COMPUTER SCIENCE

PROJECT



CAR RENTAL SYSTEM

Midhun Sushil

XII-J ( 2017-18 )

THE INDIAN SCHOOL BAHRAIN

TABLE OF CONTENTS

1. Certificate
2. Acknowledgement
3. Introduction
4. About C++
5. System Analysis
6. System Design
7. Sample Outputs
8. Source Code
9. Bibliography

Certificate

This is to certify that **MIDHUN .P. SUSHIL,** student of **XII-J, INDIAN SCHOOL BAHRAIN** has successfully completed the project based on the topic **“SOFTWARE FOR CAR RENTAL MANAGEMENT SYSTEM”**

in fulfilment of Computer Science Practical Examination conducted by AISSCE during the academic year **2017-2018**.

\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_

Signature of Signature of

Subject Teacher Examiner

Acknowledgement

In accomplishment of this project successfully, many people have best owned me their blessings and heart pledged support, this time I am utilizing to thank all the people who have been concerned with the project.

I would like to express my gratitude towards our

Computer Science Teacher Mrs. Smitha John and the

HOD of Computer Science Mr. Pius Mathew, whose valuable guidance has been the ones that helped me patch this project and make it full proof success.

Last but not the least I wish to thank my teammates Milan Mani Mathew, Joel George, Mervin D’Costa and Adarsh Chandran who have supported us in various phases of the completion of the project.

Introduction

The mini project **‘Car Rental Management System’** is the sample project for managing car rental. The project aims at developing car rental system using the C++ language that enables an organization to maintain its cars and customers.

The project demonstrates the creation of a user interface of a system. The application uses basic C++ function to generate login screen, menus, show boxes and print text on the screen. It also effectively applies the various C++ concepts such as file operations, looping and branching constructs and string manipulation functions.

It uses DOS file support as back end to save the records.

The application also implements the concept of classes to define the car and customer records and the concept of encapsulation to make all data private or protected, and to make public only those functions that are part of minimal interface for users of the class. This can hide the details of data implementation, allowing the developer to later change the implementation without changing the interface in any way.

About C ++

Loops in c++

Loops are used to repeat a block of code. Being able to have your program repeatedly execute a block of code is one of the most basic but useful tasks in programming.

**FOR** - *FOR* loops are the most useful type. The syntax is:

for ( variable initialization; condition; variable update ) {

Code to execute while the condition is true

}

**WHILE** - *WHILE* loops are very simple. The basic structure is:

while ( condition )

{ Code to execute while the condition is true }

The true represents a boolean expression which could be x == 1 or while ( x != 7 ) (x does not equal 7). It can be any combination of Boolean statements that are legal. Even, (while x ==5 || v == 7) which says execute the code while x equals five or while v equals 7.

DO.WHILE - *DO.WHILE*  loops are useful for things that want to loop at least once. The structure is:

do {

} while ( condition );

Loop control statements

Jump statements allow altering the flow of a program by performing jumps to specific locations.

**The break statement**

break leaves a loop, even if the condition for its end is not fulfilled. It can be used to end an infinite loop, or to force it to end before its natural end.

**The continue statement**

The continue statement causes the program to skip the rest of the loop in the current iteration, as if the end of the statement block had been reached, causing it to jump to the start of the following iteration.

**The goto statement**

goto allows to make an absolute jump to another point in the program. This unconditional jump ignores nesting levels, and does not cause any automatic stack unwinding. Therefore, it is a feature to use with care, and preferably within the same block of statements, especially in the presence of local variables.

## **Infinite Loop**

There may exist some loops which can iterate or occur infinitely. These are called Infinite Loop. These loops occur infinitely because their condition is always true.

CLASSES

A **class** in **C++** is a user defined type or data structure declared with keyword *class* that has data and functions (also called methods) as its members whose access is governed by the three access specifiers *private*, *protected* or *public* (by default access to members of a class is *private*). The private members are not accessible outside the class; they can be accessed only through methods of the class. The public members form an interface to the class and are accessible outside the class.

Instances of a class data type are known as [objects](https://en.wikipedia.org/wiki/Object_(computer_science)) and can contain [member variables](https://en.wikipedia.org/wiki/Variable_(programming)), [constants](https://en.wikipedia.org/wiki/Variable_(programming)), [member functions](https://en.wikipedia.org/wiki/Method_(computer_science)), and [overloaded operators](https://en.wikipedia.org/wiki/Operator_overloading) defined by the programmer.

An important feature of the C++ class and structure are **member functions**. Each datatype can have its own built-in functions (referred to as methods) that have access to all (public and private) members of the datatype. In the body of these non-static member functions, the keyword [this](https://en.wikipedia.org/wiki/This_(computer_science)) can be used to refer to the object for which the function is called. This is commonly implemented by passing the address of the object as an implicit first argument to the function.

#### Encapsulation

[Encapsulation](https://en.wikipedia.org/wiki/Information_hiding) is the hiding of information to ensure that data structures and operators are used as intended and to make the usage model more obvious to the developer. C++ provides the ability to define classes and functions as its primary encapsulation mechanisms. Within a class, members can be declared as either public, protected, or private to explicitly enforce encapsulation. A public member of the class is accessible to any function. A private member is accessible only to functions that are members of that class and to functions and classes explicitly granted access permission by the class ("friends"). A protected member is accessible to members of classes that inherit from the class in addition to the class itself and any friends.

#### Inheritance

[Inheritance](https://en.wikipedia.org/wiki/Inheritance_(computer_science)) allows one data type to acquire properties of other data types. Inheritance from a [base class](https://en.wikipedia.org/wiki/Base_class) may be declared as public, protected, or private. This access specifier determines whether unrelated and derived classes can access the inherited public and protected members of the base class. Only public inheritance corresponds to what is usually meant by "inheritance". The other two forms are much less frequently used. If the access specifier is omitted, a "class" inherits privately, while a "struct" inherits publicly. Base classes may be declared as virtual; this is called [virtual inheritance](https://en.wikipedia.org/wiki/Virtual_inheritance). Virtual inheritance ensures that only one instance of a base class exists in the inheritance graph, avoiding some of the ambiguity problems of multiple inheritance.

[Multiple inheritance](https://en.wikipedia.org/wiki/Multiple_inheritance) is a C++ feature not found in most other languages, allowing a class to be derived from more than one base class; this allows for more elaborate inheritance relationships. For example, a "Flying Cat" class can inherit from both "Cat" and "Flying Mammal". Some other languages, such as [C#](https://en.wikipedia.org/wiki/C_Sharp_(programming_language)) or [Java](https://en.wikipedia.org/wiki/Java_(programming_language)), accomplish something similar (although more limited) by allowing inheritance of multiple [interfaces](https://en.wikipedia.org/wiki/Interface_(computer_science)) while restricting the number of base classes to one (interfaces, unlike classes, provide only declarations of member functions, no implementation or member data). An interface as in C# and Java can be defined in C++ as a class containing only pure virtual functions, often known as an [abstract base class](https://en.wikipedia.org/wiki/Abstract_base_class) or "ABC". The member functions of such an abstract base class are normally explicitly defined in the derived class, not inherited implicitly. C++ virtual inheritance exhibits an ambiguity resolution feature called [dominance](https://en.wikipedia.org/wiki/Dominance_(C%2B%2B)).

Polymorphism

[Polymorphism](https://en.wikipedia.org/wiki/Type_polymorphism) enables one common interface for many implementations, and for objects to act differently under different circumstances: A **polymorphic type** is one whose operations can also be applied to values of some other type, or types. There are several fundamentally different kinds of polymorphism:

* [*Ad hoc polymorphism*](https://en.wikipedia.org/wiki/Ad_hoc_polymorphism): when a function denotes different and potentially heterogeneous implementations depending on a limited range of individually specified types and combinations. Ad hoc polymorphism is supported in many languages using [function overloading](https://en.wikipedia.org/wiki/Function_overloading).
* [*Parametric polymorphism*](https://en.wikipedia.org/wiki/Parametric_polymorphism): when code is written without mention of any specific type and thus can be used transparently with any number of new types. In the [object-oriented programming](https://en.wikipedia.org/wiki/Object-oriented_programming) community, this is often known as *generics* or [*generic programming*](https://en.wikipedia.org/wiki/Generic_programming). In the [functional programming](https://en.wikipedia.org/wiki/Functional_programming) community, this is often shortened to *polymorphism*.
* [*Subtyping*](https://en.wikipedia.org/wiki/Subtyping) (also called *subtype polymorphism* or *inclusion polymorphism*): when a name denotes instances of many different classes related by some common superclass. In the object-oriented programming community, this is often referred to as simply *Inheritance*

Pointers

In [computer science](https://en.wikipedia.org/wiki/Computer_science), a **pointer** is a [programming language](https://en.wikipedia.org/wiki/Programming_language) object, whose value refers to (or "**points** to") another value stored elsewhere in the [computer memory](https://en.wikipedia.org/wiki/Computer_memory) using its [memory address](https://en.wikipedia.org/wiki/Memory_address). A pointer *references* a location in memory, and obtaining the value stored at that location is known as [*dereferencing*](https://en.wikipedia.org/wiki/Dereference_operator) the pointer. As an analogy, a page number in a book's index could be considered a pointer to the corresponding page; dereferencing such a pointer would be done by flipping to the page with the given page number and reading the text found on the indexed page.

Pointers to data significantly improve performance for repetitive operations such as traversing [strings](https://en.wikipedia.org/wiki/String_(computer_science)#String_processing_algorithms), [lookup tables](https://en.wikipedia.org/wiki/Lookup_table), [control tables](https://en.wikipedia.org/wiki/Control_table) and [tree](https://en.wikipedia.org/wiki/Tree_(data_structure))structures. In particular, it is often much cheaper in time and space to copy and dereference pointers than it is to copy and access the data to which the pointers point.

File Handling

Files are a means to store data in a storage device. C++ file handling provides a mechanism to store output of a program in a file and read from a file on the disk. So far, we have been using <iostream> header file which provide functions cin and cout to take input from console and write output to a console respectively. Now, we introduce one more header file <fstream> which provides data types or classes ( ifstream , ofstream , fstream ) to read from a file and write to a file.

**File Opening Modes**  
A file can be opened in different modes to perform read and write operations. Function to open a file i.e **open( )**takes two arguments : **char \*filename** and **ios :: mode**. C++ supports the following file open modes :

|  |  |
| --- | --- |
| **Mode** | **Explanation** |
| ios :: in | Open a file for reading |
| ios :: out | Open a file for writing |
| ios :: app | Appends data to the end of the file |
| ios :: ate | File pointer moves to the end of the file but allows to writes data in any location in the file |
| ios :: binary | Binary File |
| ios :: trunc | Deletes the contents of the file before opening |

SYSTEM ANALYSIS

The program is user friendly and coding has been done with proper efficiency. The software is made as an administrator operated programs. So, the features of this program can be used by the customers only with the supervision of the administrator.

The Structure of the program:

*Login Screen*

The admin must enter the username and password to login.

*Main Menu*

The Main Menu contains 8 options where the option number has to be entered to select the option.

The options are:

*1. Rent A Car:* The customer has to enter the required details along with from and to dates of rent period. After entering the details, the inventory is displayed, and the customer has to choose an available car by entering the serial number. Then the contract is displayed with the options Save and Cancel.

*2. Add Car:* The administrator is asked about the number of cars to be added and necessary details of the cars to be added will asked as many times.

*3. Search and Display:* On opening this option 5 more options are displayed.

They are:

1. *Display All Cars:* In this option all the details of the cars available in the system are displayed one by one.
2. *Display All Customers:* Here the contract of each customer is displayed one by one.
3. *Search Car:* In this option any one field among the asked options must be entered to search for a car.
4. *Search Customer:* Here any one field among the asked fields must be entered and the contract of that customer will be displayed.
5. *Go Back:* To return to the main menu.

*4. Reports:* On entering this function

The user is asked to enter the one among the following options:

1. *View All Car reports:* Where the inventory is displayed.
2. *View Customer Report:* As the name suggests.
3. *View Customer-Car Report:* Here the details of the customers are displayed in a tabular form with only relevant details.
4. *Go Back:* To return to the main menu.

*5. Edit Record:* This includes editing of all the record that the system handles with. They are:

1. *Update Car:* To Update car details
2. *Update Customer:* To update contract.
3. *Delete Car:* To delete a car by entering any one among the asked details.
4. *Delete Customer:* To delete the contract pf any customer.
5. *Edit Status:* This has further options like.
6. Change all status to Available.
7. Change all status to Booked
8. Switch Status.
9. Go Back: To return to Edit Record menu.
10. *Go Back:* To return to the main menu.

*6. Return and Billing:* This option is to be used when the customer returns the car.

Any one among the required field must be entered to view the contract.

Then on proceeding to the bill the current date is asked. The billing works in this manner- If it matches with the earlier mentioned to date in the contract the customer would not be fined. Whereas if it’s a late return the 150% of the rent per day will be fined for each day.

But as the dates in the contract are fixed early return will not be refund.

*7. About:* About our company

*8. Exit*.

*Header Files Used:*

<fstream.h> <iomanip.h> <stdio.h> <string.h> <stdlib.h> <conio.h> <ctype.h>

*Special User Defined Functions Used:*

* 1. ventry():

For validated input of a string of specific size (N) which can be passed as argument. This prevents users from inputting invalid characters such as inputting alphabets in a string that stores phone number.

Usage: ventry(string,code,size); //by default size=1

code=0 >> for inputting alphabets(a-Z), space, comma(,) & dot(.)

code=1 >> for inputting numbers(0-9), dot(.) & '/'

code=2 >> for inputting username / ID

code=3 >> for normal input without space, ':' & ';'

code=4 >> for menu input (only numbers 0-9)

* 1. clear():

For clearing specific character, characters or lines

This function basically uses gotoxy() and outputs space over to clear the specified no. of characters or lines. It's more convenient than the library function clrscr() (clears entire console output) as it can clear specified locations.

Usage: clear(x, y, noc, nol ). //by default nol=1

(x,y) >> Margin/position of first character;

noc >> number of characters to clear;

nol >> number of lines to clear;

* 1. border():

For creating border of length l(horizontal) & width w(vertical).

usage: border(x, y, l, w, type)

Where (p,q)-margin (coordinate of top-left vertices of box border);

type >> 0(+ \* !);

type >> 1(single border), 4(slow-motion)(\* ! +);

type >> 2(double character border)(/);

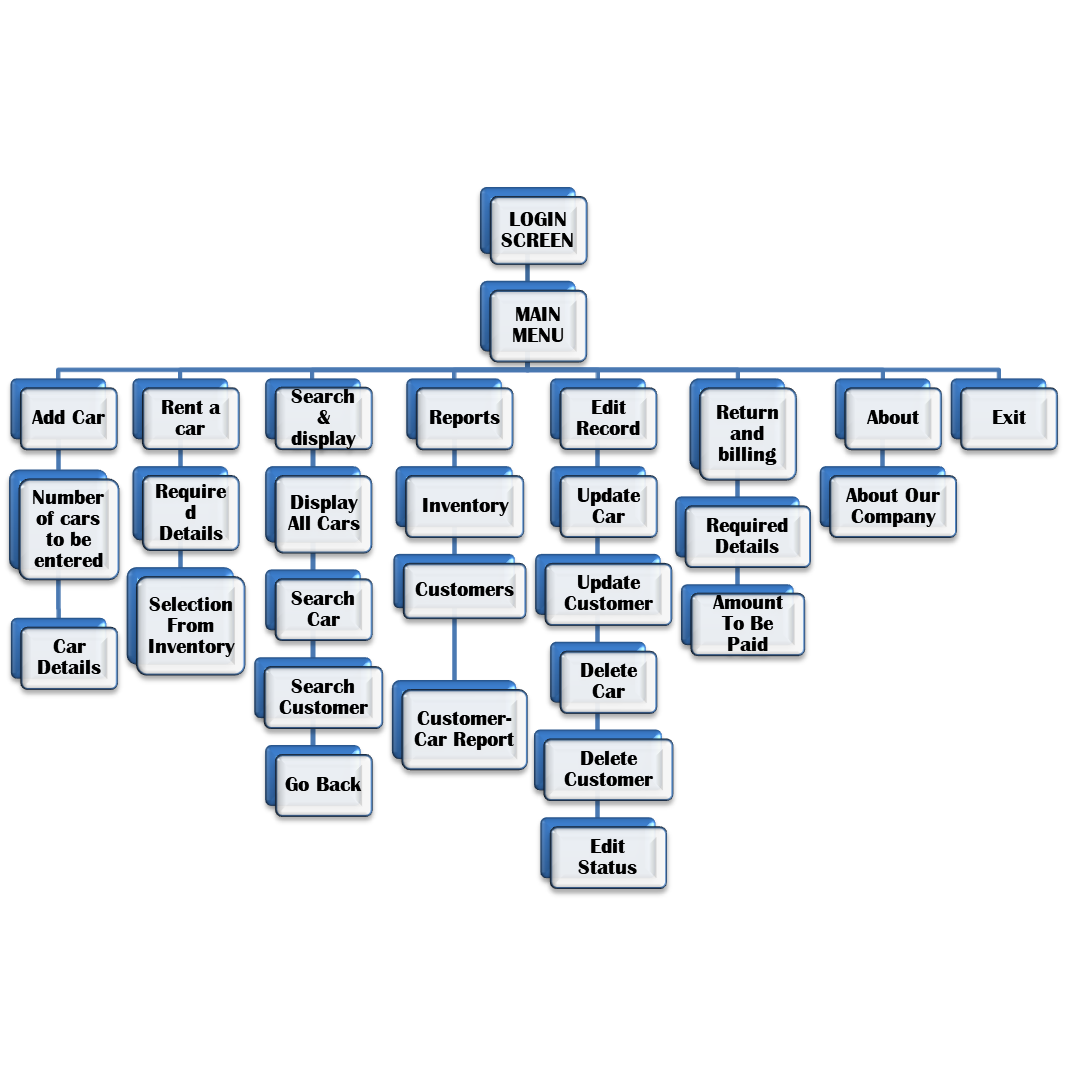
* 1. namecmp():

For checking for a word in a string; Used in the program to find name inside fullname; If string size less than word size function returns (-1);

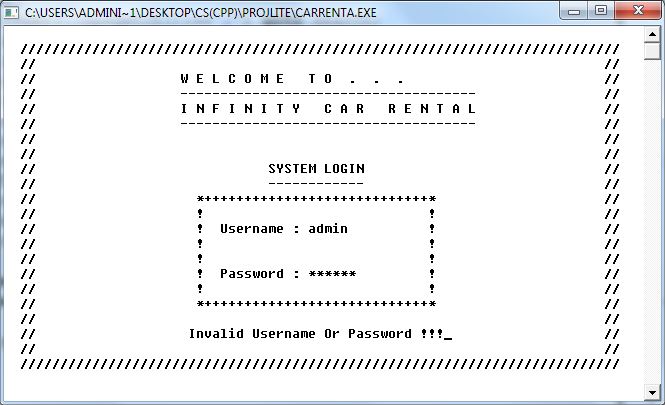
If the entire word is in the string, function returns (1) else returns (0).

Usage: namecmp(string, word).

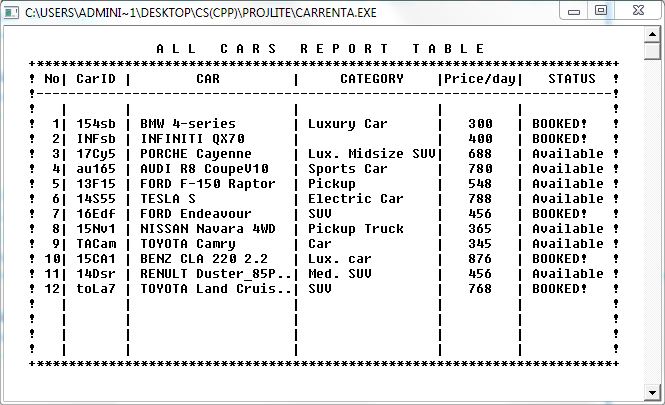
SYSTEM DESIGN

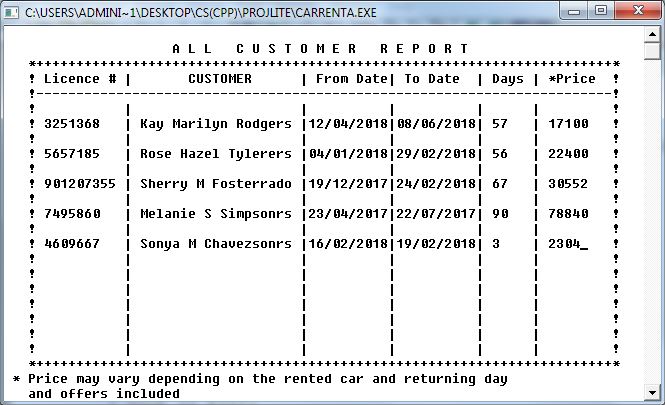


SCREENSHOTS

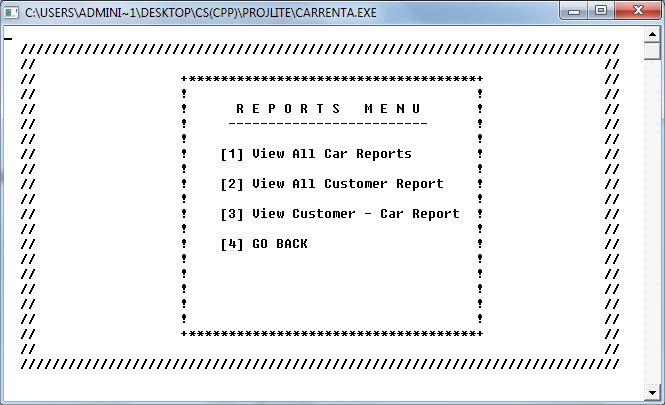
1. ***Login Screen***
2. ***Main Menu***

******

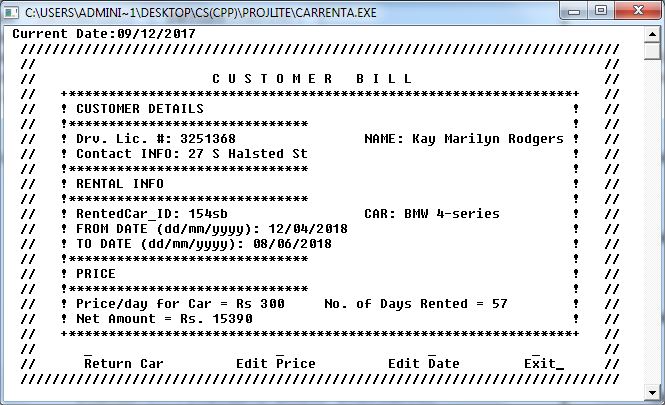
1. ***Car Report Table***
2. ***Customer Report***



1. ***Report Menu***

******

1. ***View Bill***

******

Bibliography

* C++ Text Book
* Previous year’s projects
* <http://cppqa.blogspot.com>
* https://en.wikipedia.org
* <https://stackoverflow.com>
* <https://github.com>
* http://hilite.me