# Classes and Objects – Homework Exercises

Write C++ code for solving the tasks on the following pages.

Code should compile under the C++03 or the C++11 standard.

Please submit a single.cpp file for each task.

.cpp files for the tasks should be named with the task number followed by what you feel describes the exercise in a few words.

E.g. a good name for task 2 of this homework would be:  
2.distance.cpp

Don’t worry about the name too much, just make sure the number and the file extension are correct.

Tasks 4 to 6 in this homework require you to be creative about the implementation. They simulate real-world examples of non-strict and sometimes vague descriptions of client requirements. Part of the exercise is to learn to convert paragraphs of text into classes which solve the needs described in the paragraphs, and writing code which is easily modified if, for example, the format of the input/output data changes slightly. Also note that some of these exercises mention e.g. “array” or “string” and so on – do not take these literally, you can use any data structures you find appropriate.

# Task 1 – Randomize

You are given a **list of words in one line**. **Randomize their order** and print each word at a separate line.

Also, to implement this task you should write a class that is initialized with a list (or array or vector) of words and which has a getRandomized() method which returns the words randomized. Each call to getRandomized() should return a different ordering of the words.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| Welcome to SoftUni and have fun learning programming | learning  Welcome  SoftUni  and  fun  programming  have  to | The order of the words in the output will be different after each program execution. |

# Task 2 – Distance

Write a program to calculate the (Euclidean) distance between two points **p1** {**x1**, **y1**} and **p2** {**x2**, **y2**}. You should write a class to represent such points and a method in it which calculates the distance from the point to another point.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3 4  6 8 | 5.000 |
| 3 4  5 4 | 2.000 |
| 8 -2  -1 5 | 11.402 |

# Task 3 – Sales

Write a class Sale holding the following data: **town**, **product**, **price**, **quantity**. Read a **list of sales** and calculate and print the **total sales by town** as shown in the output. Order the towns **alphabetically** in the output.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 5  Sofia beer 1.20 160  Varna chocolate 2.35 86  Sofia coffee 0.40 853  Varna apple 0.86 75.44  Plovdiv beer 1.10 88 | Plovdiv -> 96.80  Sofia -> 533.20  Varna -> 266.98 | Plovdiv -> 1.10 \* 88 = 96.80  Sofia -> 1.20 \* 160 + 0.40 \* 853 = 533.20  Varna -> 2.35 \* 86 + 0.86 \* 75.44 = 266.98 |

# Task 4 – Cars

Create classes to represent cars with registered owners, for records kept by the Road Transport Administration Agency. Each car registration should have a manufacturer name, a model name, an owner (a pointer to a Person object), horsepower (a number) and a registration number (a string). A Person has a name, age, and a unique numerical id starting from 0 (the first Person has an id of 0, the second – an id of 1 and so on). A single Person can have multiple cars registered to them. A car’s owner and registration number can be changed, but they always change together (i.e. when you change a car’s owner, you also change its registration number). A person’s name and age can change, but their id always stays the same. Write a program which can create Person objects and create car registrations by reading input from the console, and can print out information about a car registration (and the owner it is registered to) – the input and output format is up to you, just make sure it is easy to enter the input and read the output.

Make sure you create the proper classes, constructors, access modifiers and methods for the above task. You should submit your program in a single .cpp file.

# Task 5 – Save to File

Write classes which can save an array of Person objects and an array of car registrations (from task 4) into a file, as well as read them back from a file. Hint: make sure you first read/write the Person objects – notice that you can’t store pointers to Person objects in a file for your car registration (you can, but they won’t be valid, since the memory addresses will change), but you should instead store the id of the person and then find that person in the array to re-create the car registration.

Make sure you create the proper classes, constructors, access modifiers and methods for the above task. You need to write two programs for this task, 5A and 5B, and submit the .cpp files for 4A and 4B.

## Task 5A

Write a program which writes several Person objects and car registrations to a file.

You should submit your program in a single .cpp file.

## Task 5B

Write a program which reads several Person objects and car registrations from a file. Test if the objects you created when reading from the file are correct by writing functions which assert the information stored in the objects (a good way to do that would be to have getInfo() methods on the Person and car registration classes and assert if that info is correct).

You should submit your program in a single .cpp file.

# Task 6 – Astronomical Objects

You are tasked with creating a console application which will store and display information about astronomical objects. Each object has a name of a home solar system, a position in the system, a mass in kg, a radius in meters, and a type (star, rocky planet, gas giant, unknown) and a nickname. Only some astronomical objects have a nickname. The name of the home solar system of an object can change (the solar system could be renamed), and the type can ONLY change from unknown to star, rocky planet, or gas giant. Objects of type star always have their position in the system set to 1. The designation of an astronomical object has the format:

home solar system – position in the system (nickname)

Where (nickname) is omitted if the object has no nickname. E.g. the sun of the Cancri system will have a designation Cancri-1, while the 8th planet named “Steel World” will have a designation of Cancri-9 (Steel World).

Write a program, which allows the user to create info about astronomical objects as well as search for all planets of a star system (by typing the name of the system) or of a specific object of a system (by typing the name and position of the object). When displaying info about objects the user searched for, use the following format:

designation { mass: *mass in kg*, radius: *radius in meters* }

For example, if Steel World had a mass of 5.972e+24 kg and a radius of 6 137 000 meters, displaying it to the user would look like:   
Cancri-9 (Steel World) {mass: 5.972e+24, radius: 6137000} (note: don’t worry about the exact format of the numbers, just show the numbers however cout decides to print them).

Creating objects is up to you – just make sure a user can add the info of any type of astronomical object.

The program should store the info in a text file (each time an object is created) and should load the info each time it starts (so that users can look-up objects they created previously).

Make sure you create the proper classes, constructors, access modifiers and methods for the above task. You should submit your program in a single .cpp file, but also add a file which contains input which can be copy-pasted into the console to demonstrate creating astronomical objects and searching for astronomical objects.