Pseudocode Style Guide

Updated: 21st February, 2020

Note: This document is just a guide, it develops on how to write pseudocode in a format that is easily accepted and understood by your markers.

Please be cautious if you decide to use your own style. Deviations are not punished, but your style must be **clear**, **consistent**, **language agnostic**, and **unambiguous**, if it lacks any of these traits you can expect to loose marks.

1 Introduction

Writing Pseudocode is an art that takes a lot of practice. The lecture notes provide Pseudo code examples, the whole point of pseudo code is to express what the algorithm needs to do without being constrained by the rigid syntax of a particular language. The use of similar words to generic code (e.g. `IF`, `WHILE`) is strongly encouraged.

Note: "If you are concerned with the syntax of pseudocode, you are no longer writing pseudocode"
- Previous Curtin Computing Student

In OOPD pseudo code is used in the design process for all of your programming and thus to assess your understanding of the concepts taught in the unit. Hence, readability and understandability of pseudocode is of the utmost importance. *Usually* pseudocode has no rigid format (See above quote), however, as you may lack the skills and knowledge to understand what is required, and what is not, this guide has been created to help you.

2 User Interaction

User interactions takes the forms of data that the user provides to your program (input) or text and variables that your program gives back to the user (output). Input and output are steps of an algorithm and should be specified exactly where they would occur in your program. The type of data you are inputting should also be included (see the section on Data Types).

2.1 Input

Inputting data requires the variable name it is being assigned to as well as its data type. Multiple variables may be input at once, with grouped variables sharing the same data type.

Pseudo Code	Java
`INPUT myVar (Real)`	`myVar = sc.nextDouble();`
`INPUT yourVar (Integer)`	`yourVar = sc.nextInt();`
`INPUT length, width (Real)`	`length = sc.nextDouble();` `width = sc.nextDouble();`
`INPUT flag (Character), name (String)`	`flag = sc.nextLine().charAt(0);` `name = sc.nextLine();`

2.2 Output

To output a variable you may simply state the variable name. Wrapper text only needs to be included if it is *algorithmically significant*. Similar to input, multiple variables may be output at once.

Pseudo Code	Java
`OUTPUT "Actual text to appear"`	`System.out.println("Actual Text to appear");`
`OUTPUT myVar`	`System.out.println("Wrapper text" + myVar);`
`OUTPUT length, width, area`	`System.out.println(length + width + area);`
`OUTPUT "l:" + len + "w:" + wid`	`System.out.println("l:" + len + "w:" + wid);`

3 Variable Assignment

The variable you are assigning a value to must **always be on the left-hand side**. It is a good idea to distinguish between assignment and equality, hence we are using `":="` for assignment.

Pseudo Code	Java
`x := 3`	`x = 3;`
`area := length x width`	`area = length * width;`
`diff := sum - count`	`diff = sum - count;`
`even := num MOD 2`	`even = num % 2;`
`avg := sum / count`	`avg = sum / count;`

Note: Be cautious of the difference between integer division and real division. ÷, `/`, `DIV` and `MOD` are all percieved differently depending on the context.

4 Data Types

When talking about data types in pseudocode we are only concerned with conceptual differences, that is `Integer`, `Real`, `Character`, and `String`. You **must avoid** language specific implementations such as `int`, and `long`.

4.1 When to Specify

The datatype need **only** to be specified in relation to `INPUT`, `IMPORT`, and `EXPORT`.

4.2 Converting & Real vs. Integer Operations

At times, it is algorithmically important to show when you are converting to a different datatype. Especially in regards to `Integer` operations and avoiding mixed mode arithmetic.

`Integer` operations such as modulus and integer division should be specified using `DIV` and `MOD` (as shown above). `Real` division can be shown using ÷ or `/`.

To avoid mixed mode arithmetic if you are using literal numbers you should also specify the fraction for a real number (even if it is 0.0).

Pseudo Code	Java
`truncX := (Convert to Real) diameter / 2.0`	`truncX = (double) diameter / 2.0;`
`myVar := (Convert to Integer) (yourVar * 10.0)`	`myVar = (int) (yourVar * 10.0);`
`someVar := (Convert to Integer) yourVar * 10`	`someVar = (int) yourVar * 10;`
`divVar := sum DIV (Convert to Integer) var`	`divVar = sum / (int) var;`
`modVar := sum MOD (Convert to Integer) var`	`modVar = sum % (int) var;`

4.3 The Do Not's

- Do not "declare" variables, that is do not explicitly state the datatype of a variable anywhere other than `INPUT`, `IMPORT`, and `EXPORT`.
- Do not use language specific terms ie: `int`, `long`, `float`, `double`, `%`, `typecast` etc...

5 Arrays and Strings

Type	Pseudo Code	Java
Array Data Type	myArray (ARRAY OF Integer OF SIZE size)	int[] myArray = new int[size];
Array Element	myArray[0] myArray[i]	myArray[0] myArray[i]
Array Size	(SIZE OF myArray) (LENGTH OF myArray)	myArray.length myArray.length
String Data Type	name (String)	String name;
Equality	name EQUALS otherName	name.equals(otherName)

6 Submodules

6.1 How To Declare

Note: You should try to avoid having a submodule spread across two pages. If you do not think there will be enough room to write the entire algorithm just go to the next page.

Pseudo Code	Java

SUBMODULE: **calcAverage**
IMPORT: **sum (Real), count (Integer)**
EXPORT: **avg (Real)**
ALGORITHM:
    <Indented algorithm>
	public double calcAverage(**double** sum, **int** count)
	{
	    /* Code goes here */
	    return avg;
	}

SUBMODULE: **outputAverage**
IMPORT: **sum (Real), count (Integer)**
EXPORT: **none**
ALGORITHM:
    <Indented algorithm>
	public void outputAverage(**double** sum, **int** count)
	{
	    /* Code goes here */
	    /* No return statement */
	}

SUBMODULE: **exportMethod**
IMPORT: **none**
EXPORT: **myVar (Character)**
ALGORITHM:
    <Indented algorithm>
	public char exportMethod()
	{
	    /* Code goes here */
	    return myVar;
	}

Pseudo Code	Java

SUBMODULE: **neitherMethod**
IMPORT: **none**
EXPORT: **none**
ALGORITHM:
    <Indented algorithm>
	public void neitherMethod()
	{
	    /* Code goes here */
	    /* No return statement */
	}

6.2 How To Call

To be consistent you must use the same syntax to assigning the export value of a sub-module call as you do to assign a value to a variable.

Pseudo Code	Java
avg := calcAverage <- sum, count	avg = calcAverage(sum, count);
calcOutputAverage <- sum, count	calcOutputAverage(sum, count);
myChar := exportMethod <- none	myChar = exportMethod();
neitherMethod <- none	neitherMethod();

7 Selection

Note: Avoid "IF (something == true) THEN" and "IF (something == false) THEN" as these are already boolean expressions and do not need comparisons.

Pseudo Code	Java

IF (x = y) THEN
    <Indented algorithm>
END IF
	if (x == y)
	{
	    /* Code goes here */
	}

IF (x = y) THEN
    <Indented algorithm>
ELSE
    <Indented algorithm>
END IF
	if (x == y)
	{
	    /* Code goes here */
	}
	else
	{
	    /* Code goes here */
	}

IF (x = y) THEN
    <Indented algorithm>
ELSE IF (x < y) AND (x > 5) THEN
    <Indented algorithm>
END IF
	if (x == y)
	{
	    /* Code goes here */
	}
	else if ((x < y) && (x > 5))
	{
	    /* Code goes here */
	}

Pseudo Code	Java

IF (x IS 'A' OR 'a') THEN
    <Indented algorithm>
END IF
	if ((x == 'A') || (x == 'a'))
	{
	    /* Code goes here */
	}

IF (LOWER <= x < UPPER) THEN
    <Indented algorithm>
END IF
	if ((LOWER <= x) && (x < UPPER))
	{
	    /* Code goes here */
	}

CASE x
    2: 3:
        <Indented algorithm>
    1: 4:
        <Indented algorithm>
    5:
        <Indented algorithm>
    DEFAULT:
        <Indented algorithm>
END CASE
	switch(x)
	{
	    case 2: **case** 3:
	        /* Code goes here */
	    break;
	    case 1: **case** 4:
	        /* Code goes here */
	    break;
	    case 5:
	        /* Code goes here */
	    break;
	    default:
	        /* Code goes here */
	}

8 Looping

Pseudo Code	Java

FOR i := 0 TO 10 (exclusive) INC BY 1
    <Indented algorithm>
END FOR
	for(**int** i = 0; i < 10; i++)
	{
	    /* Code goes here */
	}

FOR i := 10 TO 0 (exclusive) DIV BY 2
    <Indented algorithm>
END FOR
	for(**int** i = 10; i >= 0; i = i/2)
	{
	    /* Code goes here */
	}

WHILE (myInt < 10) DO
    <Indented algorithm>
END WHILE
	while (myInt < 10)
	{
	    /* Code goes here */
	}

DO <Indented algorithm> WHILE (10 > myInt >= 50)	**do** { /* Code goes here */ } **while**((myInt < 10) && (myInt >= 50))

9 Object Orientation

9.1 Object Creation

This example assumes that wheel is a variable of type `WheelClass` and shows how to indicate use of the default constructor, the alternate constructor (using variables, and values), and the copy constructor.

Pseudo Code	Java
`CONSTRUCT wheel USING default`	`wheel = new Wheel();`
`CONSTRUCT wheel USING sizeString, pressure, rimString, manufacturer`	`wheel = new Wheel(sizeString, pressure, rimString, manufacturer);`
`CONSTRUCT wheel USING 210, 101, 15, 31.43, "SteEl", "Bob Jayne"`	`wheel = new Wheel(210, 101, 15, 31.43, "SteEl", "Bob Jayne");`
`CONSTRUCT wheel USING inEngine`	`wheel = new Wheel(inEngine);`

9.2 Using Objects

This example assumes that wheel is a variable of type `WheelClass` and shows how to indicate use of the constructed object.

Pseudo Code	Java
`wheel.getSize <- none`	`wheel.getSize();`
`wheel.setRim <- "Steel"`	`wheel.setRim("Steel");`
`wheel.toString <- none`	`wheel.toString();`
`wheel.equals <- otherWheel`	`wheel.equals(otherWheel);`

End of Worksheet