Candidate Name: _____ CT Group: _____

Index no. _____

PIONEER JUNIOR COLLEGE
JC 2 PRELIMINARY EXAMINATION

H2 COMPUTING **9597/01**

Paper 1

Tuesday **13 SEP 2016** **3 hours 15 min**

TIME 0800 - 1115

Additional Materials:

- Removable storage device
- Question 1: `AIRCON.txt, BLIND.txt, COMP.txt,EMPEROR.txt`
- Question 2: `ENCRYPT_KEY.txt, PASSWORDS.txt`
- Question 4: `SUBJECT.TXT`
- `EVIDENCE.docx`

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.

---

1. Four stories are given in four data files: `AIRCON.txt, BLIND.txt, COMP.txt, EMPEROR.txt`. The task is to find the frequency of alphabets used in the stories.

Task 1.1

Count the alphabets in each story and display the frequency of each alphabet in a table form. Do not distinguish between upper and lower cases. Do not count non-alphabets. Frequency should be displayed in descending order. Display alphabets with same frequency in ascending order.

Sample output:

Alphabet	Frequency
e	20
t	14
a	12
i	12
o	10
u	6
g	3
j	3
z	1

a and i have same frequency of 12, display a before i, since a comes before i.

g and j have same frequency of 3, display g before j, since g comes before j.

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

Evidence 2: Screenshot of running `AIRCON.txt` data file. **[1]**

Task 1.2

Amend your program code to display alphabets and frequencies in **all four** files in a table form, with alphabets in first column and frequencies of alphabets in subsequent columns.

Display in alphabetical order as shown in following sample output:

Alphabet	AIRCON.txt	BLIND.txt	COMP.txt	EMPEROR.txt
a	115	471	604	490
b	25	87	111	56
c	46	117	230	107
⋮	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	⋮
x	3	6	9	0
y	38	121	128	130
z	2	3	4	4

Evidence 3: Your amended program code for task 1.2. **[8]**

Evidence 4: Screenshot of running your program code. **[1]**

2. Write a program to encrypt and decrypt user passwords.

Task 2.1

A set of encryption key is given in the data file `ENCRYPT_KEY.txt`. The encryption key will map each alphabet and number according to the table below.

a	→	m	m	→	q	y	→	c
b	→	h	n	→	s	z	→	a
c	→	t	o	→	l	0	→	7
d	→	f	p	→	n	1	→	3
e	→	g	q	→	i	2	→	8
f	→	k	r	→	u	3	→	9
g	→	b	s	→	o	4	→	5
h	→	p	t	→	x	5	→	6
i	→	j	u	→	z	6	→	0
j	→	w	v	→	y	7	→	1
k	→	e	w	→	v	8	→	4
l	→	r	x	→	d	9	→	2

For example, `a` is mapped to `m`, `b` mapped to `h`, `m` mapped to `q`, `9` mapped to `2`, etc.
The mapping is case-sensitive for alphabets.
Therefore a password `WhizKid123` will be encrypted to `VpjaEjf389`.

Conversely, decryption works in the opposite way. Hence the password `VpjaEjf389` will be decrypted to `WhizKid123`.

There is no mapping for symbols. Some examples of symbols are `!, @, #, $, %, ^.` Hence, a password `Extr@123` will be encrypted to `Gdxu@389`, where the symbol remains the same.

Copy and paste the encryption key (from data file `ENCRYPT_KEY.txt`) ***into your program code*** and ***make use of it*** to write a program that

- Allows a user to select option to ***encrypt*** or ***decrypt*** a user password
- Gets user input of the password,
- Encrypts (or decrypts) the password according to user's option,
- Displays the encrypted or decrypted password

Evidence 5: Your program code for task 2.1. **[8]**

Evidence 6: Screenshot of running program by encrypting `WhizKid123` and decrypting `XgteV^cg84`. **[1]**

Task 2.2

Amend your program code into a function that accepts **two** parameters – `"password"` and `"encrypt"`, and returns the encrypted or decrypted password.

```
FUNCTION Cryptograph(password:STRING, encrypt:BOOLEAN):RETURN STRING
```

Evidence 7: Your program code for task 2.2. **[4]**

Task 2.3

Write program code that makes use of the function `Cryptograph` from task 2.2 that reads all the passwords in data file `PASSWORDS.txt`, encrypts and writes them into another file `CONVERTED.txt`.

Sample input file:

```
WhizKid123
Extr@123
```

Sample output file:

```
VpjaEjf389
Gdxu@389
```

Evidence 8: Program code for task 2.3. **[3]**

Evidence 9: Screenshot of output file `CONVERTED.txt`. **[1]**

3. 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 3.1

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

Task 3.2

Write a function **`displayBoard`** that will display the game board clearly to the players. You should use the **`board`** as a parameter in **`displayBoard`**.

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

Evidence 11: Your functions **`displayBoard`** and **`displayInstructions`**. **[4]**

Task 3.3

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 12: Your function **`getPlayerMove`**. **[5]**

Task 3.4

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 13: Your function **`checkWin`**. **[5]**

Task 3.5

Write a **`main`** function that makes use of the identifiers and functions from task 3.1 to task 3.4 and allows two human players to play a game of tic-tac-toe.
The **`main`** function 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 14: Your **`main`** function. **[8]**

Evidence 15: Run your **`main`** function and produce screenshots of **three** games where player 1 wins one game, player 2 wins another game, and a drawn game. **[3]**

4. A college uses a binary tree structure to store its subjects offered to students.

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

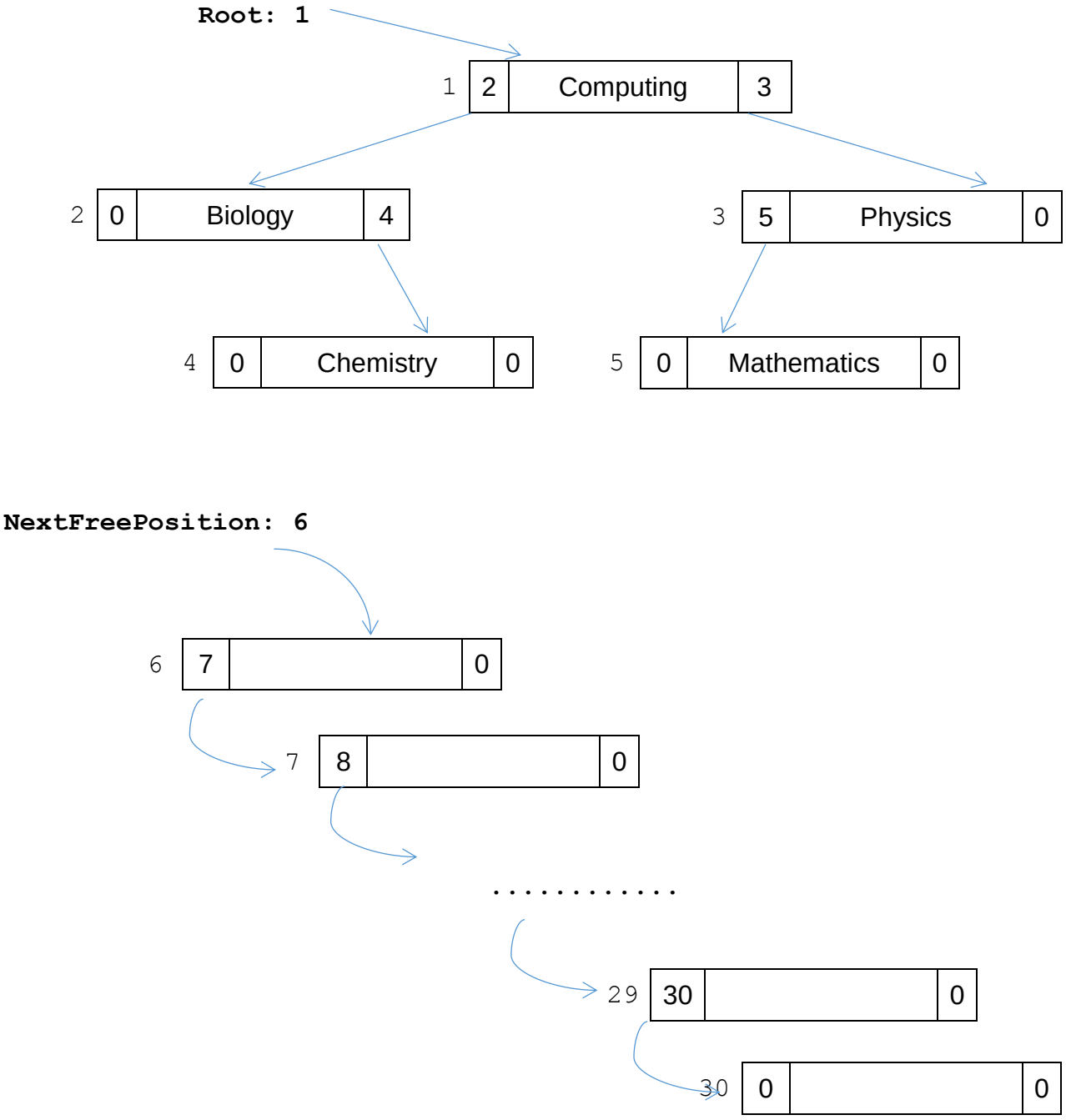
Identifier	Data type	Description
`Subject`	`STRING`	The node's value for subject offered
`LeftPtr`	`INTEGER`	The left pointer for the node
`RightPtr`	`INTEGER`	The right pointer for the node

A linked list is maintained of all unused nodes which do not form part of the tree. The first available node which is used for a new item is indicated by **`NextFreePosition`**. Items in the unused list are linked using their left pointers.

The binary tree and linked list are implemented using variables as follows:

Identifier	Data type	Description
`SubjectTree`	`ARRAY[30] of Node`	The tree data
`Root`	`INTEGER`	Index position of the root node
`NextFreePosition`	`INTEGER`	Index for the next unused node

The diagram shows the binary tree and linked list after five subjects have been added.



Task 4.1

Write the program code to declare all the required variables and create the initial linked list which contains all 30 nodes.

Add statement(s) to initialise the empty tree.

Evidence 16: Your program code for task 4.1. **[8]**

The following incomplete pseudocode inserts a data value into the binary tree structure.

```
PROCEDURE InsertBinaryTree(NewItem)
    ... ...
    IF tree is empty
        THEN
            ... ...
        ELSE
            //traverse the tree to find the insert position
            ... ...
            LastMove = 'X'
            REPEAT
                PreviousPtr ← CurrentPtr
                IF NewItem < CurrentPtr item
                    THEN
                        //move left
                        CurrentPtr ← CurrentPtr's left
pointer
                        LastMove = 'Left'
                    ELSE
                        //move right
                        CurrentPtr ← CurrentPtr's right
pointer
                        LastMove = 'Right'
                ENDIF
            UNTIL CurrentPtr = NULL
    ENDIF
    IF LastMove = 'Left'
    ... ...
    ELSE IF LastMove = 'Right'
    ... ...
    ENDIF
    ... ...
END PROCEDURE
```

Task 4.2

Complete the pseudocode and write a module `AddSubject` to add a subject into the binary tree.

Evidence 17: Your `AddSubject` program code. **[7]**

Task 4.3

Write a module `Display` to display the value of `Root`, the value of `NextFreePosition` and the contents of `SubjectTree` in index order.

Evidence 18: Your `Display` program code. **[4]**

Task 4.4

Write a module `BuildTree` to construct a binary tree using the data provided in the data file `SUBJECT.TXT`. Read in all the data from the data file and use the `AddSubject` module.

Evidence 19: Your `BuildTree` program code. **[2]**

Evidence 20: Run your program and produce screenshot of contents of binary tree. **[1]**

Deleting a node from a tree may change the structure of the tree. To simplify the deletion process, label a node as "**`deleted`**" but do not remove the node from the tree structure. After that, regenerate the entire tree structure.

Task 4.5

Write a module `LabelDelete` that labels a node as "**`deleted`**" but **do not remove** the node from the tree structure.

Evidence 21: Your `LabelDelete` program code. **[6]**

Task 4.6

Write program code to regenerate the entire binary tree.

Evidence 22: Your program code to regenerate the binary tree. **[6]**

Evidence 23: Display a screenshot of the regenerated binary tree after deleting "`Chemistry`" and "`History`". **[1]**

END OF PAPER