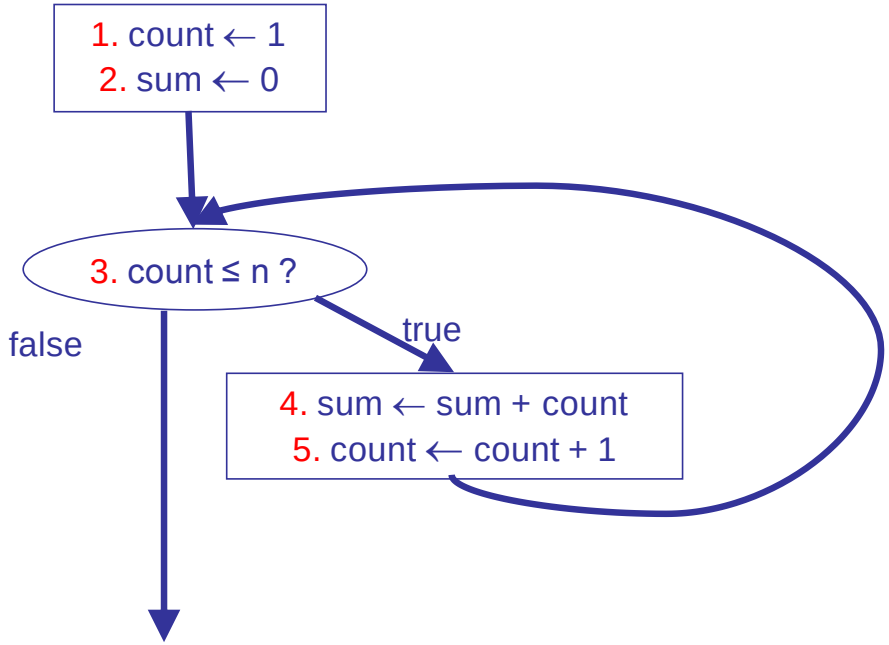
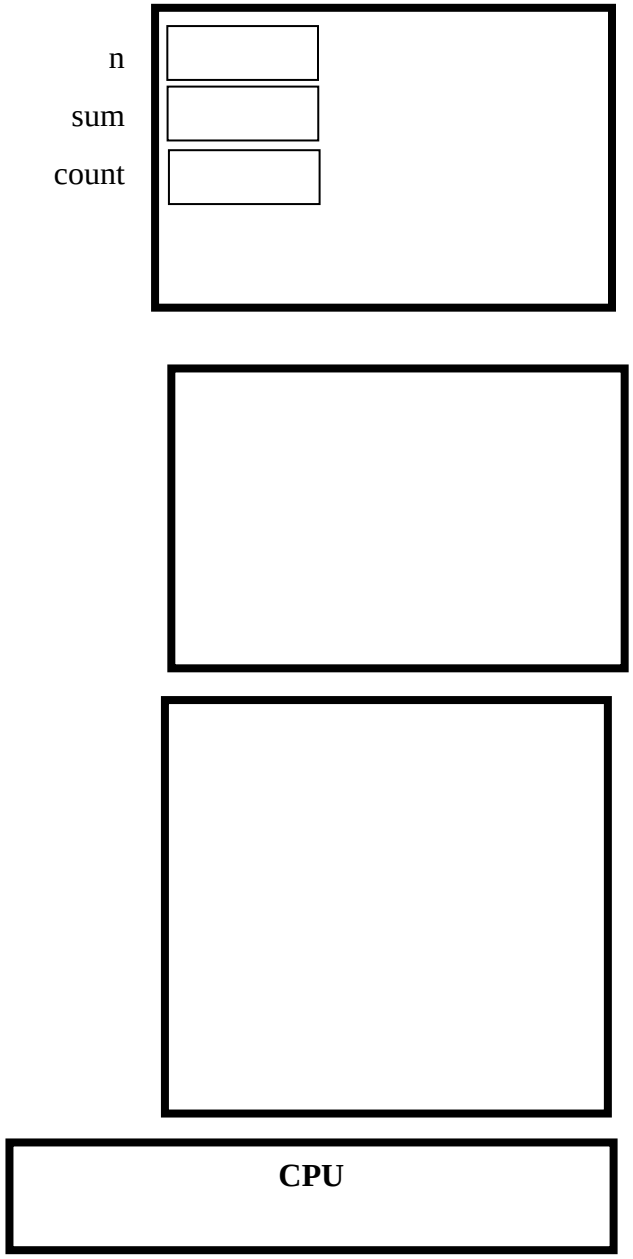
Program Memory

Exercise 6-1 - Sum from 1 to N

Working memory

GIVEN: n (a positive integer)
INTERMEDIATE: count (index going from 1 to n)
RESULT: sum (sum of integers 1 to N)
HEADER: sum ← sum1ToN(n)
BODY:

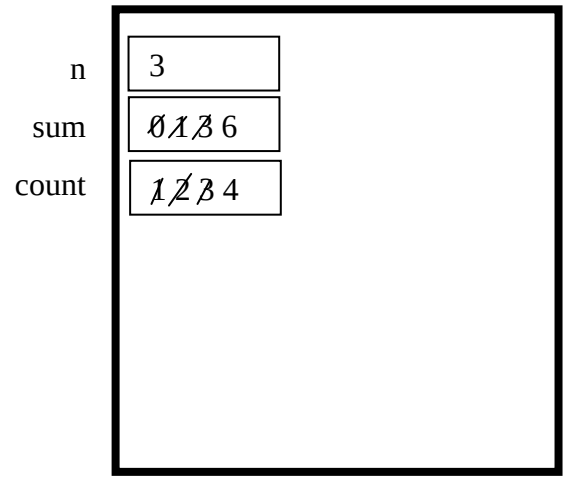




Exercise 6-1: Trace of sum1toN(3)

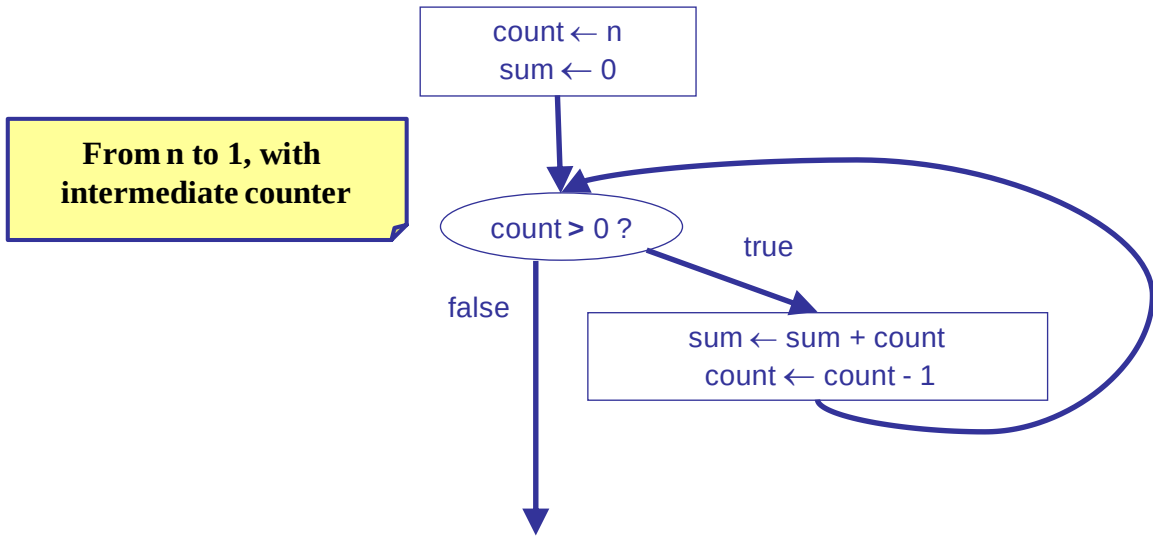
	n	count	sum
Init.	**3**	**?**	**?**
1.		**1**	
2.			**0**
3. TRUE			
4.			**1**
5.		**2**	
3. TRUE			
4.			**3**
5.		**3**	
3. TRUE			
4.			**6**
5.		**4**	
3. FALSE			

Working Memory



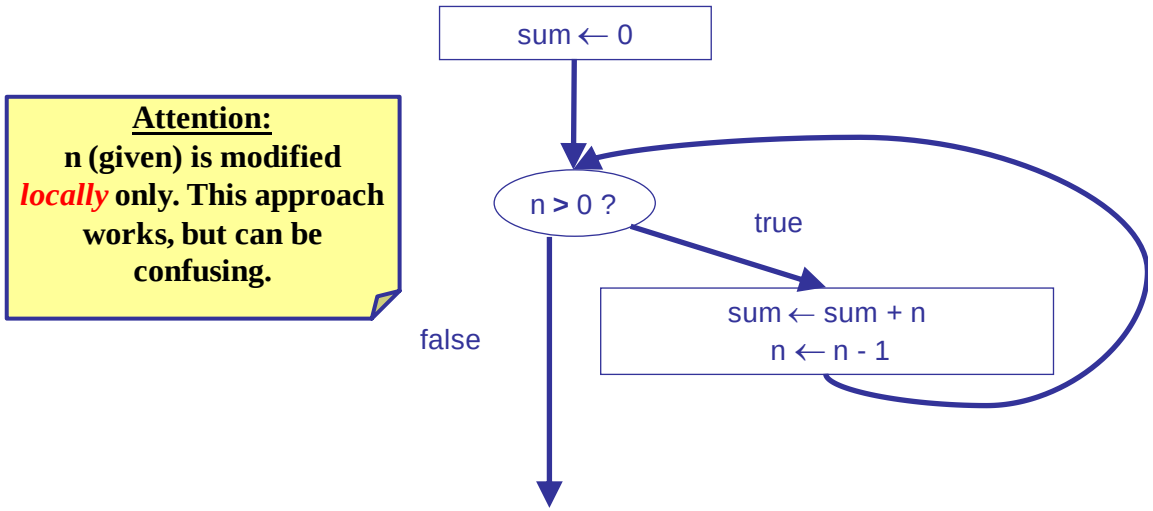
Exercise 6-1 - Sum from 1 to N: Variation (I)

GIVEN: n (a positive integer)
INTERMEDIATE: count (index going from n to 1)
RESULT: sum (sum of integers 1 to n)
HEADER: sum ← sum1ToN(n)
BODY:

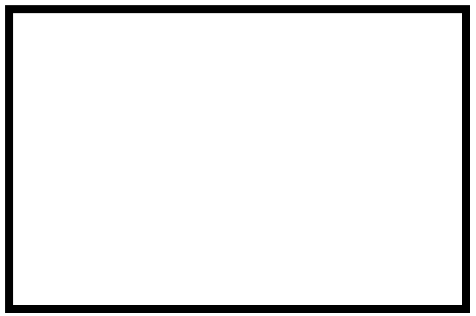


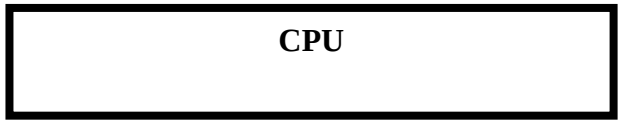
CPU

GIVEN: n (a positive integer)
INTERMEDIATE: (none)
RESULT: sum (sum of integers 1 to N)
HEADER: sum ← sum1ToN(n)
BODY:



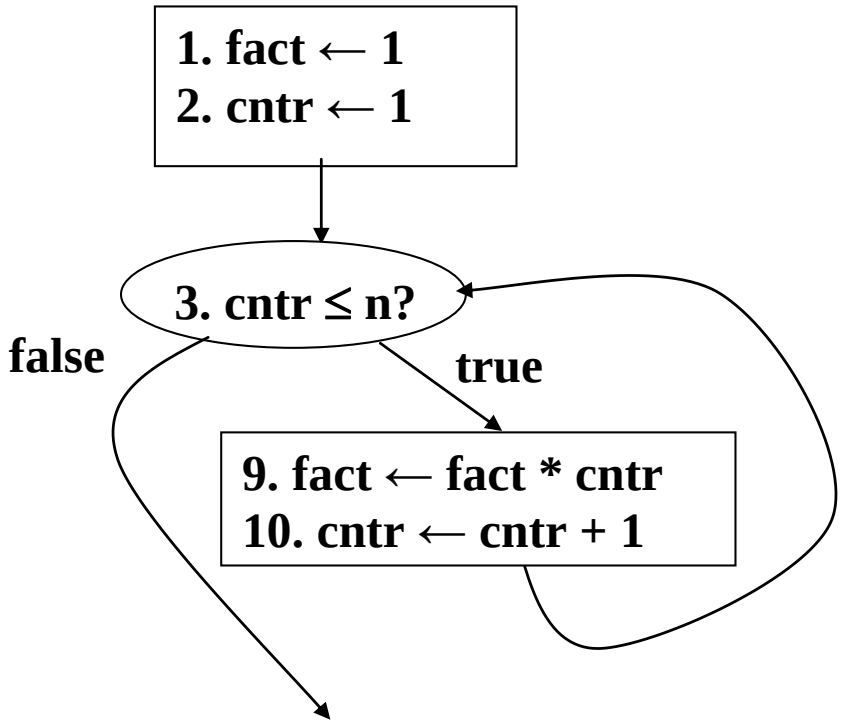
Attention:
n (given) is modified *locally* only. This approach works, but can be confusing.





Program Memory **Exercise 6-2 – A definite loop** **Working memory**

GIVENS: n (number to compute factorial)
RESULTS: fact (n factorial)
INTERMEDIATES: cntr (to count from 1 to N)
ASSUMPTIONS: n should be positive
HEADER: fact ← factorial (n)
BODY:



CPU

GIVENS:

aNumber (number)
divisor (divisor – serves as log base)

RESULTS:

intPart (integer part of the logarithm, number of times aNumber divisible by divisor)

INTERMEDIATES:

quotient (quotient from division)

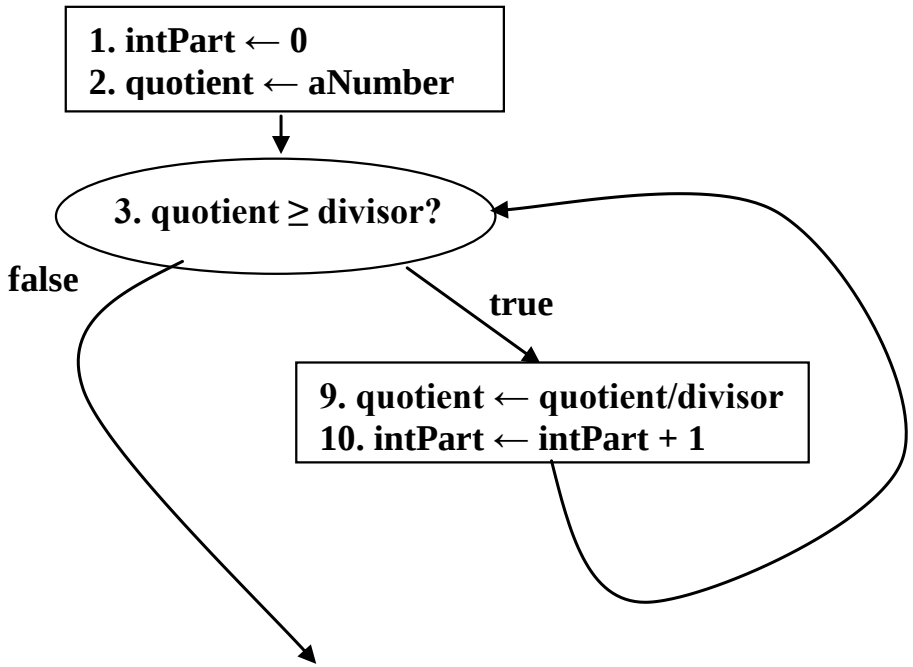
ASSUMPTIONS:

aNumber and divisor should be positive

HEADER:

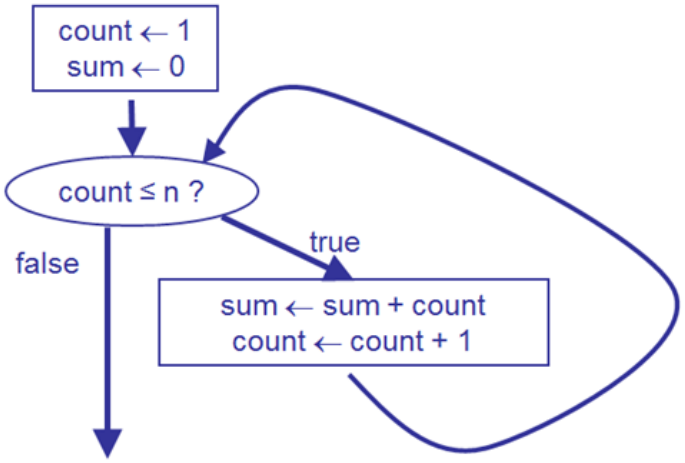
intPart ← logInt (aNumber, divisor)

BODY:



CPU

GIVEN: n (a positive integer)
INTERMEDIATE: count (index going from 1 to N)
RESULT: sum (sum of integers 1 to N)
HEADER: sum ← sum1ToN(n)
BODY:




```
public static int sum1ToN (int n)
   {
      // initialize counter
      // and accumulator sum
      int count = 1;
      int sum = 0;
      // loop
      while (count <= n)
      {
         sum = sum + count;
         count = count + 1;
      }
      // return the result
      return sum;
   }
```

CPU

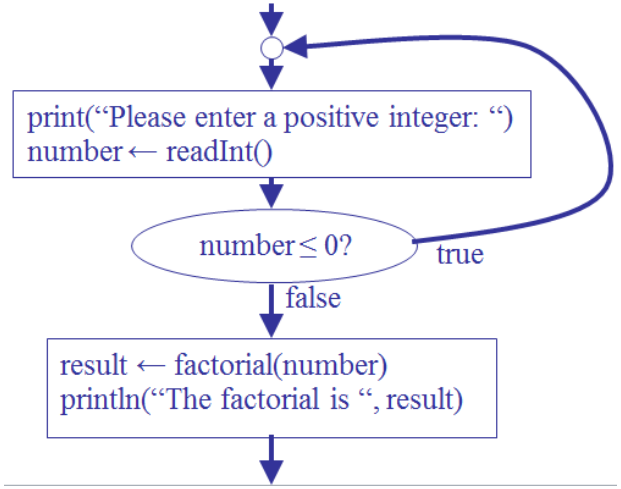
Program Memory

Exercise 6-5 – Example of Post-Test Loop

Working memory

GIVENS: (none)
RESULTS: (none)
INTERMEDIATES: number (number to compute factorial)
result (the factorial, i.e. number!)
CONSTRAINTS: number should be positive
HEADER: main

BODY:



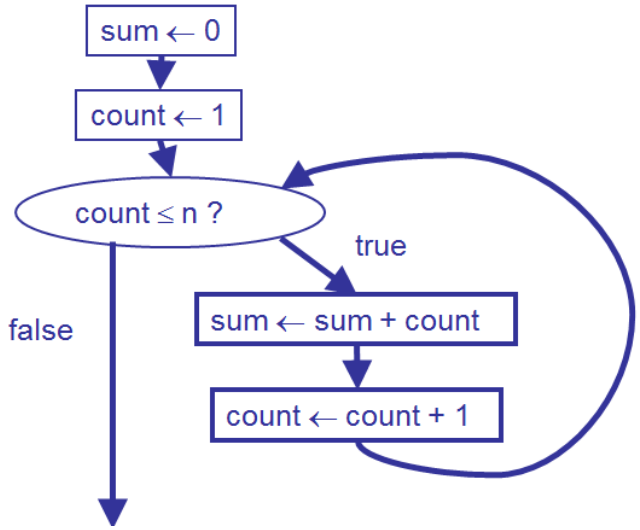

```
// Translation to Java
public static void main(String args)
{
    // Variables
    int number; // number provided by user
    int results; // factorial, i.e. number!

    // Body
    do // Post-test loop
    {
      System.out.print("Please enter a positive integer: ");
      number = ITI1120.readInt();
    } while(number <= 0);
    reslt = factorial(number);
    System.out.println("The factorial is "+number);
}
```

CPU

Program Memory

Exercise 6-6 – FOR loop to add 1 to N

Working memory

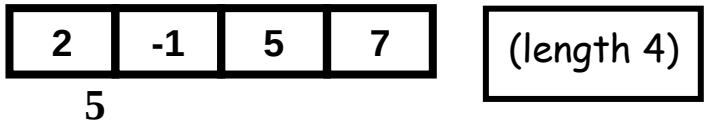



```
// Translation to Java
sum = 0 ;
for ( count = 1; count <= n; count = count + 1 )
{
   sum = sum + count ;
}
```

CPU

Exercise 6-7 – Array Indexing

- The index (subscript) of an array of length l may be any integer expression that returns a value in the range 0...(l-1).
  - Suppose k =2, and A references

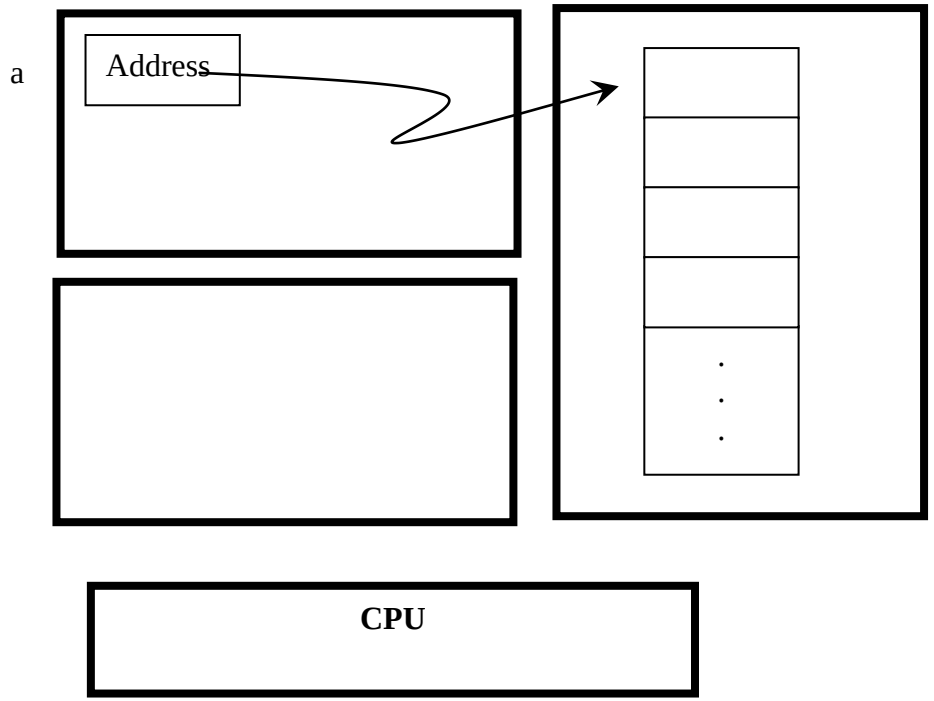


a[2] = **5**
a[k] = **5**
a[2*k-1] = **7**
a[a[0]+1] = **7**

- A[expression] is just like any ordinary variable and can be used anywhere an ordinary variable can be used.
- Remember a is a reference variable

Program Memory **Exercise 6-8 – Value in Middle of Array** **Working memory** **Global Memory**

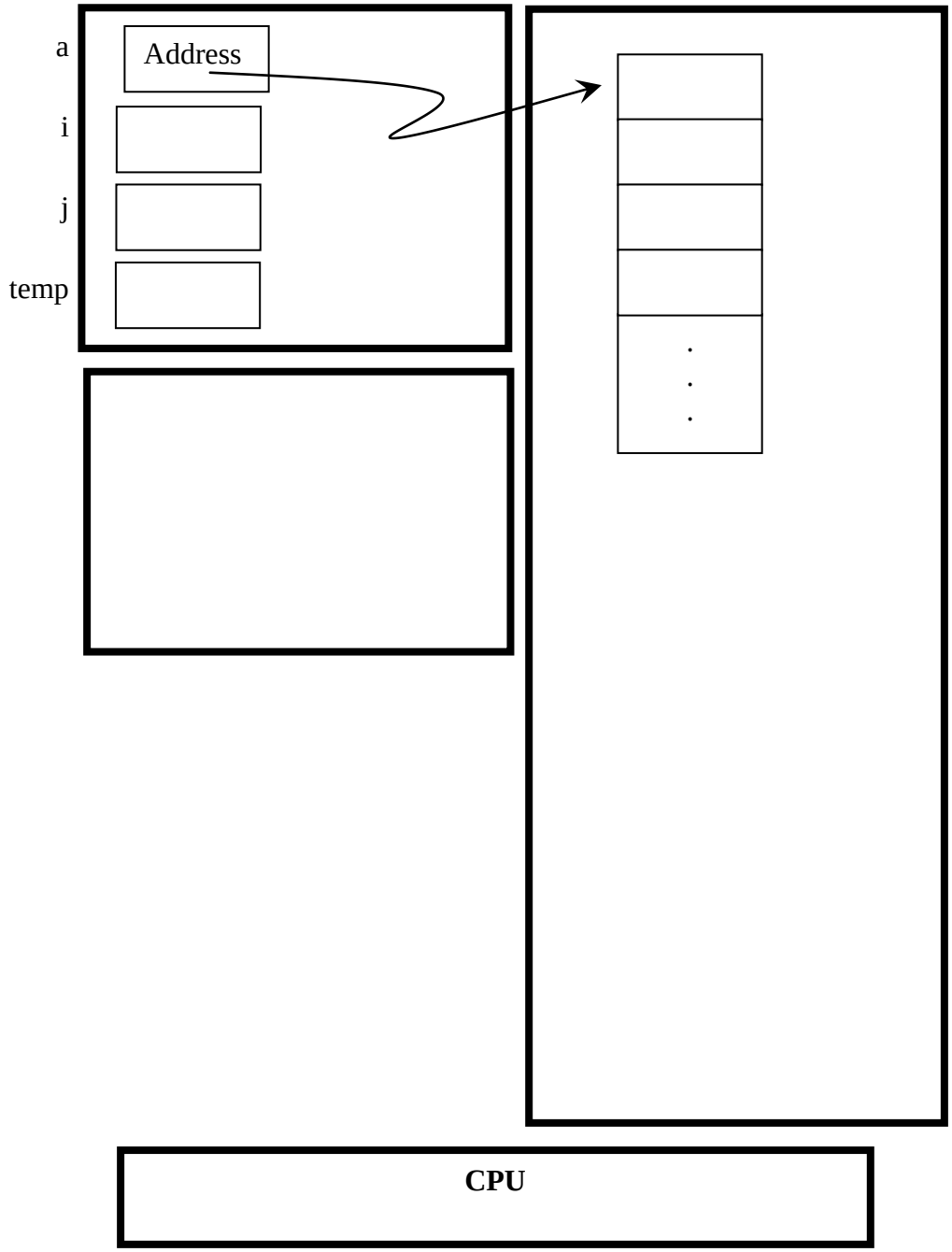
GIVENS: a (an array of real numbers)
n (an odd integer, the length of a)
RESULT: mid (the middle member of a)
INTERMEDIATES: **(none)**
HEADER: **mid ← middle (a, n)**
BODY
mid ← a [(n – 1) / 2]



Program Memory **Exercise 6-9 – Swap Values in an Array** **Working memory** **Global Memory**

GIVENS: a (an array of integers)
i, j (two indices)
MODIFIEDS: **a (with a[i] and a[j] swapped)**
INTERMEDIATES:
temp (stores a[i] temporarily)
RESULT: **(None)**
HEADER: swap(a, i, j)
BODY

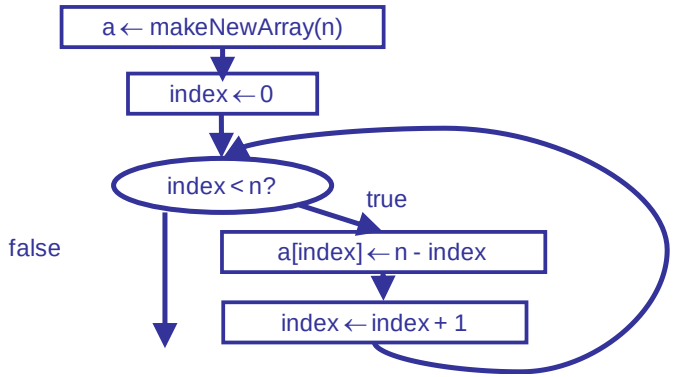
temp ← a[i]
a[i] ← a[j]
a[j] ← temp



Program Memory

Exercise 6-10 – Creating an Array

GIVENS: n (a positive integer)
RESULTS: a (an reference to an array containing n…1)
INTERMEDIATES: **index (an index for A)**
HEADER: **a ← createNDownTo1(n)**
BODY:

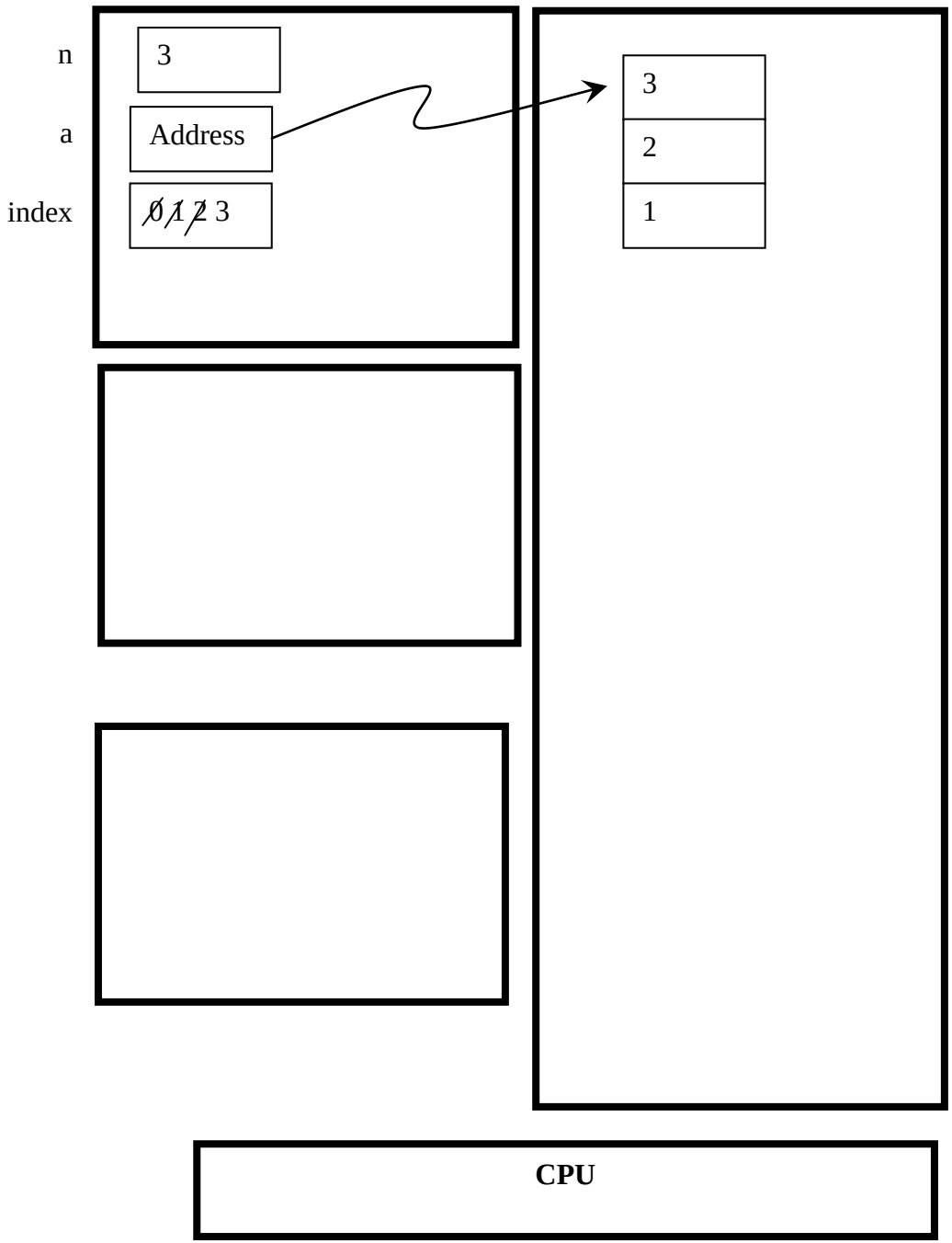


Trace for N=3

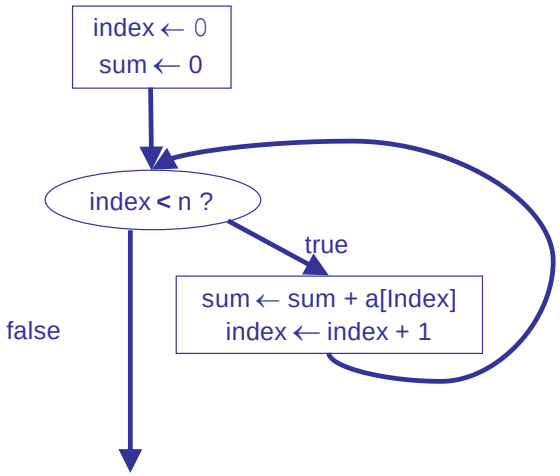
statements	n	index	a
initial values	**3**	**?**	**?**
1. a ← makeArray(3)			**{?, ?, ?}**
2. index ← 0		**0**	
3. index < n? true			
4. a[index] ← n – index			**{3,?,?}**
5. index ← index + 1		**1**	
3. index < n? true			
4. a[index] ← n – index			**{3,2,?}**
5. index ← index + 1		**2**	
3. index < n? true			
4. a[index] ← n – index			**{3,2,1}**
5. index ← index + 1		**3**	
3. index < n? false			

Working memory

Global Memory



Algorithm Model	Java
Exercise 6-11 – Find the sum of the values in an array containing N values	

GIVENS: a (An array of numbers)
n (Number of array elements)
INTERMEDIATE: index (Array index going from 0 to n-1)
RESULT: sum (Sum of array contents)
HEADER: sum ← sumArray(a,n)
BODY:
 
 	
	`public static int[] sumArray(int[] a)`
	`{`
	`  // note, given n is a.length`
	`  // Results`
	`  int sum;`
	`  // intermediate`
	`  int index;`
	`  //`
	`  sum = 0;  // intialiase sum`
	`  // loop with for`
	`  for(index = 0; index < n; index++)`
	`  {`
	`  sum = sum + a[index];`
	`  }`
	`  // return the result`
	`  return sum;`
	`}`

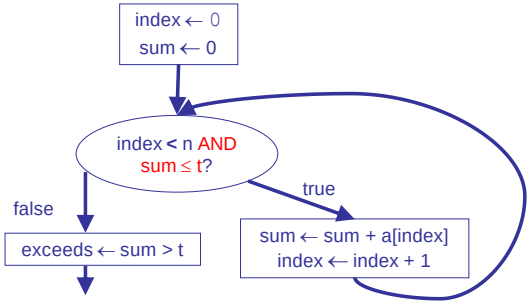
Exercise 6-12 (a) – Given a value T and an array X containing N values, check if the sum of X's values exceeds T.	

GIVENS: a (An array of numbers)
n (Number of array elements)
t (A "threshold" value)
INTERMEDIATE: sum (The sum of the array, from example 2)
RESULT: exceeds (Boolean: True if sum > t and False otherwise)
HEADER: exceeds ← sumLargerThanT(a,n,t)
BODY:
sum ← sumArray(a,n)
exceeds ← sum > t
	`public static boolean sumLargerThanT(int[] a, int t)`
	`{`
	`  // note, given n is a.length`
	`  // Results`
	`  int exceeds;`
	`  // intermediate`
	`  int sum;`
	`  //`
	`  sum = sumArray(a);  // get sum`
	`  exceeds = sum > t;`
	`  // return the result`
	`  return exceeds;`
	`}`

Algorithm Model	Java

Exercise 6-12 (b) – Given a value T and an array X containing N values, check if the sum of X's values exceeds T.

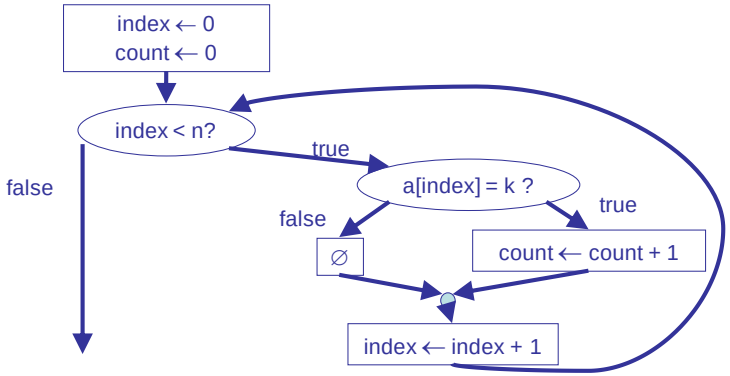
GIVENS: a (An array of numbers)
n (Number of array elements)
t (A "threshold" value)
INTERMEDIATE:
index (Array index going from 0 to n-1)
sum (The sum of the array elements)
RESULT: exceeds (Boolean: True if sum > t and False otherwise)
HEADER: exceeds ← sumLargerThanT(a,n,t)
BODY:




```
public static boolean sumLargerThanT(int[] a, int t)
{
   // note, given n is a.length
   // Results
   int exceeds;
   // intermediate
   int index
   int sum;
   // loop using while
   sum = 0;
   index = 0;
   while(index < n && sum <= t)
   {
        sum = sum+a[index];
        index = index+1;
   }
   exceeds = sum > t;
   // return the result
   return exceeds;
}
```

Exercise 6-13 – Count how many times K occurs in an array containing N values.

GIVENS: a (An array of numbers)
n (Number of array elements)
k (Value for which to count instances)
INTERMEDIATES:
Index (Array index going from 0 to n-1)
RESULT: count (Number of times value of k is contained in A)
HEADER: count ← countK(a,n,k)
BODY:




```
public static int countK(int[] a, int k)
{
   // Results
   int count;
   // intermediate
   int index;
   // intialiase sum and index
   count = 0;
   for(index = 0; index < a.length; index=index+1) // loop
   {
      if(a[index] == k)
      {
        count = count + 1;
      }
      else { /* do nothing */ }
   }
   return count; // return the result
}
```

Exercise 6-14 (a) – Given an array X of N values and a number K, see if K occurs in X or not.

Algorithm Model

GIVENS: a (An array of numbers)
n (Number of array elements)
k (Value to find)
INTERMEDIATES:
count (Number of times K is in array, from algorithm 4)
RESULT: found (Boolean : true if K is in array and false otherwise)
HEADER: found ← findK(a,n,k)
BODY:
count ← countK(a,n,k)
found ← count > 0

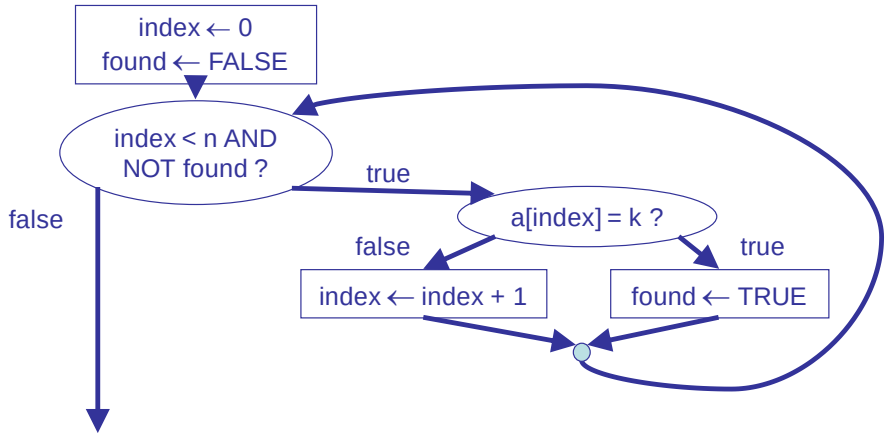
Java

```
public static boolean findK(int[] a, int k)
{
  // Results
  boolean found;
  // intermediates
  int count;
  // Body
  count = countK(a, k);
  found = count > 0;
  // Return results
  return found;
}
```

Exercise 6-14 (b) – Given an array X of N values and a number K, see if K occurs in X or not.

Algorithm Model

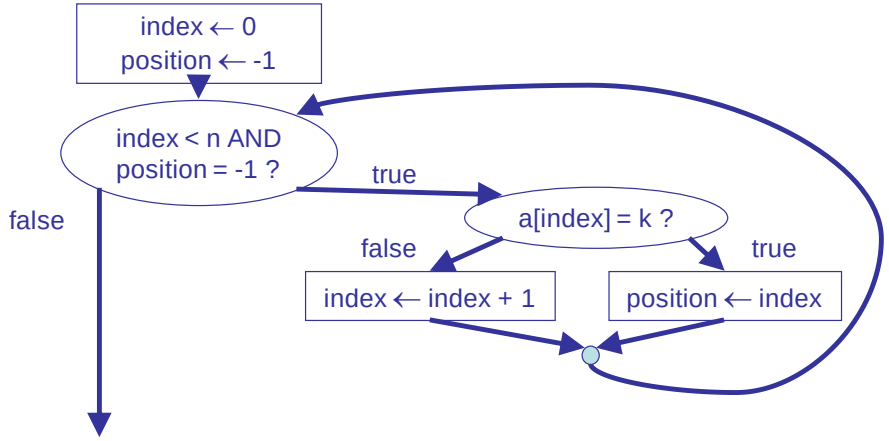
GIVENS: a (An array of numbers)
n (Number of array elements)
k (Value to find)
INTERMEDIATES:
index (Array index going from 0 to n-1)
RESULT: found (Boolean : true if K is in array and false otherwise)
HEADER: found ← findK(a,n,k)
BODY:



Java

```
public static boolean findK(int[] a, int k)
{
   boolean found; // results
   int index; // intermediate
   // initialize found and index
   found = false;
   index = 0;
   while(index < a.length && !found) // loop
   {
      if(a[index] == k)
      {
        found = true;
      }
      else { /* do nothing */ }
      index=index+1;
   }
   return found; // return the result
}
```

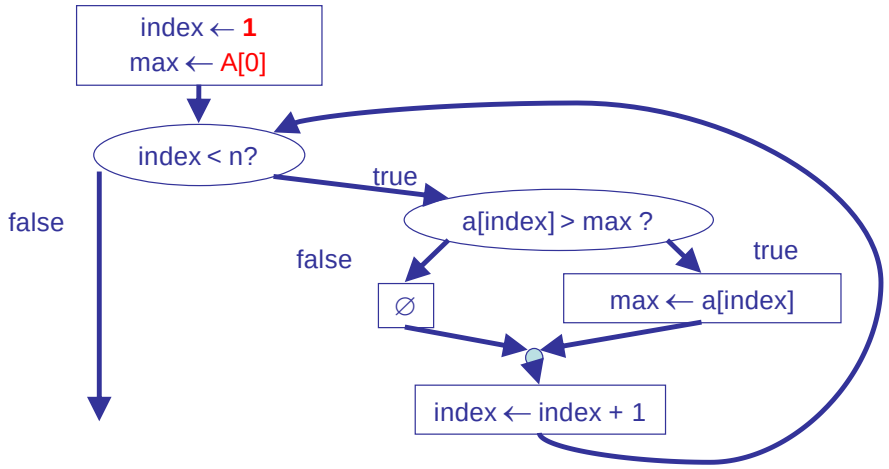
Algorithm Model	Java
Exercise 6-15 – Given an array X of N values and a number K, find the position of the first occurrence of K. (If K does not occur, return -1 as the position.)	

GIVENS: a (An array of numbers)
n (Number of array elements)
k (Value to find)
INTERMEDIATES:
index (Array index going from 0 to n-1)
RESULT: position (Position of K in array, or -1 if K is not contained in array)
HEADER: position ← whereIsK(a,n,k)
BODY:
 
 	
	`public static int whereIsK(int a[], int n, int k)`
	`{`
	`int position; // result`
	`int index; //intermediate`
	`// Body`
	`index = 0;`
	`position = -1;`
	`while( (index < n) && (position == -1) )`
	`{`
	`if(a[index] == k)`
	`{`
	`position = index;`
	`}`
	`else {/* do nothing */ }`
	`index = index + 1;`
	`}`
	`// Return results`
	`return position;`
	`}`
	`}`

Algorithm Model	Java

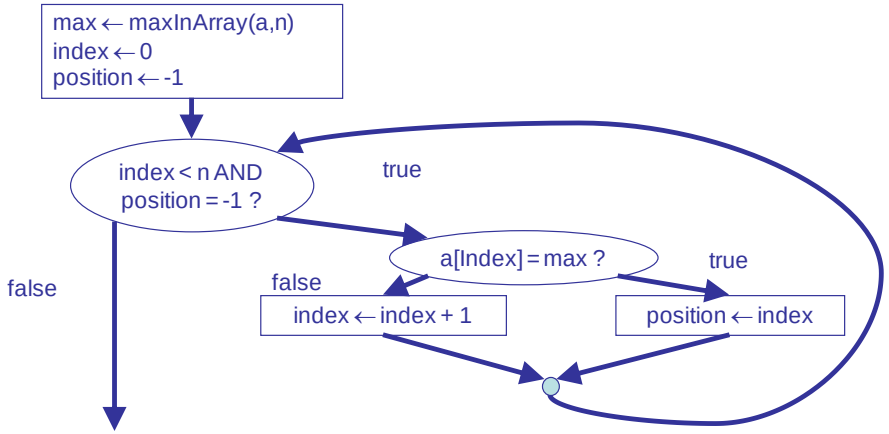
Exercise 6-16 – Find the maximum value in an array containing N values.

GIVENS: n (a positive integer)
a (array containing N values)
INTERMEDIATE: index (indices for a)
RESULT: max (maximum member of a)
HEADER: max ← maxInArray(a, n)
BODY

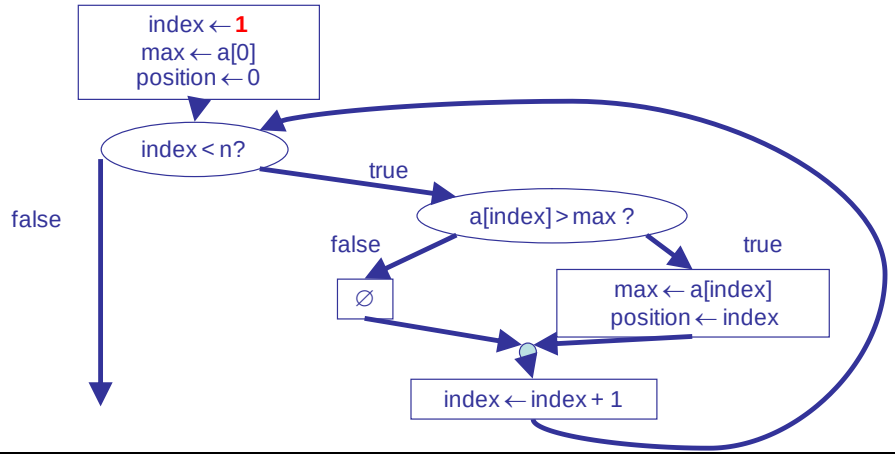



```
public static int maxInArray(int [] a)
{
   // DATA DICTIONARY
   // GIVEN: a An array of int`s
   // RESULT:
   int max; // The maximum in the array
   // INTERMEDIATES:
   int n;  // The length of array a
   int index; //Index into array
   // Initialise n
   n = a.length;
   // ALGORITHM BODY
   max = a[0];
   index = 1;
   while (index < n)
   {
      if (a[index] > max)
      {
         max = a[index]; //update
      }
      else
      {
         /* do nothing */ ;
      }
      index = index + 1;
   }

   // Return results
   return max;
}
```

Algorithm Model	Java
Exercise 6-17 (a) – Find the position of the first occurrence of the maximum value in an array containing N values.	

GIVENS: a (An array of numbers)
n (Number of array elements)
INTERMEDIATES:
index (Array index going from 0 to n-1)
max (Maximum value contained in a)
RESULTS: position (Position of Maximum value contained in a)
HEADER: position ← maxPosInArray(a,n)
BODY:
 
 	
	`public static int maxPosInArray(int [] a)`
	`{`
	`// GIVEN: a An array of int's`
	`int position; //RESULT: Position of max in the array`
	`// INTERMEDIATES:`
	`int n;// The length of array a`
	`int index; //Index into array`
	`int max; // maximum value`
	`// Body`
	`max = maxInArray(a); // get the max value`
	`index = 0; // search for max value in the array`
	`position = -1;`
	`while(index < a.length && position == -1)`
	`{`
	`if(a[index] == max)`
	`position = index;`
	`else`
	`index = index + 1;`
	`}`
	`return position; // Return the results`
	`}`

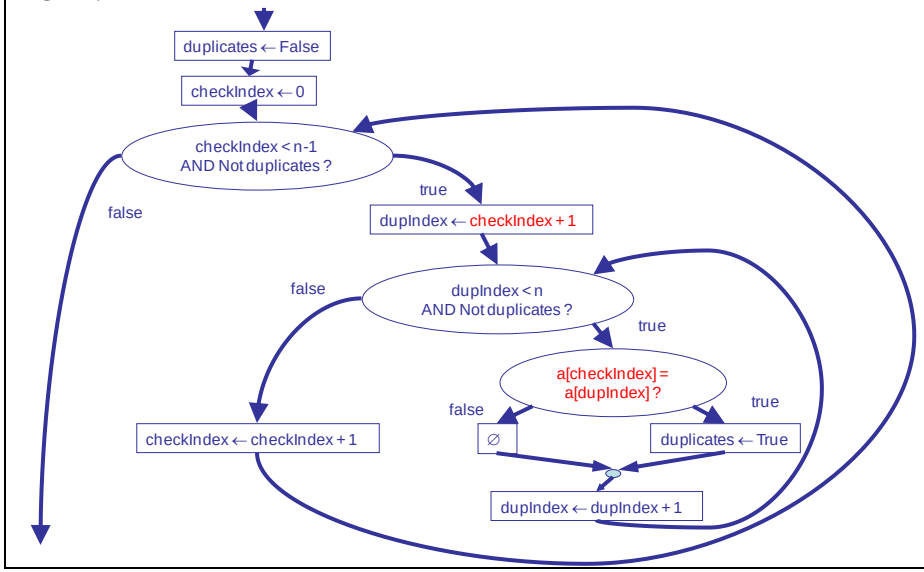
Exercise 6-17 (b) – Find the position of the first occurrence of the maximum value in an array containing N values.	
GIVENS: a (An array of numbers) n (Number of array elements) INTERMEDIATES: max (Maximum in array from example 7) RESULTS: position (Position of Maximum value contained in A) HEADER: position ← maxPosInArray(a,n) BODY: max ← maxInArray(a,n) position ← whereIsK(a,n,max)	`public static int maxPosInArray(int [] a)` `{` `// DATA DICTIONARY` `// GIVEN: a An array of int`s` `int position; // // RESULT: Position of max in array` `// INTERMEDIATES:` `int n;// The length of array a` `int index; //Index into array` `int max; // maximum value` `// Body` `max = maxInArray(a); // get the max value` `position = whereIsK(a, max); // get the position` `return position; // Return the results` `}`

Algorithm Model	Java
Exercise 6-17 (c) – Find the position of the first occurrence of the maximum value in an array containing N values.	
GIVENS: a (An array of numbers) n (Number of array elements) INTERMEDIATES: index (Array index going from 0 to n-1) max (Maximum value contained in a) RESULTS: position (Position of Maximum value contained in a) HEADER: position ← maxPosInArray(a,n) BODY:  	`public static int maxPosInArray(int [] a)` `{` `// GIVEN: a An array of int`s` `int position; // // RESULT: Position of max in array` `// INTERMEDIATES:` `int index; //Index into array` `int max; // maximum value` `// Body` `index = 1;` `max = a[0];` `position = 0;` `while(index < a.length)` `{` `if(a[index] > max)` `{` `max = a[index];` `position = index;` `}` `else index = index + 1;` `}` `return position; // Return the results` `}`

Exercise 6-18 – Check if an array of N values contains any duplicates.

Algorithm Model

GIVENS: a (An array of numbers)
n (Number of array elements)
INTERMEDIATES:
checkIndex (Array index for current element)
dupIndex (Array index for duplicate checking of current element)
RESULT: duplicates (True if there are duplicates in A and false otherwise)
HEADER: duplicates ← hasDuplicates(a,n)
BODY:



Java

```
public static boolean hasDuplicates(int [] a)
{
   // GIVEN: a An array of int`s, a.length used for n
   boolean duplicates; // RESULT: true if dupls found
   int checkIndex; // INTERMEDIATE: Index of current element
   int dupIndex; // INTMEDT: checking of dupls for current
   // ALGORITHM BODY
   duplicates = false;   checkIndex = 0;
   while (checkIndex < a.length-1 && !duplicates)
   {
      dupIndex = checkIndex + 1;
      while(dupIndex < a.length && !duplicates)
      {
         if (a[checkIndex] == a[dupIndex])
         {
            duplicates = true; // found a duplicate
         }
         else { /*do nothing*/ }
         dupIndex = dupIndex + 1;
      }
      checkIndex = checkIndex + 1;
   }
   return duplicates; // Return results
}
```

Exercise 6-19: Comparing Strings

- What is the value of `result` for these examples?
  - Example 1:

```
String str1 = "abcde" ;
String str2 = "abcfg" ;
int result = str1.compareTo(str2);
```

`result` **is less than zero**

  - Example 2:

```
String str1 = "abcde" ;
String str2 = "ab" ;
int result = str1.compareTo(str2);
```

`result` **is greater than zero**