Cambridge Assessment International Education

Cambridge International AS & A Level

COMPUTER SCIENCE **9618/41**

Paper 4 Practical **October/November 2025**

2 hours 30 minutes

You will need: Candidate source files (listed on page 2)
evidence.doc

INSTRUCTIONS

- Carry out every instruction in each task.
- Save your work using the file names given in the task as and when instructed.
- You must **not** have access to either the internet or any email system during this examination.
- You must save your work in the evidence document as stated in the tasks. If work is **not** saved in the evidence document, you will **not** receive marks for that task.
- You must use a high-level programming language from this list:
  - Java (console mode)
  - Python (console mode)
  - Visual Basic (console mode)
- A mark of **zero** will be awarded if a programming language other than those listed here is used.

INFORMATION

- The total mark for this paper is 75.
- The number of marks for each question or part question is shown in brackets [ ].

DC (SL) 343264/2

[Turn over

You have been supplied with the following source files:

`HashTableData.txt`

Open the evidence document, **evidence.doc**

Make sure that your name, centre number and candidate number will appear on every page of this document. This document must contain your answers to each question.

Save this evidence document in your work area as:

evidence_ followed by your centre number_candidate number, for example: **evidence_zz999_9999**

A class declaration can be used to declare a record. If the programming language used does **not** support arrays, a list can be used instead.

One source file is used to answer **Question 3**. The file is called **`HashTableData.txt`**

1 A program stores integers in a stack. The stack is represented as a 1D array of 30 elements with the identifier `Stack`

The global pointer `TopOfStack` stores the index of the last element inserted into the stack. `TopOfStack` is initialised to `-1`

(a) Write program code to declare `Stack`, initialise each element in the array with a null value and declare and initialise `TopOfStack`

Save your program as **Question1_N25**. Copy and paste the program code into part **1(a)** in the evidence document.

[2]

(b) The function `Push()` takes an integer parameter. If the stack is full, the function returns `FALSE`. If the stack is not full, the parameter is inserted into the stack, the pointer is updated and the function returns `TRUE`

Write the program code for `Push()`

Save your program. Copy and paste the program code into part **1(b)** in the evidence document.

[4]

(c) The function `Pop()` returns the next integer in the stack and updates the pointer as appropriate. If there is no data in the stack, the function returns the value `-999`

Write program code for `Pop()`

Save your program. Copy and paste the program code into part **1(c)** in the evidence document.

[4]

(d) The main program generates 40 random integers between 0 and 1000 (inclusive) and attempts to insert each one into the stack using the appropriate function. If the return value from the function call indicates the stack is full, no more integers are generated and `"Stack full"` is output.

Write program code for the main program.

Save your program. Copy and paste the program code into part **1(d)** in the evidence document.

[4]

(e) The procedure `FindValues()`:

- pops each integer from the stack until the stack is empty
- finds and outputs the largest number that was in the stack in an appropriate message
- finds and outputs the smallest number that was in the stack in an appropriate message.

Write program code for `FindValues()`

Save your program. Copy and paste the program code into part **1(e)** in the evidence document.

[4]

(f) **(i)** Extend the main program to call `FindValues()`

Save your program. Copy and paste the program code into part **1(f)(i)** in the evidence document.

[1]

(ii) Test your program.

Take a screenshot of the output(s).

Save your program. Copy and paste the screenshot(s) into part **1(f)(ii)** in the evidence document.

[1]

2 A program stores data about trains and train stations using Object-Oriented Programming (OOP).

The class `Train` stores the data about the trains:

Train	
`TrainIDNumber : String`	stores the train ID number
`Route : Integer`	stores the route number the train is travelling
`Constructor()`	initialises `TrainIDNumber` and `Route` to the parameter values
`GetTrainIDNumber()`	returns the train ID number
`GetRoute()`	returns the route number the train is travelling

(a) (i) Write program code to declare the class `Train` and its constructor.

Do **not** declare the other methods.

All attributes should be private.

Use your programming language appropriate constructor.

If you are writing in Python, include attribute declarations using comments.

Save your program as **Question2_N25**.

Copy and paste the program code into part **2(a)(i)** in the evidence document.

[4]

(ii) The methods `GetTrainIDNumber()` and `GetRoute()` return the appropriate attribute.

Write program code for `GetTrainIDNumber()` and `GetRoute()`

Save your program.

Copy and paste the program code into part **2(a)(ii)** in the evidence document.

[3]

(b) The program is tested with four trains:

train ID number	route
12ADV	134
33ART	20
9FKF	3
21VBC	24

Write program code to declare an instance of `Train` for each of the **four** trains.

Save your program.

Copy and paste the program code into part **2(b)** in the evidence document.

[2]

(c) The class `Station` stores the data about the stations:

Station	
`StationID : String`	stores the station ID
`NumberPlatforms : Integer`	stores the number of platforms at the station
`Trains[0:9] : Train`	stores the trains currently at the station platforms
`NumberTrains : Integer`	stores the number of trains currently at the station platforms
`Constructor()`	initialises `StationID` and `NumberPlatforms` to the parameter values, initialises `Trains` to an empty array and `NumberTrains` to `0`
`GetTrains()`	returns a string containing data about the trains currently at the station platforms
`AddTrain()`	takes a `Train` parameter and stores it if there is a platform available; each platform can only have one train

(i) Write program code to declare the class `Station` and its constructor.

Do **not** declare the other methods.

All attributes should be private.

Use your programming language appropriate constructor.

If you are writing in Python, include attribute declarations using comments.

Save your program.

Copy and paste the program code into part **2(c)(i)** in the evidence document.

[3]

(ii) The method `AddTrain()` takes a `Train` parameter. The method compares the attributes `NumberTrains` and `NumberPlatforms` to identify if there is a platform available (a platform currently with no train).

The method returns `FALSE` if there are no platforms available.

If there is a platform available, the method:

- stores the `Train` parameter in the array `Trains`
- updates the appropriate attribute(s)
- returns `TRUE`

Write program code for `AddTrain()`

Save your program.

Copy and paste the program code into part **2(c)(ii)** in the evidence document.

[4]

(iii) The method `GetTrains()` returns the string `"There are no trains"` if there are no trains at the station platforms.

If there are trains at the station platforms, the method returns a string in the format:

```
The trains at station <StationID> are:
<TrainIDNumber> on route number <Route>
```

The line `<TrainIDNumber> on route number <Route>` is repeated for each train at the station platforms.

For example: If the station with the station ID `"NT1"` has two trains with the train ID numbers `"48RTG"`, `"6UFH"`, the method will produce this output:

```
The trains at station NT1 are:
48RTG on route number 43
6UFH on route number 12
```

Write program code for `GetTrains()`

Save your program.

Copy and paste the program code into part **2(c)(iii)** in the evidence document.

[6]

(d) The program is tested with two stations:

station ID	number of platforms
STH	2
NTH	1

(i) Write program code to amend the main program to declare an instance of `Station` for each of the **two** stations.

Save your program.

Copy and paste the program code into part **2(d)(i)** in the evidence document.

[2]

(ii) The four trains attempt to stop at the following stations in the order given:

- Train 12ADV, station STH
- Train 33ART, station STH
- Train 9FKF, station STH
- Train 21VBC, station NTH

Write program code to amend the main program to:

- add each train to the given station using `AddTrain()`
- output `"Station is full"` for any train where the return value indicates it cannot be added to the station
- output the trains at each station using `GetTrains()`

Save your program.

Copy and paste the program code into part **2(d)(ii)** in the evidence document.

[4]

(iii) Test your program.

Take a screenshot of the output(s).

Save your program.

Copy and paste the screenshot(s) into part **2(d)(iii)** in the evidence document.

[2]

3 A program stores records in the 2D array `HashTable`. Each record is stored at a specific index of the array that is calculated using a hashing algorithm with the record's key field.

The array has 100 × 10 elements. The hashing algorithm uses the key to generate an index between 0 and 99 (inclusive). If two key fields generate the same index, there is a collision. Any records that have a collision are stored in the next space in the same index.

For example: In this table two record keys generated the same hash value of 1. Four record keys generated the same hash value of 3.

Index	0	1	2	3	4	5	…	9
0	record							
1	record	record						
2								
3	record	record	record	record				
4								
…								
99								

The program uses Object-Oriented Programming (OOP).

(a) The class `Record` stores data about the records:

Record	
`Key : Integer`	stores the integer key field for the data
`Data : String`	stores the string data
`Constructor()`	initialises `Key` and `Data` to its parameter values

The attributes `Key` and `Data` are public.

Write program code to declare the class `Record` and its constructor.

Use your programming language appropriate constructor.

Save your program as **Question3_N25**.

Copy and paste the program code into part **3(a)** in the evidence document.

[2]

(b) The procedure `InitialiseHashTable()` initialises each element in the array to an empty or null record.

Write program code to declare the global 2D array `HashTable` and the procedure `InitialiseHashTable()`

Save your program. Copy and paste the program code into part **3(b)** in the evidence document.

[2]

(c) The function `Hash()`:

- takes an integer key field as a parameter
- calculates and returns the hash value of the key field.

The hash value is the result from the formula: `key MOD 100`

Write program code for `Hash()`

Save your program. Copy and paste the program code into part **3(c)** in the evidence document.

[2]

(d) The procedure `InsertData()`:

- takes an object of type `Record` as a parameter
- calculates the hash value for the parameter using the appropriate function
- stores the parameter in the correct position in `HashTable`

You can assume there will be no more than 10 objects that generate the same hash value.

Write program code for `InsertData()`

Save your program. Copy and paste the program code into part **3(d)** in the evidence document.

[4]

(e) The file `"HashTableData.txt"` stores 200 key values and string data items in the format:

```
key,string
```

For example, the first row in the text file is:

```
528,permission
```

The key is `528` and the string data is `"permission"`

The procedure `ReadData()`:

- opens the text file and reads each line
- splits each line into the key and data
- calls `InsertData()` with an object containing each key and matching data.

Write program code for `ReadData()`

Save your program. Copy and paste the program code into part **3(e)** in the evidence document.

[5]

(f) The function `GetRecord()`:

- takes an integer key field as a parameter
- calculates the hash value for the key field using the appropriate function
- searches the hash table for the record with the matching key field
- returns the data for the record if the record is found
- returns `"Not found"` if the record is not found.

Write program code for `GetRecord()`

Save your program. Copy and paste the program code into part **3(f)** in the evidence document.

[5]

(g) The main program:

- calls `InitialiseHashTable()` and `ReadData()`
- takes **five** integer key fields as input from the user
- calls `GetRecord()` with each input and outputs the return value.

(i) Write program code for the main program.

Save your program. Copy and paste the program code into part **3(g)(i)** in the evidence document.

[3]

(ii) Test your program with the following **five** inputs in the order given:

528

1128

1828

1062

39

Take a screenshot of the output(s).

Save your program. Copy and paste the screenshot(s) into part **3(g)(ii)** in the evidence document.

[2]