CS 202 - Computer Science II

Project 3

Due date (FIXED): Wednesday, 2/12/2020, 11:59 pm

Objectives: The two main objectives of this project is to test your ability to (1) create and use pointers, and (2) create and use C++ classes. A review of your knowledge of structs, arrays, iostream, file I/O and C-style strings is also included.

Description:
This project will expand Project 2 by adding additional functionality, using pointers, and implementing abstract data types (ADTs) through classes. **Pointers must be used for all array manipulation**, including arrays with ADTs (structs, classes) e.g, rental cars, rental agencies.
Important Restriction:

- **Pointers must be used in function prototypes and function parameter lists** - not square brackets. The only allowed square bracket usage is when declaring an array variable.
  Make sure all your C-string functions (e.g. string copy, string compare, etc.) work with pointers (parameters list and function implementation).
  Remember: pointer arithmetic (e.g., ++ , +=, - -, -=) and **setting the pointer back to the base address** using the array name **can be used to move through arrays**.
  All pointers must be passed by value. (*Note*: Try to also get accustomed to using the arrow operator (->) with Class Object pointers for member access if you use such in your code.)

The new functionality is as follows: You are given an updated data file (e.g. Agencies.txt) where there are 3 rental Car Agency locations, where **each of the 3** locations (**RentalAgency**) has **5** cars (**RentalCar**). You will have **similar menu options**, but the **functionality has been updated** below.

The RentalCar Class will contain the following private data members:

- **m_year**, an `int` (year of production)
- **m_make**, a C-string (`char` array of 256 maximum size)
- **m_model**, a C-string (`char` array of 256 maximum size)
- **m_price**, a `float` (price per day)
- **m_available**, a `bool` (1 = true; 0 = false; try to display true/false using the "`std::boolalpha`" manipulator like: `cout << boolalpha << boolVariable;`)

and will have the following public methods:

- **Default Constructor** – will set the aforementioned data members to default initial values of your selection, and m_available specifically to false. (*Hint*: Remember to use properly named constants where appropriate).
- **Parameterized Constructor** – will create a new object based on the values of the arguments passed into it. In its parameters list: a) there should be a parameter corresponding to every one of the class data members, b) there should also be a default parameter for the parameter used to specify whether the car is available.
- Separate **get** and **set methods** for all data members.
- **print** – will print out to terminal all the car's data (including their "available" status).
- **estimateCost** – will estimate the car's cost *given* (via a parameter passed to it) a number of days to rent it for.

The RentalAgency ADT will be a struct and will contain the following data members:

- **name**, a C-string (`char` array of 256 maximum size)
- **zipcode**, an `int` array of size 5
  (*Hint*: You will *NOT* be able to use `cin` and `cout` –or any `fstream` objects– directly with this `int` array as you were doing so far with C-strings. The reason is that reading/writing is specially handled by C++ for `char` array types. You will need to manage reading/writing to an `int` array on your own.)
- **inventory**, a `RentalCar` object array of size of 5

You should again have a **userMenuPrompt()** function that displays the following **6 possible options** to the terminal, and takes **user input (1-6)** to execute the corresponding functionality. Each option has to be accompanied with a **descriptive prompt** that informs the user what functionality will be executed. Options **outside the possible** range have to be appropriately handled. The User Menu should be **displayed again** after an option is selected and executed (except for the case of option 7 which should terminate the program). The options to implement are:

- **Option 1) Ask** the user for the **input file name**, and then open and **read ALL** data from that **file** (the file has been **structured** where the first line is the Car Agency info, followed by 5 cars). The data have to be stored into **arrays of Class type Objects**.
  You have to declare and implement a function **`readCars()`** that takes whatever parameters are necessary, and completely handles the above described functionality.
  You have to make use of the Car class public interface method(s) to do so.
- **Option 2) Print out to terminal ALL** data for **all Agencies** and **all their corresponding Cars** in a way that demonstrates this relationship (see Sample Output section).
  You have to declare and implement a function **`printAgenciesToTerminal()`** that takes whatever parameters are necessary, and completely handles the above described functionality.
  You have to make use of the Car class public interface method(s) to do so.
- **Option 3) Estimate** car **rental cost**. You should **prompt** for a particular combination of:
  a) an **Agency** (e.g., Hertz – you can have the user provide a 1-3 indexing number per-Agency),
  b) a **Car number** (with a 1-5 indexing number per-Car of each agency),
  c) a number of **days to rent** it for (`rental_cost=days_to_rent*price_per_day`), and print the corresponding `rental_cost` out to terminal.
  You have to declare and implement a function **`estimateRentalCost()`** that takes whatever parameters are necessary, and completely handles the above described functionality.
  You have to make use of the Car class public interface method(s) to do so.
- **Option 4) Find** the **cheapest available** Car – **Print to terminal** all the data of the single least expensive Car out of all 3 Agencies, that is still available.
  You have to declare and implement a function **`findCheapestRental()`** that takes whatever parameters are necessary, and completely handles the above described functionality.
  You have to make use of the Car class public interface method(s) to do so.
- **Option 5) Reserve** a Car and **Refresh** list – You should **prompt** for a particular combination of: a) an **Agency**, b) a **Car** using the same 1-3 and 1-5 indexing as in Option 3. You should then update that Agency's Car as reserved, and refresh the list on the terminal by **`printAgenciesToTerminal()`**. If the car is not available, you should print a warning message to the user.
  You have to declare and implement a function **`reserveCar()`** that takes whatever parameters are necessary, and completely handles the above described functionality.
  You have to make use of the Car class public interface method(s) to do so.

- **Option 6) Exit** program.

The following minimum functionality and structure is required:

- Ask the **user** for the **input file** name.
- The list(s) of cars must be stored into **array(s) of Class type Objects**.
- The list of agencies must be stored into an **array of Struct Objects**.
- Use **character arrays** to hold your strings (i.e., C-strings) exclusively (using the `std::string` data type is still not allowed).
- Write **multiple functions** (Hint: You could have each menu option be a function).
- At least one function must use **pass by-Reference**. Otherwise, as before, you are free to use **pass by-Value, pass by-Reference**, **pass by-Address** for your function parameters. (*Note*: Remember that using pass by-Value will make the function work on a local internal copy of whatever variable you pass as an argument, therefore the change will not be made on the actual argument itself, and it will be left unaffected after the function call is complete).
- **Pointers** must be used for **all array manipulation** (iterating over elements to read/modify cannot be performed with bracket operator accessing for this Project).
- **Pointers** must be used in **function prototypes** and **function parameter lists** (the bracket notation is not allowed in parameters lists for this Project).
- **Pointers** can only be **moved by incrementing or decrementing** (pointer arithmetic):

```
double d[3] = {1,2,3};
double * d_Pt = d;
for (int i=0; i<3; ++i,++d_Pt){ cout << *d_Ptd; }
```

Or by **setting** the pointer **back to the base address** using the array name.

```
d_Pt = d;  cout << *d_Pt << endl;
```

- Write your **own C-string length**, **compare**, **copy**, **concatenate** functions. Their prototypes will have the form (use the prototypes exactly as provided, with `char *` parameters):

```
// counts characters in str array until a NULL-character '\0' is
found, then it returns that number excluding the '\0' one
// the return type size_t represents an unsigned integral number
large enough to contain the maximum possible number of a storage
size that can appear on a target architecture
size_t myStringLength(const char * str);

// returns 0 when the C-strings match, i.e. their characters are
equal one-by-one until a NULL-character '\0' is found in both
strings and at the same position as well
// returns a value <= -1 if the first character that does not
match has a lower value in str1 than in str2
// returns a value >= 1 if the first character that does not
match has a higher value in str1 than in str2
int myStringCompare(const char * str1, const char * str2);

// copies characters from source to destination array until a
NULL-character '\0' is found in source, then it NULL-terminates
destination too
// returns a pointer to the destination array
char * myStringCopy(char * destination, const char * source);

// appends the content of source to the destination array
// this means that the NULL-terminator of destination is
overwritten by the first character of source and a NULL-character
```

```
'\0' is appended at the end of the concatenated Cstring in
destination
// returns a pointer to the destination array
char * myStringCat(char * destination, const char * source);
```

- Your code has to have the following structure:

- **proj3.cpp** : The source code file that contains your main() function. It has to include all necessary headers.
- **my_string.h** : The header file that provides the declarations of the functions: **`myStringLength`**, **`myStringCompare`**, **`myStringCopy`**, **`myStringCat`**.
- **my_string.cpp** : The source code file that provides the implementations for the above mentioned functions of my_string.h.
- **RentalCar.h**: The header file that provides the declaration of the class RentalCar.
- **RentalCar.cpp**: The source code file with the implementation of the class RentalCar.
- **Agency.h**: The header file that provides the declaration of the struct RentalAgency. RentalAgency has no methods to implement, so you will not need a source file.
- **menu.h** : The header file that provides the declarations of the functions: **`userMenuPrompt`**, **`readCars`**, **`printAgenciesToTerminal`**, **`estimateRentalCost`**, **`findCheapestRental`**, **`reserveCar`**.
- **menu.cpp** : The **source** code file that provides the implementations for the above mentioned functions of menu.h.

Important Requirement:

- Based on this codebase structure and the implied dependencies, you also have to create a **Makefile** that appropriately handles building your entire project.

The following are a list of restrictions:

- Compile your code using either the C++98 or C+03 standard but no higher (**`g++ -std=c++98 ...`** -or- **`g++ -std=c++03 ...`**).
- No usage of external libraries for C-string manipulation is allowed (e.g. **`<cstring>`** **`<string.h>`**), or any **`std::string`** libraries and data types.
- No libraries except **`<iostream>`** and **`<fstream>`** are allowed.
- No global variables except **`const`** ones.
- No usage of dynamic memory.
- You are expected to employ code abstraction and reuse by implementing and using functions. Copy-pasting code segments (that each performs a specific functionality which can be wrapped within a function) throughout your program *will be penalized*. Declare, Implement, and Use functions to achieve your objectives. The already provided code structure in the project description will be considered sufficient.

Advice: Implement the concepts of encapsulation and data hiding!

Making the RentalCar data members private and implementing a public interface through the class Constructors and other methods is your first acquaintance with these concepts in practice.

Sample Output for menu option 2:

Hertz 93619

2014 Toyota Tacoma , $115.12 per day , Available: true

2012 Honda CRV , $85.1 per day , Available: false

2015 Ford Fusion , $90.89 per day , Available: false

2013 GMC Yukon , $110.43 per day , Available: false

2009 Dodge Neon , $45.25 per day , Available: true

Alamo 89502

2011 Toyota Rav4 , $65.02 per day , Available: true

2012 Mazda CX5 , $86.75 per day , Available: true

2016 Subaru Outback , $71.27 per day , Available: false

2015 Ford F150 , $112.83 per day , Available: true

2010 Toyota Corolla , $50.36 per day , Available: true

Budget 93035

2008 Ford Fiesta , $42.48 per day , Available: false

2009 Dodge Charger , $55.36 per day , Available: true

2012 Chevy Volt , $89.03 per day , Available: false

2007 Subaru Legacy , $59.19 per day , Available: false

2010 Nissan Maxima , $51.68 per day , Available: true

The completed project should have the following properties:

- Written, compiled and tested using Linux.
- It must compile successfully using the g++ compiler on department machines or the provided Xubuntu VM image.
- The code must be commented and indented properly.
  Header comments are required on all files and recommended for the rest of the program.
  Descriptions of functions commented properly.
- A one page (minimum) typed sheet documenting your code. This should include the overall purpose of the program, your design, problems (if any), and any changes you would make given more time.

Turn in: Compressed .cpp and .h files, and your Makefile. Also, your project documentation.

Submission Instructions:

- You will submit your work via WebCampus
- Compress your:
  1. Source code files
  2. Header files
  3. Makefile
  4. Documentation

  Do not include executable
- Name the compressed folder:
  PA#_Lastname_Firstname.zip
  ([PA] stands for [ProjectAssignment], [#] is the Project number)
  Ex: PA3_Smith_John.zip

Verify: After you upload your .zip file, re-download it from WebCampus. Extract it, compile it and verify that it compiles and runs on the ECC systems.

- Code that does not compile will be heavily penalized –may even cost you your *entire* grade–. Executables that do not work 100% will receive partial grade points.
- It is better to hand in code that compiles and performs partial functionality, rather than broken code. You may use your Documentation file to mention what you could not get to work exactly as you wanted in the given timeframe of the Project.

Late Submission:
A project submission is "late" if any of the submitted files are time-stamped after the due date and time. Projects will be accepted up to 24 hours late, with 20% penalty.