# 04. OOP Basics with C++ - 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 only the .cpp files for tasks 1, 2, 3, 4B and 4B, and a .cpp file and example input file for task 5 (see the task for more details).

.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.string-formatter.cpp

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

Tasks 3 to 5 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.

## Task 1

Implement a LineParser class, which takes in a string and has methods to parse the string into an array of numbers, an array of strings (by getting the words separated by spaces) and an array of chars (by getting only non-whitespace chars). The return type of the methods can either be pointers to arrays (along with a int& parameter to get the length of the returned array – remember the parseNumbers function from one of the previous exercises?), or the SmartArray class we implemented, or the std::vector class. So, if you pick SmartArray, the class should look somewhat like this (note: no need to exactly match syntax, just have methods with this amount of parameters and similar return types) :

|  |
| --- |
| class LineParser {  LineParser(const string& line);  IntSmartArray getNumbers() const;  StringSmartArray getStrings() const;  CharSmartArray getChars() const;  void changeLine(const string& line);  } |

Test your class by calling it from main() (or another function) with at least 10 examples of input data (including empty strings, strings which would yield 1-element arrays, bigger strings, etc.). You don’t need to handle incorrect input – assume that if getNumbers() is called, the entire string can be parsed into an array of numbers (i.e. there won’t be words mixed with the numbers).

## Task 2

Write a StringFormatter class, which takes a reference to an output string as a parameter, as well as a string formatPrefix parameter. The class should have a   
void format(string insertArr[], int insertArrSize) and a   
void format(int insertArr[], int insertArrSize) method, which replaces the concatenation of formatPrefix and i with the value of insertArr[i] (where 0 < i < insertArrSize). NOTE: you can also use SmartArray or std::vector as a single parameter for the format() method. Simply said, you have to make a class which takes a string and “formats” it by replacing the formattable substrings with the values of the passed in array

|  |
| --- |
| class StringFormatter {  StringFormatter(string& stringToFormat, const string& formatPrefix);  void format(string insertArr[], int insertArrSize);  void format(int insertArr[], int insertArrSize);  // or like this if you use SmartArray:  //void format(StringSmartArray insertArr)  //void format(IntSmartArray insertArr)  }  //example call *(advice: copy-paste this in a text/cpp file for readability)*:  string s = “Dear \*:0, Our company \*:1 wants to make you a Donation Of \*:2 Million \*:3 in good faith. Please send us a picture of your credit card”  StringFormatter formatter(s, “\*:”);  formatter.format(new string[]{“Ben Dover”, “Totally Legit NonSpam Ltd”, “13”, “Leva”}, 4);  cout << s; //*should print “Dear Ben Dover, Our company* Totally Legit NonSpam Ltd *wants to make you a Donation Of 13 Million Leva in good faith. Please send us a picture of your credit card”* |

The format methods should verify that the insertArr is not bigger than the number of formatPrefixes in the text and cause an error if it is. Test your class by calling it from main() (or another function) with at least 10 examples of input data (including arrays larger than 9 elements, empty arrays, etc.)

## Task 3

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 const methods for the above task. You should submit your program in a single .cpp file.

## Task 4

Write classes which can save an array of Person objects and an array of car registrations (from task 3) 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 const methods for the above task. You need to write two programs for this task, 4A and 4B, and submit the .cpp files for 4A and 4B.

## Task 4A

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 4B

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 5

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 const 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.