2017 MAD 3004\_2

Mid-Term Exam

Student:

C0676975

Pedro Fernandes

Part 1:

Q1 (10%)

Write a program that takes an integer (in a variable) and displays, using asterisks, a hollow inverted triangle of the given side length. For example, if the side length is 3, the program should display.

//Draw triangle method receives an integer which is its number of lines.

//Since it should be the number of empty lines, startLength starts with the number of lines + 1;

public func drawTriangle(lines: Int) {

let startLength = lines + 1;

var numStar = startLength

var numSpaces = 0

//Loop trough the number of lines.

for i in 1...startLength {

var str = ""

//Check if number of initial spaces is bigger than 0

if numSpaces != 0 {

//Concatenate empty spaces based on the number of spaces at the beginning

for \_ in 1...numSpaces {

str += " ";

}

}

//After spaces are added to the line, loop through the number of remaining stars

for j in 1...numStar {

//If is the first star or the last, then add the star to the line.

if j == 1 || j == numStar || i == 1 {

str += "\* "

} else {

//Since the middle should be empty, add the empty space instead of the stars in the middle

str += " "

}

}

//Descrease number of stars after every loop

numStar -= 1;

//Increase number of space after every loop

numSpaces += 1;

//Print the final line string

print(str);

}

}

drawTriangle(lines: 3)

print("")

Q2 (15%)

Write a program that takes an integer (in a variable) and displays, using asterisks, a filled and hollow square, placed next to each other. For example if the side length is 4, the program should display:

//DrawSquare accepts the number of empty line of the square.

public func drawSquare(length: Int) {

//Since the length is the number of empty lines, newLength adds one to know the real numbers of printed lines

let newLength = length + 1;

var str: String = ""

//Loop through the number of printed lines

for x in 1...newLength {

str = "";

//Populated the numbers of lines at the beginning of the square, before start the square logic.

for \_ in 1...newLength {

str += "\* "

}

str += " \*";

//After the initials stars are printed, start the square logic looping through the second position of the newLength, since the first one was already added to the string.

for y in 2...newLength {

//If the position is the first, or last, it should be a star, if not it should be emptys

if y == newLength || x == 1 || x == newLength {

str+=" \*"

}

else {

str+=" "

}

}

//Print the line of the square

print(str)

}

}

Q3 (25%)

Write and test a function that takes a String as a parameter and returns back a tuple that contains:

1. The total number of characters in this string

2. The total number of words in this string

3. How many times each character is repeated in this string (use a String array, where each element in the array represents the character concatenated with the number that represents how many times it occurs)

4. How many times each word is repeated in this string (use a String array, where each element in the array represents the word concatenated with the number that represents how many times it occurs)

// TreatString received the text as parameter and return a Tuple with the length, number of word, dictioray with the number of times each char is in the string, and a dictionary with the number of the repetition for each word

public func treatString(text: String) -> (length: Int, numWords: Int, charCount: [Character:Int], wordsCount: [String:Int]) {

//Grab the length using the characters.count variable.

let length = text.characters.count;

//Split the text by empty spaces to get an array of the real words

let wordsArray = text.components(separatedBy: " ");

//Number of words is basically the size of the array now.

let numWords = wordsArray.count;

//Instantiate an empty dictionary to be the counter for each character of the whole text. The key is the character and the value is the number of times this characted is in the text

var charCount: [Character:Int] = [:]

//Loop through the text length

for i in 0...length-1 {

//Find the value of each char of the text using index

let index = text.index(text.startIndex, offsetBy: i)

let char: Character = text[index]

//check if it is not empty space.

if char != " " {

//Check if the char already exists in the dictionary

let keyExists = charCount[char] != nil

//If the char exists, get the number of time and add one, if the char does not exists add this as the key and 1 as the value since it is the first time the char appears.

if keyExists {

let counter = Int(charCount[char]!)

charCount[char] = counter + 1;

} else {

charCount[char] = 1;

}

}

}

//Instantiate an empty dictionary to be the counter for each word of the whole text. The key is the word and the value is the number of times this word is in the text

var wordsCount: [String:Int] = [:]

//Loop through the number of words

for i in 0...numWords-1 {

//Get the word from the wordsArray previously found

let word = wordsArray[i];

//Check if word is already added to the dictionary

let keyExists = wordsCount[word] != nil

//If the word exists, get the number of time and add one, if the word does not exists add this as the key and 1 as the value since it is the first time the word appears.

if keyExists {

let counter = Int(wordsCount[word]!)

wordsCount[word] = counter + 1;

} else {

wordsCount[word] = 1;

}

}

// Return the tuple with all the requirements.

return (length, numWords, charCount, wordsCount)

}

print(treatString(text: "testing testing this func func"))

Part 2:

Q1 (10%) [File: Employee.swift]

Create a class called Employee that has the following stored properties:

· EmpID

· First Name

· Last Name

Class Employee should have read/write properties, initializer(s) and should conform Printable protocol.

//Printable protocol is a pattern used to print the variables values of the objects that inherits this protocol. Description should be set to all the variables followed by its values.

protocol Printable {

var description: String { get }

}

//Class Employee inherits from Printable and it implements description variable get method returning all the variables followed by its values.

//This class holds an id, firstName and lastName of the Employee and will be used as a parent class to the subclasses following.

//Init method can be used to instantiated the class empty and also giving values to the properties.

//All the properties has gets and sets implemented.

public class Employee : Printable {

private var \_id : Int = 0;

private var \_firstName : String = ""

private var \_lastName : String = ""

public var id: Int {

get {

return self.\_id

}

set {

self.\_id = newValue

}

}

public var firstName: String {

get {

return self.\_firstName

}

set {

self.\_firstName = newValue

}

}

public var lastName: String {

get {

return self.\_lastName

}

set {

self.\_lastName = newValue

}

}

public var description: String {

get {

return "id: \(self.id), First Name: \(self.firstName), Last Name: \(self.lastName)"

}

}

init () {

}

init (id: Int, firstName: String, lastName: String) {

self.id = id

self.firstName = firstName

self.lastName = lastName

}

}

**Q2 (5%) [File: Taxable.swift]**

Create a protocol Taxable that declares one method called income that returns the weekly income for employees

//Taxable is a protocol and declare the income function to be used for the subclasses, this function return an Double which represents the income.

protocol Taxable {

func income() -> Double;

}

**Q3 (7.5%) [File: SalariedEmployee.swift]**

Create a class called SalariedEmployee that inherits from Employee and has the following members:

· yearlySalary

Class SalariedEmployee should have read/write property, initializer(s) and should override the description property. It should also conform the protocol Taxable.

// SalariedEmployee inherits from Employee and Taxable which makes it be able to access all the variables and functions from the parents.

// It has one property called yearlySalary which has get and set

// Override description protocol pattern and return its properties and values including yearlySalary

// The income of this employee is based on weekly income.

public class SalariedEmployee : Employee, Taxable {

private var \_yearlySalary : Double = 0.0

public var yearlySalary: Double {

get {

return self.\_yearlySalary

}

set {

self.\_yearlySalary = newValue

}

}

override public var description: String {

get {

return "id: \(super.id), First Name: \(super.firstName), Last Name: \(super.lastName), Yearly Salary: \(self.yearlySalary)"

}

}

//Contructor makes possible to instantiate the object without set values to its properties.

override init() {

super.init();

}

//Constructor makes it able to give all the properties since its parent and the new of this object.

init (id: Int, firstName: String, lastName: String, yearlySalary: Double) {

super.init(id: id, firstName: firstName, lastName: lastName)

self.yearlySalary = yearlySalary

}

//Return weekly salary, since the year has 52 weeks.

internal func income() -> Double {

return yearlySalary / 52;

}

}

**Q4 (7.5%) [File: HourlyEmployee.swift]**

Create a class called HourlyEmployee that inherits from Employee and has the following members:

· hourlyRate

· hoursWorked

Class HourlyEmployee should have read/write properties, initializer(s) and should override the description property. It should also conform the protocol Taxable.

// HourlyEmployee inherits from Employee and Taxable which makes it be able to access all the variables and functions from the parents.

// Declares hourly rate and hours worked since it is an hourly employee, both are Double objects and have get and set for each.

// Override description protocol pattern and return its properties and values including hourly rate and hours worked

public class HourlyEmployee : Employee, Taxable {

private var \_hourlyRate : Double = 0.0

private var \_hoursWorked : Double = 0.0

public var hourlyRate: Double {

get {

return self.\_hourlyRate

}

set {

self.\_hourlyRate = newValue

}

}

public var hoursWorked: Double {

get {

return self.\_hoursWorked

}

set {

self.\_hoursWorked = newValue

}

}

override public var description: String {

get {

return "id: \(self.id), First Name: \(self.firstName), Last Name: \(self.lastName), Hourly Rate: \(self.hourlyRate), Hours Worked: \(self.hoursWorked)"

}

}

//Contructor makes possible to instantiate the object without set values to its properties.

override init() {

super.init();

}

//Constructor makes it able to give all the properties since its parent and the new of this object.

init (id: Int, firstName: String, lastName: String, hourlyRate: Double, hoursWorked: Double) {

super.init(id: id, firstName: firstName, lastName: lastName)

self.hourlyRate = hourlyRate

self.hoursWorked = hoursWorked

}

//Income function is based on how many hours worked times the hour rate.

internal func income() -> Double {

return self.hourlyRate \* self.hoursWorked;

}

}

**Q5 (7.5%) [File: BasePlusCommissionEmployee.swift]**

Create a class called BasePlusCommissionEmployee that inherits from Employee and has the following members:

· weeklySalary

· weeklySales

· commissionRate

Class BasePlusCommissionEmployee should have read/write properties, initializer(s) and should override the description property. It should also conform the protocol Taxable.

// BasePlusCommissionEmployee inherits from Employee and Taxable which makes it be able to access all the variables and functions from the parents.

// Declares weeklySalary, weeklySales and commissionRate to be used to calculate the income of its employee, all the properties have its own get and set.

// Override description protocol pattern and return its properties and values including hourly rate and hours worked

public class BasePlusCommissionEmployee : Employee, Taxable {

private var \_weeklySalary : Double = 0.0

private var \_weeklySales : Double = 0.0

private var \_commissionRate : Double = 0.0

public var weeklySalary: Double {

get {

return self.\_weeklySalary

}

set {

self.\_weeklySalary = newValue

}

}

public var weeklySales: Double {

get {

return self.\_weeklySales

}

set {

self.\_weeklySales = newValue

}

}

public var commissionRate: Double {

get {

return self.\_commissionRate

}

set {

self.\_commissionRate = newValue

}

}

override public var description: String {

get {

return "id: \(self.id), First Name: \(self.firstName), Last Name: \(self.lastName), Weekly Salary: \(self.weeklySalary), Weekly Sales: \(self.weeklySales), Comission Rate: \(self.commissionRate)"

}

}

//Contructor makes possible to instantiate the object without set values to its properties.

override init() {

super.init();

}

//Constructor makes it able to give all the properties since its parent and the new of this object.

init (id: Int, firstName: String, lastName: String, weeklySalary: Double, weeklySales: Double, commissionRate: Double) {

super.init(id: id, firstName: firstName, lastName: lastName)

self.weeklySales = weeklySales

self.weeklySalary = weeklySalary

self.commissionRate = commissionRate

}

//Income of this employee is bases on its weeklySalary plus the calculation of weekly sales times the committion rate.

internal func income() -> Double {

return self.weeklySalary + (self.weeklySales \* self.commissionRate);

}

}

**Q6I (12.5%) [File: main.swift]**

Create an array that can hold Employee objects. Populate this array with two objects of Hourly, Salaried and BasePlusCommission employee objects (6 objects in total).

Display all the objects on screen.

Sort and display the employees based on their income from low to high and from high to low.

//Instantiate an empty Employee Array

var employeeArr: [Employee] = []

//Instantiate two SalariedEmployee passing the values for its properties from the Init contructor method

var salaried = SalariedEmployee(id: 1, firstName: "John", lastName: "Doe", yearlySalary: 55000)

var salaried2 = SalariedEmployee(id: 2, firstName: "Marta", lastName: "Sta", yearlySalary: 120000)

//Instantiate two HourlyEmployee passing the values for its properties from the Init contructor method

var hourly = HourlyEmployee(id: 3, firstName: "Frank", lastName: "Stwood", hourlyRate: 50, hoursWorked: 200)

var hourly2 = HourlyEmployee(id: 4, firstName: "Lucas", lastName: "Turner", hourlyRate: 20, hoursWorked: 100)

//Instantiate two BasePlusCommissionEmployee passing the values for its properties from the Init contructor method

var comm = BasePlusCommissionEmployee(id: 5, firstName: "Kan", lastName: "Jum", weeklySalary: 1000, weeklySales: 10, commissionRate: 50)

var comm2 = BasePlusCommissionEmployee(id: 6, firstName: "Jun", lastName: "Yang", weeklySalary: 5000, weeklySales: 1, commissionRate: 100)

//Adding all the instantiated Employees to the array

employeeArr.append(salaried)

employeeArr.append(salaried2)

employeeArr.append(hourly)

employeeArr.append(hourly2)

employeeArr.append(comm)

employeeArr.append(comm2)

//Loop through the array and print each object description since they implements description pattern.

for empl in employeeArr {

print(empl.description);

}

//Sort function returns a new array of Employee based on the sort order, if isAsc is true then it will sort from the lower income to the higher

//If isAsc is false it will sort from the higher income to the lower.

//Uses a specific swift method to sort the arrays, since each element of the array is an Taxable, cast the element to be able to use income function.

//sorted function uses the comparable pattern used in many programming languages, it will compare each object income to the others and swift the positions in the array.

func sort(isAsc: Bool) -> [Employee] {

return employeeArr.sorted { (emp1, emp2) -> Bool in

let tax1 = emp1 as! Taxable

let tax2 = emp2 as! Taxable

if isAsc {

return tax1.income() < tax2.income();

} else {

return tax1.income() > tax2.income();

}

}

}

print("");

//Instantiate the employees array sorted ascending by income.

let lowToHigh = sort(isAsc: true);

//Loop through the lowToHigh array and print the elements description and its income.

for empl in lowToHigh {

let tax1 = empl as! Taxable

let income = tax1.income();

print("\(empl.description), Income: \(income)");

}

print("");

//Instantiate the employees array sorted descending by income.

let highToLow = sort(isAsc: false);

//Loop through the highToLow array and print the elements description and its income.

for empl in highToLow {

let tax1 = empl as! Taxable

let income = tax1.income();

print("\(empl.description), Income: \(income)");

}