HWA CHONG INSTITUTION
C2 PRELIMINARY EXAMINATION 2018

COMPUTING
Higher 2

10 September 2018 **Paper 1 (9597 / 01)** **0815 -- 1130 hrs**

Additional Materials:
Electronic version of `IBANS.txt` data file
Electronic version of `TRANSACTIONS.txt` data file
Electronic version of `ACCOUNTS.txt` data file
Electronic version of `NAMES.txt` data file
Electronic version of `COMMANDS.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. A programmer is writing a treasure island game to be played on the computer.

The island is a rectangular grid, 30 squares by 10 squares. Each square of the island is represented by an element in a 2D array. The top left square of the island is represented by the array element [0, 0].

There are 30 squares across and 10 squares down.

The computer will:

- generate three random locations where treasure will be buried
- prompt the player for the location of one square where the player choose to dig
- display the contents of the array by outputting for each square:
  - `'.'` for only sand in this square
  - `'T'` for treasure still hidden in sand
  - `'X'` for a hole dug where treasure was found
  - `'O'` for a hole dug where no treasure was found.

Here is an example display after the player has chosen to dig at location [9, 3]:

```
..............................
..............................
..............................
..............................
................T.............
..............................
...T..........................
..............................
..............................
...X..........................
```

The game is to be implemented using object oriented programming.

The programmer has designed the class `IslandClass`. The identifier table for this class is:

Identifier	Data Type	Description
`Grid`	`ARRAY[0:9, 0:29] OF CHAR`	2D array to represent the squares of the island
`Constructor()`		instantiates an object of class `IslandClass` and initialises all squares to sand
`HideTreasure()`		generates a pair of random numbers used as the grid location of treasure and marks the square with `'T'`
`DigHole(Row, Column)`		takes as parameter a valid grid location and marks the square with `'X'` or `'O'` as appropriate

`GetSquare(Row, Column)`	`CHAR`	takes as parameter a valid grid location and returns the grid value for that square from the `IslandClass` object
`DisplayGrid()`		shows the current grid data. `DisplayGrid` should make use of the getter method `GetSquare` of the `IslandClass` class

Task 1.1
Write program code for the class `IslandClass` including the `Constructor`, `GetSquare` and `DisplayGrid` methods. The code should follow the specification given.

The value to represent sand should be declared as a constant.

Do not attempt to write the methods `HideTreasure` or `DigHole` at this stage.

Evidence 1
Program code for Task 1.1 [5]

Task 1.2
Write program code for the `HideTreasure` method. Your method should check that the random location generated does not already contain treasure.

The value to represent treasure should be declared as a constant.

Evidence 2
Your program code for Task 1.2 [3]

Task 1.3
Write a main program to:

- create an `IslandClass` object
- generate three random locations where treasures will be buried
- your program will then call the `DisplayGrid` method.

Evidence 3
The program code for Task 1.3 [3]

Evidence 4
Screenshot showing the output from running the program in Task 1.3 [1]

Task 1.4
Write program code for the `DigHole` method. This method takes two integers as parameters. These parameters form a valid grid location. The location is marked with `'X'` or `'O'` as appropriate.

The values to represent treasure, found treasure and hole should be declared as constants.

Evidence 5
Program code for Task 1.4. [3]

Task 1.5
Add code to the main program in Task 1.3.

The program is to:

- prompt the player for a location to dig
- validate the user input
- call the `DigHole` method and then the `DisplayGrid` method.

Evidence 6
The program code. [3]

Task 1.6
Run the program by inputting a location where:

- the treasure is not found
- the treasure is found.

Evidence 7
Screenshot evidence similar to that shown which shows:

- The player has chosen to dig at location [2, 25] where no treasure was found

```
. . . . . .
. . . .T . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . O. . . .
. . . . . .
. . . . . .T . . . . . . . . . . . . . . . . . .
. . . . . .
. . . . . . . . . . . . . . . . . .T . . . . . . . . . . .
. . . . . .
. . . . . .
. . . . . .
```

- The player has chosen to dig at location [5, 3] where treasure was found

```
. . . . . .
. . . . . .
. . . . . . . . . . . . . . T. . . . . . . . . . . . . .
. . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . T. . . .
. . . X. . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
```

[2]

2. The International Bank Account Number (IBAN) is an international bank account identification standard used in many countries. It uses modulo arithmetic to perform validation.

The IBAN consists of up to 34 alphanumeric characters, as follows:

- country code – two letters
- check digits – two digits, and
- basic bank account number – up to 30 alphanumeric characters that are country-specific

An example of an IBAN in the United Kingdom Great Britain which is 22 characters is `GB82WEST12345698765432` (`GB` - country code, `82` - check digits)

The following is the IBAN check digits generation algorithm:

1. Initialize the two check digits by `00` (e.g. `GB00WEST12345698765432`).

2. Move the four initial characters to the end of the string (e.g. `WEST12345698765432GB00`).

3. Replace each alphabet in the string with two digits, using the mapping A = 10, B = 11, C = 12, ... , Z = 35 (i.e. ASCII value of uppercase letters - 55) (e.g. `3214282912345698765432161100`).

4. Convert the string to an integer.

5. Calculate the remainder of dividing this number by 97 (e.g. `3214282912345698765432161100 mod 97 = 16`).

6. Subtract the remainder from 98 to give the two check digits (e.g. check digits = `98 - 16 = 82`). If the result is a single digit number, pad it with a leading 0 to make a two-digit number.

Task 2.1
Write program code for the `CheckDigits` function using the following specification.

```
FUNCTION CheckDigits (IBAN : STRING) : STRING
```

The function has a single string parameter `IBAN` and returns a two-digit string result.

Use the sample data provided in the text file `IBANS.txt` and paste this into your program code.

Evidence 8
Your program code. [6]

Evidence 9
One screenshot verifying that your program generates the correct check digits for the data in `IBANS.txt`. [1]

An IBAN is validated by converting it into an integer and performing a basic *mod-97* operation on it. If the IBAN is valid, the remainder equals 1.

The algorithm of IBAN validation is as follows:

1. Move the four initial characters to the end of the string
   (For the IBAN `GB82WEST12345698765432`
   e.g. `WEST12345698765432GB82`).

2. Replace each letter in the string with two digits, thereby expanding the string, where A = 10, B = 11, C = 12, ... , Z = 35
   (e.g. `3214282912345698765432161182`).

3. Interpret the string as a decimal integer and compute the remainder of that number on division by 97
   (e.g. `3214282912345698765432161182 mod 97 = 1`).

If the remainder is 1, the check digit test is passed and the IBAN might be valid.

Task 2.2
Write a Boolean function `ValidateIBAN` to determine if a given IBAN is valid. This function should have a parameter which allows it to be used for any IBAN.

Evidence 10
Your `ValidateIBAN` program code. [2]

Task 2.3
A `TRANSACTIONS.txt` file contains transaction details of customers of a bank. Each transaction record takes up one line and has three data fields: customer IBAN, transaction mode (`W` - withdrawal or `D` - deposit) and transaction amount.

Write a procedure `CheckIBAN` to read in the IBANs in `TRANSACTIONS.txt` and display on the screen:

- If an IBAN is valid, the valid IBAN followed by "`OK`".
- If an IBAN is invalid, the invalid IBAN followed by "`Invalid. Expected check digits: ??`", where `??` represents the computed check digits.
- For an invalid IBAN, update the record in `TRANSACTIONS.txt` with the expected computed check digits.

Evidence 11
Your `CheckIBAN` program code. [7]

Evidence 12
One screenshot showing the output and contents of `TRANSACTIONS.txt` from running the program. [2]

Task 2.4
A master file `ACCOUNTS.txt` contains the customer IBANs, names and current balances.

Write a procedure `UpdateBalance` to update the current balances of the customers in `ACCOUNTS.txt` from `TRANSACTIONS.txt`. At the end of the process, your program will output the message:

```
x records updated.
```

Evidence 13
Your program code for the procedure `UpdateBalance`. [8]

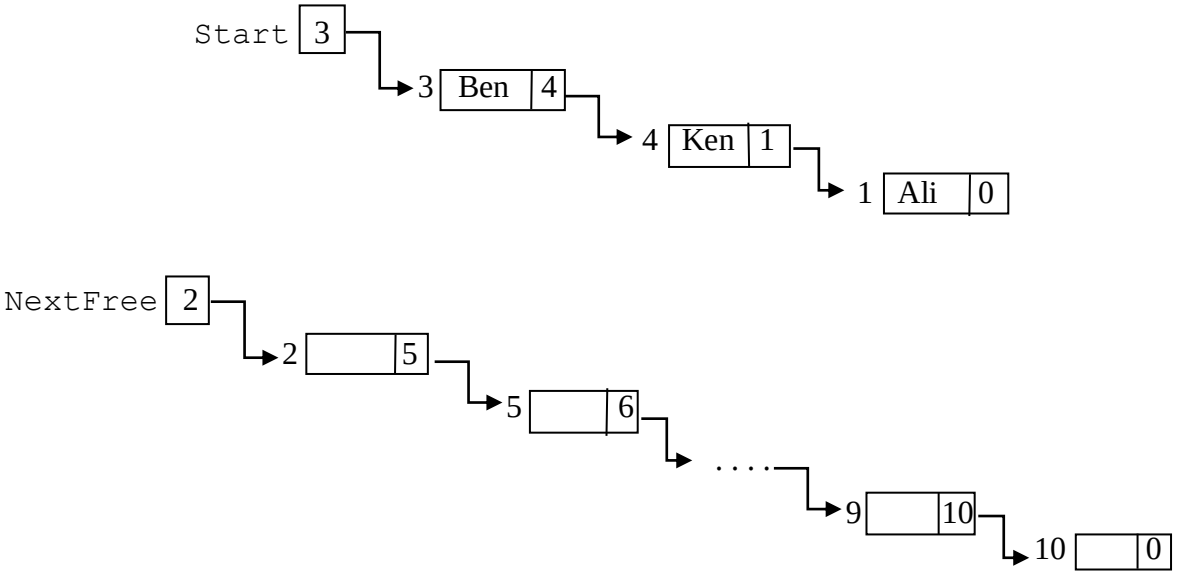
Evidence 14
One screenshot showing the program output and contents of `ACCOUNTS.txt` from running the program. [2]

3. A linked list Abstract Data Type (ADT) has the following operations defined:

- `Create()` -- creates an empty linked list;
- `Insert(item, p)` -- inserts new value, `item`, into linked list so that it is at position `p` in the linked list. Assume that the linked list contains at least `(p - 1)` items before the insertion.
- `Delete(p)` -- deletes the item at position `p` in the linked list;
- `Length()` -- returns the number of items in the linked list;
- `IsEmpty()` -- returns true if linked list is empty;
- `IsFull()` – returns true if linked list is full;

The linked list is implemented by the use of a collection of nodes that have two parts: the item data and a pointer to the next item in the list. In addition there is a `Start` pointer which points to the first item in the list.

The unused nodes are linked and the first unused node is the position where the next new data item is to be stored. Node removed from the linked list should be returned to `NextFree` list.



The diagram shows the linked list after the following sequence of commands have been executed.

```
Create()
Insert('Ali', 1)
Insert('Jack', 1)
Insert('Ben',2)
Delete(1)
Insert('Jane', 2)
Insert('Ken', 3)
Delete(2)
```

The program to implement this ADT will use the classes `ListNode` and `LinkedList` as follows:

ListNode
`Name : STRING` `Pointer : INTEGER`

`Constructor()`
`SetName(Name:STRING)`
`SetPointer(Pointer:INTEGER)`
`GetName():STRING`
`GetPointer():INTEGER`

LinkedList

`Node : ARRAY [1..10] OF ListNode`
`Start : INTEGER`
`NextFree : INTEGER`

`Constructor()`
`Insert (name: STRING, position: INTEGER)`
`Delete (position: INTEGER)`
`Length (): INTEGER`
`IsEmpty(): BOOLEAN`
`IsFull() : BOOLEAN`

Task 3.1
Write program code to define the classes `ListNode` and `LinkedList`.

Evidence 15
Program code for the `ListNode` and `LinkedList` classes. [18]

Task 3.2
A method `Display()` is to be added, which displays the value of `Start`, the value of `NextFree` and the contents of `Node` array in index order. Write program code to implement this method.

Evidence 16
Your program code for Task 3.2. [4]

Task 3.3
Write code to create a `LinkedList` object in the main program. Paste the sequence of commands in `COMMANDS.txt` into your program code. Your program will then call the `Display` method.

Execute your program to test it.

Evidence 17
Screenshot showing the output from running the program in Task 3.3. [2]

A linear queue is implemented using the `LinkedList` class as a super class.

The subclass `Queue` has the following methods:

- `Enqueue(item)` -- inserts `item` at the rear of the queue;
- `Dequeue()` -- deletes the item at the front of the queue;
- `Display()` -- displays the contents of the queue using the format given below.

```
Steven   ← Front
Celine
Tom
Ryan     ← Rear

Number in queue:  4
```

Task 3.4
Write program code for the subclass `Queue`.
Use appropriate inheritance and polymorphism in your design.

Evidence 18
Your program code for Task 3.4. [5]

Task 3.5
Write program code to:

- create a new queue and add the data in the file `NAMES.txt` to the queue
- remove two items from the queue
- display final contents of the queue

Evidence 19
Your program code for Task 3.5. [2]

Evidence 20
Screenshot showing the output from running the program in Task 3.5. [1]

4. The Romans had their own system of number representation which used a sequence of upper case letter characters to represent a number. We shall consider the denary number 1 to 20 only.

The following letters represent each of the values shown:

Roman Numeral	Represents
I	One
V	Five
X	Ten
L	Fifty

A number is always written with the smallest number of characters, with the letters in sequence starting with the character with the largest value.

- For example, 6 is written VI (not IIIIII)

The exceptions to this sequence are as follows:

- one less than 5 -- which is written as IV
- one less than ten -- which is written as IX
- ten less than fifty -- which is written as XL

Task 4.1
Write program code with the following specification:

- Input a denary integer number in the range 1 to 20
- Validate the input
- Calculate the Roman numeral representation (write this code as a function)
- Output the Roman number.

Evidence 21
Your program code. [6]

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

Evidence 22
Annotated screenshots for each test data run. [3]

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

- Input two Roman numeral numbers between 1 and 20
- Ouput the sum of the numbers as a Roman numeral number.

Evidence 23
Your program code. [8]

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

Evidence 24
Annotated screenshots for each test data run. [3]