**Team Project**

**Name: \_\_\_\_\_\_Jorge A. Serrano\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** **ID#: \_\_\_121260\_\_\_\_\_\_\_\_\_\_**

1. Copy the source code developed for the Team Project and paste it as **text** below. (*15 points*)

Cars.h

//Cars.h

#pragma once

#include <iostream>

#include <string>

using namespace std;

class Cars

{

private:

static int count; // The quantity of Cars objects created

static size\_t brandLength; // the length of the brand field

static size\_t modelLength; // the length of the model field

string brand; // the car's brand name

string model; // the car's model name

int number; // the car's number

string serialNumber; // the car's serial number

string setSerialNumber(); // private method to create the serial number

void setBrandLength(); // private method to set the length of the brand field

void setModelLength(); // private method to set the length of the model field

string uppercase(string);

string setCount(int) const;

int getNumber() const; // get the number value

public:

Cars(); // the default constructor

Cars(string, string); // parameterized constructor

~Cars();

void setBrand(string); // set the car's brand

void setModel(string); // set the car's model

string getBrandName() const; // gets the car's brand name

string getModelName() const; // gets the car's model name

string getSerialNumber() const; // gets the car's serial number

size\_t getBrandLength() const; // gets the brand field's length

size\_t getModelLength() const; // gets the model field's length

void printCar() const; // prints the car's information

};

Dealer.h

#pragma once

/\*

\* CECS 2223, Computer Programming II Lab

\* Fall 2025, Sec. 05

\* Date: October 14, 2022

\* Topic: Lab 5 - Composition

\* File name: Dealer.h

\* This file declares a class named Dealer

\* Name: Prof. de la Cruz

\*/

/\* A SOLUTION TO THIS EXERCISE WAS DISCUSSED DURING THE OCTOBER 17 LAB SESSION \*/

// What to include???

#include "Cars.h"

class Dealer {

private:

int carCount; // total cars in inventory, i.e., the size of the array

Cars\*\* inventory; // array to store the dealer's car inventory

string name; // car dealer's name

public:

Dealer(); // the default constructor

Dealer(string); // parameterized constructor, includes car dealer's name

void setName(string); // set the car dealer's name

void addCarToInventory(string, string); // adds a car to inventory

void sellCar(string, string); // sell a car from inventory

int findCar(string, string) const; // searches for a car in inventory

string getName() const; // to get the dealer's name

int getCarCount() const; // gets the total number of cars in inventory

void printInventory() const; // prints a list of each car in inventory, including the serial number

void orderInventory(); // orders the cars in inventory alphabetically

//Methods Added by Team Project:

void sellCar(string); // sells a car by serial number

int findCar(string) const; // finds a car by serial number

int getBrandCount(string) const; // gets the total number of cars of the same brand

int getModelCount(string) const; // gets the total number of cars of the same brand and model

void printBrandInventory() const; // prints a list of the brands and the counts for each brand

void printModelInventory() const; // prints a list of each different brand and model, and the count

};

Cars.cpp

//Cars.cpp

#include "Cars.h"

#include <string>

// initialize the class variables to 0

int Cars::count = 0; // The quantity of Cars objects created

size\_t Cars::brandLength = 0; // the length of the brand field

size\_t Cars::modelLength = 0; // the length of the model field

// Define the constructors. Each constructor adds 1 to the

// count class variable. The parameterized constructor calls

// the setSerialNumber method.

// The default constructor

Cars::Cars() {

brand = "";

model = "";

serialNumber = "";

number = ++count;

}

// The parameterized constructor

Cars::Cars(string aBrand, string aModel) {

brand = aBrand;

setBrandLength();

model = aModel;

setModelLength();

number = ++count;

serialNumber = setSerialNumber();

}

Cars::~Cars()

{

cout << "\nDestructor\n";

}

// The setSerialNumber method creates the serial number for the car.

// The serial number is made up of the first three letters of the

// brand plus the first three letters of the model plus three numbers

// from the count. For example, for a Toyota Yaris with count 1 the

// serial number would be TOYYAR001

string Cars::setSerialNumber() {

return uppercase(brand.substr(0, 3) + model.substr(0, 3)) + setCount(getNumber());

}

// The uppercase method is a private method used to convert characters into their

// uppercase representation

string Cars::uppercase(string str) {

for (size\_t i = 0; i < str.size(); i++)

str[i] = toupper(str[i]);

return str;

}

// The setCount method converts the count value into a 3 character string

string Cars::setCount(int aCount) const {

if (aCount < 10)

return "00" + to\_string(aCount);

else if (aCount < 100)

return "0" + to\_string(aCount);

else

return to\_string(aCount);

}

int Cars::getNumber() const {

return number;

}

// The setBrandLength and setModelLength methods determine the size of

// the corresponding field. The size of each field in the longest string

// plus 3 spaces.

void Cars::setBrandLength() {

if (brand.size() + 3 > brandLength)

brandLength = brand.size() + 3;

}

void Cars::setModelLength() {

if (model.size() + 3 > modelLength)

modelLength = model.size() + 3;

}

// Define the setters and getters.

// The setters for brand and model will call the field length methods.

void Cars::setBrand(string aBrand) {

brand = aBrand;

setBrandLength();

if (serialNumber.empty() && !model.empty())

serialNumber = setSerialNumber();

}

void Cars::setModel(string aModel) {

model = aModel;

setModelLength();

if (serialNumber.empty() && !brand.empty())

serialNumber = setSerialNumber();

}

string Cars::getBrandName() const {

return brand;

}

string Cars::getModelName() const {

return model;

}

string Cars::getSerialNumber() const {

return serialNumber;

}

size\_t Cars::getBrandLength() const {

return brandLength;

}

size\_t Cars::getModelLength() const {

return modelLength;

}

// The printCar method prints the data for a car in a

// table ready format. The data to be printed is brand,

// model, and serial number. The information must be

// printed in a single line, make sure to add the line

// termination instruction.

// This method DOES NOT print the table header

void Cars::printCar() const {

printf("%-\*s%-\*s%-s\n", getBrandLength(), getBrandName().c\_str(), getModelLength(), getModelName().c\_str(), getSerialNumber().c\_str());

}

Dealer.cpp

/\*

\* CECS 2223, Computer Programming II Lab

\* Fall 2022, Sec. 05

\* Date: October 14, 2022

\* Topic: Lab 5 - Composition

\* File name: Dealer.cpp

\* This file defines the Dealer class

\* Name: Jorge A. Serrano

\*/

/\* A SOLUTION TO THIS EXERCISE WAS DISCUSSED DURING THE OCTOBER 17 LAB SESSION \*/

#include "Dealer.h"

// The default constructor initializes carCount to 0, inventory to the null pointer (nullptr)

// and name to the empty string ""

Dealer::Dealer() {

carCount = 0;

inventory = nullptr;

name = "";

}

// The parameterized constructor receives the string with the dealer's name as parameter,

// initializes carCount to 0, and inventory to the null pointer

Dealer::Dealer(string aName) {

carCount = 0;

inventory = nullptr;

name = aName;

}

// The setName method recieves the string with the dealer's name as parameter

void Dealer::setName(string name) {

this->name = name;

}

// addCarToInventory receives a car's brand and model as parameters, and adds it to

// the inventory array as a pointer to a Cars object. This method increments carCount

// by 1. Once the car has been added to inventory, the method orderInventory

// must be called to order the array.

void Dealer::addCarToInventory(string brand, string model) {

// first, find out if the array exists

if (getCarCount() > 0) { //the array exists

Cars\*\* temp = inventory; // temporary array to store the handles

inventory = new Cars \* [++carCount]; // the new array

// Trandfer values from the "old" array to the new one

for (int i = 0; i < getCarCount() - 1; i++)

inventory[i] = temp[i];

delete[] temp;

}

else { // the array must be created

inventory = new Cars \* [++carCount]; // the new array

}

// add the handle to the new car into inventory

inventory[getCarCount() - 1] = new Cars(brand, model);

orderInventory();

}

// The sellCar method deletes a Cars object from inventory. It must first make sure that

// the car exists in inventory by calling the findCar method. If the car is not found,

// print the message "The selected car, [brand] - [model], was not found in inventory"

// If the car is found, create a new array with one less position, and copy the

// contents of the old array into the new one, except for the car being sold. Once all

// remaining pointers have been copied to the new array, make usre to call the Cars

// destructor to remove the Cars object. This method decrements carCount by 1.

// Once the car has been removed from inventory, the method orderInventory

// must be called to order the array.

void Dealer::sellCar(string brand, string model) {

const int INDEX = findCar(brand, model);

if (INDEX > -1) { // the car exists in the array

Cars\*\* temp = inventory;

inventory = new Cars \* [--carCount];

int currentIndex = 0;

for (int i = 0; i < getCarCount() + 1; i++) {

if (i != INDEX) { // not the car to be removed

if (currentIndex < getCarCount()) { // the "new" array has space to store values

inventory[currentIndex] = temp[i];

currentIndex++;

}

}

}

temp[INDEX]->~Cars(); // invoke the destructor for the car to be removed

delete[] temp; // release the memory

}

else {

cout << "\nThe selected car, " << brand << " - " << model << ", was not found in inventory\n\n";

}

}

// This method searches inventory to find a match for the given car.

// If found, the method returns the index of the car in the array,

// otherwise, it returns -1.

int Dealer::findCar(string brand, string model) const {

if (getCarCount() > 0) { // the array exists, must search it

for (int i = 0; i < getCarCount(); i++) {

if (brand.compare(inventory[i]->getBrandName()) == 0 && model.compare(inventory[i]->getModelName()) == 0)

return i;

}

return -1;

}

else // the array does not exist

return -1;

}

// The getter for dealer's name

string Dealer::getName() const {

return name;

}

// getCarCount returns the total number of cars in inventory

int Dealer::getCarCount() const {

return carCount;

}

// This method prints a list of all cars in inventory in a table format.

// The method must print the phrase "Dealer [name] has [carCount] cars in inventory."

// Use the strings "BRAND", "MODEL", and "SERIAL NUMBER" as the table's header.

// Make sure to include code for the case where there are no cars in inventory

void Dealer::printInventory() const {

cout << "Dealer " << getName() << " has " << getCarCount() << " cars in inventory.\n\n";

if (getCarCount() > 0) {

printf("\t%-\*s%-\*s%-s\n", inventory[0]->getBrandLength(), "BRAND", inventory[0]->getModelLength(), "MODEL", "SERIAL NUMBER"); // table header

for (int i = 0; i < getCarCount(); i++) {

cout << "\t";

inventory[i]->printCar();

}

cout << endl;

}

}

// This methods orders the cars in inventory alphabetically, and has no

// parameters. Implement a selection sort algorithm, where the first item

// to be ordered is the brand name, followed by the model name.

// Recall that inventory ordering is only required if there is more than

// one car in inventory

void Dealer::orderInventory() {

if (getCarCount() > 1) { // there is more than one car in inventory

string lowest = "";

int lowIndex = -1;

Cars\* temp = nullptr;

for (int i = 0; i < getCarCount() - 1; i++) {

lowest = inventory[i]->getSerialNumber();

for (int j = i + 1; j < getCarCount(); j++) {

if (lowest.compare(inventory[j]->getSerialNumber()) > 0) {

lowest = inventory[j]->getSerialNumber();

lowIndex = j;

}

}

if (lowIndex > -1) { // a swap is required

temp = inventory[i];

inventory[i] = inventory[lowIndex];

inventory[lowIndex] = temp;

lowIndex = -1;

}

}

}

}

void Dealer::sellCar(string serialNum)

{

const int INDEX = findCar(serialNum);

if (INDEX > -1) { // the car exists in the array

Cars\*\* temp = inventory;

inventory = new Cars \* [--carCount];

int currentIndex = 0;

for (int i = 0; i < getCarCount() + 1; i++) {

if (i != INDEX) { // not the car to be removed

if (currentIndex < getCarCount()) { // the "new" array has space to store values

inventory[currentIndex] = temp[i];

currentIndex++;

}

}

}

temp[INDEX]->~Cars(); // invoke the destructor for the car to be removed

delete[] temp; // release the memory

}

else {

cout << "\nThe selected car with the serial number"<<serialNum<< ", was not found in inventory\n\n";

}

}

int Dealer::findCar(string serialNum) const

{

if (getCarCount() > 0) { // the array exists, must search it

for (int i = 0; i < getCarCount(); i++) {

if (serialNum.compare(inventory[i]->getSerialNumber()) == 0)

return i;

}

return -1;

}

else // the array does not exist

return -1;

return 0;

}

int Dealer::getBrandCount(string brandName) const

{

int brandCount = 0;

for (int i = 0; i < carCount; i++) {

if (brandName.compare(inventory[i]->getBrandName()) == 0) {

brandCount++;

}

}

return brandCount;

}

int Dealer::getModelCount(string modelName) const

{

int modelCount = 0;

for (int i = 0; i < carCount; i++) {

if (modelName.compare(inventory[i]->getModelName()) == 0) {

modelCount++;

}

}

return modelCount;

}

void Dealer::printBrandInventory() const

{

cout << "Dealer " << getName() << " has " << getCarCount() << " cars in inventory.\n\n";

if (getCarCount() > 0) {

printf("\t%-\*s%-s\n", inventory[0]->getBrandLength(), "BRAND", "COUNT"); // table header

for (int i = 0; i < getCarCount(); i++) {

printf("\t%-\*s%-d\n", inventory[i]->getBrandLength(), inventory[i]->getBrandName().c\_str(), getBrandCount(inventory[i]->getBrandName()));

}

cout << endl;

}

else

{

cout << "There are no cars on inventory." << endl;

}

}

void Dealer::printModelInventory() const

{

cout << "Dealer " << getName() << " has " << getCarCount() << " cars in inventory.\n\n";

if (getCarCount() > 0) {

printf("\t%-\*s%-\*s%-s\n", inventory[0]->getBrandLength(), "BRAND", inventory[0]->getModelLength(), "MODEL", "COUNT");// table header

for (int i = 0; i < getCarCount(); i++) {

printf("\t%-\*s%-\*s%-d\n", inventory[i]->getBrandLength(), inventory[i]->getBrandName().c\_str(), inventory[i]->getModelLength(), inventory[i]->getModelName().c\_str(), getModelCount(inventory[i]->getModelName()));

}

cout << endl;

}

else

{

cout << "There are no cars on inventory." << endl;

}

}

TeamProject.cpp

#include "Dealer.h"

#include"Cars.h"

#include <string>

using namespace std;

int menu(); // options menu

bool execute(int, Dealer&); // implements the action from the menu

int main() {

Dealer poli("Poli Auto Sales");

while (execute(menu(), poli) == true);

cout << "Program developed by Jorge A. Serrano, ID#121260";

system("pause");

return 0;

}

int menu() {

int option = 0;

cout << "Select one of the following options:\n";

cout << "\t1. Add a car to inventory\n";

cout << "\t2. Sell a car from inventory\n";

cout << "\t3. Print the inventory\n";

cout << "\t4. Print the Brands in inventory\n";

cout << "\t5. Print the Models in inventory\n";

cout << "\t6. Exit the program\n";

cout << "\n\tYour selection: ";

cin >> option;

cout << endl;

cin.ignore(256, '\n'); // to "clear" the stdin buffer

return option;

}

bool execute(int option, Dealer& aDealer) {

string brand = "", model = "";

switch (option) {

case 1:

cout << "Enter the car's brand name: ";

getline(cin, brand, '\n');

cout << "Enter the car's model name: ";

getline(cin, model, '\n');

aDealer.addCarToInventory(brand, model);

cout << endl;

return true;

case 2:

cout << "Enter the car's brand name: ";

getline(cin, brand, '\n');

cout << "Enter the car's model name: ";

getline(cin, model, '\n');

aDealer.sellCar(brand, model);

cout << endl;

return true;

case 3:

aDealer.printInventory();

return true;

case 4:

aDealer.printBrandInventory(); // prints a list of the brands and the counts for each brand

return true;

case 5:

aDealer.printModelInventory();

return true;

case 6:

cout << "Thank you for using this program!\n\n";

return false;

default:

cout << "\n\tERROR! Option #" << option << " is not valid.\n\tTry again!\n\n";

return true;

}

}

1. Paste the screenshots of the program’s execution below. (*5 points*)

[PASTE SCREENSHOT HERE]

1. **In a separate document**, complete your teammate’s review and submit it at the appropriate Blackboard drop. (*5 points*)

The program gave me many problems on the printf() of the printBrandInventory() and printModelInventory(). The program did compile but it wouldn’t print does two options, but after careful considerations I realized that using the getBrandName.c\_str() along with the info in the prinCar() method. In the end to solve the problem I made another printf() inside the if statement from the printBrandInventory() and printModelInventory().