Q1:

```
def countCharacters(arr)
  if (arr.class != Array)
    return "The argument is not an array."
  end
  arr2 = arr.sort
  arr2.each {|ele| puts "#{ele}, ch_count= #{ele.length}"}
end

arr = ["Adam", "Eve", "Mark", "Franklin", "John"]
countCharacters(arr)
```

Q2:

```
# This method counts the characters including punctuation.
# If you need a method counting the characters excluding punctuation, remove
the "#" in line 27,
# "wordsArray = line.split#(/\W/)" becomes "wordsArray = line.split(/\W/)"
def calcARI(fileName)
  if (fileName.class != String)
    return "The argument is not a valid file name."
  end
  level = ["5-6 (Kindergarten)",
           "6-7 (First/Second Grade)",
           "7-9 (Third Grade)",
           "9-10 (Fourth Grade)",
           "10-11 (Fifth Grade)",
           "11-12 (Sixth Grade)",
           "12-13 (Seventh Grade)",
           "13-14 (Eighth Grade)",
           "14-15 (Ninth Grade)",
           "15-16 (Tenth Grade)",
           "16-17 (Eleventh Grade)",
           "17-18 (Twelfth grade)",
           "18-24 (College student)",
           "24+ (Professor)"]
  file = File.open(fileName)
  characters = 0
  words = 0
  sentences = 0
  file.each do |line|
    wordsArray = line.split#(/\W/)
    wordsArray.each do |word|
      characters += word.length
    end
    words += line.split.size
    sentences += line.split(/\.\s+/).size
  end
  ari = 4.71 * characters / words + 0.5 * words / sentences - 21.43
  puts "Total # of characters: #{characters}"
  puts "Total # of words: #{words}"
  puts "Total # of sentences: #{sentences}"
  printf("Automated Readability Index: %.1f\n", ari)
  score = ari.floor
  puts "Grade level: #{level[score-1]}"
```

```
end

calcARI("paragraph.txt")
```

Q3:

```
# maintain a Hash for the number of each car maker. for example {"Toyota" =>
2, "Mercedes" => 1}
class CarMaker
  attr_accessor :carCounter
  @@carCounter = Hash.new
  def self.carCounter
    @@carCounter
  end
end

# maintain all the cars with all 12 features as instance variable.
class CarModel < CarMaker
  attr_accessor :car_maker
  include Comparable

  #This class_variable is used for converting a feature from its String
expression to
  # instance variable name
  @@variableNameMap = {"#km"=>"@km", "Type"=>"@type",
"Transmission"=>"@transmission",
                       "stock#"=>"@stock",
"Drivetrain"=>"@drivetrain","Status"=>"@status",
                       "Fuel Economy"=>"@fuel", "car_maker"=>"@car_maker",
"Year"=>"@year",
                       "Trim"=>"@trim",
"set_of_features"=>"@set_of_features","Model"=>"@model"}
  #car is a hash that has all the information to create a CarModel Object
  def initialize(car)
    #first, update the number of objects
    value = @@carCounter[car["car_maker"]].to_i
    @@carCounter[car["car_maker"]] = value + 1
    #use @@variableNameMap to assign all the instance variable dynamically
    car.each { |feature, value|
instance_variable_set(@@variableNameMap[feature], value)}
  end

  def to_s
    return
"#{@car_maker},#{@model},#{@trim},#{@km},#{@year},#{@type},#{@drivetrain},"+
        "#{@transmission},#{@stock},#{@status},#{@fuel},#{@set_of_features}"
  end
end

$catalogue = Array.new # to save all the Car objects in an array, which is a
global variable

# take a String fileName, extract all the information from the file, and
store all the
# cars in an array, where each element is a car, represented by a Hash
```

```ruby
def convertListings2Array(fileName)
  File.write(fileName, File.read(fileName).gsub(/\n+/,"\n"))
  vehicles = Array.new
  file = File.open(fileName)
  file.each do |line|
      car = convertLine2Car(line) unless line.chomp.empty?
      vehicles.push(car)
  end
  return vehicles
end

# take a fileName and save all the information in an array of CarModel
Objects
def convertListings2Catalogue(fileName)
  vehicles = convertListings2Array(fileName)
  vehicles.each do |car|
    car = CarModel.new(car)
    $catalogue.push(car)
    puts "We are creating car:\n#{car}\n\n"
  end
end

# take a line of type String, use regex to convert it to a Hash, called car.
Return this
# Hash as a single car
def convertLine2Car(line)
  car = Hash.new
  if(line.class != String)
    abort("ABORTED! ")
  end
  if(line == "")
    return
  end
  line.scan(/{.+}|[^,\s]+/) do |feature|
    case feature
    when /^[^\/]+km$/
      car["#km"] = feature
    when /^Sedan$|^coupe$|^hatchback$|^station$|^SUV$/i
      car["Type"] = feature
    when /^Auto$|^manual$|^steptronic$/i
      car["Transmission"] = feature
    when /^(?!\d+$)(?![a-zA-Z]+$)\w+(?<!km)$/i
      car["stock#"] = feature
    when /^FWD$|^RWD$|^AWD$/i
      car["Drivetrain"] = feature
    when /^Used$|^new$/
      car["Status"] = feature
    when /L\/\d+km$/
      car["Fuel Economy"] = feature
    when /^Honda$|^Toyota$|^Mercedes$|^BMW$|^Lexus$/i
      car["car_maker"] = feature
    when /^\d{4}$/
      car["Year"] = feature
    when /^[A-Z]{2}$/
      car["Trim"] = feature
    when /^{.*}$/
      car["set_of_features"] = feature
    else
      car["Model"] = feature
    end
  end
  return car
end

# Parameter: hash: searching criteria
# iterate all cars in the inventory, and compare with the hash criteria. If
it matches,
# print this car
def searchInventory(hash)
  puts "Search inventory using hash #{hash}, the result is:\n"
  $catalogue.each do |car|
    match = true
    hash.each_pair do |key, value|
      variableName = CarModel.class_variable_get(:@@variableNameMap)[key]
      myValue = car.instance_variable_get(variableName)
      if (myValue != value)
        match = false
      end
    end
    if (match == true)
      puts car.to_s
    end
  end
end

# take a new feature line as type of String, add to the existing file.
extract information and
# add to inventory,$catalogue
def add2Inventory(features, fileName)
  puts "\nAdd a new listing to the inventory:\n#{features}\n"
  file = File.open(fileName, "a")
  file.print("\n#{features}")
  car = convertLine2Car(features)
  $catalogue.push(CarModel.new(car))
end

def displayInventory
  puts "\nDisplaying all the cars in our inventory: \n"
  $catalogue.each do |car|
    puts car
  end
end

# Sort all the cars in $catalogue, then output to a new file, "output.txt"
def saveCatalogue2File
  $catalogue.sort! { |a, b|  a.car_maker <=> b.car_maker }
  file = File.open("output.txt", 'w')
  $catalogue.each do |car|
    file.puts(car)
  end
  puts "\nWe have created a new file output.txt and printed all the cars in
order.\n"
end
```

```
# save all the cars in $catalogue array
convertListings2Catalogue("listing.txt")

# pass a hash and search in $catalogue. If match, print it in console
searchInventory({"car_maker" => "Toyota", "Year"=>"2010"})

# a new line of features
features = "SUV,900km,auto,RWD, Toyota,CLK,LX ,1234A4A,2010,{AC, Heated
Seats,"+
            "Heated Mirrors, Keyless Entry, Power seats},6L/100km,Used"
# add this feature in the inventory
add2Inventory(features, "listing.txt")

# display all the cars in inventory.
displayInventory

# Sort and output to "output.txt"
saveCatalogue2File

# check how many cars of each Car maker in the inventory
puts "\nCheck how many cars we have:\n #{CarModel.carCounter}"
```

Q4:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int** matrixTranspose (int **matrix, int r, int c);
int printMatrix(int **matrix, int row, int col);

int main() {

//    Generate a new 2D array.
    srand((unsigned)time(NULL));
    int row, col;
    printf("Enter the number of rows: ");
    scanf("%d",&row);
    printf("Enter the number of columns: ");
    scanf("%d",&col);
    int ** b = (int**)malloc(row * sizeof(int*));
    for(int i =0 ; i < row;i++)
        b[i] = (int*)malloc(col * sizeof(int));
    for(int i = 0; i < row; i++)
        for(int j = 0; j < col; j++)
            b[i][j] = rand() % 10;

//    print the array
    printf("Randomly generated Two Dimensional array:\n");
    printMatrix(b, row, col);

//    Transpose
    int ** newMatrix = matrixTranspose(b, row, col);

//    print it again
```

```c
    printf("\nTranspose:\n");
    printMatrix(newMatrix, col, row);

    return 0;
}

int printMatrix(int **matrix, int row, int col){
    for(int i = 0; i < row; i++) {
        for(int j = 0; j < col; j++) {
            printf("%d ", matrix[i][j]);
            if(j == col - 1){
                printf("\n");
            }
        }
    }
    return 0;
}

int** matrixTranspose (int **matrix , int r, int c){
    if (r == c){
        for(int i=0; i<r; i++) {
            for(int j=i+1;j<c;j++) {
                int temp = matrix[i][j];
                matrix[i][j] = matrix[j][i];
                matrix[j][i] = temp;
            }
        }
    } else {
        int **newMatrix = (int**)malloc(c * sizeof(int*));
        for(int i =0 ; i < c;i++)
            newMatrix[i] = (int*)malloc(r * sizeof(int));
        for(int i = 0; i < r; i++) {
            for(int j = 0;j<c;j++) {
                newMatrix[j][i] = matrix[i][j];
            }
        }
        return newMatrix;
    }
    return matrix;
}
```

Q5:

```c
//IMPORTANT NOTES: I use Clion IDE, so the output.txt file
//                  must be placed in the cmake-build-debug folder
//The version of c compiler is c99
//Add2Inventory is passing a fix String now. The String in line 61 can be
modified for testing

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
struct carMaker {
    char car_maker[30];
    struct carMaker *next;
    struct carModel *blow;
```

```c
};
struct carModel {
    char km[30];
    char type[30];
    char transmission[30];
    char stock[30];
    char drivetrain[10];
    char status[10];
    char fuel[30];
    char car_maker[30];
    char car_model[30];
    char year[10];
    char trim[10];
    char set_of_features[200];
    struct carModel *next;
};
struct carMaker *makerHead = NULL;
struct carModel * populate(char *line, struct carModel *car);
int numberOfCars = 0;

int searchInventory(char *searchMaker);
int Add2Structure(char *line);
int Add2Inventory(char *line);
int write2File(struct carModel *carArray);
int saveCatalogue2File();
int carCompare(const void *c1, const void *c2);
int main() {
    //IMPORTANT NOTES: I use Clion IDE, so the output.txt file
    //                  must be placed in the cmake-build-debug folder
    FILE *stream;
    char line[500];
    stream = fopen("output.txt", "r" );
    while (fgets(line, sizeof(line), stream)) {
        if (strcmp(line, "\n")){
            numberOfCars++;
            printf("We are creating car #%d: \n%s\n", numberOfCars, line);
            Add2Structure(line);
        }
    }
    fclose( stream );

    //SearchInventory Test
    char maker[20];
    printf("\nEnter the car maker name you want to search: ");
    scanf("%s", maker);
    searchInventory(maker);

    //Add2Inventory Test
    char newLine[] = "Mercedes,GLK,LX, 888km,2018,coupe, RWD, auto,
18FO724A,Used,6L/100km,{AC, Heated Seats, Heated Mirrors, Keyless Entry,
Power seats}";
    Add2Inventory(newLine);

    //SearchInventory again
    printf("Search the same car maker again: ");
    scanf("%s", maker);
    searchInventory(maker);
    //saveCatalogue2File Test, print all the cars to the file ascending
    saveCatalogue2File();

}

struct carModel * populate(char *line, struct carModel *car){
    char *feature;
    int i = 0;
    while (line[i] != '{'){
        i++;
    }
    feature = &line[i];
    strcpy(car->set_of_features, feature);
    char *token = NULL;
    const char s[3] = ", ";
    token = strtok(line , ", ");
    strcpy(car->car_maker, token);

    token = strtok(NULL, s);
    strcpy(car->car_model, token);

    token = strtok(NULL, s);
    strcpy(car->trim, token);

    token = strtok(NULL, s);
    strcpy(car->km, token);

    token = strtok(NULL, s);
    strcpy(car->year, token);

    token = strtok(NULL, s);
    strcpy(car->type, token);

    token = strtok(NULL, s);
    strcpy(car->drivetrain, token);

    token = strtok(NULL, s);
    strcpy(car->transmission, token);

    token = strtok(NULL, s);
    strcpy(car->stock, token);

    token = strtok(NULL, s);
    strcpy(car->status, token);

    token = strtok(NULL, s);
    strcpy(car->fuel, token);

    car->next = NULL;
    return car;
}

int searchInventory(char *searchMaker){
    struct carMaker *current;
    current = makerHead;
    while (current != NULL && strcmp(searchMaker, current->car_maker) != 0){
        current = current->next;
    }
    if (current){
        struct carModel *car;
        car = current->blow;
        if (car == NULL){
            printf("Error: Out of memory.\n");
            return 1;
        }
        printf("Searching result is:\n");
        while (car){
            printf("%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s\n",
                   car->car_maker, car->car_model, car->trim, car->km,
car->year,car->type,
                   car->drivetrain, car->transmission, car->stock,
car->status, car->fuel,
                   car->set_of_features);
            car = car->next;
        }
    } else {
        printf("%s", "This car maker does not exist in our inventory");
    }
    return 0;
}

//Accepts a new listing as a single line of ORDERED listing features
int Add2Structure(char *line){
    struct carModel *newCar;
    newCar = malloc(sizeof(struct carModel));
    newCar = populate(line, newCar);
    struct carMaker *current;
    current = makerHead;
    while (current != NULL && strcmp(current->car_maker,
newCar->car_maker) != 0){
        current = current->next;
    }
    if (current == NULL){
        struct carMaker *newMaker;
        newMaker = malloc(sizeof(struct carMaker));
        newMaker->next = makerHead;
        strcpy(newMaker->car_maker, newCar->car_maker);
        makerHead = newMaker;
        newMaker->blow = newCar;
    } else {
        newCar->next = current->blow;
        current->blow = newCar;
    }
    return 0;
}

int Add2Inventory(char *line){
    FILE *stream;
    stream = fopen("output.txt", "a" );
    if( stream == NULL ){
        printf( "The file output.txt was not opened\n" );
        return 1;
    }
    fprintf(stream, "\n%s", line);
    printf("We are adding this car to the inventory:\n%s\n\n",line);
    Add2Structure(line);
    numberOfCars++;
    fclose( stream );

    return 0;
}

int saveCatalogue2File(){
    struct carModel carArray[20];
    struct carMaker *current;
    current = makerHead;
    int i = 0;
    struct carModel *carCurrent;
    while (current){
        carCurrent = current->blow;
        while (carCurrent){
            strcpy(carArray[i].car_maker, carCurrent->car_maker);
            strcpy(carArray[i].car_model, carCurrent->car_model);
            strcpy(carArray[i].trim, carCurrent->trim);
            strcpy(carArray[i].km, carCurrent->km);
            strcpy(carArray[i].year, carCurrent->year);
            strcpy(carArray[i].type, carCurrent->type);
            strcpy(carArray[i].drivetrain, carCurrent->drivetrain);
            strcpy(carArray[i].transmission, carCurrent->transmission);
            strcpy(carArray[i].stock, carCurrent->stock);
            strcpy(carArray[i].status, carCurrent->status);
            strcpy(carArray[i].fuel, carCurrent->fuel);
            strcpy(carArray[i].set_of_features, carCurrent->set_of_features);
            carArray[i].next = NULL;
            i++;
            carCurrent = carCurrent->next;
        }
        current = current->next;
    }
    qsort(&carArray, (size_t) numberOfCars, sizeof(struct carModel),
carCompare);
    write2File(carArray);
    return 0;
}

int write2File(struct carModel *carArray){
    FILE *stream;
    stream = fopen("output2.txt", "w" );
    if( stream == NULL ){
        printf( "The file output2.txt was not opened\n" );
        exit(1);
    }
    printf("\nWe are writing cars to file...\n\n");
        for (int i = 0; i < numberOfCars; ++i) {
            fprintf(stream,
"%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s\n",
                    carArray[i].car_maker,
carArray[i].car_model,carArray[i].trim,
                    carArray[i].km, carArray[i].year,carArray[i].type,
                    carArray[i].drivetrain, carArray[i].transmission,
carArray[i].stock,
                    carArray[i].status, carArray[i].fuel,
                    carArray[i].set_of_features);
        }
    printf("Finish! All the cars have been printed in a new file
output2.txt.");
    fclose( stream );

    return 0;
}

int carCompare(const void *c1, const void *c2) {
    char *l = ((struct carModel *)c1)->car_maker;
    char *r = ((struct carModel *)c2)->car_maker;
    if (strcmp(l, r) == 0){
        l = ((struct carModel *)c1)->car_model;
        r = ((struct carModel *)c2)->car_maker;
        return strcmp(l, r);
    }
    return strcmp(l, r);
}
```