Pseudocode

Pseudocode is a generalized English-based language that describes an algorithm. There are many versions of pseudocode so expect deviations from this handout.

An **algorithm** are the steps needed to solve a problem in computer programming.

Once an algorithm has been described in pseudocode it can be coded into any computer-based programming language.

Indentation is part of pseudocode.

Declaration and initialisation. Data type is not specified but is implied

```
1 size <- 4
2 runningAvg <- 0
3 flag <- false
4 name <- "Billy"
5 letter <- 'Z'
6 name <- Get user input: "What is your name?"
```

Arrays. Declaration and initialization

```
1 agentName <- {"Hermione","Fazul","Jean","Thabo"}
2 agentGender <- {'F','M','F','M'}
```

Pseudocode "reserved" words. Often the following words are used with an implied meaning

```
1 size - often refers to the size of an array
2 pos - often means the position (index) in an array
3 flag - often used for Boolean
4 count – a counter variable - number of occurrences
5 begin end - used to define a block of code
```

Where pseudocode is very similar to Java

```
 1 if (condition) // executes if true
 2 if (condition AND condition)  // executes if true
 3 if (condition OR condition)  // executes if true
 4 if (variable1 = variable2)  // executes if they are equal
 5 if (variable1 != variable2)  // executes if not equal (T)
 6 if (variable1 <> variable2)  // executes not equal (T)
 7 while (condition)  // executes if true
 8 while (condition > condition)  // executes if true
 9 while (condition < condition)  // // executes if true
10 while (condition <> variable)  // executes if not equal (T)
11 i++ // increase i by one
12 i--  // decrease i by one
13 i = i + 1  // increase i by one
14 i = i - 1  // decrease i by one
15 break  // used to break out of a loop
16 return  // the instruction to return a value to the call
```

```
17 [ ]  // implies an index within an array
18 ( )  // implies a condition
19 { }  // used when declaring an literal array
```

For loop with a nested if statement

Here a for loop executes looking for "value" inside an array. When it finds it, it breaks out of the loop. Note that indentation is part of pseudocode.

```
1 for (i < size) // size of array. i increases by 1 (implied)
2 begin
3    if (value = valArr [ i ] )
4       pos <- i
5       break  // break out of the loop when found
6    endif
7 end for
```

While loops with a nested if statement

Here a while loop executes looking for "value" inside an array. When it finds it, the flag becomes true and the position is assigned to "pos"

```
1 while (i < size AND flag = false)
2 begin
3    if (value = valArr [ i ] )
4       flag <- true
5       pos <- i
6    endif
7    i = i + 1
8 end while
```

A typed method that returns a value

Here a method is coded in pseudocode; the method is looking for the position of a value in an array. The method accepts a **parameter** i.e. the value being searched for. The position in the array (an integer) is returned.

```
1 Method searchStop (int value) for parameter
2 begin
3    code . . . uses a loop
4    code . . . to find the position
5    pos <- i
6    return pos
7 end method
```

Output to the monitor

Here we are looking to count the factors of a certain number (The factors of 12 are 1,2,3,4,6 and 12). We use a while loop with a nested if statement to do this.

The if statement looks for division where the remainder is equal to zero (uses MOD for this). The number of factors found is printed to the monitor.

```
while (count <= number)
begin
   if (number MOD count = 0)  //  finds factors
   begin
      numOfFactors <- numOfFactors + 1
   end if
   count <- count + 1
end while

Display "The number has " + numOfFactors
```

Parallel arrays

The first array has the person's name. The second array stored their gender.

Here we are looking for a name inside an array. We want to exit out of the array as soon as the name is found, therefore we use a flag. Once found we record the position in "pos". We then display the name that we found as well as their gender (found in the matching **parallel** gender array)

```
agentName <- {"Hermione", "Fazul", "Jean", "Thabo"}
agentGender <- {'F', 'M', 'F', 'M'}  // their gender

size <- 4  // Four members in each array
i <- 0
pos <- 0
flag <- false

while (i < size AND flag = false)
begin
   if (name = agentName[i]
   begin
      flag <- true
      pos <- I  // store position for output
   end if

   i++
end while

Display name[pos], agentGender[pos]
```

Calling methods inside an instantiated object

We can call a method inside an instantiated object in the normal way **using dot notation** e.g.

```
popular <- bookArr[i].getTimesBorrowed( )

return "The book is: " + BookArr[pos].getTitle( )
```

Here bookArr is an array of book objects.

- Each book object has a "getTimesBorrowed" method that returns a value i.e. how many times the book has been borrowed.
- The book that has been borrowed the most is therefore the most popular book
- Each book object has a "getTitle" method that returns a string i.e. The title of the book

```
pos <- 0
popular <- 0
size <- number of elements in bookArr

for (i <- to size)
begin
   if (bookArr[i].getTimesBorrowed( ) > popular
   begin
      popular <- bookArr[i].getTimesBorrowed()
      pos <- i
   end if
end loop

return "Most popular book is: " + BookArr[pos].getTitle()

end for loop
```