

Teaching guide: Pseudo-code

The pseudo-code described below is provided to assist students preparing for their AQA GCSE Computer Science examination (8520).

In all assessment material, AQA will use a consistent style of pseudo-code shown. This will ensure that, given sufficient preparation, candidates will understand the syntax of the pseudo-code easily. It is not the intention that candidates should use this style of pseudo-code in their own work (NEA or written assessments), although they are free to do so. The only direction to candidates when answering questions or describing algorithms in pseudo-code is that their code is clear and consistent.

This document may be updated as required and the latest version will always be available on our website. Updates will not be made mid-year unless an error is discovered that must be corrected. If this happens centres will be notified of the changes. Ordinary updates will be made over the summer period with the new version for the following 12 months posted on our website at the start of the academic year, if any updates were made.

The document is not confidential and can be freely shared with students.

General Syntax

- `IntExp`, `RealExp`, `BoolExp`, `CharExp` and `StringExp` means any expression which can be evaluated to an integer, real, Boolean, character or string respectively.
- `Exp` means any expression
- Emboldened pseudo-code is used to indicate the keywords/operators.
- Exam paper questions will assume that indexing for arrays and strings starts at 0 unless specifically stated otherwise.

Variables and Constants

| | | |
|---------------------|--|---|
| Variable assignment | <code>Identifier ← Exp</code> | <code>a ← 3</code> <code>b ← a + 1</code> <code>c ← c - 2</code> |
| Constant assignment | constant <code>IDENTIFIER ← Exp</code> | <code>constant PI ← 3.141</code> <code>constant</code> <code>CLASS_SIZE ← 23</code> |

Arithmetic Operations

| | | |
|--------------------------------|--------------------------|--|
| Standard arithmetic operations | + - * / | Standard use using brackets to make precedence obvious. The / symbol is used instead of ÷ for division (for integer division use DIV.) |
| Integer division | IntExp DIV IntExp | 9 DIV 5 evaluates to 1 5 DIV 2 evaluates to 2 8 DIV 4 evaluates to 2 |
| Modulus operator | IntExp MOD IntExp | 9 MOD 5 evaluates to 4 5 MOD 2 evaluates to 1 8 MOD 4 evaluates to 0 |

Relational Operators for types that can be clearly ordered

| | | |
|--------------------------|-----------|--------------------|
| Less than | Exp < Exp | 4 < 6 |
| Greater than | Exp > Exp | 4.1 > 4.0 |
| Equal to | Exp = Exp | 3 = 3 |
| Not equal to | Exp ≠ Exp | True ≠ False |
| Less than or equal to | Exp ≤ Exp | 3 ≤ 4 4 ≤ 4 |
| Greater than or equal to | Exp ≥ Exp | 4 ≥ 3 4.5 ≥ 4.5 |

Boolean Operations

| | | |
|-------------|----------------------------|--------------------------|
| Logical AND | BoolExp AND BoolExp | (3 = 3) AND (3 ≤ 4) |
| Logical OR | BoolExp OR BoolExp | (x < 1) OR (x > 9) |
| Logical NOT | NOT BoolExp | NOT (another_go = False) |

Condition-controlled Iteration

| | | |
|---|---------------------------------------|-----------------------------|
| Repeat-until (repeat the statements until | REPEAT # statements here | a ← 1 REPEAT OUTPUT a |
|---|---------------------------------------|-----------------------------|

| | | |
|---|--|---|
| the Boolean expression is True) | UNTIL BoolExp | <pre> a ← a + 1 UNTIL a = 4 # will output 1, 2, 3 </pre> |
| While (while the Boolean expression is True, repeat the statements) | WHILE BoolExp # statements here ENDWHILE | <pre> a ← 1 WHILE a < 4 OUTPUT a a ← a + 1 ENDWHILE # will output 1, 2, 3 </pre> |

Count-controlled Iteration

| | | |
|-----|---|---|
| For | FOR Identifier ← IntExp TO IntExp # statements here ENDFOR | <pre> FOR a ← 1 TO 3 OUTPUT a ENDFOR # will output 1, 2, 3 </pre> |
|-----|---|---|

Selection

| | | |
|---------|---|--|
| If | IF BoolExp THEN # statements here ENDIF | <pre> a ← 1 IF (a MOD 2) = 0 THEN OUTPUT 'even' ENDIF </pre> |
| If-else | IF BoolExp THEN # statements here ELSE # statements here ENDIF | <pre> a ← 1 IF (a MOD 2) = 0 THEN OUTPUT 'even' ELSE OUTPUT 'odd' ENDIF </pre> |
| Else-if | IF BoolExp THEN # statements here ELSE IF BoolExp THEN # statements here # possibly more ELSE IFs ELSE # statements here ENDIF | <pre> a ← 1 IF (a MOD 4) = 0 THEN OUTPUT 'multiple of 4' ELSE IF (a MOD 4) = 1 THEN OUTPUT 'leaves a remainder of 1' ELSE IF (a MOD 4) = 2 THEN OUTPUT 'leaves a remainder of 2' ELSE </pre> |

```

OUTPUT 'leaves
a remainder of 3'
ENDIF

```

Arrays

| | | |
|---|----------------------------------|--|
| Assignment | Identifier ← [Exp, Exp,..., Exp] | primes ← [2, 3, 5, 7, 11, 13] |
| Accessing an element | Identifier[IntExp] | primes[0] # evaluates to 2 (questions on exam # papers will start indexing at # 0 unless specifically stated # otherwise) |
| Updating an element | Identifier[IntExp] ← Exp | primes[5] ← 17 # array is now [2,3,5,7,11,17] |
| Accessing an element in a two-dimensional array | Identifier[IntExp][IntExp] | tables ← [[1, 2, 3], [2, 4, 6], [3, 6, 9], [4, 8, 12]] tables[3][1] # evaluates to 8 as second element # (with index 1) of fourth array # (with index 3) in tables is 8 |
| Updating an element in a two-dimensional array | Identifier[IntExp][IntExp] ← Exp | tables[3][1] ← 16 # tables is now #[[1, 2, 3], # [2, 4, 6], # [3, 6, 9], # [4, 16, 12] #] |

| | | |
|--------------|-------------------------|--|
| Array length | LEN (Identifier) | <pre>LEN(primes) # evaluates to 6 using example above LEN(tables) # evaluates to 4 using example above LEN(tables[0]) # evaluates to 3 using example above</pre> |
|--------------|-------------------------|--|

Subroutines

| | | |
|-------------------------|--|--|
| Subroutine definition | <pre>SUBROUTINE Identifier(parameters) # statements here ENDSUBROUTINE</pre> | <pre>SUBROUTINE show_add(a, b) result ← a + b OUTPUT result ENDSUBROUTINE SUBROUTINE say_hi() OUTPUT 'hi' ENDSUBROUTINE</pre> |
| Subroutine return value | RETURN Exp | <pre>SUBROUTINE add(a, b) result ← a + b RETURN result ENDSUBROUTINE</pre> |
| Calling a subroutine | Identifier(parameters) | <pre>show_add(2, 3) answer ← add(2, 3)</pre> |

String Handling

| | | |
|---------------|--------------------------------------|--|
| String length | LEN (StringExp) | <pre>LEN('computer science') # evaluates to 16 (including space)</pre> |
| Position of a | POSITION (StringExp, CharExp) | <pre>POSITION('computer science', 'm')</pre> |

| | | |
|--|--|--|
| character | | # evaluates to 2 (as with arrays, # exam papers will start indexing # at 0 unless specifically stated # otherwise) |
| Substring (the substring is created by the first parameter indicating the start position within the string, the second parameter indicating the final position within the string and the third parameter being the string itself). | SUBSTRING (IntExp, IntExp, StringExp) | SUBSTRING(2, 9, 'computer science') # evaluates to 'mputer s' |
| Concatenation | StringExp + StringExp | 'computer' + 'science' # evaluates to 'computerscience' |

String and Character Conversion

| | | |
|------------------------------|-----------------------------------|--|
| Converting string to integer | STRING_TO_INT (StringExp) | STRING_TO_INT('16') # evaluates to the integer 16 |
| Converting string to real | STRING_TO_REAL (StringExp) | STRING_TO_REAL('16.3') # evaluates to the real 16.3 |
| Converting integer to string | INT_TO_STRING (IntExp) | INT_TO_STRING(16) # evaluates to the string '16' |
| Converting real to | REAL_TO_STRING (RealExp) | REAL_TO_STRING(16.3) # evaluates to the string '16.3' |

| | | |
|--|-------------------------------|--|
| string | | |
| Converting character to character code | CHAR_TO_CODE (CharExp) | CHAR_TO_CODE('a') # evaluates to 97 using # ASCII/Unicode |
| Converting character code to character | CODE_TO_CHAR (IntExp) | CODE_TO_CHAR(97) # evaluates to 'a' using # ASCII/Unicode |

Input/Output

| | | |
|------------|-------------------------|---------------|
| User input | USERINPUT | a ← USERINPUT |
| Output | OUTPUT StringExp | OUTPUT a |

Random Number Generation

| | | |
|--|------------------------------------|--|
| Random integer generation (between two integers inclusively) | RANDOM_INT (IntExp, IntExp) | RANDOM_INT(3, 5) # will randomly generate 3, 4 or 5 |
|--|------------------------------------|--|

Comments

| | | |
|----------------------|----------------------------------|--|
| Single line comments | # comment | |
| Multi-line comments | # comment # comment and so on | |