**HWA CHONG INSTITUTION**

**C2 PRELIMINARY EXAMINATION 2019**

**COMPUTING**

**Higher 2**

**3 September 2019 Paper 1 (9597 / 01) 0815 -- 1130 hrs**

**Additional Materials:**

Electronic version of CARPARK.txt data file

Electronic version of NUMBERS.txt data file

Electronic version of STUDENT.txt data file

Electronic version of SCORES1.txt data file

Electronic version of SCORES2.txt data file

Electronic version of EVIDENCE.docx document

**-----------------------------------------------------------------------------------------------------------------**

**READ THESE INSTRUCTIONS FIRST**

Type in the EVIDENCE.docx document the following:

* Candidate details
* Programming language used

Answer **all** questions.

The maximum mark for this paper is 100.

All tasks must be done in the computer laboratory. You are not allowed to bring in or take out any pieces of work or materials on paper or electronic media or in any other form.

All tasks and required evidence are numbered.

The number of marks is given in brackets [ ] at the end of each task.

Copy and paste required evidence of program listing and screenshots into the EVIDENCE.docx document.

**At the end of the examination, print out your EVIDENCE.docx and fasten your printed copy securely together.**

1. The manager of a private carpark wants to process the daily data of people using the carpark. The carpark is open from 8am to 11pm and the carpark charge is as follows:

* Before 5pm: $1.50 per hour or part thereof
* After 5pm, $3.00 per entry regardless of the duration

However, if a car enters before 5pm and leaves after 5pm, the charge involves both rules. For example, if the car stays in the carpark from 2.45pm to 6.30pm, it is broken down to 2 hours 15 minutes before 5pm and 1 hour 30 minutes after 5pm. Hence the charge will be $1.50 \* 3 + $3.00 = $7.50.

Each day the carpark electronic system generates a file CARPARK.txt. Each record in the file has the following format:

<CARPLATE NUMBER>,<START TIME>,<END TIME>

For example, SLX2315A,0940,1415 means that a car with car plate number SLX2315A entered the carpark at 9.40am and left at 2.15pm.

You are **not** allowed to use any built-in functions for time processing.

|  |
| --- |
| **Task 1.1**  Write program code for the Price function using the following specification:  FUNCTION Price (start: STRING, end: STRING) : FLOAT  The function has two string parameters start, end which refers to the start and end time when the car parked. The function returns the carpark charge as a float.  **Evidence 1**  Your program code. [5]  **Task 1.2**  Write a program code to perform the following task for the manager:   * Read data from CARPARK.txt * Write all the car plate numbers and corresponding carpark charges to another file CHARGE.txt, in the following format:   <CARPLATE NUMBER>,<CARPARK CHARGE>  <CARPLATE NUMBER>,<CARPARK CHARGE>   * Output the total charges for the day   **Evidence 2**  Your program code. [7]  **Evidence 3**  One screenshot showing the program output and contents of CHARGE.txt from running the program. [3] |

1. When a list of integers has repeated numbers, the searching and sorting algorithms can be different. The task is to perform an insertion sort before a binary search is executed.

**Task 2.1**

Write the program code for a procedure to implement insertion sort in ascending order. The input parameter is a list of integers which may have repeated numbers.

**Evidence 4**

Your program code. [4]

**Task 2.2**

Write the program code for a procedure to implement binary search for a targeted integer. The input parameter is an ordered list of integers which may have repeated numbers. The procedure outputs all the indices at which the target appears, or −1 if the target is not found.

**Evidence 5**

Your program code. [9]

**Task 2.3**

The file NUMBERS.txt contains one integer at each line. Write a program that uses the procedures in previous tasks and performs the following task:

• Reads the text file NUMBERS.txt,

• Perform insertion sort, and outputs the list of integers in a row,

• Prompts the user to provide a target to be searched,

• Perform a binary search and output an appropriate message

**Evidence 6**

Your program code. [4]

**Task 2.4**

Draw up **three** suitable tests and provide screenshot evidence for your testing.

**Evidence 7**

Annotated screenshots for each test data run. [3]

3. The examinations department of a school needs to keep long-term records of the overall examination achievements of its students.

Students at the school have two main choices. Firstly, they can take a variety of subjects and achieve an Academic Diploma. A diploma gives them the opportunity to go to university. Secondly they can achieve a Skills Certificate where they focus on one particular area (such as IT). This gives them the necessary skills to start a career in their chosen area.

The examinations department decides to store the following data:

* StudID is used to uniquely identify a particular student and is six digits. The first four digits represent the year that the student started at the school and the last two digits are used to make the StudID unique e.g. 201804.
* Name is the name of the student and is at most 30 characters.
* StudType is the type of student and can have the values ‘D’ or ‘S’.
* SkillArea is text and gives the area that the student acquired skills in. It can have one of three values: ‘IT’, ‘Business’ or ‘Accountancy’.
* NoOfSub is the number of subjects studied by those taking the Diploma.
* Result is a single character and is used to indicate the overall grade awarded. For those students who took the Skills Certificate the grades could be Distinction (D), Merit (M), Pass (P) or Fail (F). For those who took the Diploma the grade could be one of the letters A to F. Grade A to E are passes. Grade F is a fail.

The program design for a solution to this problem is to be implemented with object-oriented programming with the following three classes:

Student

Diploma SkillsCert

**Task 3.1**

Write program code to define the classes Student, Diploma and SkillsCert.

**Evidence 8**

Program code for the three classes in Task 3.1. [6]

Assume that a file, STUDENT.txt, which contains details of each student, has been created for you. The format of each student record is as follows:

<StudID>|<Name>|<StudType>|<SkillArea>|<NoOfSub>|<Result>

* SkillArea would have the value ‘Diploma’ if the student is taking a Diploma.
* NoOfSub would have the value 0 for those taking the Skills Certificate.
* Result is left blank initially.

**Task 3.2**

Write a module, ENTER\_RESULT, which, when called, will ask the user for a particular StudID whose result is to be entered. Using the student ID that has been input, the corresponding student record will be located in STUDENT.txt. The student data will be displayed to the user. The user will be allowed to enter the result for the student. The amended record will be stored back in STUDENT.txt.

The student ID and result that have been input should be validated.

If the StudID does not exist, the user will be given an appropriate message.

**You are expected to make use of the classes you designed in Task 3.1.**

Run the program **three** times. Use the following data input, and produce a screenshot for each.

StudID Result

201701 A

201801 B

201901 M

**Evidence 9**

Program code for Task 3.2 [8]

**Evidence 10**

**Three** screenshots showing the test runs and final contents of STUDENT.txt to show evidence that successful updates have been carried out. [2]

**Task 3.3**

Implement code as specified below.

A report should be generated and displayed which will list the students whose result has still not been entered into the STUDENT.txt file. The report will list, for each different starting year:

• StudID

• Name

• StudType

• SkillArea or NoOfSub depending upon the value of StudType

In addition the number of each student type for each year will also be output.

A sample output is shown below.

Year: 2017

--------------------------------------------------------------

201715 FLoo D 6

201708 BLang D 5

201710 LArms S IT

Diplomas: 2

Skills: 1

Year: 2018

--------------------------------------------------------------

201813 EJean D 7

201817 ABright D 7

Diplomas: 2

Skills: 0

Year: 2019

---------------------------------------------------------------

201905 Alfie S Business

201903 GKoh D 8

Diplomas: 1

Skills: 1

**Evidence 11**

Program code for Task 3.3. [8]

**Evidence 12**

Screenshot of the output produced. [2]

4. A game maintains the player IDs and their scores in an ordered linked list. The player with the highest score is stored at the first node while the player with the lowest score is stored at the last node.

The program to implement the linked list abstract data type will use two classes, ListNode and LinkedList.

The ListNode class has the following properties:

|  |  |  |
| --- | --- | --- |
| Identifier | Data Type | Description |
| ID | STRING | The ID of the player. All IDs are unique and have the format L999 where L is any uppercase letter and 9 is a digit. |
| Score | INTEGER | The score of the player. |
| Ptr | INTEGER | The pointer to the next node. |

The LinkedList class has the following properties:

|  |  |  |
| --- | --- | --- |
| Identifier | Data Type | Description |
| Node | ARRAY[1..20] OF ListNode | 1-D array stores the nodes that make the ordered linked list. The unused nodes are linked together into a free list. |
| HeadPtr | INTEGR | Pointer to the first node in the ordered list. |
| FreePtr | INTEGER | Pointer to the first node in the free list. |

The following diagram shows an example of a linked list object. This example list consists of three nodes, linked in descending order of the game scores. The unused nodes are linked to form a free list.

HeadPtr 2 3 1

2 C731 30 3 A145 28 1 C412 13 0

ID Score Ptr

FreePtr 4 5 20

4 5 6 . . . . . 0

**Task 4.1**

Write program code for the classes ListNode and Linkedlist to declare all the required variables and create the initial empty linked list which contains all 20 nodes.

Add statement(s) to initialise the empty ordered linked list.

**Evidence 13**

Your program code for Task 4.1. [6]

**Task 4.2**

Write code to implement a method AddInOrder that will add a new node with player’s ID and score into the ordered linked list in descending order of the scores. Node added to the ordered linked list should be taken from the free list.

Assume that all players have different scores.

**Evidence 14**

Your program code for Task 4.2. [7]

**Task 4.3**

Write a procedure OutputData which displays the value of HeadPtr, the value of FreePtr and the contents of Node array in index order.

**Evidence 15**

Your program code for Task 4.3. [3]

The files SCORES1.txt and SCORES2.txt contain the game data. Each entry has the following format: <Player ID>,<Score>

**Task 4.4**

Write a main program to:

• Create a linked list object

• Read all player data from SCORES1.txt and add them to the linked list by

calling procedure AddInOrder.

• Your program will then call procedure OutputData.

**Evidence 16**

Screenshot showing the output from running the program in Task 4.4 using SCORES1.txt file. [2]

**Task 4.5**

Amend your AddInOrder program code in Task 4.2 so that if two or more players have the same score, they are stored in alphabetical player ID order. Use the file SCORES2.txt to test your program code.

The following diagram shows an example of an ordered linked list where players C412 and B321 have the same game score of 13 points.

HeadPtr 2 3 4 1

2 C731 30 3 A145 28 4 B321 13 1 C412 13 0

**Evidence 17**

The amended program code for method AddInOrder. [4]

**Evidence 18**

Screenshot showing the output from running the program in Task 4.4 using SCORES2.txt file. [2]

**Task 4.6**

A method DisplayByRank is to be added, which outputs all player IDs and their scores stored in the ordered linked list in rank order. If multiple players record the same score, they will have the same rank.

Below is a sample of screen output:

Rank Player ID Score

1 F111 45

1 G333 45

1 Z333 45

4 C333 38

5 B111 25

5 Q333 25

7 E333 12

Write program code to:

• Implement this method

• Test the program code with the data from Task 4.5.

**Evidence 19**

Program code for Task 4.6. [7]

**Evidence 20**

Screenshot of the program output. [2]

**Task 4.7**

Write a recursive ReverseTraversal procedure that will traverse the linked list in reverse order and output players’ IDs and scores in ascending scores order.

Include a call to the procedure from your main program.

Test the program code with the data from Task 4.5.

**Evidence 21**

Your program code for Task 4.7. [4]

**Evidence 22**

Screenshot showing the program execution to test the ReverseTraversal method.

[2]