| Name: | | Index Number: | | Class: | |
|---|---|---|---|---|---|

# DUNMAN HIGH SCHOOL
Preliminary Examination
Year 6

**COMPUTING PAPER 2** **9597**

**Higher 2**
100 marks

**5 September 2013**
**3 hours 15 minutes**

Additional Materials:

**Instructions: Answer all questions.**

Type in the `EVIDENCE.doc` document the following:

- Candidate details
- Programming language used

All data files and `EVIDENCE.doc` are provided in the CD-ROM.

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

Copy and paste required evidence of program listing and screen shots into the `EVIDENCE.doc` document.

**At the end of the examination, print out your `EVIDENCE.doc` and fasten your printed copy securely together with the cover page provided.**

Data files

Q1 – `WIDEST.txt`
Q2 – `NUMBERS.txt`
Q3 – `CURRENCIES.txt, UPDATED.txt`
Q4 – `GAME.dat`

**This paper consists of 4 questions in 4 printed pages.**

**1** A program is to record the highest and lowest temperature readings of selected cities in a day. All temperature readings are integers in the range -90 to 60 inclusive.

The program reads from the file `WIDEST.txt`, the city with the current greatest absolute difference in temperature readings, generated from running the program on previous days. The program specification is to:

- input **up to** 3 cities and their corresponding pairs of highest and lowest temperatures
- calculate and display on screen:
  - the city name and its greatest absolute difference in temperature readings today
  - a message saying the number of days elapsed since the greatest absolute difference in temperature readings last occurred
- update the file `WIDEST.txt` if a greater absolute difference in temperature readings was captured today.

**Task 1.1**
Write program code for this task.

**Evidence 1**: Your program code. [7]

**Task 1.2**
Draw up a set of test data which tests the functionality of your program. Consider carefully all cases which could occur for both the temperature readings input and the processing requirements.

**Evidence 2**: A screenshot for each test case you considered. Annotate the screenshot explaining the purpose of each test. [8]

**2** The following is a recursive pseudocode algorithm for a binary search on an array X. This array contains N integers arranged in **descending** order.
The algorithm is poorly presented and also contains some errors.

```
Mid = (Low + High) div 2
If X(Mid) = Target then output "found", exit
If X(Mid) > Target then Search(X, Low, Mid) Else Search(X, Mid, High)
```

**Task 2.1**
Write program code for this algorithm including all the improvements you would make to:

- follow good programming practice
- make the algorithm work correctly

Use the sample array X data available from the data file `NUMBERS.txt` and paste this into your program code.

**Evidence 3**: Your program code. [9]

**Task 2.2**
A quick sort routine is to be written to sort the contents of an array in **ascending** order. Use the sample data provided in the array X.

**Evidence 4**: Your quick sort routine program code. [4]

**Evidence 5**: One screenshot showing the output from running the program code. [2]

**3** An application is to update the exchange rates of countries against the US dollar stored in a sequential file `CURRENCIES.txt`. The new data for countries with updated exchange rates are provided in `UPDATED.txt`. A new updated sequential file `NEWCURRENCIES.txt` is to be created.

**Task 3.1**
Write program code to perform the update process efficiently.

**Evidence 6**: Your program code. [11]

To facilitate searching, a direct access file is to be created from `NEWCURRENCIES.txt`. The address of each record is calculated from a **hashing function** as follows:

- The ASCII code is calculated for each character within the country name
- The total of all ASCII values is calculated
- The total is divided by a suitable prime number and the remainder calculated with modulo arithmetic
- This value + 1 is the address for the record.

**Task 3.2**
Write program code for the hashing function `HashKey` which takes in a single parameter `Country` and returns the hashed key (i.e. the address).

**Evidence 7**: Your `HashKey` program code. [6]

**Task 3.3**
Write program code for a procedure `CreateCurrency` which does the following:

- **all the country names** are read from `NEWCURRENCIES.txt`
- their addresses are calculated using `HashKey`
- the country names and exchange rates are written to `DIRECTCURRENCIES.txt`

**You must ensure that when a collision occurs your program design will deal with this situation with the result that all records are written to `DIRECTCURRENCIES.txt`.**
Add comment lines to your program code at the start of the procedure to describe your design for dealing with a collision.

**Evidence 8**: Your `CreateCurrency` program code. [11]

**Task 3.4**
Write program code for a procedure `LookUpCurrency` which does the following:

- the user inputs the country name
- the address is hashed from the country name
- the country and exchange rate is located in `DIRECTCURRENCIES.txt`
- the address, the country name and exchange rate are output, each on a new line

Run the program and retrieve the Singapore exchange data from `DIRECTCURRENCIES.txt`

**Evidence 9**: Your `LookUpCurrency` program code. [4]

**Evidence 10**: A screenshot confirming the retrieval of the Singapore exchange rate data. [2]

**Task 3.5**
Write program code for a procedure `FindCollisions` to find all records affected by collisions. You must ensure that the address output is the one where the exchange rate data has been stored.

**Evidence 11**: Your `FindCollisions` program code. [4]

**Evidence 12**: Produce two screenshots showing the retrieval of collided records. [2]

**4** A game maintains the player ids and scores in a linked list with the highest score first in the list. If two or more players have the same highest score, they are stored in alphabetical player id order. All player ids are alphanumeric but must begin with a letter and be at least 3 characters in length. Player ids are not case-sensitive. A valid score is an integer in the range 0 to 20 inclusive.

Every time a player completes a new game its score is added to its current total score. It is then necessary to adjust the linked list to keep it accurate.

Initially all game data is stored in `GAME.dat`.

**Task 4.1**
Assume that all player ids input are valid, write program code to:

- Create a linked list from `GAME.dat`
- Input and validate a player id and score
- Insert to and update the linked list
- Output the player's old and new ranks (players with the same score have the same rank)

**Evidence 13**: Your program code. [12]

**Task 4.2**
Draw up a list of five suitable tests and provide screenshot evidence for your testing.

**Evidence 14**: Annotated screenshots for each test data run. [5]

**Task 4.3**
Write additional program code with appropriate data validation for the following specification:

- Input a rank range e.g. 1 3
- Output for each rank in the range, all player ids and the number of players having this rank

**Evidence 15**: Your program code. [10]

**Task 4.4**
Draw up a list of three suitable tests and provide screenshot evidence for your testing.

**Evidence 16**: Annotated screenshots for each test data run. [3]

**- END OF PAPER -**