Cambridge International AS & A Level

Cambridge International Examinations
Cambridge International Advanced Subsidiary and Advanced Level

COMPUTER SCIENCE **9608/22**

Paper 2 Fundamental Problem-solving and Programming Skills **October/November 2017**

PRE-RELEASE MATERIAL

No Additional Materials are required.

This material should be given to the relevant teachers and candidates as soon as it has been received at the Centre.

READ THESE INSTRUCTIONS FIRST

Candidates should use this material in preparation for the examination. Candidates should attempt the practical programming tasks using their chosen high-level, procedural programming language.

This document consists of **8** printed pages and **4** blank pages.

DC (JP) 154212

CAMBRIDGE International Examinations

This material is intended to be read by teachers and candidates prior to the November 2017 examination for 9608 Paper 2.

Reminders

The syllabus states:

- there will be questions on the examination paper which do not relate to this pre-release material.
- you must choose a high-level programming language from this list:
  - Visual Basic (console mode)
  - Python
  - Pascal / Delphi (console mode)

Note: A mark of **zero** will be awarded if a programming language other than those listed is used.

Questions on the examination paper may ask the candidate to write:

- structured English
- pseudocode
- program code

A program flowchart should be considered as an alternative to pseudocode for the documenting of an algorithm design.

Candidates should be confident with:

- the presentation of an algorithm using either a program flowchart or pseudocode
- production of a program flowchart from given pseudocode and vice versa.

There is an **Appendix** at the end of this document. Some tasks will refer you to this information. There will also be a similar appendix at the end of the question paper.

Some tasks require a candidate to write program code. These have been carefully chosen to encourage the candidate's programming skills to be at a standard in line with the question paper.

Declaration of variables

The syllabus document shows the syntax expected for a declaration statement in pseudocode.

```
DECLARE <identifier> : <data type>
```

If Python is the chosen language, each variable's identifier (name) and its intended data type must be documented using a comment statement.

Structured English – Variables

An algorithm in pseudocode uses variables, which should be declared. An algorithm in structured English does not always use variables. In this case, the candidate needs to use the information given in the question to complete an identifier table. The table needs to contain an identifier, data type and description for each variable.

TASK 1 – Programming basics: Selection

A program design is written in structured English:

1. PROMPT for Day
2. INPUT Day
3. IF Day = “Monday” OUTPUT “Red socks”
4. IF Day = “Tuesday” OUTPUT “Blue socks”
5. IF Day = “Wednesday” OUTPUT “Yellow socks”
6. OTHERWISE OUTPUT “Sandals”

Key focus:

Selection statements

TASK 1.1

Write the **pseudocode** equivalent using a nested `IF` structure.

TASK 1.2

Write the **pseudocode** equivalent using a `CASE` structure.

TASK 1.3

A program design is intended to produce a single output message depending on the value of variable `Temperature`, only when one of the following conditions is met.

- “Hot” for `Temperature` > 25
- “Just right” for `Temperature` between 20 and 25 inclusive
- “Cold” for `Temperature` < 20

Key focus:

Nested or un-nested?

The first attempt at the pseudocode is:

```
IF Temperature > 25
   THEN
      OUTPUT "Hot"
ENDIF
IF Temperature > 20
   THEN
      OUTPUT "Just right"
   ELSE
      OUTPUT "Cold"
ENDIF
```

Complete the trace table by performing a dry run of the preceding pseudocode.

Temperature	Output
10	
20	
22	
25	
28	

TASK 1.4

Correct the pseudocode to produce the required output.

Key focus:

`IF` structure in a HLL

TASK 1.5

Rewrite the corrected pseudocode from task 1.4 in a high-level language (HLL).

TASK 1.6

Change the HLL code to use a `CASE` statement in place of an `IF` statement.

Key focus:

`CASE` structure in a HLL

TASK 2 – Program flowcharts and arrays

A 1D array, `StudentName`, of type `STRING` consists of 40 elements.

Each string is the name of a student and all the strings are made up of lower case characters.

Unused elements are assigned a dummy value of `"####"`.

The following pseudocode represents an algorithm that:

- converts the first character of each element of `StudentName` to upper case
- counts the number of unused elements and outputs this number.

```
DECLARE Index : INTEGER
DECLARE Temp : CHAR
DECLARE NameLength : INTEGER
DECLARE UnusedCount : INTEGER
CONSTANT DummyValue = "####"

UnusedCount ← 0

FOR Index ← 1 TO 40
   IF StudentName[Index] = DummyValue           // an unused element
      THEN
         UnusedCount ← UnusedCount + 1
      ELSE
         NameLength ← LENGTH(StudentName[Index])
         Temp ← LEFT(StudentName[Index], 1)
         Temp ← UCASE(Temp)
         StudentName[Index] ← Temp & RIGHT(StudentName[Index], NameLength - 1)
   ENDIF
ENDFOR

OUTPUT "There are " & UnusedCount & " unused elements"
```

Key focus:

Use of program flowchart to describe an algorithm

TASK 2.1

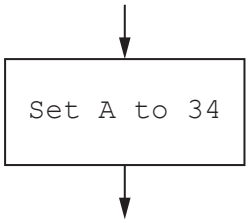
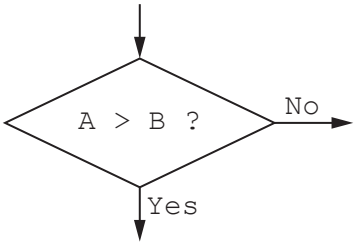
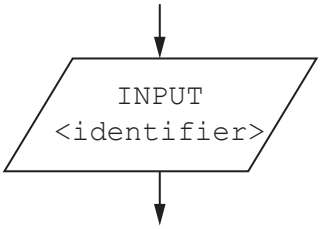
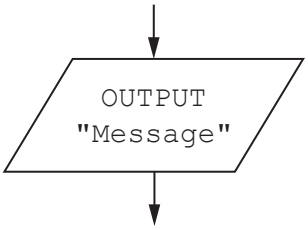
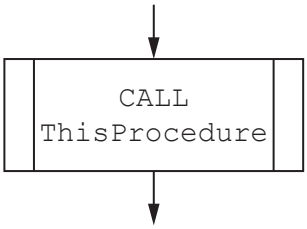
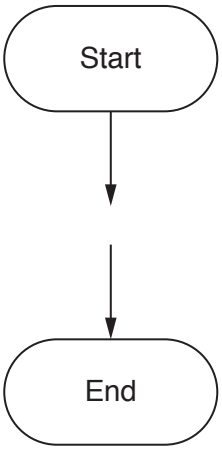
Draw a program flowchart to represent this algorithm. Ensure that the correct symbols are used for all types of operation. Standard symbols are shown on the following page.

TASK 2.2

Write **pseudocode** for a procedure to perform a bubble sort on the `StudentName` array into ascending order.

Key focus:

Applying a bubble sort

	Flowchart symbol
Assignment	
Selection	
Input	
Output	
Procedure call	
Terminator	

TASK 3 – Built-in functions: Number formatting and random numbers

TASK 3.1

Write **program code** that uses a HLL built-in function to format a real number as currency by applying the following features:

Key focus:

HLL built-in functions

- a leading currency character, such as $
- comma separators for thousands
- rounding to a defined number of decimal places
- padding with leading and/or trailing zeros

TASK 3.2

Write **program code** for a procedure to generate a random number within a given range.

Use built-in features of the HLL to avoid repeating the same sequence of random numbers.

Appendix

Built-in functions (pseudocode)

Each function returns an error if the function call is not properly formed.

```
MOD(x : INTEGER, y : INTEGER) RETURNS INTEGER
```

returns the remainder when `x` is divided by `y` using integer arithmetic.
Example: `MOD(5, 2)` returns `1`

```
LEFT(ThisString : STRING, x : INTEGER) RETURNS STRING
```

returns leftmost `x` characters from `ThisString`.
Example: `LEFT("ABCDEFGH", 3)` returns string `"ABC"`

```
RIGHT(ThisString: STRING, x : INTEGER) RETURNS STRING
```

returns rightmost `x` characters from `ThisString`.
Example: `RIGHT("ABCDEFGH", 3)` returns string `"FGH"`

```
LENGTH(ThisString : STRING) RETURNS INTEGER
```

returns the integer value representing the length of string `ThisString`.
Example: `LENGTH("Happy Days")` returns `10`

```
LCASE(x : CHAR) RETURNS CHAR
```

returns the lower case equivalent character of `x`.
Example: `LCASE('W')` returns `'w'`

```
UCASE (x : CHAR) RETURNS CHAR
```

returns the upper case equivalent character of `x`.
Example: `UCASE('h')` returns `'H'`

```
INT(x : REAL) RETURNS INTEGER
```

returns the integer part of `x`.
Example: `INT(27.5415)` returns `27`

Operators (pseudocode)

Operator	Description
`&`	Concatenates (joins) two strings Example: `"Summer" & " " & "Pudding"` produces `"Summer Pudding"`
`AND`	Performs a logical `AND` on two Boolean values Example: `TRUE AND FALSE` produces `FALSE`
`OR`	Performs a logical `OR` on two Boolean values Example: `TRUE OR FALSE` produces `TRUE`

BLANK PAGE

BLANK PAGE

BLANK PAGE

BLANK PAGE