A-LEVEL COMPUTER SCIENCE • EXERCISE SHEET

10.4 Introduction to Abstract Data Types (ADT)

Name: ____________ Class: ________ Date: __________ **Total: 19 marks**

KEY WORDS linked list 链表 • queue 队列 • stack 栈 • pointer 指针 • node 节点

Objective

Build the skills to answer exam questions on **Abstract Data Types 抽象数据类型** (ADTs) — the stack, queue and linked list, and how they are implemented with arrays.

You must be able to:

- explain that an **ADT** is data **plus** the operations on it
- use a **stack** (LIFO), a **queue** (FIFO) and a **linked list**
- describe how each is implemented using an **array**

1 Worked examples

Study these first. Each one shows the method for a question type used later — follow the steps and you can do the Practice and Exam-style questions yourself.

What is an ADT

An **ADT** is a **collection of data** together with a **set of operations** on it. You use it through its operations only — **how** it is stored is hidden, so the implementation can change without breaking your code.

Stack — LIFO (Last In, First Out)

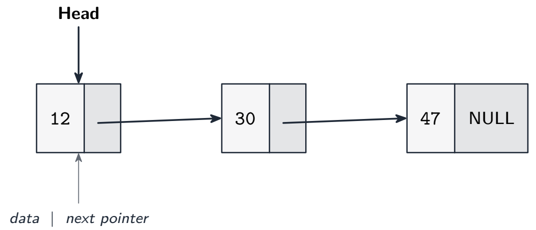
- **push** — add to the **top**; **pop** — remove from the **top**.
- Stored in `Stack[1:MaxSize]` with an integer `Top` (0 = empty).
- **Push**: full when `Top = MaxSize` (**overflow**); else `Top ← Top + 1`, `Stack[Top] ← x`.
- **Pop**: empty when `Top = 0` (**underflow**); else return `Stack[Top]`, `Top ← Top - 1`.

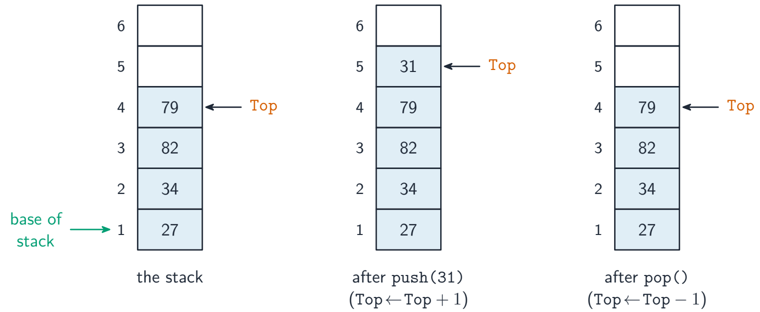
Queue — FIFO (First In, First Out)

- **enqueue** — add to the **rear**; **dequeue** — remove from the **front**.
- Stored in `Queue[1:MaxSize]` with a `Front` pointer, a `Rear` pointer and a `Count`.
- **Enqueue**: full when `Count = MaxSize`; else `Rear ← (Rear MOD MaxSize) + 1`, `Queue[Rear] ← x`, `Count ← Count + 1`.
- **Dequeue**: empty when `Count = 0`; else return `Queue[Front]`, `Front ← (Front MOD MaxSize) + 1`, `Count ← Count - 1`.
- The `MOD` makes it a **circular array**, so the pointers wrap round instead of running off the end.

Linked list

Each **node** holds a value and a **pointer** to the next node. `Head` marks the first node; the last node's pointer is `NULL`.





A stack is another abstract data type (push/pop)

Traverse — follow the pointers from `Head` to `NULL`:

```
Current ← Head
WHILE Current <> NULL DO
    OUTPUT Current.Value
    Current ← Current.Next
ENDWHILE
```

Advantage over an array: cheap insert/delete (adjust pointers). Disadvantage: slow random access (you must follow pointers from `Head`).

2 Practice

Now apply the methods above.

2.1 State what an **Abstract Data Type** is. [1]

2.2 State the order rule of a **stack** and of a **queue** (use the terms LIFO / FIFO). [2]

2.3 Name the stack operation that **adds** an item and the one that **removes** an item. [2]

2.4 In a linked list, state what the **Head** pointer identifies and what the **last node's** pointer holds. [2]

3 Exam-style questions

3.1 A stack is implemented as `Stack[1:10]` with an integer `Top`.

(i) Write pseudocode for `Push(NewItem)` that checks for **overflow**. [4]

(ii) State what **underflow** means. [1]

3.2 The items `A`, `B`, `C`, `D` are pushed onto an empty stack in that order, then **two** items are popped. State the two items removed (**in order**) and which item is now on top. [3]

3.3 A linked list stores nodes that each have a `Value` and a `Next` pointer.

(i) Describe how to **traverse** the list and output every value. [3]

(ii) State **one** advantage of a linked list over an array. [1]