MINISTRY OF EDUCATION, SINGAPORE
in collaboration with
UNIVERSITY OF CAMBRIDGE LOCAL EXAMINATIONS SYNDICATE
General Certificate of Education Advanced Level
Higher 2

COMPUTING

9597/01

Paper 1

October/November 2018

3 hours 15 minutes

Additional Materials: Pre-printed A4 paper
Removable storage device
Electronic version of `STAR.TXT` data file
Electronic version of `QUICKSORT.TXT` file
Electronic version of `SCORES.TXT` data file
Electronic version of `HASHEDDATA.TXT` file
Electronic version of `MAZE.TXT` file
Electronic version of `EVIDENCE.DOCX` file

READ THESE INSTRUCTIONS FIRST

Type in the `EVIDENCE.DOCX` document the following:
- Candidate details
- Programming language used

Answer **all** questions.

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` document and fasten your printed copy securely together.

Singapore Examinations and Assessment Board

CAMBRIDGE International Examinations

1 A program is required to input and process the number of steps taken by members of a walking club each week. The number of steps taken by each member is an integer in the range 0 to 100000.

Each week, the "Star of the Week" is the member who has taken the greatest number of steps.

The name and number of steps taken by the **previous** week's "Star of the Week" are stored in the text file, `STAR.TXT`.

The program specification is as follows:

- Input **up to** 10 names and the number of steps each has taken. Assume that each number of steps is unique.
- Find the walker who has taken the greatest number of steps from this data.
- Read the data about the previous "Star of the Week" from the text file `STAR.TXT`.
- Display a message on screen to show the previous star of the week **and** the new star of the week, each with their number of steps. For example,

```
Last week, Jenny Smith was 'Star of the Week' with 75827 steps
taken.
This week, Vanessa Lim is 'Star of the Week' with 67152 steps taken.
```

- Update the text file, `STAR.TXT`, with the details of the new "Star of the Week".

Task 1.1
Write program code for this task that includes validation of data entered.

[8]

The program needs to be tested with different test cases. Consider carefully, test cases for input of names and steps.

Task 1.2
Copy the table with the following headings. Add other test cases to the table. One type of test case has already been added to the table.

Test data	Purpose of test data	Expected results
Yi Ling Aw, 10232 Ryan Batisah, 42231 Lee Casmir, 35020 Daniel Bennett, 60192 Sarah Heng Chee, 29389 Vanessa Lim, 67152 Wong Yip, 53231 Rin Xie, 34200 Tin Wee, 49480 David Bala, 32010	Test the maximum of 10 values entered into the program.	10 values entered and star of the week is Vanessa Lim with 67152 steps taken.

[4]

Task 1.3
Use **three** of the test cases in the table

[3]

2 The following algorithm is an implementation of a quick sort that operates on an array `Scores`.

This algorithm assumes that the first element of an array is the zeroth element. This means that `Scores[0]` is the first element in the array.

This pseudocode is available in the file `QUICKSORT.TXT`

```
FUNCTION QuickSort(Scores)
  QuickSortHelper(Scores, 0, LENGTH(Scores) - 1)
  RETURN Scores
ENDFUNCTION

FUNCTION QuickSortHelper(Scores, First, Last)
  IF First < Last
    THEN
       SplitPoint ← Partition(Scores, First, Last)
       QuickSortHelper(Scores, First, SplitPoint - 1)
       QuickSortHelper(Scores, SplitPoint + 1, Last)
  ENDIF
  RETURN Scores
ENDFUNCTION

FUNCTION Partition(Scores, First, Last)
  PivotValue ← Scores[First]
  LeftMark ← First + 1
  RightMark ← Last
  Done ← FALSE
  WHILE (Done = FALSE)
    WHILE LeftMark <= RightMark AND Scores[LeftMark] <= PivotValue
      LeftMark ← LeftMark + 1
    ENDWHILE
    WHILE Scores[RightMark] >= PivotValue AND RightMark >= LeftMark
      RightMark ← RightMark - 1
    ENDWHILE
    IF RightMark < LeftMark
      THEN
        Done ← TRUE
    ELSE
      Temp ← Scores[LeftMark]
      Scores[LeftMark] ← Scores[RightMark]
      Scores[RightMark] ← Temp
    ENDIF
  ENDWHILE

  <swap Scores[First] with Scores[RightMark]>

  RETURN RightMark
ENDFUNCTION
```


Task 2.1
Write program code to implement this algorithm. Ensure that you add the missing code to complete the algorithm. The area of missing code is highlighted as:

```
<swap Scores[First] with Scores[RightMark]>
```

Copy the sample data available in the `SCORES.TXT` file. Paste this into your programming code to set up the data to be sorted.

[12]

Task 2.2
Add a function to your code to output `Scores`. Call this function before and after the operation of the quick sort so that the unsorted and sorted data is displayed.

[2]

[1]

3 The file, HASHEDDATA.TXT, holds details of the names and telephone numbers of 250 people.

There are a total of 500 lines in the file, and a number of these lines are empty of name and telephone number.

An index is stored for each line of the file.

The format of the data in the file is:

<Index>,<PersonName>,<TelephoneNumber>

The first 10 lines from the file are shown as follows:

```
0,,
1,,
2,,
3,Boon Keng V.,07492 546415
4,,
5,,
6,Ahmad Yusof,07439 778665
7,Durno Peter,07662 863518
8,Batisah Wong,07362 156265
9,,
```

The values in the file are separated by the comma character.

A record structure is used to store a name and telephone number. A data structure of 500 records is needed to store all the names and telephone numbers. Each line in the file is written to a corresponding position in the data structure.

The records with index six to eight from the data structure are:

Index	PersonName	TelephoneNumber
6	Ahmad Yusof	07439 778665
7	Durno Peter	07662 863518
8	Batisah Wong	07362 156265

Task 3.1
Use program code to create a:

- record structure to hold the name and telephone number for one person
- data structure, using this record structure to store 500 records.

[6]

Task 3.2
Write program code to:

- read the lines from the file
- extract the <Index>, <PersonName> and <TelephoneNumber> values
- store these values in the data structure.

Create a procedure called `DisplayValues` that will loop though the data structure and display the index, name and telephone number for every record where the name is present.
Ensure your procedure uses headings to identify the data displayed.

[13]

[1]

A hashing function was used to create the file. The same hashing function can be used to search the data structure for a particular name.

The hashing function generates a hash. This is calculated as follows:

```
Get SearchName
Set HashTotal to 0
FOR each Character in SearchName
    Get the ASCII code for Character
    Multiply the ASCII code by the position of Character in SearchName
    Add the result to the HashTotal
Calculate Hash as HashTotal MOD 500
RETURN Hash
```

Task 3.3
Add the program code for the hashing function. Use the following specification:

```
FUNCTION GenerateHash(SearchName : STRING) : INTEGER
```

The function has a single parameter `SearchName` and returns an integer value.

Write additional code for your program to allow you to test the implementation of this function.

The following test data will assist you.

"Tait Davinder" should return a hash of 87
"Anandan Yeo" should return a hash of 156

[8]

[2]

The hash calculated from the `SearchName` can be used to find a corresponding record in the data structure.

If the `SearchName` is not found in the record given by the hash **and** the record is not empty:

- compare `SearchName` with the next record
- until the `SearchName` is found or an empty record is found.

If an empty record is found then the program will report that the name is "NOT FOUND".

If the record is found, the program will output the index, name and telephone number.

Task 3.4
Add program code to implement the search as described

[7]

showing the result of the following searches:

Search 1: Charlie Love

Search 2: Chin Tan

Search 3: John Barrowman

[3]

4 In a computer game, a player ("O") moves around a maze measuring 10 metres by 11 metres to collect a prize ("P"). The prize is placed at a random position within the maze. The prize position is not where a wall ("X") appears in the maze. An empty position is indicated with a full-stop (".").

The maze is represented on the screen by a rectangular grid. Each square metre of the maze is represented by an x-coordinate and a y-coordinate. The top left square metre of the puzzle display has x = 0 and y = 0.

The player moves left, right, up or down according to a direction entered by the user. The game is turn-based; a user enters the direction, their player moves one position in that direction. If the direction would place the player on a wall, then the player does not move. The maze is displayed after each move.

```
X X X X X X X X X X X
X . . . . . . . . . X
X . X . X . X X . X . X
X . X . . P . . . X . X
X . X X X X X X . X . X
X . . . O . . . . X . X
X . X . X X . X . X . X
X . X . . . . X . X . X
X . X X . X X X . X . X
X . . . . . . . . . X
X X X X X X X X X X X
```

Task 4.1
Write a program to display the maze as shown.

- The maze should be stored in a suitable data structure.
- The data structure will allow fixed loop(s) to be used to display the maze.

The maze is given in the text file MAZE.TXT. You may read in the data from this file **or** place the data in your program using any suitable method.

[6]

Task 4.2
The prize is placed randomly on the maze. It cannot appear in the same grid position as a wall ("X").

Add to your program code to place the prize at a random position.

[4]

output your program. [1]

The player is represented by the character "O". The character starts the game in a central position on the grid, for example, x = 4 and y = 5.

To move the character, the user is prompted for a direction. The following are valid inputs:

Input character	Action
"U"	Player moves up
"D"	Player moves down
"L"	Player moves left
"R"	Player moves right
""	Continue with previous move. If no previous move, do nothing

If the next position for the player ("O") is a wall ("X"), then the player stays in their current position; this is called collision detection.

When the player enters the move, a new position for the player ("O") is calculated and the maze is displayed. The previous position is changed back to a "." when the player has a new position. The moves are repeated until the player is at the same position as the prize.

Task 4.3
Add to your program code to:

- place the player on the grid at a central position on the grid
- take in and validate a direction
- calculate a new position
- check this position is not a wall
- update the grid so that the previous position of "O" is replaced with a "." and "O" is located in its new position
- continue this until the player is at the same position as the prize.

[16]

When the player and the prize are at the same position, the message "Player has reached the prize" is displayed and the game ends.

Task 4.4

Add to your program, code to end the game when this condition is met, and display the required message. show key elements of your program

- entering each direction
- player changing position
- end of game

[1]

- end of game (player wins)

[2]