Cambridge International AS & A Level

Cambridge International Examinations
Cambridge International Advanced Subsidiary and Advanced Level

CANDIDATE NAME

CENTRE NUMBER | | | | | |

CANDIDATE NUMBER | | | | |

6366267429

COMPUTER SCIENCE **9608/42**

Paper 4 Further Problem-solving and Programming Skills **May/June 2017**

2 hours

Candidates answer on the Question Paper.

No Additional Materials are required.

No calculators allowed.

READ THESE INSTRUCTIONS FIRST

Write your Centre number, candidate number and name in the spaces at the top of this page.
Write in dark blue or black pen.
You may use an HB pencil for any diagrams, graphs or rough working.
Do not use staples, paper clips, glue or correction fluid.
DO **NOT** WRITE IN ANY BARCODES.

Answer **all** questions.
No marks will be awarded for using brand names of software packages or hardware.

At the end of the examination, fasten all your work securely together.
The number of marks is given in brackets [ ] at the end of each question or part question.

The maximum number of marks is 75.

This document consists of **17** printed pages and **3** blank pages.

© UCLES 2017

CAMBRIDGE International Examinations

1 The following table shows part of the instruction set for a processor which has one general purpose register, the Accumulator (ACC), and an Index Register (IX).

Instruction		Explanation
Op code	**Operand**	
LDM	#n	Immediate addressing. Load the number n to ACC.
LDD	<address>	Direct addressing. Load the contents of the location at the given address to ACC.
LDI	<address>	Indirect addressing. The address to be used is at the given address. Load the contents of this second address to ACC.
LDX	<address>	Indexed addressing. Form the address from <address> + the contents of the index register. Copy the contents of this calculated address to ACC.
STO	<address>	Store the contents of ACC at the given address.
INC	<register>	Add 1 to the contents of the register (ACC or IX).
CMP	<address>	Compare the contents of ACC with the contents of <address>.
JMP	<address>	Jump to the given address.
JPE	<address>	Following a compare instruction, jump to <address> if the compare was True.
JPN	<address>	Following a compare instruction, jump to <address> if the compare was False.
AND	<address>	Bitwise AND operation of the contents of ACC with the contents of <address>.
XOR	<address>	Bitwise XOR operation of the contents of ACC with the contents of <address>.
OR	<address>	Bitwise OR operation of the contents of ACC with the contents of <address>.
IN		Key in a character and store its ASCII value in ACC.
OUT		Output to the screen the character whose ASCII value is stored in ACC.
END		Return control to the operating system.

(a) A programmer writes a program that:

- reads two characters input from the keyboard (you may assume they will be capital letters in ascending alphabetical sequence)
- outputs the alphabetical sequence of characters from the first to the second character. For example, if the characters ‘B’ and ‘F’ are input, the output is:

  BCDEF

© UCLES 2017

The programmer has started to write the program in the following table. The Comment column contains descriptions for the missing program instructions, labels and data.

Complete the following program. Use op codes from the given instruction set.

Label	Op code	Operand	Comment
START:			// INPUT character
			// store in CHAR1
			// INPUT character
			// store in CHAR2
			// initialise ACC to ASCII value of CHAR1
			// output contents of ACC
			// compare ACC with CHAR2
			// if equal jump to end of FOR loop
			// increment ACC
			// jump to LOOP
ENDFOR:	END		
CHAR1:			
CHAR2:			

[9]

(b) The programmer now starts to write a program that:

- converts a positive integer, stored at address NUMBER1, into its negative equivalent in two's complement form
- stores the result at address NUMBER2

Complete the following program. Use op codes from the given instruction set.
Show the value stored in NUMBER2.

Label	Op code	Operand	Comment
START:			
		MASK	// convert to one's complement
			// convert to two's complement
	END		
MASK:			// show value of mask in binary here
NUMBER1:	B00000101		// positive integer
NUMBER2:			// negative equivalent

[6]

2 An ordered binary tree Abstract Data Type (ADT) has these associated operations:

- create tree
- add new item to tree
- traverse tree

The binary tree ADT is to be implemented as a linked list of nodes.

Each node consists of data, a left pointer and a right pointer.

(a) A null pointer is shown as Ø.

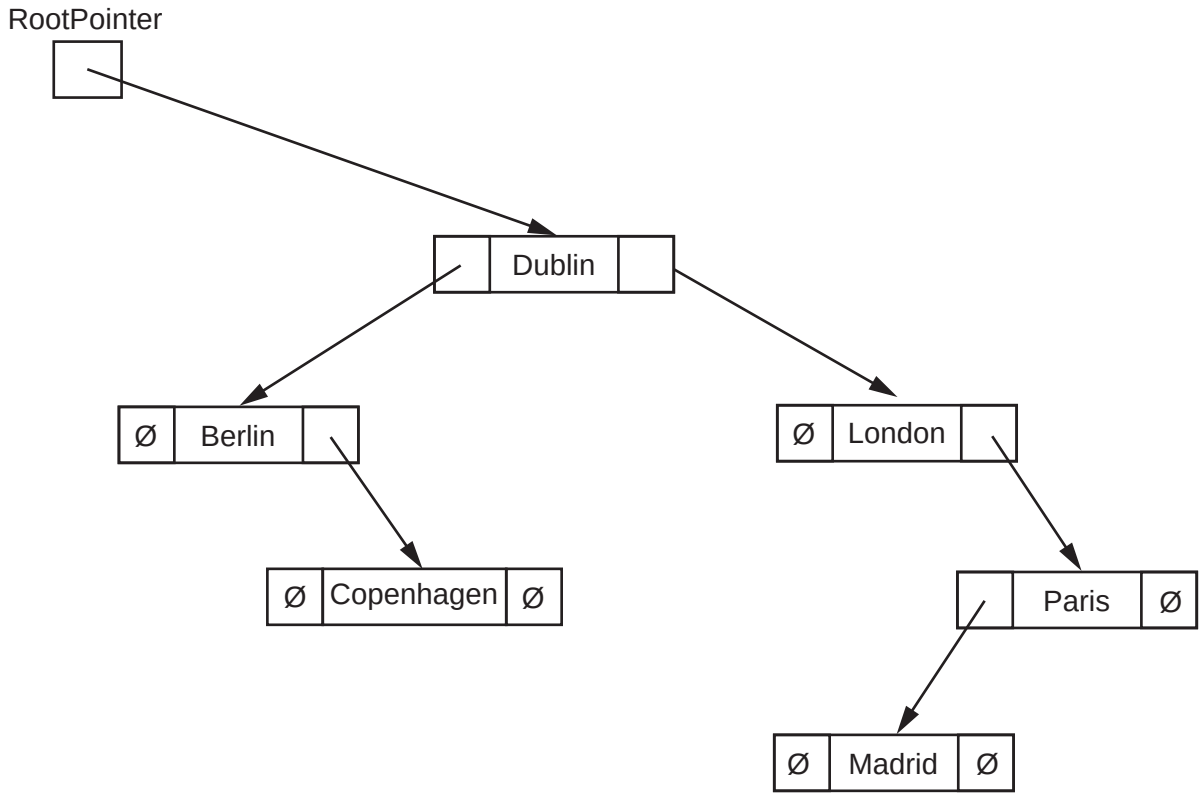
Explain the meaning of the term **null pointer**.

.....

.....[1]

(b) The following diagram shows an ordered binary tree after the following data have been added:

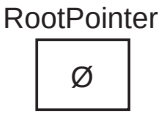
Dublin, London, Berlin, Paris, Madrid, Copenhagen



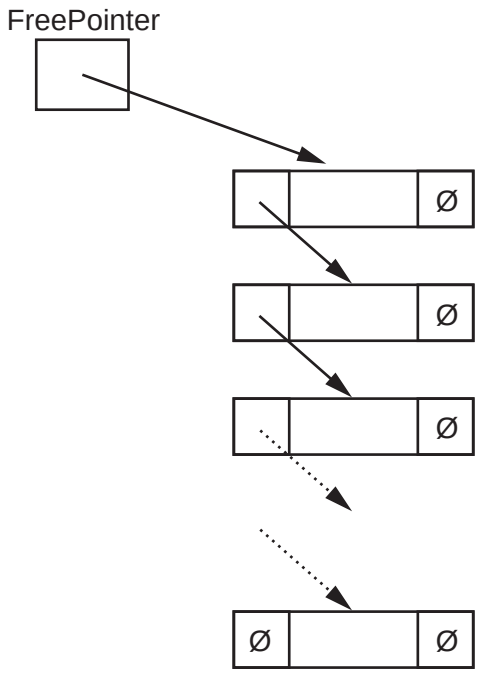
Another data item to be added is Athens.

Make the required changes to the diagram when this data item is added. [2]

(c) A tree without any nodes is represented as:



Unused nodes are linked together into a free list as shown:



The following diagram shows an array of records that stores the tree shown in **part (b)**.

(i) Add the relevant pointer values to complete the diagram.

RootPointer

0

FreePointer

	LeftPointer	Tree data	RightPointer
[0]		Dublin	
[1]		London	
[2]		Berlin	
[3]		Paris	
[4]		Madrid	
[5]		Copenhagen	
[6]		Athens	
[7]			
[8]			
[9]			

[5]

(ii) Give an appropriate numerical value to represent the null pointer for this design. Justify your answer.

.....

.....

.....

.....[2]

(d) A program is to be written to implement the tree ADT. The variables and procedures to be used are listed below:

Identifier	Data type	Description
`Node`	`RECORD`	Data structure to store node data and associated pointers.
`LeftPointer`	`INTEGER`	Stores index of start of left subtree.
`RightPointer`	`INTEGER`	Stores index of start of right subtree.
`Data`	`STRING`	Data item stored in node.
`Tree`	`ARRAY`	Array to store nodes.
`NewDataItem`	`STRING`	Stores data to be added.
`FreePointer`	`INTEGER`	Stores index of start of free list.
`RootPointer`	`INTEGER`	Stores index of root node.
`NewNodePointer`	`INTEGER`	Stores index of node to be added.
`CreateTree()`		Procedure initialises the root pointer and free pointer and links all nodes together into the free list.
`AddToTree()`		Procedure to add a new data item in the correct position in the binary tree.
`FindInsertionPoint()`		Procedure that finds the node where a new node is to be added. Procedure takes the parameter `NewDataItem` and returns two parameters: • `Index`, whose value is the index of the node where the new node is to be added • `Direction`, whose value is the direction of the pointer (“`Left`” or “`Right`”).

© UCLES 2017

(i) Complete the pseudocode to create an empty tree.

```
TYPE Node
    .....
    .....
    .....
ENDTYPE
DECLARE Tree : ARRAY[0 : 9] .....
DECLARE FreePointer : INTEGER
DECLARE RootPointer : INTEGER

PROCEDURE CreateTree()
   DECLARE Index : INTEGER
   .....
   .....
   FOR Index ← 0 TO 9   // link nodes
      .....
      .....
   ENDFOR
   .....
ENDPROCEDURE
```

[7]

(ii) Complete the pseudocode to add a data item to the tree.

```
PROCEDURE AddToTree(BYVALUE NewDataItem : STRING)
// if no free node report an error
   IF FreePointer ..........
      THEN
         OUTPUT("No free space left")
      ELSE // add new data item to first node in the free list
         NewNodePointer ← FreePointer
         ..........
         // adjust free pointer
         FreePointer ← ..........
         // clear left pointer
         Tree[NewNodePointer].LeftPointer ← ..........
         // is tree currently empty ?
         IF ..........
            THEN // make new node the root node
               ..........
            ELSE   // find position where new node is to be added
               Index ← RootPointer
               CALL FindInsertionPoint(NewDataItem, Index, Direction)
               IF Direction = "Left"
                  THEN   // add new node on left
                     ..........
                  ELSE   // add new node on right
                     ..........
               ENDIF
         ENDIF
   ENDIF
ENDPROCEDURE
```

[8]

(e) The traverse tree operation outputs the data items in alphabetical order. This can be written as a recursive solution.

Complete the pseudocode for the recursive procedure `TraverseTree`.

```
PROCEDURE TraverseTree(BYVALUE Pointer : INTEGER)
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
ENDPROCEDURE
```

[5]

3 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 chooses 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 parameters 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

© UCLES 2017

(a) The programmer designed the pseudocode for the main program as follows:

```
DECLARE Island : IslandClass.Constructor()     // instantiate object
CALL DisplayGrid()                           // output island squares
FOR Treasure ← 1 TO 3                          // hide 3 treasures
   CALL Island.HideTreasure()
ENDFOR
CALL StartDig()                      // user to input location of dig
CALL DisplayGrid()                           // output island squares
```

Write **program code** to implement this pseudocode.

Programming language used .....

Program code .....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....[3]

(b) Write **program code** to declare the `IslandClass` and write the constructor method.

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

Programming language used .....

Program code .....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....[5]

(c) The procedure `DisplayGrid` shows the current grid data. `DisplayGrid` makes use of the getter method `GetSquare` of the `Island` class.

An example output is:

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

(i) Write **program code** for the `GetSquare(Row, Column)` getter method.

.....

.....

.....

.....

.....[2]

(ii) Write **program code** for the `DisplayGrid` procedure.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....[4]

(d) 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.

Programming language used .....

Program code .....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....[5]

(e) (i) The `DigHole` method takes two integers as parameters. These parameters form a valid grid location. The location is marked with `'X'` or `'O'` as appropriate.

Write **program code** for the `DigHole` method. The values to represent treasure, found treasure and hole should be declared as constants.

Programming language used .....

Program code .....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....[3]

(ii) The `StartDig` procedure:

- prompts the player for a location to dig
- validates the user input
- calls the `DigHole` method from **part (e)(i)**.

Write **program code** for the `StartDig` procedure. Ensure that the user input is fully validated.

Programming language used ....

Program code ....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....[5]

(f) **(i)** The squares in the `IslandClass` grid could have been declared as objects of a `Square` class.

State the term used to describe the relationship between `IslandClass` and `Square`.

.....

.....[1]

(ii) Draw the appropriate diagram to represent this relationship. Do not list the attributes and methods of the classes.

[2]

BLANK PAGE

© UCLES 2017

BLANK PAGE

© UCLES 2017

BLANK PAGE

Permission to reproduce items where third-party owned material protected by copyright is included has been sought and cleared where possible. Every reasonable effort has been made by the publisher (UCLES) to trace copyright holders, but if any items requiring clearance have unwittingly been included, the publisher will be pleased to make amends at the earliest possible opportunity.

To avoid the issue of disclosure of answer-related information to candidates, all copyright acknowledgements are reproduced online in the Cambridge International Examinations Copyright Acknowledgements Booklet. This is produced for each series of examinations and is freely available to download at www.cie.org.uk after the live examination series.

Cambridge International Examinations is part of the Cambridge Assessment Group. Cambridge Assessment is the brand name of University of Cambridge Local Examinations Syndicate (UCLES), which is itself a department of the University of Cambridge.