Student ID	:
Student Name	:

Total	*⇨ 100*

1	2	3	4
21	*34*	*15*	*30*

Marmara University – Faculty of Engineering

Computer Engineering

Spring 2020
CSE1142 Computer Programming II

Online Homework #2

25 June 2020

I hereby swear that the work done on this online homework is totally my own; and on my honor, I have neither given nor received any unauthorized and/or inappropriate assistance for this homework. I understand that by the school code, violation of these principles will lead to a zero grade and is subject to harsh discipline issues.

Full Name: *Signature:*

Solution Notes:

1) Textbooks, slides and notes are closed.
2) Internet usage and cell phones are not allowed. You will only use your computer to create *.java* and *.c* files for questions 1 and 2.
3) This online homework consists of 4 questions and 9 pages. Solve questions 1 and 2 using computer, create *.java* and *.c* files. However, you are not allowed to use computer for questions 3 and 4, solve these on paper.
4) Please write your answers clearly and neatly to the appropriate place.
5) **Show all your work.**

Checlist and Submission Notes:

6) Write your student ID, first name and last name on top of each solution page.
7) If you have not printed this PDF file (9 pages), write the sentences above in the frame on top of the first page and the date in your handwriting, and sign it.
8) If you have not printed this PDF file (9 pages), solve related questions in your handwriting on blank A4 sheets.
9) Solve the questions in the order given in this document.
10) Scan the solution pages for questions 3-4 to a single PDF file. **Your scans should be clear, readable, small in size, cropped and fixed, portrait oriented, and pages ordered.** Name your scanned file as "FirstName_LastName.pdf".
11) **Combine *.java* files (question 1), *.c* files (question 2), and *.pdf* file (questions 3-4) into a single *.zip* file. Name it as "studentID.zip".**
12) Upload your ZIP file to http://ues.marmara.edu.tr before deadline.

1) [21 pts] **JAVA PROGRAMMING**.

a) [2 pts] Implement an interface *Drivable* using the following UML diagram.

	<<*interface*>> **Drivable**
+	*howToDrive(): void*

- This interface represents the ability to be driven.
- The method *howToDrive* explains how a specific machine is driven.

b) [3 pts] Implement an abstract class Machine using the following UML diagram.

	Machine
-	Type: int
+	Machine(type: int)
+	getType(): int
+	*mainProperty(): void*
+	*howItWorks: void*

- The data field type represents the type of the seat of the machine.
- In the constructor, you will set type.
- The method getType is the getter for the field type.
- The other two methods are abstract methods.

c) [4 pts] Implement a class Car that extends the class Machine and implements the interface Drivable using the following UML diagram.

	Car
+	Car(type: int)

- In the constructor, you will call the super constructor.
- In the method *howToDrive*, you will print "Use my steering wheel to drive." on the screen.
- In the method *mainProperty*, you will print "I have 4 wheels." on the screen.
- In the method *howItWorks*, you will print "I need fuel to run." on the screen.

d) [4 pts] Implement a class Ship that extends the class Machine and implements the interface Drivable using the following UML diagram.

Ship	
+	`Ship(type: int)`

- In the constructor, you will call the super constructor.
- In the method *howToDrive*, you will print "Use my rudder." on the screen.
- In the method *mainProperty*, you will print "I have anchor." on the screen.
- In the method *howItWorks*, you will print "I need fuel to run." on the screen.

e) [3 pts] Implement a class Computer that extends the class Machine using the following UML diagram.

Computer	
+	`Computer(type: int)`

- In the constructor, you will call the super constructor.
- In the method *mainProperty*, you will print "I have CPU." on the screen.
- In the method *howItWork*, you will print "I need electricity to run." on the screen.

f) [5 pts] Write a Test class with the following specifications:

- Create an array Machines of size 5. Create 1 Car, 2 Computer and 2 Ship objects.
- The field type for Car, Computer and Ship objects are 1, 2 and 3, respectively.
- For each machine in the array, print machine type.
- For Drivable objects, call *howToDrive* method.
- For objects with type value 3, call the method *mainProperty*.
- For objects with type value 2, call the method *howItWorks*.

2) [34 pts] **C PROGRAMMING**. Implement a C program with the following specifications:

a) [1 pt] Define a constant *SIZE* with the value 10.

b) [1 pt] Define a global variable *totalNumberOfCars* and initialize it to 0.

c) [3 pts] Create a **struct** with name **carInformation**. This struct has to contain a char array of size 20, named *model*; an integer named *seriesNo*; a char array of size 20, named *brand*; an integer named *hp*; and a double *gasPerKm*. **typedef** it to **Car**.

d) [5 pts] **void addCar(Car *car)**: Create a function *addCar* that takes a Car pointer and returns void. In the function:

i) Ask the user for *model*, *seriesNo*, *brand*, *hp*, *gasPerKm* of the car and store them in the Car pointer.

ii) Increment *totalNumberOfCars* by 1.

e) [6 pts] **void changeCarInfo(Car car[])**: Create a function *changeCarInfo* that takes a Car array and returns void. In the function:

i) Ask the user *seriesNo* of the car whose information will be changed.

ii) Iterate through the Car array to find the desired car.

iii) Ask the user for *model*, *seriesNo*, *brand*, *hp*, *gasPerKm* of the car and update the old information.

iv) If no car found with the given *seriesNo*, inform the user.

f) [7 pts] **void searchWithModel(Car car[])**: Create a function *searchWithModel* that takes a Car array and returns void. In the function:

i) Ask the user *model* of the car to be searched.

ii) Iterate through the Car array to find the desired car.

iii) Print all information of the car.

iv) If no car found with the given *model*, inform the user.

g) [3 pts] **void printAll(Car car[])**: Create a function *printALL* that takes a Car array and returns void. In the function, print all information of all cars.

h) [7 pts] **void menu()**: Create a function menu that returns void. In the function:

i) Create a Car array with size *SIZE*.

ii) Print the following lines:

```
Choose an option:
1 - Add a car
2 - Modify car information
3 - Search a car with respect to model 4 - Print all cars
0 - Exit
```

iii) According to the choice entered, call the following functions:

(1) If choice is 1, check if there is room in the array. If so, call *addCar* function with address of the first empty slot of the Car array as the parameter. Otherwise, inform the user.

(2) If choice is 2, call *changeCarInfo* with Car array as the parameter.

(3) If choice is 3, call *searchWithModel* with Car array as the parameter.

(4) If choice is 4, call *printAll* with Car array as the parameter.

(5) If choice is 0, end the function.

i) [1 pt] In main function, call menu function.

SAMPLE OUTPUT:

```
Choose an option:
1 - Add a car
2 - Modify car information
3 - Search a car with respect to model
4 - Print all cars
0 - Exit
1
Enter car model: mxrcxdxs
Enter car series no: 123456
Enter car brand: mxrcxdxsbxnz
Enter car hp: 120
Enter car gasPerKm: 1.2
Choose an option:
1 - Add a car
2 - Modify car information
3 - Search a car with respect to model
4 - Print all cars
0 - Exit
1
Enter car model: fxcxs
Enter car series no: 654321
Enter car brand: fxrd
Enter car hp: 100
Enter car gasPerKm: 0.2
Choose an option:
1 - Add a car
2 - Modify car information
3 - Search a car with respect to model
4 - Print all cars
0 - Exit
4
Car model: mxrcxdxs
Car seriesNo: 123456
Car brand: mxrcxdxsbxnz
Car hp: 120
Car gasPerKm: 1.200000
Car model: fxcxs
Car seriesNo: 654321
Car brand: fxrd
Car hp: 100
Car gasPerKm: 0.200000 Choose an option:
```

```
1 - Add a car
2 - Modify car information
3 - Search a car with respect to model
4 - Print all cars
0 - Exit
2
Car seriesNo to change?: 6
No car with given seriesNo!
Choose an option:
1 - Add a car
2 - Modify car information
3 - Search a car with respect to model
4 - Print all cars
0 - Exit
2
Car seriesNo to change?: 654321
Enter car model: fxcxs
Enter car brand: fxrd
Enter car hp: 100
Enter car gasPerKm: 0.4
Choose an option:
1 - Add a car
2 - Modify car information
3 - Search a car with respect to model
4 - Print all cars
0 - Exit
4
Car model: mxrcxdxs
Car seriesNo: 123456
Car brand: mxrcxdxsbxnz
Car hp: 120
Car gasPerKm: 1.200000
Car model: fxcxs
Car seriesNo: 654321
Car brand: fxrd
Car hp: 100
Car gasPerKm: 0.400000 Choose an option:
1 - Add a car
2 - Modify car information
3 - Search a car with respect to model
4 - Print all cars
0 - Exit
3
Car model to search?: mxrcxdxs
Car seriesNo: 123456
Car brand: mxrcxdxsbxnz
Car hp: 120
Car gasPerKm: 1.200000
Choose an option:
1 - Add a car
2 - Modify car information
3 - Search a car with respect to model
4 - Print all cars
0 - Exit
0
```

3) [15 pts] Given sequence a_n is defined by the recurrence relation

$$a_n = a_{n-1} + a_{n-2} + a_{n-3}\ ,\quad n \geq 3\ ,\quad a_0 = 0,\quad a_1 = 1,\quad a_2 = 2.$$

a) [10 pts] Write a recursive C function that implements the algorithm of the question.

```
int func(int n) {











}
```

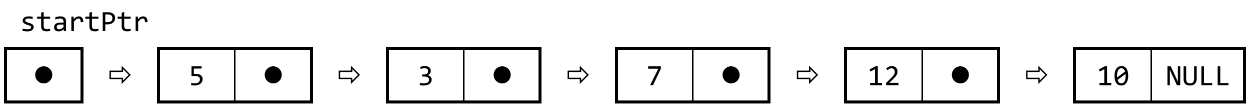
b) [5 pts] What would be the total number of calls made to this recursive function for n = 5? Justify your answer.

4) [30 pts] Assume that a linked list is made up of nodes defined as follows:

```
struct listNode{
  int data;
  struct listNode *next;
};

typedef struct listNode Node;
typedef Node *NodePtr;
```

Suppose that we have the following list:



a) [8 pts] Draw a diagram of the above list after the following lines of code have been executed:

```
NodePtr prev = startPtr->next;
NodePtr nodeToInsert = malloc(sizeof(Node));
nodeToInsert->data = 4;
nodeToInsert->next = prev->next;
prev->next = nodeToInsert;
```

b) [5 pts] Assume that the code represented above in part (a) has executed. What is the value of **`prev->data`**?

c) [7 pts] In addition to the code above, assume the following code executes. Draw a diagram of the list after this code executes as well.

```
prev = prev->next;
prev = prev->next;
Node* curr = prev->next;
prev->next = curr->next;
free(curr);
curr = NULL;
```

d) [10 pts] Write a function **`RemoveDuplicates()`** which takes a list sorted in increasing order and deletes any duplicate nodes from the list. The linked list is sorted such as 1, 3, 3, 3, 5, 7, 7, 8. Complete the missing parts.

```
void RemoveDuplicates(NodePtr start_ptr) {

   NodePtr current, nextNext;
   current = start_ptr;


   if(________________________________________) return;


   while(________________________________________) {

      if(current->data == current->next->data) {

         nextNext= ________________________________________;

         free(________________________________________);

         current->next = nextNext;

      }

      else {

         ________________________________________;

      }

   }

}
```