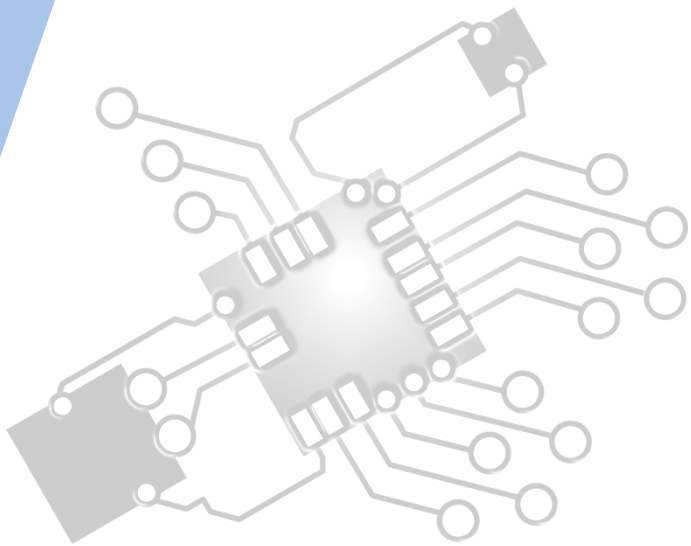
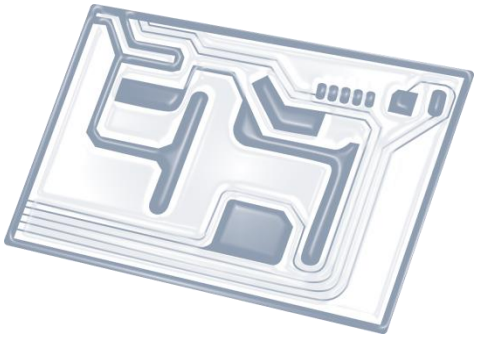
Planning & system installation

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

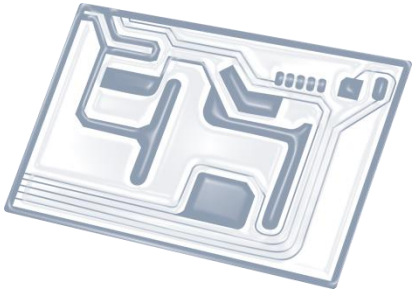
6: Resource management

7: Control

D: OOP

What type of questions to expect:

AO	Typical command terms used in questions
AO1	Classify, define, draw, label, list, state
AO2	Annotate, apply, calculate, describe, design, distinguish, estimate, identify, outline, present, trace
AO3	Analyse, comment, compare, compare and contrast, construct, contrast, deduce, demonstrate, derive, determine, discuss, evaluate, examine, explain, formulate, interpret, investigate, justify, predict, sketch, suggest, to what extent



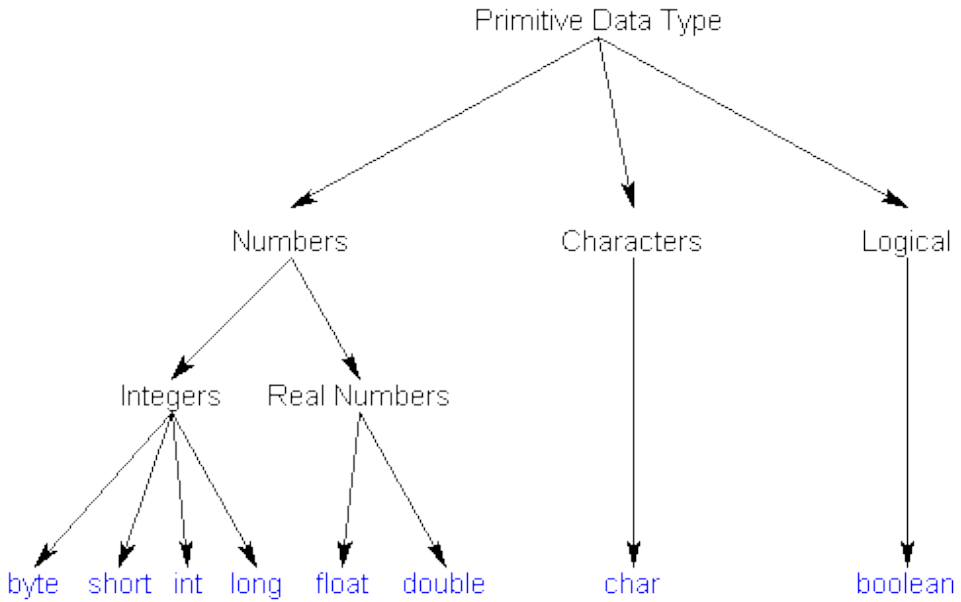
Topic 5.1.20

Suggest a **suitable structure** for a given situation

AO3

Abstract Data Structures (ADTs)

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



Stacks? Best used for:

- Perhaps the most important application of stacks is to implement **function calls (methods)**.
- Most compilers implement function calls by using a stack.
- This also provides a technique for eliminating recursion from a program: instead of calling a function recursively, the programmer uses a stack to simulate the function calls in the same way that the compiler would have done so.
- Conversely, we can often use **recursion** instead of using an explicit stack.

Queues? Best used for:

- **Computing applications**: serving requests of a **single shared resource** (printer, disk, CPU),
- **Buffers**: MP3 players and portable CD players, iPod playlist.
- **Playlist** for jukebox: add songs to the end, play from the front of the list.
- **Interrupt handling**: When programming a real-time system that can be interrupted (e.g., by a mouse click or wireless connection), it is necessary to attend to the interrupts immediately, before proceeding with the current activity. If the interrupts should be handles in the same order they arrive, then a FIFO queue is the appropriate data structure.

Linked list? Best when:

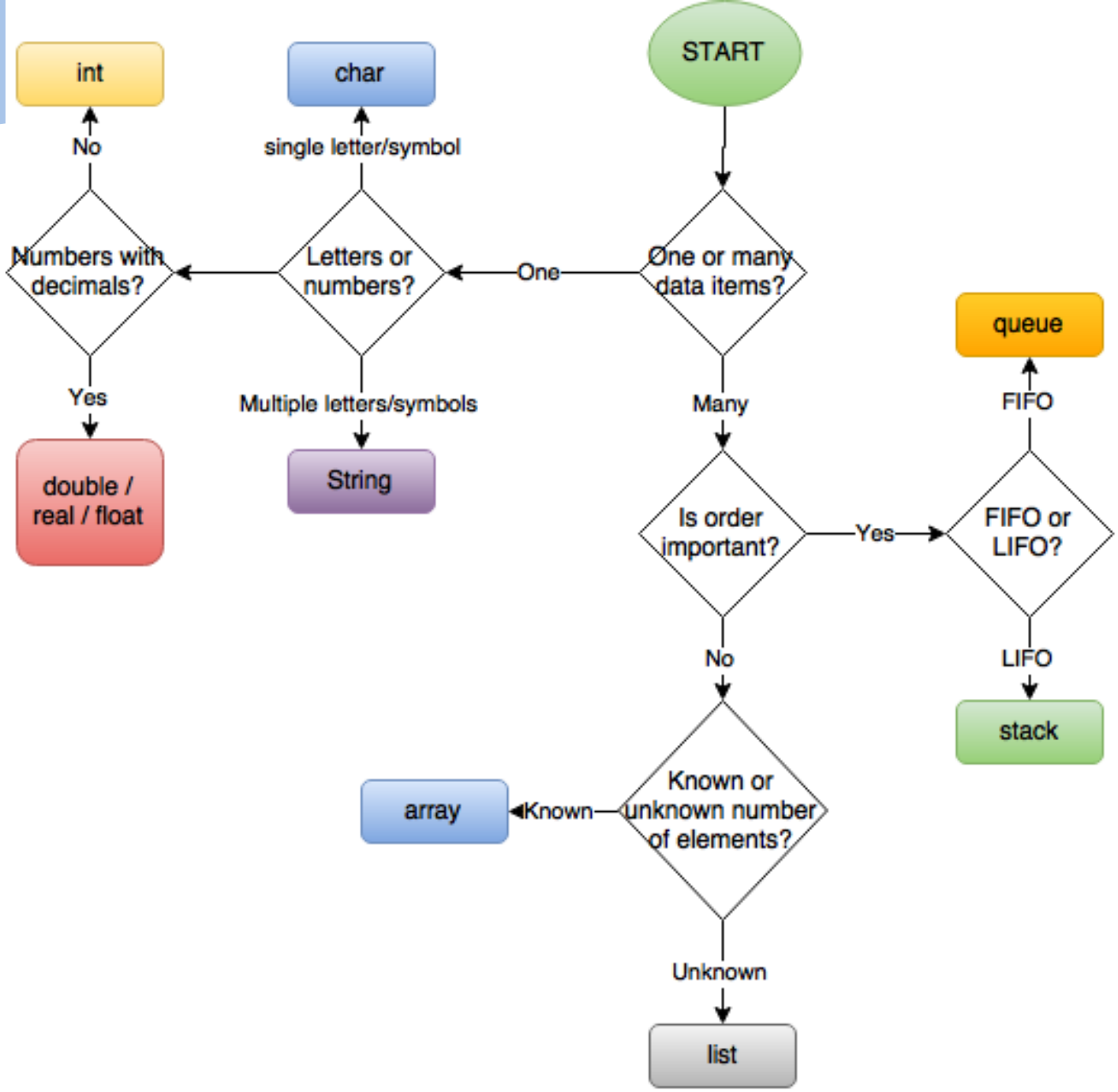
- You need constant-time insertions/deletions from the list (such as in real-time computing where time predictability is absolutely critical)
- You don't know how many items will be in the list. With arrays, you may need to re-declare and copy memory if the array grows too big
- You don't need random access to any elements
- You want to be able to insert items in the middle of the list (such as a priority queue)

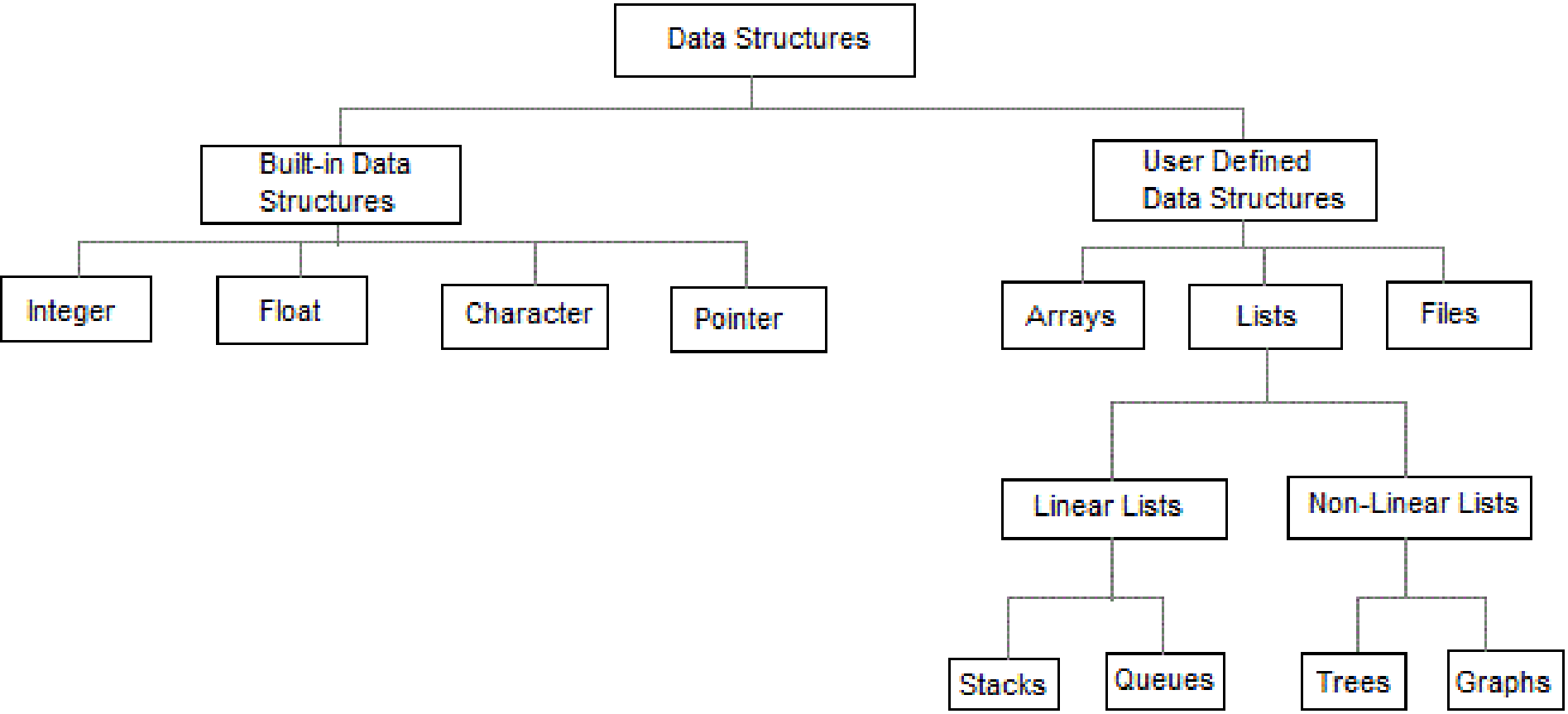
Arrays? Best when:

- You need indexed/random access to elements
- You know the number of elements in the array ahead of time so that you can allocate the correct amount of memory for the array
- You need speed when iterating through all the elements in sequence.
- **Memory is a concern. Filled arrays take up less memory than linked lists. Each element in the array is just the data. Each linked list node requires the data as well as one (or more) pointers to the other elements in the linked list.**

Binary trees? Best for:

- **Binary Search Tree** - Used in many search applications where data is constantly entering/leaving, such as the map and set objects in many languages' libraries.
- **Heaps** - Used in implementing efficient priority-queues, which in turn are used for scheduling processes in many operating systems, Quality-of-Service in routers, and A* (path-finding algorithm used in AI applications, including robotics and video games). Also used in heap-sort.
- **GGM Trees** - Used in cryptographic applications to generate a tree of pseudo-random numbers.
- **Syntax Tree** - Constructed by compilers and (implicitly) calculators to parse expressions.


START
One or many data items?
One
Many
Letters or numbers?
single letter/symbol
char
Multiple letters/symbols
String
Numbers with decimals?
No
int
Yes
double / real / float
Is order important?
Yes
FIFO or LIFO?
FIFO
queue
LIFO
stack
No
Known or unknown number of elements?
Known
array
Unknown
list


Data Structures
Built-in Data Structures
User Defined Data Structures
Integer
Float
Character
Pointer
Arrays
Lists
Files
Linear Lists
Non-Linear Lists
Stacks
Queues
Trees
Graphs