COP3014 Programming Project # 4

Project Outcomes:

- To develop and compile several interactive C programs.
- To implement algorithms that can solve problems involving several functions and arrays.
- To use structure data types and control structures in programming.
- To implement algorithms that can solve problems involving strings.
- To use functions and pass parameters by reference.
- To write simple user interfaces.

Prep Readings: Class textbook, chapter 1-10.

Project Requirements:

Your job is to write a C program that implements a simple car rental information system. Your program keeps track of the cars of a rental company and rental information. The program provides a simple user interface to add cars to the company's inventory, to make a car reservation, find a reservation, print rented and non-rented vehicles, and compute average number of days rented. Your program must use the data types and variables described below.

A type definition of a new struct *CarT* to store car information. This struct must have the following fields:

```
int       carId;          // a unique ID of a car
char      make[20];       // make of the car
char      model[20];      // model of the car
int       numDoors;       // 2 or 4 door cars
double    rate;           // the rental rate of the car
```

A type definition of a new struct *RentalT* to store rental information of a renter:

```
char      renterName[20];     // name of renter
int       daysRenting;        // days renting the car
int       carId;              // ID of the car rented
```

Two arrays in `main()` that store car information and rental information:

```
CarT      allCars[20];        // a list of cars
RentalT   allRentals[60];     // a list of rentals
```

Note that there can be more rentals than cars this is because we may consider allRentals to store records about cars currently rented and those that have been reserved. The program keeps track of the total number of cars owned by the company and the total number of rentals by the following two variables declared in `main()`:

```
int       totalCars;          // number of all cars owned
int       totalRentals;       // number of rented cars
```

Write a main program that declares the above specified variables and any other variables you may need. The main program performs the following operations:

1. It calls a function `int createInventory ( CarT *allCars )` to add a total of 3 cars to the inventory with the following information:

Car ID	Make	Model	#doors	rate
1234	VW	Golf	2	66.00
2241	Ford	Focus	4	45.00
3445	BMW	X3	4	128.00

   The new cars must be stored in the first three array positions of the array `allCars`. The function returns the number of cars that has been added successfully, which should be 3. This value can be used to initialize the local variable `totalCars`. **(5 points)**

2. Then the program prints a menu to the screen and repeatedly prompts the user to choose from several options shown below: **(5 points)**

1	Add new car to the inventory.
2	Make a reservation.
3	Find a reservation using a renter name and print it to the screen.
4	Print all rental information to the screen.
5	Print car information to the screen of a selected car using the make of the car to search the inventory.
6	Calculate and print average number of days rented.
7	Exit program.

   Except for exiting the program, each option is performed by a function. Your program must call the appropriate function. Use a `switch` statement to handle the user input and make the function calls and updates as necessary. The function header and definition of the functions are given below.

```
int addNewCar ( CarT *allCars, int totalCars, int size );
```

   This function must prompt the user for the fields to initialize the values in the structure composing a car. It must use the value of `totalCars` to select the array location for the new car and set the corresponding fields of the `CarT` struct. It must use `size` to check if there is a space in the array. Given the above definition of the array, there can be at most 20 cars in the company's inventory. The function returns the new number of total cars in the inventory. The returned value must be used to update the `totalCars` variable in `main()`. **(15 points)**

```
int addNewRental ( CarT *allCars,
                   int totalCars,
                   RentalT *allRentals,
                   int totalRentals,
                   int size );
```

This function must prompt the user for the fields to initialize the values in the structure composing a rental reservation. It must use the value of `totalRentals` to select the array location for the new rental reservation and set the corresponding fields of the `RentalT` struct. It must use `size` to check if there is a new space in the array. Given the above definition of the array, there can be at most 60 cars rented. The function returns the new number of total car rentals. The returned value must be used to update the `totalRentals` variable in `main()`. For this function, you must implement and use another function that identifies the `carId` of the car matching a given car name:

```
int findCarIDByName ( CarT *allCars,
                      int totalCars,
                      char *carMake );
```

This function searches through the list of cars in the inventory and finds the car matching the given car name as indicated by the make of the car. The function then returns the ID of the car. If the car cannot be located -1 must be returned. **(25 points)**

```
int findReservation ( RentalT *allRentals,
                      int totalRentals,
                      char *renterName );
```

This function locates a reservation in the array that matches the specified `renterName` in the `RentalT` struct. It returns the index of the field in the array that stores the matched rental information. The information can then be used to print the fields of the rental information. Instead of printing a `carID` from the rental information, you must locate the car information in the car array using the `carID` and print the car information to the screen. This can be achieved by calling the following function:

```
int findCarById ( CarT *allCars,
                  int totalCars,
                  int carId );
```

This function searches through the list of cars in the inventory and finds the car matching the given car ID. The function then returns the index of the matched car in the array `allCars`. If the car cannot be located in the inventory -1 must be returned. With the index of the field storing a `CarT` struct, the car information can be printed. **(25 points)**

```
void printAllRentals  (RentalT *allRentals, int totalRentals,
                       CarT *allCars, int totalCars );
```

This function prints all car rental information to the screen. Instead of the field `carID`, the car information must be printed using the above function to locate the index of the car matching the `carID`. The variable `allRentals` is the array of rental information, `totalRentals` is the total number of car rentals, `allCars` is the array of car information, and `totalCars` is the number of cars in the inventory.

```
void printCarInfo ( CarT *allCars,
                    int totalCars,
                    int carId );
```

This function prints the car information to the screen matching the specified `carId`. The inventory must be searched for cars matching the given Id to print all the fields in the `CarT` struct. If there is no car matching Id, then a message of "Car not found" must be printed. **(15 points)**

```
double getAverageRentalDays  (RentalT *allRentals,
                              int totalRentals );
```

This function computes the average number of days cars have been rented considering all rentals. The function examines all rental records in the array to compute the average number of days. The function returns the computed value. If no cars have been rented a value of -1 must be returned. **(10 points)**

2. Write a C program *pattern.c* that includes a main program and a function with the following function prototype:

```
int contains ( const char *text, const char *pattern );
```

The main program must test the function by calling it with 5 different values for `text` and `pattern` and printing the results to the screen using the following format string:

```
"Pattern %s occurs in %s at the location %d.\n"
```

The function accepts two text strings, `text` and `pattern` as input. It then determines the location of the specified pattern string in the text string. If the pattern occurs multiple times, it returns the first occurrence of the pattern and if it does not occur, then the function returns -1. For example, if called with the text “Hello, World!” and the pattern “orl”, the function returns 8 because the pattern starts at position 8 in the text string. **(25 points)**

3. Write a C program *coding.c* that includes a main program and two functions with the following function prototypes:

```
void encodeStr ( char *input, char *output, CodeT code );

void decodeStr ( char *input, char *output, CodeT code );
```

The first function takes the input string in `input` and generates the encoded string using the information in `code`. The result is stored in `output`. The second function takes the input string in `input` and decodes it back using the information in `code`. The result is stored in `output` again.

Both functions use the following structure as the key to encode and decode text messages:

```
typedef struct {
   char from[28];
   char to[28];
} CodeT;
```

The main function may then initialize the values as shown below:

```
CodeT code = {    .from = " abcdefghijklmnopqrstuvwxyz",
                  .to = ".172093%@#+!:_-$^*()854=6?>" };
```

The main program must prompt the user repeatedly for a line of text and then generate and print the encoded text string before decoding the encoded text string and printing the result again to the screen to test that the original text string can be retrieved through the decoding function. The program should stop prompting the user for input text when the user enters an empty text as input. **(25 points)**

Submission Requirement:

The project requires 4 source code files to be turned in.

a. `carRental.c` (the source code for solving all problems in 1)
b. `pattern.c` (the source code for solving problem2)
c. `coding.c` (the source code for solving problem 3)

Be sure to follow the Project Submission Instructions posted by your instructor in *eLearning* under Course Materials. The submission requirements are part of the grading for this assignment. If you do not follow the requirement, 5 points will be deducted from your project grade.

Important Notes:

1. Projects will be graded on whether they correctly solve the problem and whether they adhere to good programming practices.
2. Projects must be submitted by the time specified on the due date. Projects submitted after that time will get a grade of zero.
3. Please review UWF's academic conduct policy that was described in the syllabus. Note that viewing another student's solution, whether in whole or in part, is considered academic dishonesty. Also note that submitting code obtained through the Internet or other sources, whether in whole or in part, is considered academic dishonesty. All

programs submitted will be reviewed for evidence of academic dishonesty, and all violations will be handled accordingly.