**JC 1 PROMOTIONAL EXAMINATION 2016**

Candidate name:

Centre number: 3030

Index number:

Programming language used:

|  |
| --- |
| **QUESTION 1** |
| **EVIDENCE 1** |
| raceData = open("RACE.txt", "r")  raceArray = [] #store race data in a 2-D array  moreThan11 = 0  print("|{0:^15}|{1:^10}|{2:^20}|{3:^15}".format("Runner ID", "Country", "Name", "Race Time"))  print("-"\*73)  for eachRacer in raceData:  racerData = eachRacer[:-1].split(",")  runnerID = racerData[0]  countryCode = racerData[1]  runnerName = racerData[2]  raceTime = racerData[3]  raceArray.append(racerData)  if float(raceTime) > 11: #eval is evil  moreThan11 += 1  print("|{0:^15}|{1:^10}|{2:^20}|{3:^15}".format(runnerID, countryCode, runnerName, raceTime))  print()  print("Number of runner who recorded a timing >11 seconds: " + str(moreThan11))  raceData.close() |
| **EVIDENCE 2** |
|  |
| **EVIDENCE 3** |
| raceData = open("RACE.txt", "r")  raceArray = [] #store race data in a 2-D array  for eachRacer in raceData:  racerData = eachRacer[:-1].split(",")  runnerID = racerData[0]  countryCode = racerData[1]  runnerName = racerData[2]  raceTime = racerData[3]  raceArray.append(racerData)  print("|{0:^15}|{1:^10}|{2:^20}|{3:^15}".format("Runner ID", "Country", "Name", "Race Time"))  print("-"\*73)  sortedRaceArray = raceArray  #insertion sort  for i in range(len(sortedRaceArray)):  x = sortedRaceArray[i]  j = i  while j > 0 and sortedRaceArray[j - 1][3] > x[3]:  sortedRaceArray[j] = sortedRaceArray[j - 1]  j = j - 1  sortedRaceArray[j] = x  for eachRacer in range(10):  print("|{0:^15}|{1:^10}|{2:^20}|{3:^15}".format(sortedRaceArray[eachRacer][0], sortedRaceArray[eachRacer][1], sortedRaceArray[eachRacer][2], sortedRaceArray[eachRacer][3]))    raceData.close() |
| **EVIDENCE 4** |
|  |
| **QUESTION 2** |
| **EVIDENCE 5** |
| def binary\_search(array, input\_value): #array is the CITY array, input\_value is the search key  element\_found = False  low\_element = 0  high\_element = len(array) - 1  while (element\_found != True) and (low\_element <= high\_element):  index = int((low\_element + high\_element) / 2)  if array[index] == input\_value:  element\_found = True  else:  if input\_value < array[index]: #if input is higher in alphabetical order than middle  high\_element = index - 1 #left sub-array  else: #lower in alphabetical order than middle  low\_element = index + 1 #right sub-array  if element\_found == True:  return array[index] + " is found in the array in position " + str(index + 1) + "."  else:  return "Not found"  city\_file = open("CITY.txt", "r")  city\_array = [] #convert city file to list  for each\_city in city\_file:  city\_array.append(each\_city[:-1])  city\_file.close()  print(binary\_search(city\_array, "Singapore"))  print(binary\_search(city\_array, "not a city"))  print(binary\_search(city\_array, "Barcelona")) |
| **EVIDENCE 6** |
| def binary\_search(array, input\_value): #array is the CITY array, input\_value is the search key  element\_found = False  low\_element = 0  high\_element = len(array) - 1  while (element\_found != True) and (low\_element <= high\_element):  index = int((low\_element + high\_element) / 2)  if array[index] == input\_value:  element\_found = True  else:  if input\_value < array[index]: #if input is higher in alphabetical order than middle  high\_element = index - 1 #left sub-array  else: #lower in alphabetical order than middle  low\_element = index + 1 #right sub-array  if element\_found == True:  return array[index] + " is found in the array in position " + str(index + 1) + "."  else:  return "Not found"  city\_file = open("CITY.txt", "r")  city\_array = [] #convert city file to list  for each\_city in city\_file:  city\_array.append(each\_city[:-1])  city\_file.close()  user\_exited = False #checks if user has exited  while user\_exited == False:  city\_input = input("Enter the city to search for: ")  if city\_input == "XXX": #'XXX' pressed!  print("'XXX' pressed, exiting program.")  user\_exited = True  else: #keep on searching  print(binary\_search(city\_array, city\_input))  print() |
| **EVIDENCE 7** |
|  |
| **EVIDENCE 8** |
| {Paste your answers here} |
| **QUESTION 3** |
| **EVIDENCE 9** |
| class ListNode():  def \_\_init\_\_(self, Name = "", Pointer = -1):  self.\_\_Name = Name  self.\_\_Pointer = Pointer  def getPointer(self):  return self.\_\_Pointer  def setPointer(self, newPointer):  self.\_\_Pointer = newPointer  def getName(self):  return self.\_\_Name  def setName(self, newName):  self.\_\_Name = newName  class LinkedList():  def \_\_init\_\_(self, size = 20):  self.\_\_Node = [ListNode() for i in range(size)]  for i in range(size - 1):  self.\_\_Node[i].setPointer(i + 1)  self.\_\_StartPtr = -1  self.\_\_NextFreePtr = 0  def DisplayList(self):  print("{0:^10} | {1:^20} | {2:^10}".format("Node", "Country Name", "Pointer"))  print("-"\*45) #separate heading from contents  for i in range(len(self.\_\_Node)):  print("{0:^10} | {1:^20} | {2:^10}".format(i, self.\_\_Node[i].getName(), self.\_\_Node[i].getPointer()))  print()  print("Value of StartPtr: {0}".format(self.\_\_StartPtr))  print("Value of NextFreePtr: {0}".format(self.\_\_NextFreePtr))  def IsEmpty(self):  if self.\_\_StartPtr == -1 and self.\_\_NextFreePtr == 0:  return True  else:  return False  def IsFull(self):  if self.\_\_NextFreePtr == -1:  return True  else:  return False |
| **EVIDENCE 10** |
|  |
| **EVIDENCE 11** |
| def InsertNode(self, NewItem):  if self.IsFull() == True:  print("Array is full. Cannot insert node!")  else: #array not full  #find insertion point  PreviousPtr = self.\_\_StartPtr  CurrentPtr = self.\_\_StartPtr  while (CurrentPtr != -1) and (self.\_\_Node[CurrentPtr].getName() < NewItem):  PreviousPtr = CurrentPtr  CurrentPtr = self.\_\_Node[CurrentPtr].getPointer()  if (self.\_\_Node[CurrentPtr].getName() == NewItem):  print("Cannot insert repeated entries!")  else:  self.\_\_Node[self.\_\_NextFreePtr].setName(NewItem)  NewNodePtr = self.\_\_NextFreePtr  self.\_\_NextFreePtr = self.\_\_Node[self.\_\_NextFreePtr].getPointer()  if CurrentPtr == self.\_\_StartPtr: #insert new node at start of list  self.\_\_Node[NewNodePtr].setPointer(self.\_\_StartPtr)  self.\_\_StartPtr = NewNodePtr  else: #in between PreviousPtr and CurrentPtr  self.\_\_Node[NewNodePtr].setPointer(self.\_\_Node[PreviousPtr].getPointer())  self.\_\_Node[PreviousPtr].setPointer(NewNodePtr) |
| **EVIDENCE 12** |
| CountryFile = open("COUNTRY.txt", "r")  CountryList = LinkedList() #linked list of all countries to be added  for EachCountry in CountryFile:  CountryName = EachCountry[:-1] #removes newline at the end  CountryList.InsertNode(CountryName)  CountryFile.close()  CountryList.DisplayList() |
| **EVIDENCE 13** |
|  |
| **EVIDENCE 14** |
| def DeleteNode(self):  ItemToDelete = input("Enter the country name to delete: ")  if self.IsEmpty == True:  print("Array is empty. Cannot delete", ItemToDelete, "from the linked list!")  else: #array not empty  PreviousPtr = self.\_\_StartPtr  CurrentPtr = self.\_\_StartPtr  while (CurrentPtr != -1) and (self.\_\_Node[CurrentPtr].getName() != ItemToDelete): #traverse linked list  PreviousPtr = CurrentPtr  CurrentPtr = self.\_\_Node[CurrentPtr].getPointer()  if CurrentPtr == -1: #name cannot be found  print(ItemToDelete, "cannot be found. Cannot delete", ItemToDelete, "from the linked list!")  elif CurrentPtr == self.\_\_StartPtr: #deleting the first node  temp = self.\_\_Node[CurrentPtr].getPointer()  self.\_\_Node[CurrentPtr].setPointer(self.\_\_NextFreePtr)  self.\_\_StartPtr = temp  self.\_\_NextFreePtr = CurrentPtr  else: #deleting between 2 nodes or last node  temp = self.\_\_Node[CurrentPtr].getPointer()  self.\_\_Node[CurrentPtr].setPointer(self.\_\_NextFreePtr)  self.\_\_Node[PreviousPtr].setPointer(temp)  self.\_\_NextFreePtr = CurrentPtr |
| **EVIDENCE 15** |
|  |
| **EVIDENCE 16** |
| def OutputAllNodes(self):  print("{0:^10} | {1:^20} | {2:^10}".format("Node", "Country Name", "Pointer"))  print("-"\*45) #separate heading from contents  PreviousPtr = self.\_\_StartPtr  CurrentPtr = self.\_\_StartPtr  while CurrentPtr != -1: #to traverse linked list  print("{0:^10} | {1:^20} | {2:^10}".format(CurrentPtr, self.\_\_Node[CurrentPtr].getName(), self.\_\_Node[CurrentPtr].getPointer()))  PreviousPtr = CurrentPtr  CurrentPtr = self.\_\_Node[CurrentPtr].getPointer() |
| **EVIDENCE 17** |
|  |