:

- The variable **`population`** is assigned to the datatype **`long long`** to give a wider range of integer numerals which is proper for country populations.

- In the first main **`for`** loop, we used **`cin`** for numeric inputs but we used **`getline`** for the country name, the reason behind that is that if we used **`cin`** for the country names, countries with spaces won't be displayed properly (Ex.: **United States** will show up as **United** only), so, using **`getline`** and **`cin.ignore()`** to clear the new line, will read the country name correctly.

- The **`i+1`** (line 28) is to display the countries numbered starting from **1** instead of the computer's default starting index of **0**, so it makes the interface more user-friendly.

- It's worth noting that **`using namespace std;`** is generally bad practice in larger C++ programs that contain functions because it could lead to name clashes, *For example*, let's say you have a function named **`find()`** in your code, and you're also using a library that has a function named **`find()`**. If you use **`using namespace std;`** and the library is in the **`std`** namespace, your **`find()`** function will clash with **`std::find()`**. This can lead to confusing and hard-to-debug issues in your code.

  This is why it's generally recommended to avoid **`using namespace std;`** in global scope, and instead use the **`std::`** prefix when you want to refer to something in the std namespace, like **`std::cout`** or **`std::cin`**.

2. Here's a code that performs the required task:

(Exercise(2) on GitHub)

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

using namespace std;

class Person {
public:
    string name;
    int age;
    string gender;

    // Constructor
    Person(const string &name, int age, const string &gender)
        : name(name), age(age), gender(gender) {}

    // Member function to print the data
    void printData() {
        cout << "Name: " << name << endl;
        cout << "Age: " << age << endl;
        cout << "Gender: " << gender << endl;
    }
};

int main() {
    Person john("Shehab Mahmoud", 20, "Male");
    john.printData();
    return 0;
}
```

Additionally, here's a code that is input-based and returns the user's entries:

(Exercise(2) BONUS on GitHub)

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

using namespace std;

class Person {
public:
    string name;
    int age;
    string gender;

    // Constructor
    Person(const string &name, int age, const string &gender)
        : name(name), age(age), gender(gender) {}

    // Member function to print the data
    void printData() {
        cout << "Name: " << name << endl;
        cout << "Age: " << age << endl;
        cout << "Gender: " << gender << endl;
    }
};
```

```
int main() {
    string name;
    int age;
    string gender;

    cout << "Enter name: ";
    getline(cin, name);
    cout << "Enter age: ";
    cin >> age;
    cin.ignore();  // remove newline from input buffer
    cout << "Enter gender: ";
    getline(cin, gender);

    Person person(name, age, gender);
    person.printData();

    return 0;
}
```

3.

```
#include <iostream>
```

(**Exercise(3)** on GitHub)

```
using namespace std;

class TollBooth {
private:
    int totalCars;
    double totalCash;

public:
    // Constructor initializes both totalCars and totalCash to 0
    TollBooth() : totalCars(0), totalCash(0) {}

    // payingCar() increments the car total and adds 5 to the cash total

    void payingCar() {
        totalCars++;
        totalCash += 5;
    }

    // nopayCar() increments the car total but adds nothing to the cash
total

    void nopayCar() {
        totalCars++;
    }

    // display() displays the two totals

    void display() {
        cout << "Total cars: " << totalCars << endl;
        cout << "Total cash: " << totalCash << " L.E" << endl;
    }
};
```

```
int main() {
    TollBooth booth;
    booth.payingCar();
    booth.nopayCar();
    booth.payingCar();
    booth.display();
    return 0;
}
```

Here's a breakdown for the code's operation and its output:

1. `TollBooth booth;`: A **`TollBooth`** object named **`booth`** is created. Its **`totalCars`** and **`totalCash`** attributes are initialized to **`0`** by the constructor.

2. `booth.payingCar();`: The **`payingCar()`** method is called on **`booth`**. This increments **`totalCars`** to **`1`** (since a car has passed) and adds **`5`** to **`totalCash`** (since the car paid the toll).

3. `booth.nopayCar();`: The **`nopayCar()`** method is called on **`booth`**. This increments **`totalCars`** to **`2`** (since another car has passed) but does not change **`totalCash`** (since this car did not pay the toll).

4. `booth.payingCar();`: The **`payingCar()`** method is called on **`booth`** again. This increments **`totalCars`** to **`3`** and adds another **`5`** to **`totalCash`**, making it **`10`**.

5. `booth.display();`: The **`display()`** method is called on **`booth`**. This prints out the values of **`totalCars`** and **`totalCash`**, which are **`3`** and **`10`** respectively.

   Hence, the output: "Total cars: 3 Total cash: 10 L.E".

4. The following code's function is to create time values, and it can add two time values and get a new time value, *here are a few bullet points on how the code works:*

- The code defines a class called **`Time`** that has three private data members: **`hours`**, **`minutes`**, and **`seconds`**.
- The class has two constructors: one that initializes the data to **`0`**, and another that initializes the data to fixed values given as parameters.
- The class has a display function that prints the time in 11:59:59 format. It uses a helper function called **`addZero`** that adds a leading zero if the number is less than 10.
- The class has an add function that takes two objects of type **`Time`** as arguments and returns another object of type **`Time`** that is the **sum** of the two arguments. It handles the overflow of seconds and minutes by using a carry variable and adjusting the hours accordingly.
- The main function creates two initialized time objects and one uninitialized time object. It then calls the add function on the two initialized objects and stores the result in the third object. It then calls the display function on the third object to print the result.

- In the following code, I initialized the time objects where:
  `t1 = 10:15:30` and **`t2 = 1:45:50`** , so adding those two time objects (in HH:MM:SS format) would result in 12:01:20 (12 hrs, 1 min, 20 seconds)…
  which is the output of the program.

(**Exercise(4)** on GitHub)

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

// A class to represent time
class Time {
  // Private data members
  private:
    int hours;
    int minutes;
    int seconds;

  // Public member functions
  public:
    // Default constructor to initialize data to 0
    Time() {
      hours = 0;
      minutes = 0;
      seconds = 0;
    }

    // Parameterized constructor to initialize data to fixed values
    Time(int h, int m, int s) {
      hours = h;
      minutes = m;
      seconds = s;
    }

    // A helper function to add a leading zero if the number is less than 10
    void addZero(int n) {
        if (n < 10) {
          cout << "0";
        }
    }

    // A function to display time in 11:59:59 format
    void display() {
      // Display the hours with a leading zero if needed
      addZero(hours);
      cout << hours << ":";

      // Display the minutes with a leading zero if needed
      addZero(minutes);
      cout << minutes << ":";

      // Display the seconds with a leading zero if needed
      addZero(seconds);
      cout << seconds << endl;
    }

    // A function to add two objects of type time and return the result in
another object of type time
```

```
    Time add(Time t1, Time t2) {
      Time t3; // The result object
      int carry = 0; // To handle overflow of seconds and minutes

      // Add the seconds of t1 and t2 and store in t3
      t3.seconds = t1.seconds + t2.seconds;
      // If the sum is more than 59, set carry to 1 and reduce t3.seconds by 60
      if (t3.seconds > 59) {
        carry = 1;
        t3.seconds -= 60;
      }

      // Add the minutes of t1 and t2 along with carry and store in t3
      t3.minutes = t1.minutes + t2.minutes + carry;
      // Reset carry to 0
      carry = 0;
      // If the sum is more than 59, set carry to 1 and reduce t3.minutes by 60
      if (t3.minutes > 59) {
        carry = 1;
        t3.minutes -= 60;
      }

      // Add the hours of t1 and t2 along with carry and store in t3
      t3.hours = t1.hours + t2.hours + carry;
      // If the sum is more than 23, set t3.hours to 0 (assuming 24-hour format)
      if (t3.hours > 23) {
        t3.hours = 0;
      }

      // Return the result object
      return t3;
    }

};

int main() {
  // Create two initialized time objects
  Time t1(10, 15, 30);
  Time t2(1, 45, 50);

  // Create an uninitialized time object
  Time t3;

  // Add the two initialized objects and store the result in the third object
  t3 = t3.add(t1, t2);

  // Display the value of the third object
  cout << "The result is: ";
  t3.display();

  return 0;
}
```

5.

(Exercise(5) on GitHub)

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

// A class to represent 2d circles
class Circle {
  // Private data members
  private:
    float x; // The x-coordinate of the center
    float y; // The y-coordinate of the center
    float radius; // The radius of the circle

  // Public member functions
  public:
    // Default constructor to initialize data to 0
    Circle() {
      x = 0;
      y = 0;
      radius = 0;
    }

    // Parameterized constructor to initialize data to a fixed value
    Circle(float r) {
      x = 0;
      y = 0;
      radius = r;
    }

    // Parameterized constructor to initialize data to fixed values
    Circle(float x1, float y1) {
      x = x1;
      y = y1;
      radius = 0;
    }

    // Parameterized constructor to initialize data to fixed values
    Circle(float x1, float y1, float r) {
      x = x1;
      y = y1;
      radius = r;
    }

    // A function to print the circle data in the form "circle at (x,y) radius is
radius"
    void print() {
      cout << "circle at (" << x << "," << y << ") radius is " << radius << endl;
    }

    // A function to compute the distance between two circles
    float distance(Circle c1, Circle c2) {
      // Calculate the difference of x and y coordinates of the two circles
      float dx = c1.x - c2.x;
      float dy = c1.y - c2.y;

      // Calculate and return the distance using the formula: sqrt(dx^2 + dy^2)
      return sqrt(dx * dx + dy * dy);
```

```
    }
};

int main() {
  // Define two circles with values (1,2,1) and (3,5,4)
  Circle c1(1, 2, 1);
  Circle c2(3, 5, 4);

  // Print the data of the two circles
  cout << "The first circle is: ";
  c1.print();
  cout << "The second circle is: ";
  c2.print();

  // Find and print the distance between the two circles
  cout << "The distance between the two circles is: ";
  cout << c1.distance(c1, c2) << endl;

  return 0;
}
```

The output of this program would be:

```
The first circle is: circle at (1,2) radius is 1
The second circle is: circle at (3,5) radius is 4
The distance between the two circles is: 3.60555
```

Which is mathematically correct as well;

$$\sqrt{(3-1)^2+(5-2)^2} = \sqrt{4+9} = \sqrt{13} \approx 3.61$$

This concludes Sheet (5) Solutions, this document + source code to all programming exercises available on https://bit.ly/CSE131Sheets **.**

This also concludes all Sheet solutions; I hope they helped you with your studies 😊
If you have any questions or simply want to reach out, feel free to do so!
Visit my GitHub Site for contact info and more:
https://dizzydroid.github.io/