Pseudo code Tutorial and Exercises – Teacher's Version

Pseudo-code is an informal way to express the design of a computer program or an algorithm in 1.45. The aim is to get the idea quickly and also easy to read without details. It is like a young child putting sentences together without any grammar. There are several ways of writing pseudo-code; there are no strict rules. But to reduce ambiguity between what you are required to do and what you express let's base the pseudo code on the few defined conventions and carry out the exercises.

Pseudo-code Examples

Let's see few examples that can be used to write pseudo-code.

1. Sort

Repeatedly steps through the list to be sorted, comparing each pair of adjacent items and swapping them if they are in the wrong order.

Taking the sorting example; let's sort an array using the ***Bubble sort technique***. This sorting algorithm could be implemented in all programming languages but let's see the C implementation.

```
void ArraySort(int This[], CMPFUN fun_ptr, uint32 ub)
{
  /* bubble sort */

  uint32 indx;
  uint32 indx2;
  int temp;
  int temp2;
  int flipped;

  if (ub <= 1)
    return;

  indx = 1;
  do
  {
    flipped = 0;
    for (indx2 = ub - 1; indx2 >= indx; --indx2)
    {
      temp = This[indx2];
      temp2 = This[indx2 - 1];
      if ((*fun_ptr)(temp2, temp) > 0)
      {
        This[indx2 - 1] = temp;
        This[indx2] = temp2;
        flipped = 1;
      }
    }
  } while ((++indx < ub) && flipped);
}
```

What's your impression?

Is it easy to understand at once this C implementation?

Bubble sort is mostly used in teaching. However, its performance is slow and in 2.44 the students will discover that there are better algorithms.

Here is some pseudo code for this algorithm.

```
Set n to number of records to be sorted
 repeat
flag = false;
for counter = 1 to n-1 do
          if key[counter] > key[counter+1] then
          swap the records;
           set flag = true;
          end if
 end do
 n = n-1;
until flag = false or n=1
```

What's easier to understand, the implementation in C or pseudo-code?

OR the same can be expressed more concisely in words as below

```
repeat
set a flag to False
for each pair of keys
       if the keys are in the wrong order then
            swap the keys
            set the flag to True
       end if
next pair
until flag is not set.
```

OR even as follows

Keep swapping items until array is in order

This is easier than the programming language but is not so precise. Hence the above pseudo code examples are more useful for implementing purposes. This one-line version may raise questions such as "on what basis do I swap the items?" Therefore, it is important to be precise too.

The main part is that it is important to provide easy to read but precise instructions; this will keep the design simple and unambiguous.

Taking a practical example, if I gave you the following instructions:

(a) *Take a left, then take a right, go down the stairs, on your right enter the kitchen, pick a cup and pour some hot water and add some hot chocolate….*

OR

(b) *Please make me a hot chocolate.*

The above line of instruction depends on the reader, some prefer to (a) if not experienced while others prefer (b) because it nails it to the point. It is pretty concise too.

Let us take Example 1 and divide the algorithm implementation in stages and conquer.

Example 1: Compute Fibonacci numbers till 50.

C implementation

```
int main( )
{
  int n, k, f1, f2, f;
  if ( n < 2 ) return n;
  else {
    f1 = f2 = 1;
    for(k=2;k<n;k++)
    {
      f = f1 + f2;
      f2 = f1;
      f1 = f;
    }
  return f;
}
```

The statements with // are just comments

Let us first declare and initialise all the variables.

Declare an integer variable called n *// n is the numberlimit*
Declare an integer variable sum *// f is the sum*
Declare an integer variable f1 *// f1 is temporary storage*
Declare an integer variable f2 *// f2 is temporary storage*

set loopcounter to 2 *// assigning the variables declared above to values*
set sum to 0
set f1 and f2 to 1
set n to 50

Instead of an equal sign an arrow sign ← can also be used

Now the computation and displaying the output

repeat n times
sum = f1 + f2
f2 = f1
f1 = sum
print sum
end loop

If all the above sections of pseudo code are put together we will get something looking like the example below

Pseudo Code Example 1

1. *Declare an integer variable called n*
2. *Declare an integer variable sum*
3. *Declare an integer variable f1*
4. *Declare an integer variable f2*
5. *set sum to 0*
6. *set f1 and f2 to 1*
7. *set n to 50*
8. *repeat n times*
    a. *sum = f1 + f2*
    b. *f2 = f1*
    c. *f1 = sum*
    d. *print sum*
9. *end loop*

For example in Pseudo Code Example1 the scope of a loop starts at line 9 and ends at 10

Points to note

- Usually scope terminators such as start and end are used.
- Say if you want to consider a check if the n is more than 2 then you can have an if – endif statement

```
if n < 2  then
    print n
end if
```

If this condition is true only then continue initialising the temporary variables and the computation else exit of the function

This if – endif statement would come in before code line 7. Pseudo Code Example 1 is one of the ways that pseudo code can be written. Below is another example where it exempts the declaration of the variable and directly initialises them to a value. It is quite wordy than Pseudo Code Example 1.

Pseudo- Code Example 2

```
Initialise n to fifty
Initialise sum to zero
Initialise f1 and f2 to zero
repeat n times
        add f1 and f2, store this value in sum
        assign f1's current value to f2
        assign sum's current value to f1
end loop
```

These examples are just suggested ways of writing pseudo-code. There are various approaches that you can use. I have included a number of links in Table 2 each having variations of pseudo-code writing techniques with few examples. No strict rules are defined but just easy to read but precise.

Let's look at another example

2. Nested Function

Now let's have an example of a nested function in other words where a function calls another function.

Exercise: Create an array of size 10 and assign random values to each element of the array and print. The random values are generated by another function which we do not implement but it is just invoked to complete our need.

Example 3: The implementation is in Java programming language.

```
int arr;                // Declaring a variable called arr
                        // which will be later used to create an array
arr = new int[10];      // Initialising the array
for(int i = 0; i < arr.size(); i++)  // Loop
{
      arr[i] = random ();           // insert random numbers into the array
      print arr[i];

}
```

Pseudo Code Example 3

```
Declare an integer variable arr
Declare an integer variable loopcounter

Set arr to size 10
for loopcounter = 0  to (size of arr)-1
        arr[loopcounter] = random()
        loopcounter = loopcounter +  1
        print arr[loopcounter]
endfor
```

Since there is a use of the loopcounter to succeed to the next element in the array the *for loop* is vital.

Points to note

- There are two function calls random() and size(). size() gets the size of the initialised array and randon() generates numbers randomly.

The process could be re-written in a wordier format and quite precise than the above example. See Pseudo Code Example 4 for this.

Pseudo Code Example 4

```
fill the array with random variables
```

Pseudo Code Example 4 is very concise description of the algorithm and most programmers know how to implement it.

Desk – Checking

Desk checking is a way of testing your algorithm; it can be also called as code walk through. There are several ways of testing for example by showing it to peers or by implementing it into a programming language and just executing the code step-by-step. But desk checking is faster and it helps one to think like a compiler and a debugger.

The Desk Check Guide at the link below has a range of examples with desk checking. http://ironbark.bendigo.latrobe.edu.au/subjects/PE/2005s1/other_resources/desk_check_guide.html . This guide creates a table of input, outputs and various variables against the code lines and checks what happens to the variables at each line of code. But this seems to be time consuming especially if the algorithm is complex. Desk checking checks the internal workings of an algorithm. There is power point on the teaching on desk checking. Use this link http://www.docstoc.com/docs/15885541/Desk-Checking.

There is no fixed way of desk checking but taking the Fibonacci example I have suggested a simpler way to do it (simpler than the Desk Check Guide).

Let us make *n = 6* and the set values are of *f1 = f2 = 1.*

Table 1: Fibonacci desk-check

Code line 11	***Code line 12***	***Code line 13***	***Code line 14***	***Code line 15***
At loopcounter	*Sum (f1 + f2)*	*f2*	*f1*	***Increment loop by 1***
2	2	1	1	**3**
3	3	2	3	**4**
4	5	3	5	**5**
5	8	5	8	**6**

Side check: is it overlimit

Answer : No. then continue.

Hits the limit. **STOP Looping.**

Extra Information

There are few keywords that are common while writing pseudo-code.

Looping and selection

- Keywords :
  Do While...EndDo;
  Do Until...Enddo;
  Case...EndCase;
  If...Endif;
  Call;
  When;

- Most authors use scope terminators (Start-end, If-endif, for-endfor, do-endo) for loops and iteration.

As verbs, use the words

- Generate,Compute,Process, etc.
- Words such as set, reset, increment, compute, calculate, add, sum, multiply, ...
- Displaying :
  - print, display, input, output, edit, test , etc. with careful indentation tend to foster desirable pseudocode.

Expressions

Common Operators: =, +, -, *, /, (), <, <=, >, >=, [], %, ^. Sometimes add, sum, subtract, etc. are used instead. But *a + b* is better and quick to understand. Such are language independent while some are not such as % or mod. Thus, it is better to specifically mention modulo, e.g. *a modulo B*.

It is not necessary to include data declarations in your pseudo code.

Useful links

All these links cover all the AS expectations but one particular link does not cover all of them. Hence, I have listed all these in a table. So for further information you can refer to the following links. These include various sites of pseudo-code writing tutorials/information and desk checking.

Table 2: Useful Links with extra information and various examples to practice

	Link	Difference	Level of useful
1	http://ironbark.bendigo.latrobe.edu.au/subjects/PE/2005s1/other_resources/desk_check_guide.html	Has number of desk checking examples	Adequate
2	http://ironbark.bendigo.latrobe.edu.au/subjects/PE/2005s1/other_resources/pseudocode_guide.html	Has pseudo code examples	Basic
3	http://userpages.wittenberg.edu/bshelburne/Comp150/Algorithms.htm	Shows use of various pseudo-code jargon	Too basic
4	http://www.unf.edu/~broggio/cop3530/3530pseu.htm	Does not declare the variables first , instead initializes them directly, only examples and also shows use of parameters in pseudo code	Basic
5	http://www.dreamincode.net/forums/topic/59022-using-pseudo-code-to-design-application-logic/	A simple example but emphasises the importance of specifying in detail	Basic
6	http://www.roseindia.net/tutorialsearch/?t=pseudocode	Links to several examples of pseudo-code (with various scenarios) writing from Java	Basic
7	http://users.csc.calpoly.edu/~jdalbey/SWE/pdl_std.html	Pseudo-code conventions and simple examples, calling of sub-procedures	Basic
8	http://www.brpreiss.com/books/opus5/html/book.html	Complicated algorithms available that can be used to write pseudo - code	Advanced material for interested students (beyond AS)

There are few exercises below. These exercises are of a mixed range such as achieved, merit and excellence. They aim to cover the advanced concepts from computer science to ensure students gain an understanding of modular algorithmic structure design. These exercises can be taken further and implemented as expected for AS 2.46

Exercise 1: String reverse - Design an algorithm that reverses a string and print it.

Achieved	Uses indexed data structure- array, includes data type, specifies variables , uses the value in the data structure (prints the string in the reverse order)
Merit	×
Excellence	×

Sample Answer

Algorithm start

Declare an array of string type variable called word

Declare a loopcounter

Store a string in the array word

for loopcounter = (length of the word) – 1 to 0

loopcounter = loopcounter – 1

print arrayword[loopcounter]

endfor

Algorithm end

Exercise 2: Replace every third element in the array with the value 10. Assume that an array filled with values is already provided. (Hint: Use example 3)

Sample Answer1

```
for loopcounter = 2 to size of the array length – 1
        arr[loopcounter]= 10;
        loopcounter = loopcounter + 3;
endfor
```

Achieved	Uses indexed data structure- array, includes data type, specifies variables
Merit	Modifies the content in the indexed data-structure, has boundary cases
Excellence	×

A better way of writing for loop.

Sample Answer 2

```
Set loopcounter to 0
for every third element in the array
    replace the existing value with 10
    increment loopcounter by 3 ;
endfor
```

OR

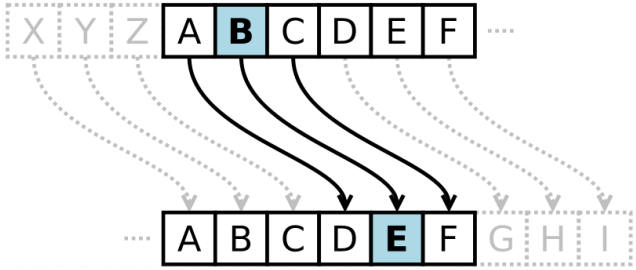
Sample Answer 3

```
for the array from 2 to size of the array length -1 by 3
    replace the existing value with 10
endfor
```

ASIDE: You can extend this exercise for example; consider there are duplicated values in the array. To delete duplicated values you will have to first sort the array and then delete them. Moreover, you will have to decrease the size of the array. In addition you can also insert values at various places in the array.

Achieved	×
Merit	Modifies the contents of the indexed data structure
Excellence	(If Sample Answer 3 is used then this level can be achieved too). It calls a module to conduct the Caesar Cipher.

Exercise 3: Caesar Cipher- it is an encryption technique for sending secret messages. Let's take the example of the alphabets.



You can have a look and experiment with various words at http://www.secretcodebreaker.com/caesar-cipher.html. This

Sample Answer

for each character c
output = (c+3) modulo 26

Alternatives for the output statement can be:

Sample Answer 2
output k characters after c in the alphabets

OR

Sample Answer 3
output code(c)

Define code (parameters: c)
return c + 3 modulo 26

For some the Sample Answer1 is easier to understand while for others the alternatives spell out all that they need. Note: If Sample Answer 3 is used then it satisfies the excellence expectation.

Achieved	×
Merit	×
Excellence	Calls modules, well-structured logical decomposed task.

Exercise 4: Multiple modules

Print a line of stars and then for five times print an asterisk, multiple spaces and print an asterisk and then finally print again a line of stars.

Sample Answer

```
FullLine
repeat five times
        HollowLine
FullLine
```

```
FullLine()
  print 5 asterisks
  newline
```

```
HollowLine()
  print *
  print 3 spaces
  print *
  newline
```

```
*****
*   *
*   *
*   *
*   *
*****
```

ASIDE: To fulfil each of the condition of the exercise two separate functions have been defined - FullLine and a HollowLine. These have no parameters. We could have compressed it all in one function as below.

```
print 5 asterisks
newline
repeat five times
     print *
     print 3 spaces
    print *
    newline
print 5 asterisks
newline
```

Which one is elegant, the one above or this one?

The example above has modules that constitute a well-structured logical decomposition for the task.

There are more similar exercises, but these use parameters. Even though the AS does not expect to let's practice with these.

Achieved	×
Merit	×
Excellence	Calls modules, well-structured logical decomposed task.

Exercise 5: Convert binary to Hexadecimal (relates to AS 2.44)

Convert an integer (a decimal) number to hexadecimal.

Sample Answer

Declare a string variable numbin
Read in value numbin to a value of binary numbers
Call ConvBintoHex(arguments:numbin) and return the hexadecimal numbers
print numhex

ConvBintoHex (parameters: binarynumber)
Divides binary number in groups each containing 4 binary number and computes a value for each group

Important to note: The word Call is significantly used when invoking another procedure or module.

The function contains a parameter and in the relevant box it shows how to write pseudo-code for functions with parameters. Note the difference when a function with parameters is defined and when it is called. When the function is invoked it uses the notation *'arguments'* but the notation used for the function header is *'parameters'*. Some websites use: `Call function name with parameter name`, for example *Call ConBintoHex with numbin*.

Other Sample Answers

Declare a string variable numbin
Declare a long variable numhex
Read in value numbin to a value of binary numbers
numhex ← Call ConvBintoHex(arguments:numbin) ***OR*** *Call ConvBintoHex(arguments:numbin) and store the result*
print numhex

Achieved	×
Merit	×
Excellence	Calls modules, well-structured logical decomposed task.

Exercise 6: Tower of Hanoi

Before starting the exercise a video at this link http://www.youtube.com/watch?v=bPgv9D0lMfs&feature=related can be shown to the students. Furthermore, http://www.youtube.com/watch?v=aGlt2G-DC8c can be shown to the students; this video shows the tower of Hanoi process in a slow motion and with few disks.

While writing the pseudo-code for tower of Hanoi for a number of disks there are few considerations. The disks of different sizes are stacked in ascending order of size; from the largest to the bottom to the smallest at the top. Stack the disks from tower one to tower three as shown in the figure below. Only one disk may be moved at a time. No disk may be placed on top of a smaller disk at any time (even in the process of stacking). Let's experiment with three disks.

Sample Answer

Algorithm start

Move from Tower A to Tower C

Move from Tower A to Tower B

Move from Tower C to Tower B

Move from Tower A to Tower C

Move from Tower B to Tower A

Move from Tower B to Tower C

Move from Tower A to Tower C

Algorithm end

The algorithm calls a module named move, this module moves the each disk one at a time. I have included a pseudo code found on a website to handle indefinite number of disks.

```
FUNCTION MoveTower(disk, source, dest, spare):
IF disk == 0, THEN:
   move disk from source to dest
ELSE:
   MoveTower(disk - 1, source, spare, dest)        // Step 1 above
   move disk from source to dest                   // Step 2 above
   MoveTower(disk - 1, spare, dest, source)        // Step 3 above
END IF
```

There is more to this pseudo code that is it calls its function recursively. (http://www.cs.cmu.edu/~cburch/survey/recurse/hanoiimpl.html)

Let's see if you can solve this.

Say, you ask a friend to go out and buy 1L of milk, and if there are eggs, to buy 10. So what does your friend buy in the end?[i]

It is precise, is it unambiguous?

MORAL: There is no one defined way of writing pseudo code but a pseudo-code should possess three elements: clarity, precision and concise.

~~~~~~~~~~~~~~~~~~~~~~~~ HAVE FUN ~~~~~~~~~~~~~~~~~~~~~~~~

[i] 10 Litres of milk
~~~~~~~~~~~~~~~~~~~~~~~~