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 2019

3 hours 15 minutes

Additional Materials: Pre-printed A4 paper
Removable storage device
Electronic version of `TIDES.TXT` data file
Electronic version of `TASK3_2.TXT` data file
Electronic version of `TASK3_3.TXT` data file
Electronic version of `TASK3_4.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.

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 text file, `TIDES.TXT`, contains the low and high tide information for a coastal location for each day of a month. Each line contains tab-delimited data that shows the date, the time, whether the tide is high or low and the tide height in metres.

Each line is in the format:

```
YYYY-MM-DD\tHH:mm\tTIDE\tHEIGHT\n
```

- The date is in the form YYYY-MM-DD, for example, 2019-08-03 is 3rd August, 2019
- The time is in the form HH:mm, for example, 13:47
- TIDE is either HIGH or LOW
- HEIGHT is a positive number shown to one decimal place
- `\t` represents the tab character
- `\n` represents the newline character

The text file is stored in ascending order of date and time.

Task 1.1
Write program code to:

- read the tide data from a text file
- find the highest high tide and print this value
- find the lowest low tide and print this value.

Use `TIDES.TXT` to test your program code.

[9]

Task 1.2
The tidal range is the difference between the heights of successive tides; from a high tide to the following low tide or from a low tide to the following high tide.

Amend your program code to:

- output the largest tidal range and the date on which the second tide occurs
- output the smallest tidal range and the date on which the second tide occurs.

Use `TIDES.TXT` to test your program code.

[9]

2 Characters are numerically encoded using ASCII codes.

- 'A' has the denary value 65; 'B' has the denary value 66 and so on.
- 'a' has the denary value 97; 'b' has the denary value 98 and so on.

The ROT-13 encoding function replaces a letter with the letter that is 13 positions after it in the alphabet. Characters that are not letters remain unchanged.

The function wraps around from the end of the alphabet back to the beginning. The case of the coded letter should match the case of the original letter.

For example:

- 'A' is replaced with 'N'; 'a' is replaced with 'n'
- 'B' is replaced with 'O'; 'b' is replaced with 'o'
- 'Z' is replaced with 'M'; 'z' is replaced with 'm'

Task 2.1
Write program code that:

- reads a string of characters as input
- encodes the string in ROT-13 form
- outputs the encoded string.

Run the program **three** times with the inputs:

```
This is a word.
ALL &&&& CAPITALS
UpperCamelCase12()
```

[9]

Task 2.2
A string is encoded using ROT-13. The resulting string is then encoded using ROT-13. The output of the second encoding should be identical to the original string.

Amend your program code to apply ROT-13 twice, in the method described. Show that the resulting string is identical to the original string.

[3]

3 A program is to be written to implement a to-do list using object-oriented programming (OOP).

The list shows tasks that need to be done.

Each task is given a category and a description.

The base class will be called `ToDo` and is designed as follows:

ToDo
`category : STRING` `description : STRING`

`constructor(c : STRING, d : STRING)`
`set_category(s : STRING)`
`set_description(s : STRING)`
`get_category() : STRING`
`get_description() : STRING`
`summary() : STRING`

The `summary()` method returns the category and description as a single string.

Task 3.1
Write program code to define the class `ToDo`.

[5]

Tasks should be sorted alphabetically by category. Within each category, tasks should be sorted alphabetically by description.

A task to be added to the list is compared to the tasks already in the list to determine its correct position in the list. If the list is empty, it is added to the beginning of the list.

This comparison will use an additional member method,

```
compare_with(td : ToDo) : INTEGER
```

This function compares the instance (the item in the list) and the `ToDo` object passed to it, returning one of three values:

–1 if the instance is before the given `ToDo`

0 if the two are equal

+1 if the instance is after the given `ToDo`

Task 3.2
There are four objects defined in the text file `TASK3_2.TXT`.

Write program code to:

- implement the `compare_with()` method
- create an empty list of `ToDo` objects
- add each of the four objects in the text file `TASK3_2.TXT` to its appropriate place in the list
- print out the list contents using the `summary()` method.

[13]

The to-do list can have items with extra information. One such item has a date by which the task should be completed.

The `DatedToDo` class inherits from the `ToDo` class, extending it to have a `due_date`, designed as follows:

DatedToDo : ToDo
`due_date : DATE`

`constructor(dt : DATE, c : STRING, d : STRING)`
`set_due_date(d : DATE)`
`get_due_date() : DATE`

The `DatedToDo` class should extend the `compare_with()` method to ensure that tasks are ordered by ascending `due_date`, and then by the ordering used by the base `compare_with()` method. The `summary()` method should also be extended to return the `due_date` and the return values of the base `summary()` method.

Task 3.3
There are seven objects defined in the text file `TASK3_3.TXT`.

Amend your program code to:

- implement the `DatedToDo` class, with `constructor`, `get_due_date` and `set_due_date`
- implement the extended `compare_with()` method
- implement the extended `summary()` method
- ensure all seven objects in the text file `TASK3_3.TXT` are added to the list
- print out the list contents using the `summary()` method.

[12]

When a task in the to-do list has been completed, it should be removed.

Task 3.4
There are four completed tasks defined in the text file `TASK3_4.TXT`.
If any of the four tasks exists in the list, it should be removed.

Amend your program to:

- recreate the list of seven tasks from Task 3.3
- check if each of the four completed tasks in the text file `TASK3_4.TXT` exists in the list and:
    - remove it from the list if it does or
    - print a warning message if the completed task does not exist
- print out the list after all four objects have been processed.

[10]

4 A stack is used to store characters.

Task 4.1
Write program code to implement the stack and the operations specified.

Your code should allow operations to:

- push an item on to the stack
- pop an item off the stack
- determine the size of the stack. A size of zero indicates that the stack is empty.

[5]

The stack is to be used to identify if an arithmetic expression is balanced.

An expression is balanced if each opening bracket has a corresponding closing bracket.
Different pairs of brackets can be used. These are: [], () or {}.

This is an example of an expression that is balanced.

([8 – 1] / (5 * 7))

This is an example of an expression that is not balanced.

[(8 – 1] / (5 * 7))

Note the change in the order of the first two open bracket symbols. The first closing bracket should be a closing bracket ')' to match the previous opening bracket '('.

Note that an expression is not balanced if the order of the brackets is incorrect, even if there are the same number of opening and closing brackets of each bracket type.

An expression is checked by iterating over it:

- If a non-bracket symbol is found, continue to the next character.
- If an opening symbol is found, push it on to the stack and continue to the next character.
- If a closing bracket is encountered:
    - If the stack is empty, return an error (because there is no corresponding opening bracket)
    - else pop the symbol from the top of the stack and compare it to the current closing symbol to see if they make a matching pair
    - If they do match continue to the next character
    - else return an error (pairs of brackets must match).
- When the last symbol is encountered:
    - return an error if the stack is not empty (too many opening symbols)
    - else return a success message.

Task 4.2
Add **five** other suitable test cases and a reason for choosing each test case.

Test case	Reason for choice	Expected value
([8 – 1] / (5 * 7))	Provided	Succeeds
[(8 – 1] / (5 * 7))	Provided	Fails
		Succeeds
		Succeeds
		Fails
		Fails
		Fails

[6]

Task 4.3
Write program code that checks expressions using the given algorithm.
Use all **seven** test cases to verify it.

[19]