HWA CHONG INSTITUTION
C2 PRELIMINARY EXAMINATION 2014

COMPUTING
Higher 2

15 September 2014 **Paper 1 (9597 / 01)** **0815 -- 1130 hrs**

Additional Materials:
Electronic version of `DATA.txt` data file
Electronic version of `HIGHEST.txt` data file
Electronic version of `MORSE.txt` data file
Electronic version of `SPORT+NAME+TEAM.txt text` 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 screen shots 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 program is to process daily high scores recorded from an online game. The program can be run every day.

All scores are integers in the range 1 to 500.

The program reads from file `HIGHEST.txt` the current highest score and player name from running the program on previous days.

The program specification is to:

- input **up to** 5 player names and their corresponding scores for today
- calculate and display on screen:
  - the highest score with the player name for today
  - a message saying whether or not the highest score today beat the current highest score
- update the file `HIGHEST.txt` if a higher score was input 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 functioning of your program. Consider carefully all cases which could occur for both the scores input and the two processing requirements.

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

A ***palindrome*** is an integer that reads the same backwards and forwards -- so 6, 11 and 121 are all palindromes, while 10, 12, 223 and 2244 are not (even though 010=10, we don't consider leading zeroes when determining whether a number is a palindrome).

Task 1.3
Write program code with the following specification:

- input two integers -- the endpoints of an interval e.g. `10  120`
- output the number of integers that are palindromes in the interval.

Evidence 3: Your program code. [8]

Evidence 4: Produce two screenshots showing the output of `1  4` and `10  120` by the user. [2]

2. The following is a pseudocode algorithm for the quick sort function.

The function takes in an array and returns a sorted array in **ascending** order. The algorithm works for most input but is missing out on a few edge cases.

```
FUNCTION quicksort(array)
     if array length = 1
          return array
     select and remove a pivot value from array
     create empty subarrays less and greater
     for each item in array
          if item < pivot
               append item to less
          else
               append item to greater
     return quicksort(less) + pivot + quicksort(greater)
```

Task 2.1
Write program code for this algorithm including all the amendments you would make to:
- make the function work for all cases
- adhere to good programming style

Use the `Team` array sample data available from text file `SPORT+NAME+TEAM.txt` and paste this into your program code.

Evidence 5: Your program code. [6]

Evidence 6: One screenshot showing the output from running the program code for array `Team`. [1]

The following is a pseudocode algorithm of a binary search written by a student to search for an item in an array `Sport`. This array stores string data arranged in **ascending** order and has a final subscript `MAX`.

The algorithm is poorly designed and also contains errors.

```
Enter k
A = 1
B = MAX
While not found
      C = (A+B) DIV 2
      If Sport(C) = k
           Print "Found"
      Else If Sport(C) < k
           B = C-1
      Else
           A = C+1
      End-If-Else
End-While
```

Task 2.2
Write program code for this algorithm including all the changes you would make to:

- follow good programming practice
- make the algorithm work properly

Use the sample array data available from text file `SPORT+NAME+TEAM.txt` and paste this into your programming code.

Evidence 7: Your program code. [7]

Task 2.3
The binary search code could be useful for many programs where a search routine is required.
Re-design the program code to have a procedure `BinarySearch`. This procedure should have parameters which allow it to be used for any sorted array of string data.

Use the data provided in the array `Name` from text file `SPORT+NAME+TEAM.txt` and test the procedure with appropriate test cases to ensure it is working properly.

Evidence 8: Your amended program code. [3]

Evidence 9: A screenshot for each appropriate test case. [3]

3. Morse Code is a type of code in which letters are represented by combinations by long or short signals, e.g. the letter ‘A’ is represented by a short followed by a long signal, i.e. ‘.-‘

Here we use the period (‘.’) to represent the short signal and the dash (‘-‘) to present the long signal. The Morse code equivalent for the letters ‘A’ to ‘Z’ is provided in the file `MORSE.txt`. In this implementation, we use a space (‘ ‘) to simulate the inter-character gap and the slash (‘/’) to represent the inter-word gap.

Task 3.1
Write the program code for a function that will convert a given word into its Morse Code equivalent using the following specification.

```
FUNCTION ConvertWord(SingleWord : STRING): STRING
```

The function has a single parameter `SingleWord` and returns the Morse Code equivalent for that word as a string.

Evidence 10: Your `ConvertWord` program code. [7]

Evidence 11: A screenshot showing the correct Morse Code equivalent for the word “COMPUTING”. [1]

Task 3.2
Write the program code that does the following:
- the user inputs a sentence of words not exceeding 5 words.
- uses the function `ConvertWord` in Task 3.1
- outputs the Morse Code equivalent of the sentence of words that was entered.

Evidence 12: Your program code. [6]

Task 3.3
Draw up **two** suitable tests to assess that your program is working properly, explaining the reason for each test and provide screenshot evidence for your testing.

Evidence 13: Annotated screenshots for each test data run. [4]

Task 3.4
Write the program code that will do the following:
- the user enters a word string in Morse Code
- the program converts the Morse Code word to its alphabetical equivalent
- outputs the converted word.

Evidence 14: Your program code. [6]

Evidence 15: A screenshot showing the correct word equivalent for the following Morse Code string, “... `---` ...” [1]

4. Many programming languages use a variety of brackets eg `()`, `{}` and `[]` in their program statement constructs. Brackets can be nested e.g. `(())` and `{[]}`. A well-formed construct must contain a matched set of optionally nested brackets e.g. `(())`, `{[]}` and `(){}[]` are well-formed whereas `(()`, `{]` and `)(` are not.

An efficient way to check if a program construct is well-formed is to use a stack to scan the brackets once from left to right as follows:

Start from an empty stack. Whenever we encounter an open bracket, push it into the stack. Whenever we encounter a close bracket, we check if it is of the same type with the item at the top of the stack. Once we have a match, we pop the topmost bracket from the stack. Only when we managed to read the last bracket and find that the stack is empty, then we know that the brackets are properly nested, and the program construct is well-formed.

The stack ADT contains character data and a top pointer, and has the following operations:

`Create()` : initializes a new stack
`Push(item)` : add `item` to top of stack
`Pop()` : remove item from top of stack
`Peep()` : inspect the topmost item of stack
`isEmpty()` : check if stack is empty

Task 4.1
Using object-oriented programming techniques, implement the stack ADT using the class name `Stack`.

Evidence 16: Your `Stack` class program code. [8]

Task 4.2
Write a function `CheckNested(construct)` which accepts a string program construct of nested brackets and returns a Boolean result indicating if construct is well-formed.

Evidence 17: Your `CheckNested` function code. [10]

Task 4.3
Test your `CheckNested` function using the data file provided in `DATA.txt`. If a construct is not well-formed, append it to a text file `ERRORS.txt`.

Evidence 18: Screenshot of `ERRORS.txt`. [1]

Task 4.4
Write a function `CheckWellformed(construct)` which contains a modified and enhanced version of `CheckNested(construct)`. When a first mismatch of bracket occurs, output the diagnostic message "`Expecting '?'`", where `?` is the type of bracket expected. For example, for the construct `())`, output the diagnostic message "`Expecting '('`".

Evidence 19: Your `CheckWellformed` function code. [10]

Task 4.5
Test your `CheckWellformed` function using `ERRORS.txt`. Display the diagnostic messages to screen.

Evidence 20: Screenshot of diagnostic messages on screen. [1]

-- END OF PAPER ---