JURONG PIONEER JUNIOR COLLEGE

JC2 Common Test 2019

COMPUTING **9597/01**

Higher 2 **26 March 2019**

Paper 1 **3 hours**

Additional materials: Removable storage device
Electronic version of MARKS.txt data file
Electronic version of CARD.txt data file
Electronic version of EVIDENCE.docx file
Cover page

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, save your EVIDENCE.docx in your removable storage device. You will be called up to print your work by the invigilator.

At the end of the examination, fasten all your work securely together.

This document consists of **9** printed pages.

[Turn over

1. The data file `MARKS.txt` contains information about students and the marks of four exams they have taken. The format of a student record is as follows:

```
<Student Name>,<exam 1>,<exam 2>,<exam 3>,<exam 4>
```

Sample record:

```
LIM BOON KEE,81.3,75.5,72.6,78.0
```

Task 1.1

Write program code to output the full list of students with their total exam marks, *sorted in descending order of total exam marks*. Use of Python *sort*, *min*, or *max* method carries no credit. Display in this format:

```
No.     Name of student          Total exam marks
1.      ... ...                  ...
2.      ... ...                  ...
...     ... ...                  ...
```

Evidence 1: Your program code for task 1.1. **[9]**

Evidence 2: Screenshot of running task 1.1. **[1]**

The civics tutor of the class would like to find out the names of the students who have not achieved `x`, the minimum number of marks entered by the civics tutor.

Task 1.2

Write program code to output a list of students who have completed less than `x` exam marks, and the total number of students in this list. Get user input for value of `x` and perform validation of `x`.

Evidence 3: Program code for task 1.2. **[5]**

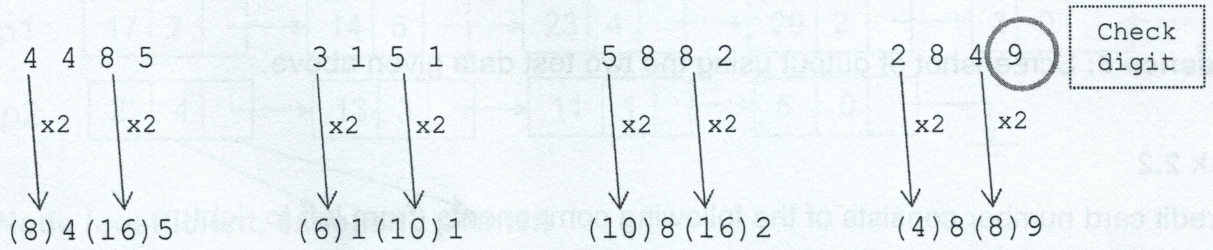
Evidence 4: Screenshot of output for value of `x` = 200. **[1]**

(16 marks)

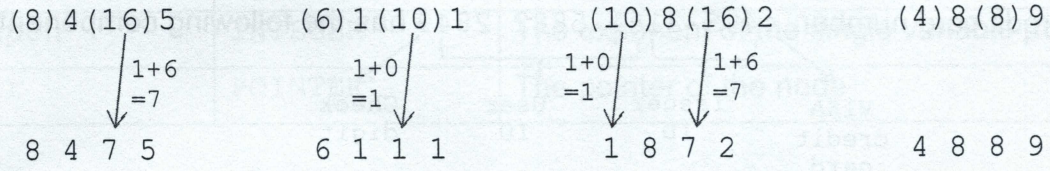
2. You have been engaged by an online merchant to write a program to validate customers' supplied credit card numbers. Consider only VISA and MasterCard credit card numbers. A VISA credit card number has **13** or **16** digits and begins with the digit **4**. A MasterCard credit card number has **16** digits and begins with any number from **51** to **55**.

Each credit card number also contains a final check digit appended according to the Luhn algorithm, illustrated by the following example:

VISA credit card number is **`4485 3151 5882 2849`**.
Starting from rightmost digit (exclude check digit), **double** every alternate number (result shown in brackets):



For all two-digit numbers, add the two digits to get one resulting single digit:



Sum up all single digits to get the total.

`8+4+7+5 + 6+1+1+1 + 1+8+7+2 + 4+8+8+9 = 80`

`80 is a multiple of 10.`
`Hence credit card number is valid.`

If total is a multiple of 10, then the credit card number is valid. Otherwise, it is invalid.

Task 2.1

Write a function to determine if a **user input** VISA or MasterCard credit card number is valid using the following specification:

```
FUNCTION validate_creditcard(card_number:STRING): BOOLEAN
```

Parameter of function is credit card number and ***return value*** is of Boolean type

Use the following credit card numbers to test your function `validate_creditcard`:

VISA: `4384 3760 0096 4231`

MasterCard: `5402 0528 1417 5528`

These credit card numbers can be found in the given file `CARD.txt`.

Evidence 5: Your program code for task 2.1. **[7]**

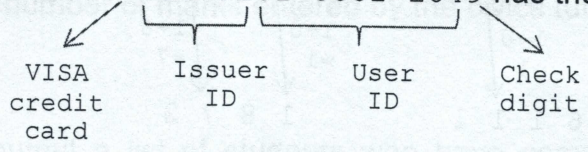
Evidence 6: Screenshot of output using the two test data given above. **[2]**

Task 2.2

A credit card number consists of the following components (from left to right):

- Digit 1: Industry identifier, e.g. 4 for VISA, 5 for MasterCard
- Digits 2-6: Issuer ID, indicates the bank that issues the credit card
- Digits 7 to n-1: User ID, where n is the total number of digits
- Last digit: Check digit

For example, credit card number `4485 3151 5882 2849` has the following components:



A bank has issuer ID `34922`, and issues a 16-digit VISA credit card number. A sample credit card issued by the bank has digits `4349 22xx xxxx xxxx`, where `x` represents digits 0 to 9. Write a procedure `gen_creditcard` that generates the first 20 valid credit card number for this bank, starting from the number `4349 2200 0000 0000` and count upwards.

Evidence 7: Your program code for task 2.2. **[5]**

Evidence 8: Screenshot of output from running `gen_creditcard`. **[1]**

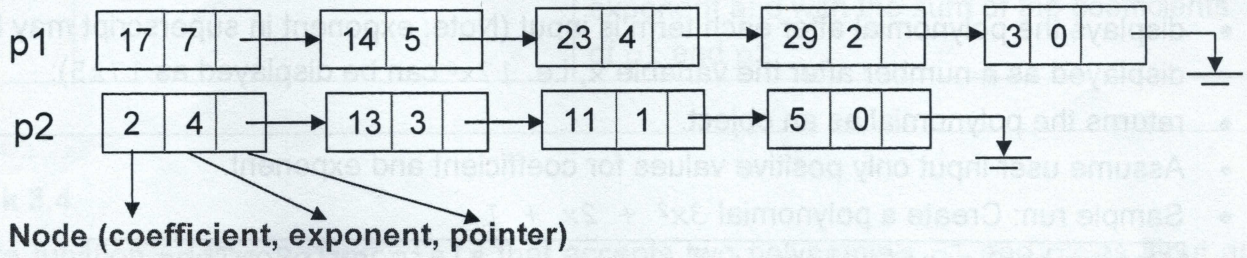
(15 marks)

3. A single variable polynomial can be generalised as $f(x) = \sum_{i=0}^{n} a_i x^i$. An example of a single variable polynomial: $3x^8 + 8x^5 - 5x + 2$. This polynomial has order 8, which is the highest exponent.

The task is to implement a single variable polynomial in a ***dynamic linked list***. Each node contains a ***coefficient***, an ***exponent***, and a ***pointer***. For example, two polynomials p1 and p2 with single variable `x` are shown below.

$p1 = 17x^7 + 14x^5 + 23x^4 + 29x^2 + 3$

$p2 = 2x^4 + 13x^3 + 11x + 5$



The program will use a user-defined type **`Node`** for each node defined as follows:

Identifier	Data type	Description
`coeff`	`FLOAT`	The coefficient of the single variable polynomial
`expon`	`INTEGER`	The exponent of the single variable polynomial
`ptr`	`POINTER`	The pointer of the node

The program will also use a user-defined type **`Polynomial`**, which has a **`start`** pointer, for each single variable polynomial.

Both `Node` and `Polynomial` have methods to ***display*** the coefficient and exponent of the single variable polynomial.

Task 3.1

Write program code to declare **`Node`** and **`Polynomial`** with the required variables and display methods.

Evidence 9: Your program code for task 3.1. **[11]**

Task 3.2

Write a function **`create_polynomial`** that:

- allows user to create a ***single variable polynomial***, using the user-defined types `Node` and `Polynomial`;
- gets user input of coefficient and exponent for each term (node) of the polynomial, starting with the highest exponent. For example, to input the polynomial $p1 = 17x^7 + 14x^5 + 23x^4 + 29x^2 + 3$, user will input coefficient `17` followed by exponent `7`, then `14` and `5`, then `23` and `4`, then `29` and `2`, then `3` and `0`;
- displays the polynomial after each term is input (Note: exponent in superscript may be displayed as a number after the variable `x`, i.e. $17x^5$ can be displayed as `17x5`);
- returns the polynomial as an object.
- Assume user input only positive values for coefficient and exponent.
- Sample run: Create a polynomial $3x^2 + 2x + 1$

```
Enter the coefficient : 3
Enter the exponent : 2
+3.0x2
Add another term [Yes/No]? y
Enter the coefficient : 2
Enter the exponent : 1
+3.0x2+2.0x1
Add another term [Yes/No]? y
Enter the coefficient : 1
Enter the exponent : 0
+3.0x2+2.0x1+1.0x0
Add another term [Yes/No]? n
Polynomial is +3.0x2+2.0x1+1.0x0
```

Evidence 10: Your program code for task 3.2. **[10]**

Task 3.3

Test your program by calling **`create_polynomial`** to input polynomial $17x^7+14x^5+32x^4+29x^2+3$ and return the result to `p1`. Display `p1` on the screen.

Evidence 11: Program code in task 3.3 and screenshot of running your program code. **[3]**

Algorithms can be designed for addition, subtraction, multiplication, differentiation and integration of polynomials stored in linked list.

When ***adding*** polynomials using a linked list representation, the process may be broken down into three cases by comparing the exponents of the terms. Assume the polynomials contain terms with the highest exponent on the left and smallest exponent on the right. Assume polynomials ***p1*** and ***p2*** are added and the result stored in ***p3***.

Case	Condition	Action
1	exponent of `p1` > exponent of `p2`	Copy node of `p1` to end of `p3`
2	exponent of `p1` < exponent of `p2`	Copy node of `p2` to end of `p3`
3	exponent of `p1` = exponent of `p2`	Create a new node in `p3` with the same exponent and with the sum of the coefficients of `p1` and `p2`

Task 3.4

Write function **`addTwoPolynomials`** that accepts two polynomials `p1` and `p2` as input and returns `p3` as a result of adding.

Evidence 12: Your program code for task 3.4. **[13]**

Task 3.5

Write program code to add two polynomials `p1` and `p2` by calling **`addTwoPolynomials`** and display the result `p3` on the screen.

$p1 = 3x^2 + 5x + 9$

$p2 = 4x^4 + 8x^3 + 2.2x^2 + 7.8x + 6$

$p3 = 4x^4 + 8x^3 + 5.2x^2 + 12.8x + 15$

Evidence 13: Your program code for task 3.5 and screenshot for running task 3.5. **[3]**

(40 marks)

4. Tic-tac-toe is a game in which two players take turns marking X and O in the spaces in a 3×3 grid. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins the game.

For example, player who marks X wins this game.

X	O	
X	X	X
O		O

The task is to write program code for a tic-tac-toe game for two human players.

Task 4.1

Decide on the design to be used for:

- the data structure to represent the 3x3 grid, using the identifier **`board`**
- the contents of the marks (X, O and blank) in the spaces
- how user input (X or O) in the spaces

Evidence 14: Your program design. Include program code for **`board`**. **[4]**

Task 4.2

Write a module **`displayBoard`** that will display the game board clearly to the players. You should use the **`board`** (defined in task 4.1) as a parameter in **`displayBoard`**.

Write another module **`displayInstructions`** to inform players on how to input X and O in the spaces on the game board.

Evidence 15: Your modules **`displayBoard`** and **`displayInstructions`**. **[4]**

Task 4.3

Write a function **`getPlayerMove`** to get players to make their move (by marking X or O) on the board. You should include validation on user input and check that the space is not already occupied. Use **`board`** as a parameter. You may include any other suitable parameters.

Evidence 16: Your function **`getPlayerMove`**. **[5]**

Task 4.4

Write a function `checkWin` that checks all the conditions for winning a game and returns True if a player has won the game, otherwise return False. Use `board` as a parameter. You may include any other suitable parameters.

Evidence 17: Your function `checkWin`. **[5]**

Task 4.5

Write a module `main` that makes use of the modules and functions from task 4.1 to task 4.4 and allows two human players to play a game of tic-tac-toe.

The module `main` should include the following:

- give instructions to players on how to input X or O;
- ask whether player 1 or player 2 starts first move;
- ensure players 1 and 2 take turns to make their move;
- display the game board **after every move made by a player**;
- check for winner
- display message on which player has won the game or whether the game ends in a draw

Evidence 18: Your module `main`. **[8]**

Evidence 19: Run your `main` module and produce screenshots of **three** games where player 1 wins one game, player 2 wins another game, and a drawn game. **[3]**

[Total: 29 marks]

END OF PAPER