COMP 2401
Introduction to Software Engineering
Assignment 1

Released: Saturday January 20, 2018, 12:00 Noon

Due: Saturday February 3, 2018, 12:00 Noon

Customer Information Storage for an Automotive Mechanic Business

Instructions

In this assignment you will create a small program which could be used by an automotive mechanic business to keep track of their customers and their vehicles. You will begin with the skeleton code posted in cuLearn. Your program **must** use the provided skeleton code **without making any changes to the existing code, data structures or function prototypes.** You will, however, add your own code to it. Below are the requirements for your program. Note: all strings described in this assignment will be declared to have a length of MAX_STR.

1. **Defined structures**
   This program makes use of two structures that need to be completed. The first is the VehicleType structure. This structure will contain all of the data for each car owned by the customers. This structure will have fields for the following pieces of data:
   - make (string)
   - model (string)
   - colour (string)
   - year (int)
   - mileage (int)

   The second structure (CustomerType) will contain all of the data regarding the individual customers. It will have the following fields:
   - first name (string)
   - last name (string)
   - number of vehicles registered with the shop (int)
   - a VehicleType array storing the individual vehicles owned by the customer (declared to have a length of MAX_VEHICLES

2. **Functions**

   This program has a number of functions which you need to complete. The function prototypes in the skeleton code file must be kept as is. Here is a description of what the functions must do:

   A. **Print_vehicle**

      This function takes a pointer to a vehicle structure as a parameter and prints out all of the information from the specific vehicle on a single line in a human readable manner.

   B. **Print_customer**

      This function takes a pointer to a customer structure as a parameter and prints out all of the information from the specific customer. The first line will contain their full name and the number of registered vehicles. You will then use a loop and call the **print_vehicle** function for each vehicle in the array.

   C. **Add_vehicle**

      This function takes a pointer to a customer structure and a pointer to a vehicle structure. The function will check if the number of registered vehicles is less than the maximum allowed. If so, it will add the vehicle to the array, increment the vehicle counter and return 0 (signifying the successful addition of the vehicle to the array. If there is no room in the array the function will return -1.

   D. **Create_customer**

      This function takes a first name and last name (strings) as parameters and is used to initialize an instance of the customer structure. It will set the new instance first name and last name appropriately, set the number of registered vehicles to 0 and return the new instance.

   E. **Create_vehicle**

      This function takes all of the required data to create an instance of the vehicle structure. It will create an instance of this structure, initialize all of the fields appropriately and return the newly created instance.

3. **Main function**

   Your main function will now make use of the functions described above. Already declared in the main function is an array of CustomerType structures. You will completely fill this array with instances of the CustomerType structure representing individual customers. As described above, each customer can have a variable number of vehicles. All of your customers must have at least one vehicle registered. On top of this, you are to have at least one customer with 2 vehicles, at least one customer with 3 vehicles and at least one customer with 4 vehicles. You must make use of the **create_customer** and **create_vehicle** functions to create instances of the structures. You then must use the **add_vehicle** function to add vehicle structures to the respective customer. For the customer with 4 vehicles, you must also attempt to add a fifth vehicle to their array. Because of how the **add_vehicle** function was implemented, this will not cause an error. It will simply see that the vehicles array is full, do nothing and return -1. For the time being we are ignoring this return value, however this is a test to ensure it will work.

Lastly you will loop over all of the CustomerType structures in the customers array and use the **print_customer** function (which itself uses the **print_vehicle** function) to display all of the customer data to the screen. See the end of this assignment for some sample output.

Constraints

- you must use the function prototypes exactly as stated
- do not use any global variables
- your program must reuse functions everywhere possible
- your program must be thoroughly commented
- your program must perform all basic error checking
- **your program must compile and run in the provided Virtual Machine**

Submission

You will submit in cuLearn, before the due date and time, one **tar** file that includes all the following:

- all source code, including the code provided, if applicable
- a readme file that includes:
    - a preamble (program author, purpose, list of source/header/data files)
    - the exact compilation command
    - launching and operating instructions

Grading [out of 15 marks]

Marking components:

- 2 marks: structures
    - 1 mark **VehicleType** structure
    - 1 mark **CustomerType** structure
- 1 mark: correctly implementing **print_vehicle** function
- 3 marks: correctly implementing **print_customer** function
- 3 marks: correctly implementing **add_vehicle** function
- 1 mark: correctly implementing **create_customer** function
- 1 mark: correctly implementing **create_vehicle** function
- 4 marks: main program
    - 1 mark for creating instances of the customer structures as described
    - 1 mark for creating instances of the vehicle structures as described
    - 1 mark for adding vehicles to customers as described
    - 1 mark for properly displaying all data

Notes:

In order to get credit for a marking component, the program must prove that the marking component executes successfully. This is usually accomplished by printing out correct data. For example, if the print_customer function does not work, then your program cannot prove that the other functions work correctly. That would result in a deduction of all the marks for adding to, removing from, and printing the array.

Deductions

- Packaging errors:
  - 10% for missing readme
- Major programming and design errors:
  - 50% of a marking component that uses global variables
  - 50% of a marking component that consistently fails to use correct design principles, including separate functions
  - 50% of a marking component where unauthorized changes have been made to provided code or prototypes
- Minor programming errors:
  - 10% for consistently missing comments or other bad style
  - 10% for consistently failing to perform basic error checking
- Execution errors:
  - 100% of a marking component that can't be tested because the code doesn't compile or execute in the VM
  - 100% of a marking component that can't be tested because the feature isn't used in the code
  - 100% of a marking component that can't be proven to run successfully because data is not printed out

Sample Console Output

```
$ gcc -Wall -o assignment1 assignment1.c
$ ./assignment1
Maurice Mooney, 1 vehicle(s)
2007 Ford Fiesta, Red, 100000KM

Abigail Atwood, 1 vehicle(s)
2016 Subaru Forester, Green, 40000KM

Brook Banding, 2 vehicle(s)
2018 Honda Accord, White, 5000KM
1972 Volkswagen Beetle, White, 5000KM

Ethan Esser, 1 vehicle(s)
2010 Toyota Camry, Black, 50000KM

Eve Engram, 3 vehicle(s)
2013 Toyota Corolla, Green, 80000KM
2015 Toyota Rav4, Gold, 20000KM
2017 Toyota Prius, Blue, 10000KM

Victor Vanvalkenburg, 4 vehicle(s)
2012 GM Envoy, Purple, 60000KM
2016 GM Escalade, Black, 40000KM
2015 GM Malibu, Red, 20000KM
2012 GM Trailblazer, Orange, 90000KM
```