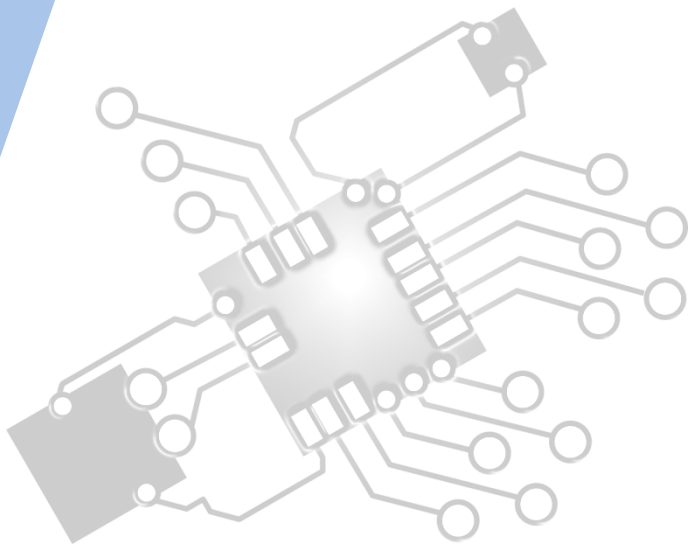
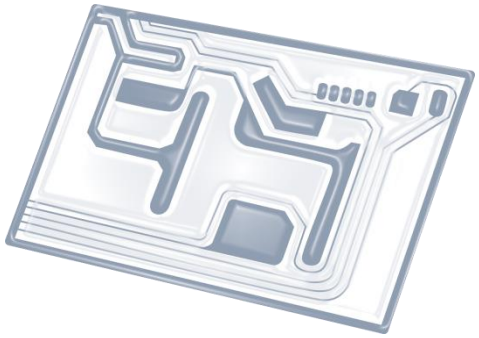
Abstract Data Structures

IB Computer Science



Content developed by
Dartford Grammar School
Computer Science Department

HL Topics 1-7, D1-4



1: System design

2: Computer Organisation

3: Networks

4: Computational thinking

5: Abstract data structures

6: Resource management

7: Control

D: OOP

HL only 5 Overview

Thinking recursively
5.1.1 Identify a situation that requires the use of recursive thinking
5.1.2 Identify recursive thinking in a specified problem solution
5.1.3 Trace a recursive algorithm to express a solution to a problem
Abstract data structures
5.1.4 Describe the characteristics of a two-dimensional array
5.1.5 Construct algorithms using two-dimensional arrays
5.1.6 Describe the characteristics and applications of a stack
5.1.7 Construct algorithms using the access methods of a stack
5.1.8 Describe the characteristics and applications of a queue
5.1.9 Construct algorithms using the access methods of a queue
5.1.10 Explain the use of arrays as static stacks and queues
Linked lists
5.1.11 Describe the features and characteristics of a dynamic data structure
5.1.12 Describe how linked lists operate logically
5.1.13 Sketch linked lists (single, double and circular)
Trees
5.1.14 Describe how trees operate logically (both binary and non-binary)
5.1.15 Define the terms: parent, left-child, right-child, subtree, root and leaf
5.1.16 State the result of inorder, postorder and preorder tree traversal
5.1.17 Sketch binary trees
Applications
5.1.18 Define the term dynamic data structure
5.1.19 Compare the use of static and dynamic data structures
5.1.20 Suggest a suitable structure for a given situation

1: System design

2: Computer Organisation

3: Networks

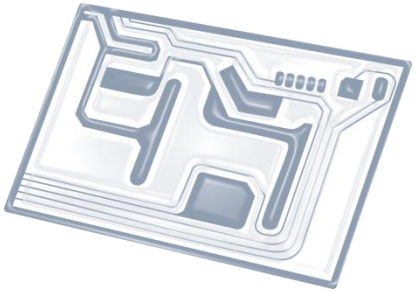
4: Computational thinking

5: Abstract data structures

6: Resource management

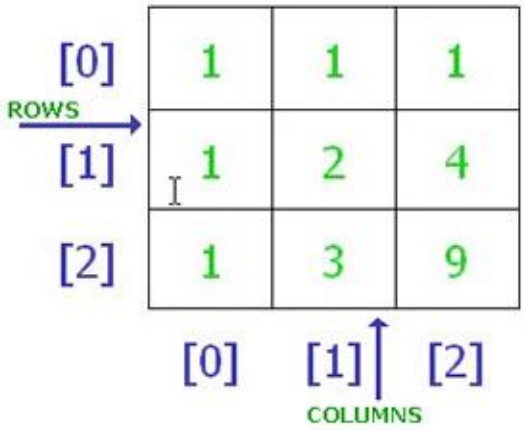
7: Control

D: OOP



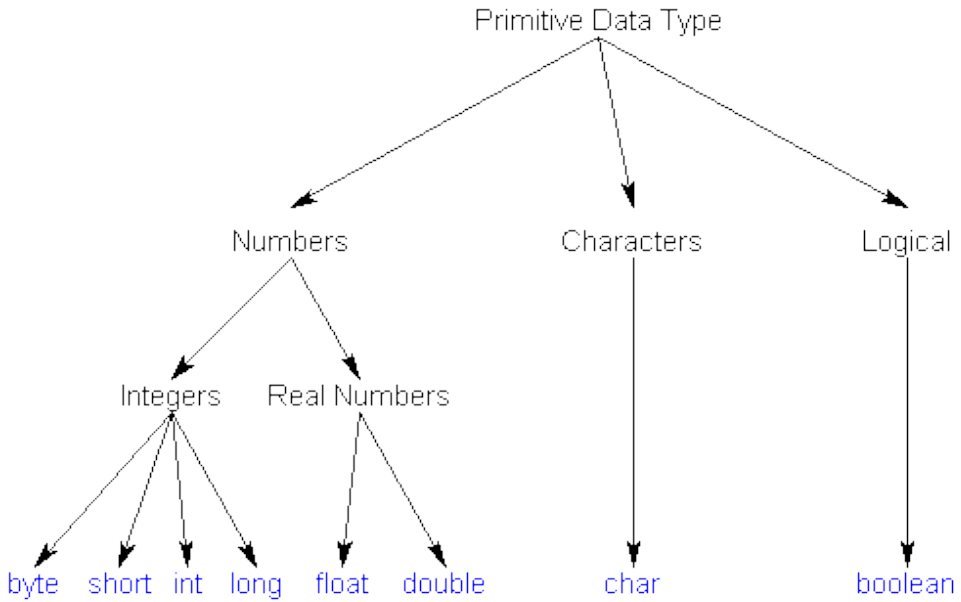
Topic 5.1.5

Construct algorithms using two-dimensional arrays



Abstract Data Structures (ADTs)

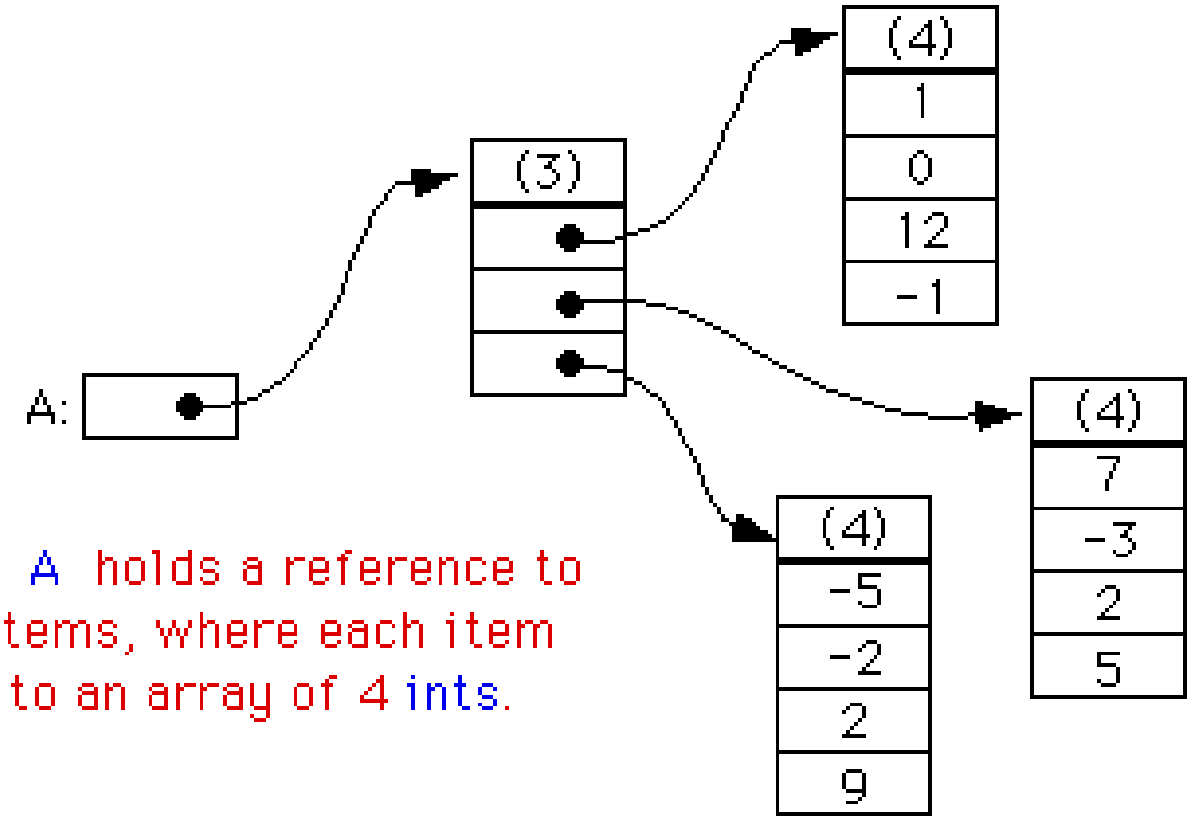
- 2D array
- Stack
- Queue
- Linked List
- (Binary) Tree
- Recursion



If you create an array `A = new int[3][4]`, you should think of it as a "matrix" with 3 rows and 4 columns.

A:

1	0	12	-1
7	-3	2	5
-5	-2	2	9

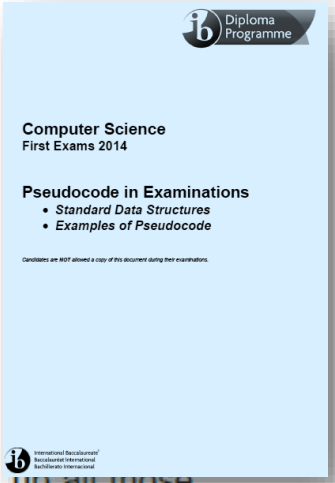


But in reality, A holds a reference to an array of 3 items, where each item is a reference to an array of 4 ints.

```
int rows = 6;
int columns = 5;

int i, j;

for (i=0; i < rows ; i++) {

    for (j=0; j < columns ; j++) {
        System.out.print( aryNumbers[i][j] + " " );
    }
    System.out.println( "" );
}
```

Task	Java Syntax	Examples
Declare a 2D array	`type[][] name`	`int[][] matrix` `Pixel[][] pixels`
Create a 2D array	`new type[nRows][nCols]`	`new int[5][8]` `new Pixel[numRows][numCols]`
Access an element	`name[row][col]`	`int value = matrix[3][2];` `Pixel pixel = pixels[r][c];`
Set the value of an element	`name[row][col] = value`	`matrix[3][2] = 8;` `pixels[r][c] = aPixel;`
Get the number of rows	`name.length`	`matrix.length` `pixels.length`
Get the number of columns	`name[0].length`	`matrix[0].length` `pixels[0].length`

Key algorithm: Averaging

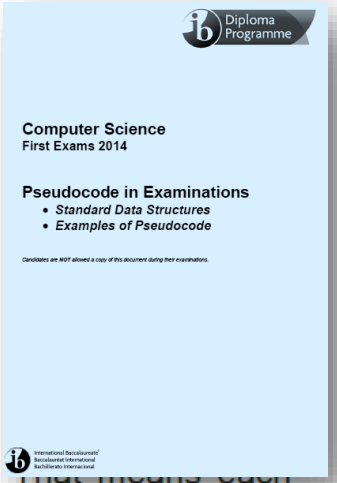


AVERAGING AN ARRAY

The array STOCK contains a list of 1000 whole numbers (integers). The following 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
```

Key algorithm: List to Array

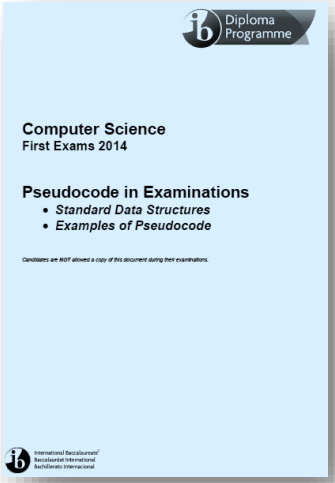


COPYING FROM A COLLECTION INTO AN ARRAY

The following pseudocode presents an algorithm that reads all the names in `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
```

Key algorithm: Factors



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 "
```