## horizontal line

Car & Bike Rental System

# Abstract Data Types

This document contains detailed description and evaluation of the data structures and algorithms to be used in the project.

Welcome to “ABC” car and bike rental system.

1. Rent a car/bike
2. Check a rental information
3. Print records (admins only)
4. Delete a record (admins only)
5. Quit

Enter your selection: \_

Ankhbayar Enkhlkhagva COS 221a Fundamental Data Structures

ID: 100132497 Spring 2020

Mail address: ANE171@aubg.edu Instructor: Professor Vladimir Georgiev

# Class Description

This project has 7 classes shown below.

# Class Rental

**class Rental {** //Parent class(abstract)

**public**:

**Rental**(){}

**void** setCustomerName() {}

**void** setPhoneNumber() {}

**void** setInvoiceID() {}

**virtual void** printIt() {

cout << “Invoice ID: “ << this->invoiceID;

cout << "\nCustomer Name: " << this->customerName;

cout << "\nPhone Number: " << this->phoneNumber << endl;

}

**virtual double** getTotalPrice() = 0;

};

# Class Car

**class** Car : **public** Rental {

**public**:

**Car**(){

setCarModel();

setPower();

setFuelConsumption();

setDiscount();

}

**Car**(string customerName, string phoneNumber, string invoidID, string carModel, string power, string fuelConsumption, double pricePerDay, double discount) : Rental(customerName, phoneNumber, invoiceID),

carModel(carModel),

power(power),

fuelConsumption(fuelConsumption),

pricePerDay(pricePerDay),

discount(discount){}

**void** setCarModel(){}

**void** setPower(){}

**void** setPricePerDay(){}

**void** setDiscount(){}

**void** printIt () {

Rental:: printIt ();

cout << "Car Model: " << this->carModel;

cout << "\nPower: " << this->power;

cout << "\nFuel Consumption: " << this->fuelConsumption;

cout << "\nPrice Per Day: " << this->pricePerDay;

cout << "\nDiscount: " << this->discount << endl;

}

**private**:

**string** carModel, power, fuelConsumption, pricePerDay;

**double** discount;

};

# Class Bike

**class** Bike : **public** Rental {

**public**:

**Bike**(){

setBikeModel();

setFrameMaterial();

setWheelSize();

setForkTravel();

setPricePerDay();

setDiscount();

setNumberOfGear();

}

**Bike**(string customerName, string phoneNumber, string invoiceID, string bikeModel, string frameMaterial, double wheelSize, double forkTravel, double pricePerDay, double discount, int numberOfGears) : Rental(customerName, phoneNumber, invoiceID),

bikeModel(bikeModel),

frameMaterial(frameMaterial),

wheelSize(wheelSize),

forkTravel(forkTravel),

pricePerDay(pricePerDay),

discount(discount),

numberOfGears(numberOfGears){}

**void** setBikeModel();

**void** setFrameMaterial();

**void** setWheelSize();

**void** setForkTravel();

**void** setPricePerDay();

**void** setDiscount();

**void** setNumberOfGear();

**void** printIt() {

Rental:: printIt ();

cout << "Bike Model: " << this->bikeModel;

cout << "\nFrame Material: " << this->frameMaterial;

cout << "\nWheel Size: " << this->wheelSize;

cout << "\nFork Travel: " << this->forkTravel;

cout << "\nPrice per Day: " << this->pricePerDay;

cout << "\nDiscount: " << this->discount;

cout << "\nNumber of Gears: " << this->numberOfGears << endl;

}

**private**:

**string** bikeModel, frameMaterial;

**double** wheelSize, forkTravel, pricePerDay, discount;

**int** numberOfGears;

};

# Class Diesel

**class** Diesel : **public** Car {

**public**:

**Diesel**(){

setAdditionalDiscount();}

**Diesel**(string carModel, string power, string fuelConsumption, double pricePerDay, double discount, double aD) : Car(carModel, power, fuelConsumption, pricePerDay, discount),

additionalDiscount(aD){}

**void** setAdditionalDiscount(){}

**double** getTotalPrice(){}

**double** getAdditionalDiscount() { **return** additionalDiscount; }

**void** printIt() {

Car::printIt();

cout << " Additional discount: " << getAdditionalDiscount() << endl;

**private**:

**double** additionalDiscount;

};

# Class Petrol

**class** Petrol : **public** Car {

**public**:

**Petrol**();

**Petrol**(string carModel, string power, string fuelConsumption, double pricePerDay, double discount) : Car(carModel, power, fuelConsumption, pricePerDay, discount){}

**double** getTotalPrice(){}

**void** printIt() {

Car::printIt();

}

};

# Class TBike

**class** TBike : **public** Bike {

**public**:

**TBike**() {

setExtraDay();

}

**TBike**(string bikeModel, string frameMaterial, double wheelSize, double forkTravel, double pricePerDay, double discount, int numberOfGears, int eD) : Bike(bikeModel, frameMaterial, wheelSize, forkTravel, pricePerDay, discount, numberOfGears),

extraDay(eD){}

**void** setExtraDay() {}

**double** getTotalPrice(){}

**string** getExtraDay(){**return** this->extraDay;}

**void** printIt() {

Bike::printIt();

cout << “Extra Day: “ << getExtraDay() << endl;

}

**private**:

**string** extraDay;

};

# Class EBike

**class** EBike : **public** Bike {

**public**:

**EBike**() {

setMotorBrand();

setMaximumRange();

}

**EBike**(string bikeModel, string frameMaterial, double wheelSize, double forkTravel, double pricePerDay, double discount, int numberOfGears, string mB, double maxR) : Bike(bikeModel, frameMaterial, wheelSize, forkTravel, pricePerDay, discount, numberOfGears),

motorBrand(mB),

maximumRange(maxR){}

**void** setMotorBrand(){}

**void** setMaximumRange(){}

**string** getMotorBrand(){**return** this->motorBrand;}

**string** getMaximumRange(){**return** this->maximumRange;}

**double** getTotalPrice(){}

**void** printIt() {

Bike::printIt();

cout << “ Motor Brand: ” << getMotorBrand();

cout << “Maximum Range: “ << getMaximumRange();

}

**private**:

string motorBrand, maximumRange;

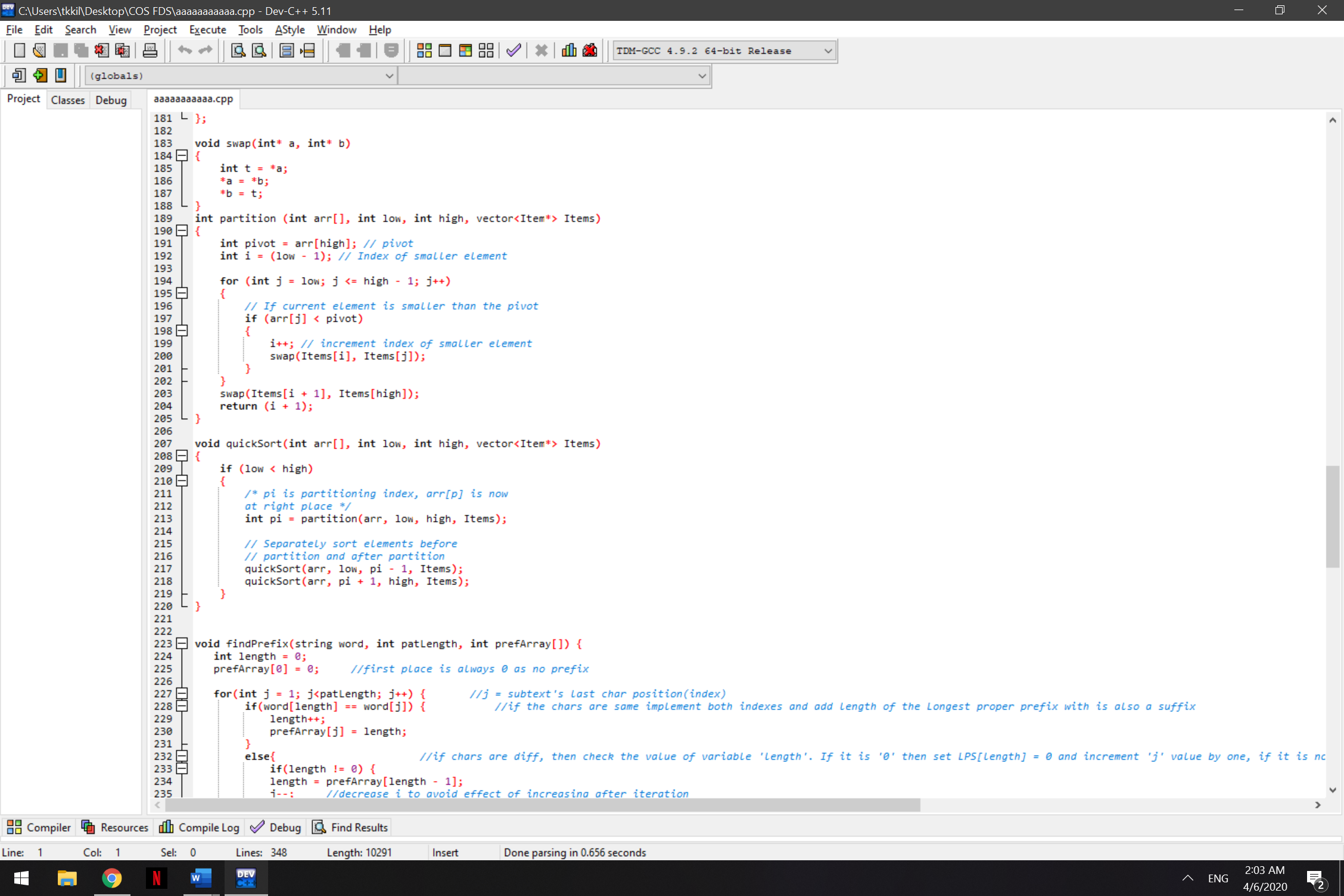
};

# Important Axioms

All total prices, calculated by the function *getTotalPrice(),* **must be** positive number.

# Algorithm description

In this project, I will be using Quick sort and KMP pattern searching algorithms.

1. Quick sort – Users will be able to see the list of the collections sorted(descending) by the Price Per Day. Admins will be able to see the list of the collections sorted by the total rental prices.
2. KMP pattern searching - The program uses KMP algorithm to search matching IDs. If there is a record matching to the ID, the program will print the user’s Invoice letter.

