Solutions

Each answer shows the steps, then the **result**.

2.1

- a **collection of data together with the operations** that can be performed on it (implementation hidden)

2.2

- stack = **LIFO** (Last In, First Out)
- queue = **FIFO** (First In, First Out)

2.3

- add = **push**
- remove = **pop**

2.4

- `Head` identifies the **first node** in the list
- the last node's pointer holds **NULL** (a null/sentinel pointer = end of list)

2.5

- `Top` becomes 7, then 8
- the third push finds `Top = 8 = MaxSize`: the stack is full, so the push is refused (**overflow**)

2.6

- `Rear` moves to 5, then **wraps round to 1**
- `Count` is **5**

2.7

- stack, any one: storing **return addresses** when procedures are called; **undo** in an editor; the back button in a browser; evaluating expressions
- queue, any one: a **print queue**; a keyboard buffer; jobs waiting for the processor

3.1

(i) test full, else increment `Top` and store

```
IF Top = 10 THEN
    OUTPUT "Stack full (overflow)"
ELSE
    Top ← Top + 1
    Stack[Top] ← NewItem
ENDIF
```

(ii) trying to **pop from an empty stack** (`Top = 0`) – there is nothing to remove

3.2

- popped: **D**, then **C** (LIFO order)
- now on top: **B**

3.3

(i) start at `Head`

- repeatedly output the current node's value and move to the node its `Next` pointer gives
- stop when the pointer is `NULL`

(ii) any one: items can be **inserted/deleted** without shifting others (just change pointers); the list can grow/shrink at run time

3.4

```
FUNCTION Pop() RETURNS STRING
    DECLARE Item : STRING
    IF Top = 0 THEN
        RETURN ""
    ELSE
        Item ← Stack[Top]
        Top ← Top - 1
        RETURN Item
    ENDIF
ENDFUNCTION
```

- test for an empty stack, `Top = 0`; empty string returned when empty; item read from `Stack[Top]`; `Top` decreased by 1 and the item returned

3.5

- start at **Head** and follow the pointers, comparing `"Bea"` with each value, to find the node it goes **after** (`"Ann"`)
- copy Ann's pointer into Bea's pointer, so **Bea points to Cal**
- change **Ann's pointer** to point to Bea
- the order of the last two steps matters: setting Ann's pointer first would lose the rest of the list; no data is moved

3.6

(a) **"Cat"** was removed: it was added first

- `Front` = **2**, `Rear` = **5**
- `Count` = **4**

(b)

```
PROCEDURE Enqueue(NewItem : STRING)
    IF Count = 6 THEN
        OUTPUT "Queue full"
    ELSE
        Rear ← (Rear MOD 6) + 1
        Queue[Rear] ← NewItem
        Count ← Count + 1
    ENDIF
ENDPROCEDURE
```

- header with a `STRING` parameter; test for a full queue, `Count = 6`; message output and nothing added when full; `Rear` moved on and wrapped from 6 to 1; item stored at `Queue[Rear]`; `Count` increased by 1

(c) when a pointer passes the last element, 6, it **wraps round** to element 1

- so spaces freed at the front of the array are used again, and the queue never runs out of room while `Count` is below 6