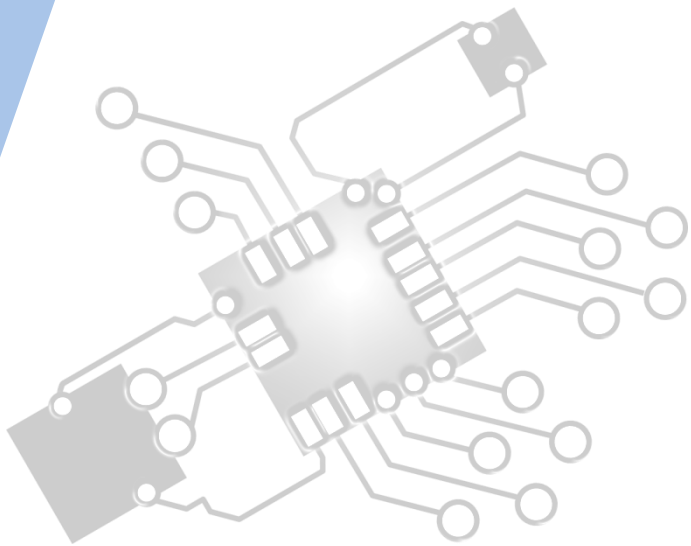
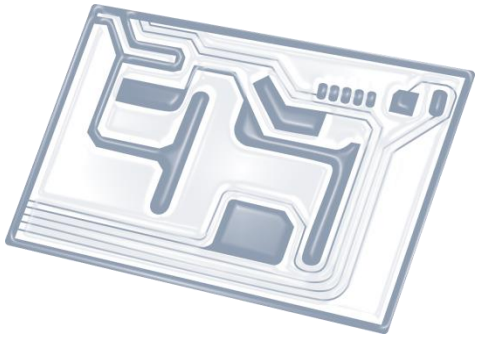
Abstract Data Structures

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

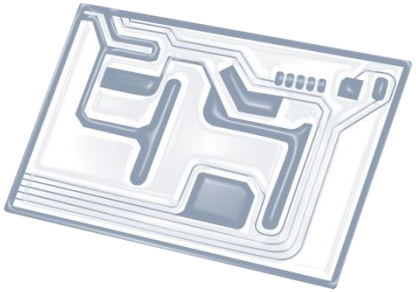
4: Computational thinking

5: Abstract data structures

6: Resource management

7: Control

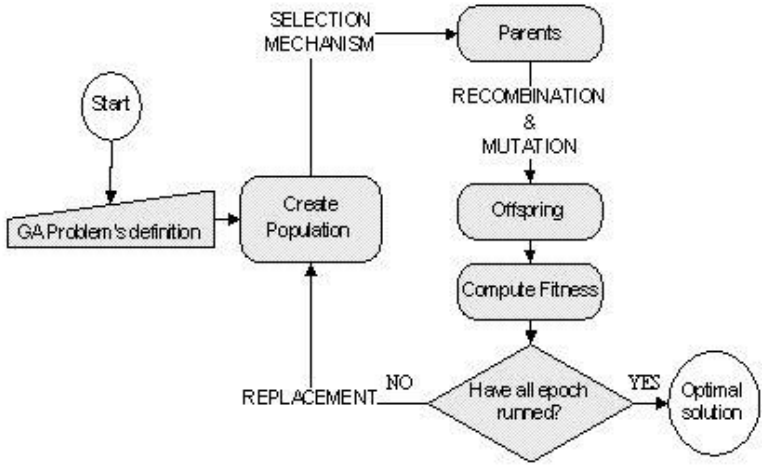
D: OOP



Topic 5.1.9

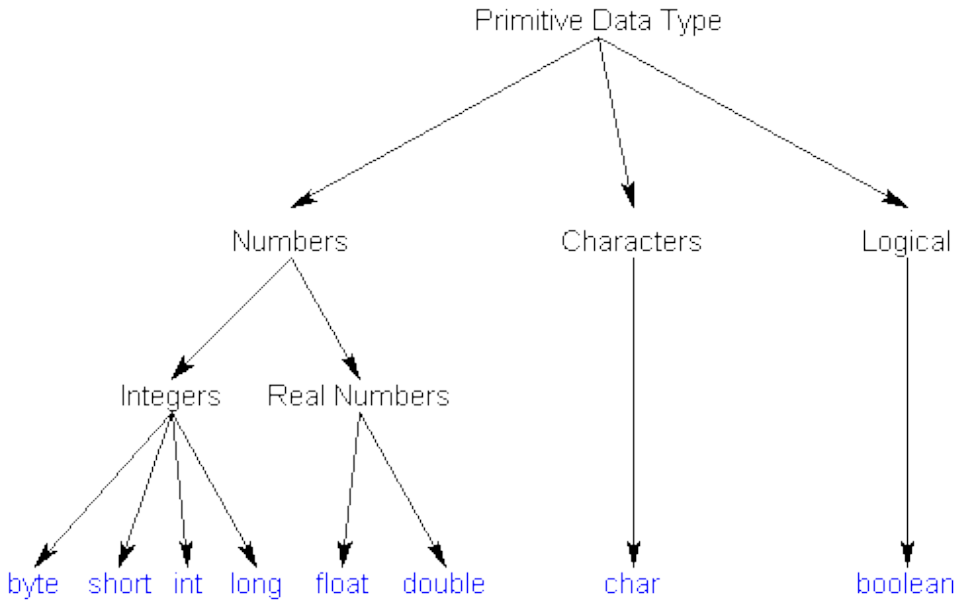
Construct **algorithms** using the access methods of a queue

Input: Instance $x \in I$ of Π_{opt}
set algorithm parameters ()
$i \leftarrow 0$
$Pop_0 \leftarrow$ Initial population ()
Evaluate_fitness (Pop_0)
while not termination condition **do**
$i \leftarrow i+1$
Selection Pop_i from Pop_{i-1}
Crossover (Pop_i)
Mutation (Pop_i)
Fitness (Pop_i)
Replacement_procedure
end while
$S_{best} \leftarrow$ optimal solution Pop_i
Output: S_{best} "candidate" to be the best found solution $x \in I$

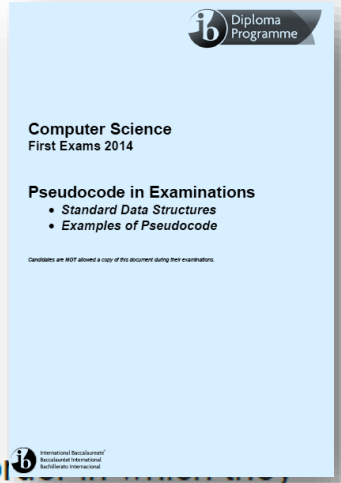


Abstract Data Structures (ADTs)

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



3 Queue Methods



Queues

A queue stores a set of elements in a particular order: Items are retrieved in the order in which they are inserted (First-in, First-out). The elements may be of any type (numbers, objects, arrays, Strings, etc.).

Method name	Brief description	Example: WAIT, a queue of Strings	Comment
`enqueue()`	Put an item into the end of the queue	`WAIT.enqueue("Mary")`	Adds an element that contains the argument, whether it is a value, String, object, etc. to the end of the queue.
`dequeue()`	Remove an item from front of the queue	`CLIENT = WAIT.dequeue()`	Removes and returns the item at the front of the queue.
`isEmpty()`	Test: queue contains no elements	`if WAIT.isEmpty() then`	Returns TRUE if the queue does not contain any elements.

Example 1: Move from array to queue

Write an algorithm that will move all the elements from a linear integer array LINE to a queue called Q.

```
int COUNTER = 0
loop COUNTER from 0 to LINE.length
    Q.enqueue(LINE[COUNTER])
end loop
```

Example 2: Print values from a queue

Write an algorithm that will print all the String values kept in a queue called Q.

```
loop while not Q.isEmpty()
    output( Q.dequeue() )
end loop
```