William George

11/23/2014

CS 161

Week 8: Assignment

UNDERSTANDING

This weeks’ assignments are all about practicing with vectors and structs. The two excercises tackled one of the two and the project for the week tackles both. A struct is basically a class of variables that are all associated with one another and associated with a single variable. For example, with the date you have a day, month and year component and while they are all integers you still need three numbers to precisely define it. Structs also give the user and developer the opportunity to group a list of variables that are not necessarily of the same data type, like strings and integers. This is convenient because you can put all of the information for a particular thing in one place. Also when working with structs you can nest structs, meaning you can have a defined variable within a defined variable. In the project this is done with the date struct which is defined before it is placed in the car struct.

Vectors are similar to arrays but have some key differences. Like arrays they hold values in sequence in contiguous memory locations. They are also accessed like arrays with the [] operator. But unlike arrays vectors are easy to add elements to and to grow dynamically, so unlike arrays you do not need to know how large your vector will be. This is a powerful aspect of the vector and something you can only really understand the value of if you’ve worked with arrays first.

The purpose of the project this week is to build a program that will keep inventory and report sales in a car lot. It will use two structs or single variables representing a group of variables and a vector to keep track of all the car information already input. The program will open with a menu allowing the user to add a new entry to the car lot, to view the current inventory of the lot and to check profits in the lot for a given month. Each of these tasks will be completed with a specific function, meaning the main program will likely be very basic, just using a switch statement to call the different functions.

I think the most challenging aspect of this project will be the input validation aspect, especially with the date. While the month and year will be easy to validate, the day will be difficult because each month has a different number of days, either 30 or 31, or in one case 28. This part of the project will likely take the most time.

DESIGN

Project: carLot.cpp

DECLARE struct: Date

int date , int month, int year

DECLARE struct Car

string make, string model, int year, Date dateDurchased, double purchasePrice, bool isSold, Date dateSold, double salePrice

FUNCTIONS:

Void displayMenu

int getChoice

void listInventory

void addEntry

double monthProfits

bool dateValid

MAIN Program

DECLARE Variables:

Vector<Car> carsInfo;

int choice

int carsInLot

int totalCars

char ch

DO

CALL displayMenu()

SET choice = getChoice()

Switch using choice

case 1: CALL addEntry Function

break

case 2: CALL listInvent() Function

break

case 3: CALL monthProfits Function

break

case 4: OUTPUT “Good-Bye”

WHILE choice does not = 4

END program.

displayMenu Function

CLEAR SCREEN

OUTPUT: Car Lot Inventory: Select the corresponding number

OUTPUT: Select 1 to add entry

OUTPUT: Select 2 to List available vehicles

OUTPUT: Select 3 to calculate profits for a specific month

OUTPUT: Select 4 to quit

END FUNCTION

getChoice Function

DELCARE local variable

int choice

WHILE choice is invalid input

OUTPUT enter a valid choice

WHILE choice is not between 1 and 4

OUTPUT enter a valid choice

RETURN choice

END FUNCTION

addEntry Function: Parameter 🡪 vector<Car> cars

DECLARE local variables:

Car newCar

Char sold

OUTPUT: Enter Make

INPUT: newCar.make

OUTPUT: Enter Model

INPUT: newCar.model

OUTPUT: Enter Year

INPUT: newCar.year

WHILE newCar.year is invalid

INPUT newCar.year

OUTPUT: Enter day purchased

INPUT: newCar.datePurchased.day

WHILE newCar.datePurchased.day is invalid

INPUT newCar.datePurchase.day

OUTPUT: Enter month purchased

INPUT: newCar.datePurchased.month

WHILE newCar.datepurchased.month is invalid

INPUT newCar.datePurchased.month

OUTPUT: Enter year purchased

INPUT: newCar.datePurchased.year

WHILE newCar.datePurchased.year is invalid

INPUT newCar.datePurchased.year

OUTPUT : Enter Purchase Price:

INPUT: newCar.purchasePrice

WHILE newCar.purchasePrice is invalid

INPUT newCar.purchasePrice

OUTPUT: Has the car been sold? Enter Y or N

WHILE INPUT is not Y or N

OUTPUT: Please Enter Y or N

INPUT Y or N

IF sold is Y or y

newCar.isSold = true

OUTPUT: Enter day sold

INPUT: newCar.dateSold.day

WHILE newCar.dateSold.day is not valid

INPUT newCar.dateSold.day

OUTPUT: Enter month sold

INPUT: newCar.dateSold.month

WHILE newCar.dateSold.month is not valid

INPUT newCar.dateSold.month

OUTPUT: Enter year sold

INPUT: newCar.dateSold.year

WHILE newCar.dateSold.year is not valid

INPUT newCar.dateSold.year

OUTPUT: Enter sale price

INPUT newCar.salePrice

WHILE newCar.salePrice is not valid

INPUT newCar.salePrice

ELSE IF sold is N or n

newCar.isSold = false

ADD newCar entry to vector Car

END FUNCTION

listInventory function 🡪 parameter: vector<Car> cars

FOR each car in vector<Car> cars

IF cars.isSold is false

OUTPUT: cars.make

OUTPUT: cars.model

OUTPUT: cars.year

OUTPUT: cars.datePurchased.day / cars.datePruchased.month / cars.datePurchased.year

OUTPUT: cars.purchasePrice

END FUNCTION

monthProfit function 🡪 parameter: vector<Car> cars

DECLARE local variables

Double totalPurchasePrice = 0.0

Double totalSalesPrice = 0.0

Double monthProfits

Int month, year, index

OUTPUT Please enter a year

INPUT year

WHILE year is not valid

INPUT year

OUTPUT Please enter a month

INPUT month

WHILE month is not valid

INPUT month

FOR each car on lot

IF cars.datePurchased.year == year AND cars.datePurchased.month == month

totalPurchasePrice += cars.purchasePrice

IF cars.dateSold.year == year AND cars.datePurchased.month == month

totalSales += cars.salePrice

OUTPUT Total sales for month/year = $ totalSales

OUTPUT Total purchases for month/year = $totalPurchases

END FUNCTION

dayValid function 🡪 parameters: int day, int month

IF month is NOT between 1 and 12

RETURN false

IF month == 1,3,5,7,8,10 or 12

IF day is not between 1 and 31

RETURN false

ELSE IF month == 4,6,9,11

IF day is not between 1 and 30

RETURN false

ELSE IF month == 2

IF day is not between 1 and 28

RETURN false

RETURN true

END function

yearValid function 🡪 parameters: int year

IF year is not between 1908 and 2015

RETURN false

RETURN true

TESTING

|  |  |  |
| --- | --- | --- |
| INPUT | EXPECTED OUTPUT | ACTUAL OUTPUT |
| Menu input: 5 | Please enter a number between 1 and 4 | Expected |
| Menu input: F | Please enter a valid choice | Expected |
| Menu Input: 1 | Enter Make | Expected |
| Year Input 2017 | Enter a valid year | Expected |
| Year input 4 | Enter a valid year | Expected |
| Year input ffff | Enter a valid year | Expected |
| Year input 1920 | Enter day | Expected |
| Day input: 31 | Enter month | Expected |
| Month input: 9 | There are not that many days in that month | Expected |
| Year input : fff | Enter a valid year | Expected |
| Year input 1982 | Enter purchase price | expected |
| Price input:ff | Invalid price | Expected |
| Price input 12.02 | Has the car been sold? Enter Y or N | Expected |
| Y or N input f | Please enter y or n | Expected |
| Y or N input Y | Enter day sold | Expected |
| Day: 31 | Enter month | Expected |
| Month: 9 | Day is not in month re-enter | Expected |
| Year 1980 | Enter sale price |  |
| Sale price 200 | Press enter to return to menu | Expected |
| Menu input: 2 | No values | Expected |
| Menu input 3 | Enter a year | Expected |
| Year: 1980 month 9 | Total sales $200.00  Total profit $200.00 | Expected |
| Menu input: 4 | Good-bye | Expected |

REFLECTION

Looking back the project was quite a bit more challenging than I originally anticipated. I had the biggest issue with the add entry part of the project because I could not figure out how to access each individual element in the vector when referring to it via a struct. For a while my whole project was not working because the date was not properly initialized and I didn’t call the struct of the struct properly. Once I figured out what I did wrong though the project came together nicely.

As I predicted beforehand the input validation was a tedious task, but the logic wasn’t as tough as I had anticipated. In fact, the program itself came together much easier than I had predicted. I think the experience of building so many programs throughout the course of this class has really helped increase my confidence when encountering challenging problems. It has gotten much easier to see where I’ve made logic and syntax errors in the composition of the program and correct them the first time. At the beginning of the class there were several small issues where I would be looking at the same problem for a long period of time trying to figure out where I went wrong, now I can get one of these programs done in a few hours. I’m excited to see what we will do for the final project.