2. Program design

Program design is all about solving problems. If you can't solve and articulate the problem by hand, you will not be able to solve it with code! There are three key strategies this course would like you to be familiar with for articulating and testing algorithms. They are:

- Pseudo code
- Flow charts
- Trace tables

2.1 Pseudo code

There are a lot of different computer programming languages available, each serving different needs. Algorithms, however, are universal. A programmer who uses one language, should be able to communicate how to create an algorithm to another programmer who uses a completely different language without knowing anything about it!

For that reason, most of the time an algorithm is being written it won't be in a language-specific format, but one of two generic forms: flow charts or pseudo-code.

Pseudo code is "Structured English".

It's intent: To clearly communicate an algorithm to other programmers regardless of the programming language(s) they are familiar with.

It uses general programming constructs rather than anything language specific.

Since it is not an actual language, there is not a fixed syntax for its use in the broader computer science industry, provided it is generic enough to achieve its aims.

That said, to achieve consistency in iGCSE & IB exam settings, these courses have their own set syntax for the manner in which questions are provided.

An example:

```
EVENS ← 0
input N
loop while N <> 0
    if N modulus 2 == 0 then
        EVENS ← EVENS + 1
    else
        ODDS ← ODDS + 1
    end if
    input N
end loop
output ODDS, EVENS
```

What is the above algorithm doing?

Bonus points: What is the error in this algorithm?

The course requires you to be able to create as well as interpret/analyse pseudo code.

Where answers are to be written in pseudocode, the examiners will be looking for clear algorithmic thinking to be demonstrated. In examinations, this type of question will be written in the approved notation, so a familiarity with it is expected. It is accepted that under exam conditions candidates may, in their solutions, use pseudocode similar to a programming language with which they are familiar. This is acceptable. The markscheme will be written using the approved notation. Provided the examiners can see the logic in the candidate's response, regardless of language, it will be credited. No marks will be withheld for syntax errors ... for answers to be written in pseudo code

Given the statements made above about there not being a "correct" way to write it, there is a set syntax that the IB will use when they write pseudo code questions for you in the exam. As per the expectations comments above, you are not obliged to follow their syntax in your responses, and you will not lose credit for doing so, but you should be familiar enough with their syntax to be able to properly interpret the questions they give you.

These methods, in their pseudocode format, may be used without explanation or clarification in examination questions. Teachers should ensure that candidates will be able to interpret these methods when presented as part of an examination question.

The following comes from the IB documents

- "Approved notation for developing pseudocode" available at https://pbaumgarten.com/ib-compsci/ib-compsci-pseudocode-flowcharts.pdf
- "Pseudocode in Examinations" available at https://pbaumgarten.com/ib-compsci/ib-compsci-pseudocode-in-detail.pdf

Approved notation for developing pseudocode

When developing pseudocode teachers must use the symbols below, which are those used in mathematics.

This information should be distributed to candidates as close as possible to the commencement of teaching of the course. This notation sheet will be available to candidates during the external examinations.

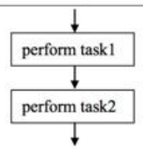
Conventions	Variable names are all capitals, for example, CITY Pseudocode keywords are lower case, for example, loop, if … Method names are mixed case, for example, getRecord Methods are invoked using the "dot notation" used in Java, C++, C#, and similar languages, for example, BIGARRAY.binarySearch(27)

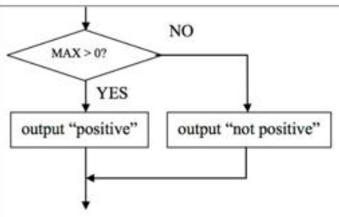
Variable names	These will be provided and comments // used, for example:
	N = 5 // the number of items in the array
	SCOREHISTORY,getExam(NUM) // get the student's score on exam NUM

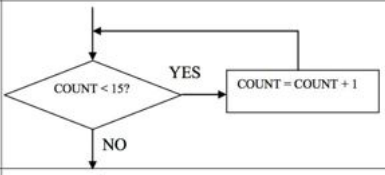
Assigning a value to a variable	Values will be assigned using = , for example:
	N = 5 // indicates the array has 5 data items
	VALUE[0] = 7 // assigns the first data item in the array a value of 7

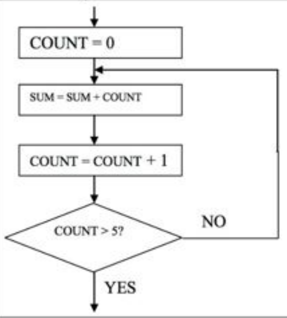
Output of information	Output—this term is sufficient to indicate the data is output to a printer, screen, for example: output COUNT // display the count on the screen

Symbol	Definition	Examples	
=	is equal to	X = 4, X = K	If X = 4
>	is greater than	X > 4	if X > 4 then
>=	is greater than or equal to	X >= 6	loop while X >= 6
<	is less than	VALUE[Y] < 7	loop until VALUE[Y] < 7
<=	is less than or equal to	VALUE[] <=12	if VALUE[Y] <= 12 then
≠	not equal to	X ≠ 4, X ≠ K	
AND	logical AND	A AND B	if X < 7 AND Y > 2 then
OR	logical OR	A OR B	if X < 7 OR Y > 2 then
NOT	logical NOT	NOT A	if NOT X = 7 then
mod	modulo	15 mod 7 = 1	if VALUE[Y] mod 7 = 0 then
div	integer part of quotient	15 div 7 = 2	if VALUE[Y] div 7 = 2 then

Operation	Flowchart example	Pseudocode example
sequential operations		perform task1 perform task2

conditional operations	 MAX > 0?
	NO
	YES
	output "positive"
	output "not positive"
		if MAX > 0 then
		    output "positive"
		else
		    output "not positive"
		end if

while-loop		loop while COUNT < 15
		    COUNT = COUNT + 1
		end loop

from/to-loop	 COUNT = 0
	SUM = SUM + COUNT
	COUNT = COUNT + 1
	COUNT > 5?
	NO
	YES
		loop COUNT from 0 to 5
		    SUM = SUM + COUNT
		end loop

Higher Level and Standard Level

Arrays

An array is an indexed and ordered set of elements. Unless specifically defined in the question, the index of the first element in an array is 0.

```
    NAMES[0]  // The first element in the array NAMES
```

Strings

A string can contain a set of characters, or can be empty. Strings can be used like any other variable.

```
    MYWORD = "This is a string"
    if MYWORD = "the" then
         output MYWORD
    end if
```

Strings should be regarded as a class of objects. However no methods belonging to that class are part of this standard. Instead, if a specialized method such as `charAt()` or `substring()` is to be used in an examination, it will be fully specified as part of the question in which it is needed.

Collections

Collections store a set of elements. The elements may be of any type (numbers, objects, arrays, Strings, etc.).

A collection provides a mechanism to iterate through all of the elements that it contains. The following code is guaranteed to retrieve each item in the collection exactly once.

```
    // STUFF is a collection that already exists
    STUFF.resetNext()
    loop while STUFF.hasNext()
         ITEM = STUFF.getNext()
         // process ITEM in whatever way is needed
    end loop
```

Collections

Method name	Brief description	Example: HOT, a collection of temperatures	Comment

`addItem()`	Add item	`HOT.addItem(42)`
		`HOT.addItem("chile")`
			Adds an element that contains the argument, whether it is a value, String, object, etc.

`getNext()`	Get the next item	`TEMP = HOT.getNext()`	`getNext()` will return the first item in the collection when it is first called. Note: `getNext()` does not remove the item from the collection.

`resetNext()`	Go back to the start of the collection	`HOT.resetNext()`
		`HOT.getNext()`
			Restarts the iteration through the collection. The two lines shown will retrieve the first item in the collection.

`hasNext()`	Test: has next item	`if HOT.hasNext() then`	Returns TRUE if there are one or more elements in the collection that have not been accessed by the present iteration: The next use of `getNext()` will return a valid element.
`isEmpty()`	Test: collection is empty	`if HOT.isEmpty() then`	Returns TRUE if the collection does not contain any elements.

Examples of Pseudocode

AVERAGING AN ARRAY

The array `STOCK` contains a list of 1000 whole numbers (integers). The following pseudocode presents an algorithm that will count how many of these numbers are non-zero, adds up all those numbers and then prints the average of all the non-zero numbers (divides by `COUNT` rather than dividing by 1000).

```
COUNT = 0
TOTAL = 0

loop N from 0 to 999
  if STOCK[N] > 0 then
    COUNT = COUNT + 1
    TOTAL = TOTAL + STOCK[N]
  end if
end loop

if NOT COUNT = 0 then
  AVERAGE = TOTAL / COUNT
  output "Average = " , AVERAGE
else
  output "There are no non-zero values"
end if
```

COPYING FROM A COLLECTION INTO AN ARRAY

The following pseudocode presents an algorithm that reads all the names from a collection, `NAMES`, and copies them into an array, `LIST`, but eliminates any duplicates. That means each name is checked against the names that are already in the array. The collection and the array are passed as parameters to the method.

```
COUNT = 0   // number of names currently in LIST

loop while NAMES.hasNext()

  DATA = NAMES.getNext()

  FOUND = false
  loop POS from 0 to COUNT-1
    if DATA = LIST[POS] then
      FOUND = true
    end if
  end loop

  if FOUND = false then
    LIST[COUNT] = DATA
    COUNT = COUNT + 1
  end if
end loop
```

FACTORS

The following pseudocode presents an algorithm that will print all the factors of an integer. It prints two factors at a time, stopping at the square root. It also counts and displays the total number of factors.

```
// recall that
//   30 div 7 = 4
//   30 mod 7 = 2

NUM = 140  // code will print all factors of this number
F = 1
FACTORS = 0

loop until F*F > NUM  //code will loop until F*F is greater than NUM
  if NUM mod F = 0 then

    D = NUM div F
    output NUM , " = " , F , "*" , D

    if F = 1 then
      FACTORS = FACTORS + 0
    else if F = D then
      FACTORS = FACTORS + 1
    else
      FACTORS = FACTORS + 2
    end if

  end if
  F = F + 1
end loop
output NUM , " has " , FACTORS , " factors "
```

COPYING A COLLECTION INTO AN ARRAY IN REVERSE

The following pseudocode presents an algorithm that will read all the names from a collection, `SURVEY`, and then copy these names into an array, `MYARRAY`, in reverse order.

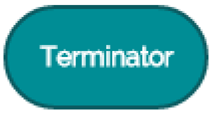
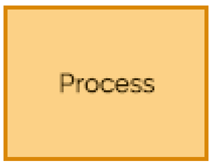
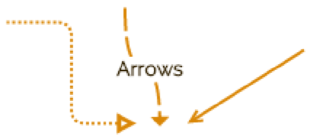
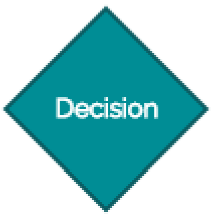
```
// MYSTACK is a stack, initially empty

COUNT = 0 // number of names

loop while SURVEY.hasNext()
  MYSTACK.push( SURVEY.getNext() )
  COUNT = COUNT + 1
end loop

// Fill the array, MYARRAY, with the names in the stack

loop POS from 0 to COUNT-1
  MYARRAY[POS] = MYSTACK.pop()
end loop
```

2.2 Flow charts

IB exams will not require you to create your own flow charts but they will present flow charts to you for analysis and interpretation.

Flow charts are usually quite intuitive. The main issue is to correctly interpret the symbols. Some guidelines are:

- Always format your flow from left to right or top to bottom.
- Run your return lines under your flowchart, making sure that they don't overlap.
- Maintain consistent spacing between symbols.
- Use the correct symbol for each step (diamond shapes are for decisions, rectangles are used for processes, start/end shapes should be the same, etc.)

Symbol	Meaning
	Start or end point of a program.
	A task or action that needs to be performed.
	The "flow" indicating what part of the program should be executed next.
	A decision is required to move forward. This could be a binary, this-or-that choice or a more complex decision with multiple choices. Make sure that you capture each possible choice within your diagram.

From:
https://cacoo.com/blog/keep-it-simple-how-to-avoid-overcomplicating-your-flowcharts/
https://www.gliffy.com/blog/how-to-flowchart-basic-symbols-part-1-of-3

2.3 Trace tables

A trace table is a method of performing a manual dry run on an algorithm where you perform the computations. It is useful for error checking of simple algorithms.

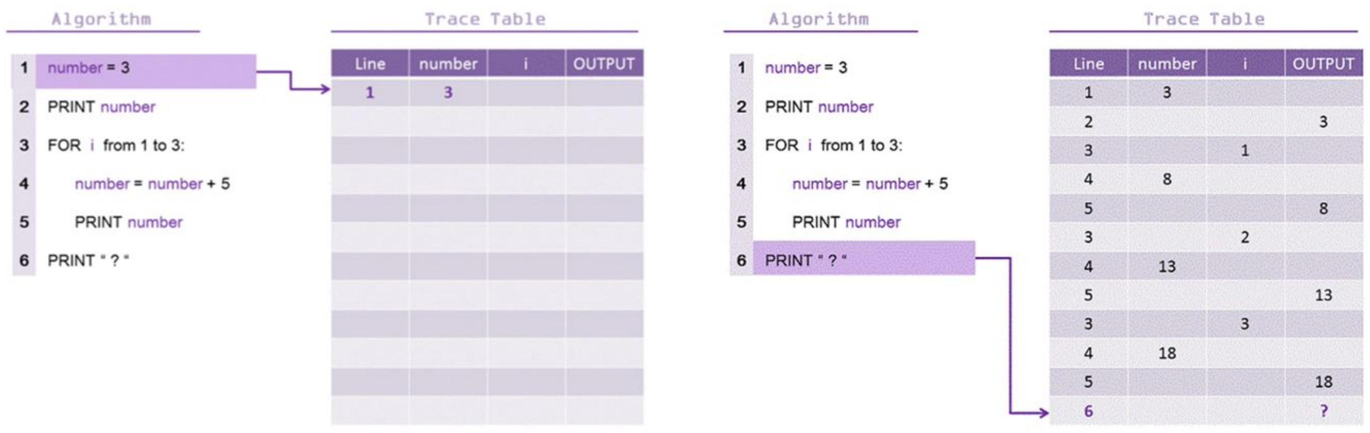
To make one, draw a table of columns, one for each variable. Then walk through the algorithm by hand, writing any changes to the data line by line using test data.

To properly test an algorithm with a trace table it is important to use a good variety of test data, both normal and erroneous data.

You are expected to be able to analyse and create trace tables for different algorithms. Some questions you could be asked to perform with trace tables include:

- Create a trace table to identify the error within a given algorithm.
- Determine the number of times a step in a given algorithm will be performed for given input data
- Suggest changes in an algorithm that would improve efficiency, for example, using a flag to stop a search immediately when an item is found
- Justify an algorithms efficiency, correctness, reliability or flexibility

An example



Algorithm

```
1  number = 3
2  PRINT number
3  FOR i from 1 to 3:
4      number = number + 5
5      PRINT number
6  PRINT " ? "
```

Trace Table

Line	number	i	OUTPUT
1	3		

Algorithm

```
1  number = 3
2  PRINT number
3  FOR i from 1 to 3:
4      number = number + 5
5      PRINT number
6  PRINT " ? "
```

Trace Table

Line	number	i	OUTPUT
1	3		
2			3
3		1	
4	8		
5			8
3		2	
4	13		
5			13
3		3	
4	18		
5			18
6			?

2.4 Practice

1. Write pseudo code that will sum all the even numbers input from a user. Stop after 10 numbers have been input.

2. Write pseudo code that reads in any three numbers and outputs them into sorted order.

3. Design an algorithm... where the user continually inputs a number, stopping when -1 is provided. For each number, the count and the sum of the numbers provided is kept and output at the end.

4. Design an algorithm... that counts numbers. Have the user input two positive integers, and the program counts up by increments of 1 from the smaller number up to but not including the larger number.

5. Write pseudo code that will perform the following:

   Read in 5 separate numbers.
   Calculate the average of the five numbers.
   Find the smallest (minimum) and largest (maximum) of the five entered numbers.
   Output the results found from steps 2 and 3.

6. Write pseudo code that will calculate a running sum. A user will enter numbers that will be added to the sum and when a negative number is encountered, stop adding numbers and write out the final result.

7. The Hailstone problem. The Hailstone Series is generated using the following high level algorithm:

   Pick a positive number (0 or greater)
   If it is odd, triple the number and add one.
   If it is even, divide the number by two.
   Go back to step 2.
   This series will eventually reach the repeating "ground" state: 4, 2, 1, 4, 2, 1

   Here is the sequence generated for an initial value of 26:

   26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1

   Your task is to convert the high-level algorithm into pseudo-code. At the end, display how many items are in the sequence and what is the largest number computed in the sequence. You can assume the "ground" state commences when the integer 4 is computed. At this stage you can terminate computation of the series.

   Once complete, produce a trace table for your Hailstone Series algorithm, given an input of 17.

8. Fizz buzz. This is a "famous" programming job interview question. Create the pseudo code for a Fizz Buzz generator, and use a trace table for values up to 15.

   Write a program that prints the numbers from 1 to 100.
   For multiples of three print "Fizz" instead of the number

For the multiples of five print "Buzz".
For numbers which are multiples of both three and five print "FizzBuzz"
Example output: 1, 2, Fizz, 4, Buzz, Fizz, 7, 8, Fizz, Buzz, 11, Fizz, …

9. An IB question. The following exercise comes from the May 2015 IB exam. Trace the following algorithmic fragment for N = 6. Show all working in a trace table.

```
SUM = 0
loop COUNT from 1 to (N div 2)
    if N mod COUNT = 0 then
        SUM = SUM + COUNT
    end if
end loop
if SUM = N then
    output "perfect"
else
    output "not perfect"
end if
```

Want more practice?

Pseudocode past paper questions for practice
https://pbaumgarten.com/ib-compsci/unit-4/assets/pseudocode-past-questions.pdf

70 pseudocode questions for practice
https://pbaumgarten.com/igcse-compsci/distribute/pseudocode-70-questions.pdf
(I have the answers for these if you are interested)