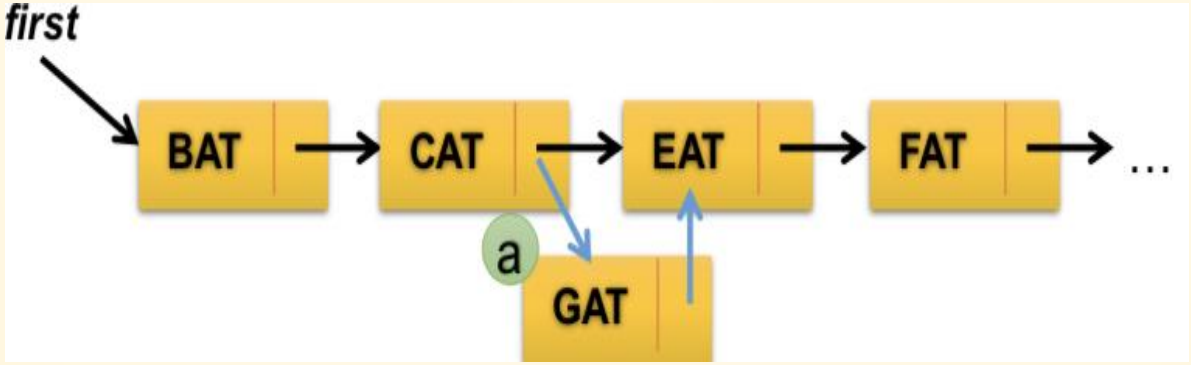
Ch.4 Linked List 參考答案

Question 1

Write a pseudo code to insert an element with value “GAT” into between element “CAT” and “EAT” in a linked list as illustrated below. (Your code should manipulate pointers to achieve the goal.)



For some more practices: https://leetcode.com/explore/learn/card/linked-list/209/singly-linked-list/1290/

For Q1~Q4:

We should have something like this:

```
class Node{
        string data;
        Node *next;

        Node(){data=empty string, next=null}
        Node(string a){data=empty string, next=null}

        friend class LinkedList;
};
```

```
class LinkedList{
        Node *first;

        //All other functions
};
```

(Question 1) Ans:

Node pointer: cur // cur stores the current node.

Find ”CAT” node and store it in cur. (if we can’t find “CAT”, return error)

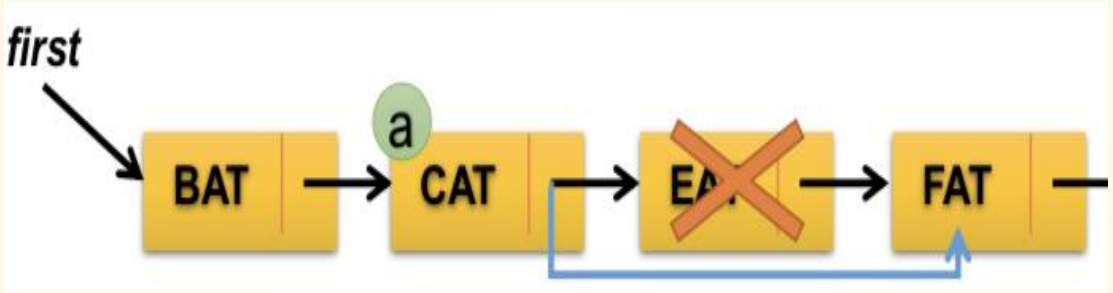
New a node with value = “GAT”

“GAT”->next = “CAT”->next //”CAT”->next is “EAT”

“CAT”->next = “GAT”

Question 2

Write a pseudo code to delete the element with value “EAT” in a linked list as illustrated below. (Your code should manipulate pointers to achieve the goal.)



(Question 2) Ans:

Node pointers: cur, prev //prev stores the previous node of cur

Find “EAT” and store it in cur. (if we can’t find “EAT”, return error)

In the searching above, store the previous node of cur in pointer prev.

if(“EAT” is first) // “EAT” is exactly the head of the list

    first = “EAT”->next

    delete “EAT”

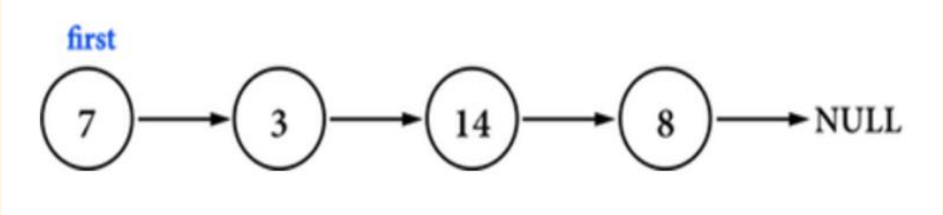
else // “EAT” is not the head of the list

    prev->next = cur->next //prev is “CAT”, and points it to cur->next which is“FAT”

    delete “EAT”

Question 3

Write a pseudo code to clean(清除) the whole linked list as illustrated below. (Your code should manipulate pointers to achieve the goal.)

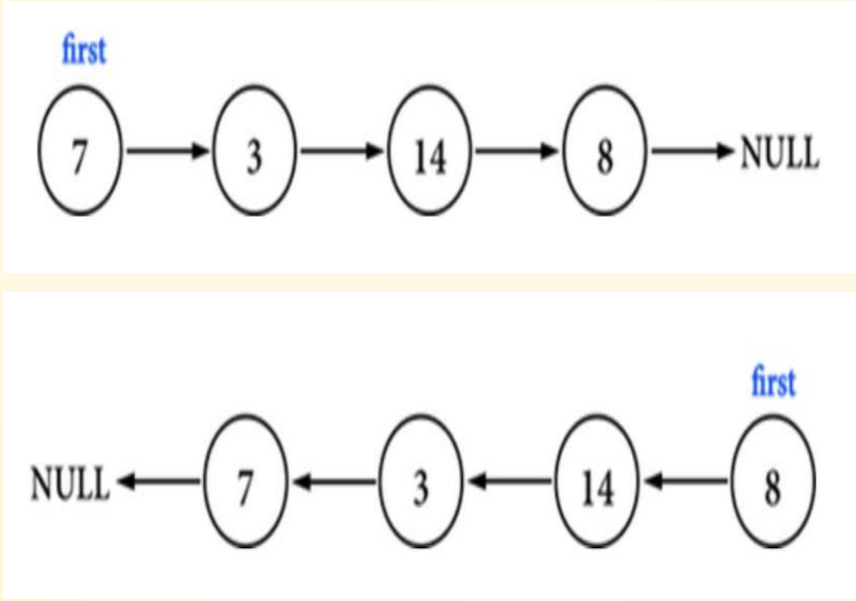


(Question 3) Ans:

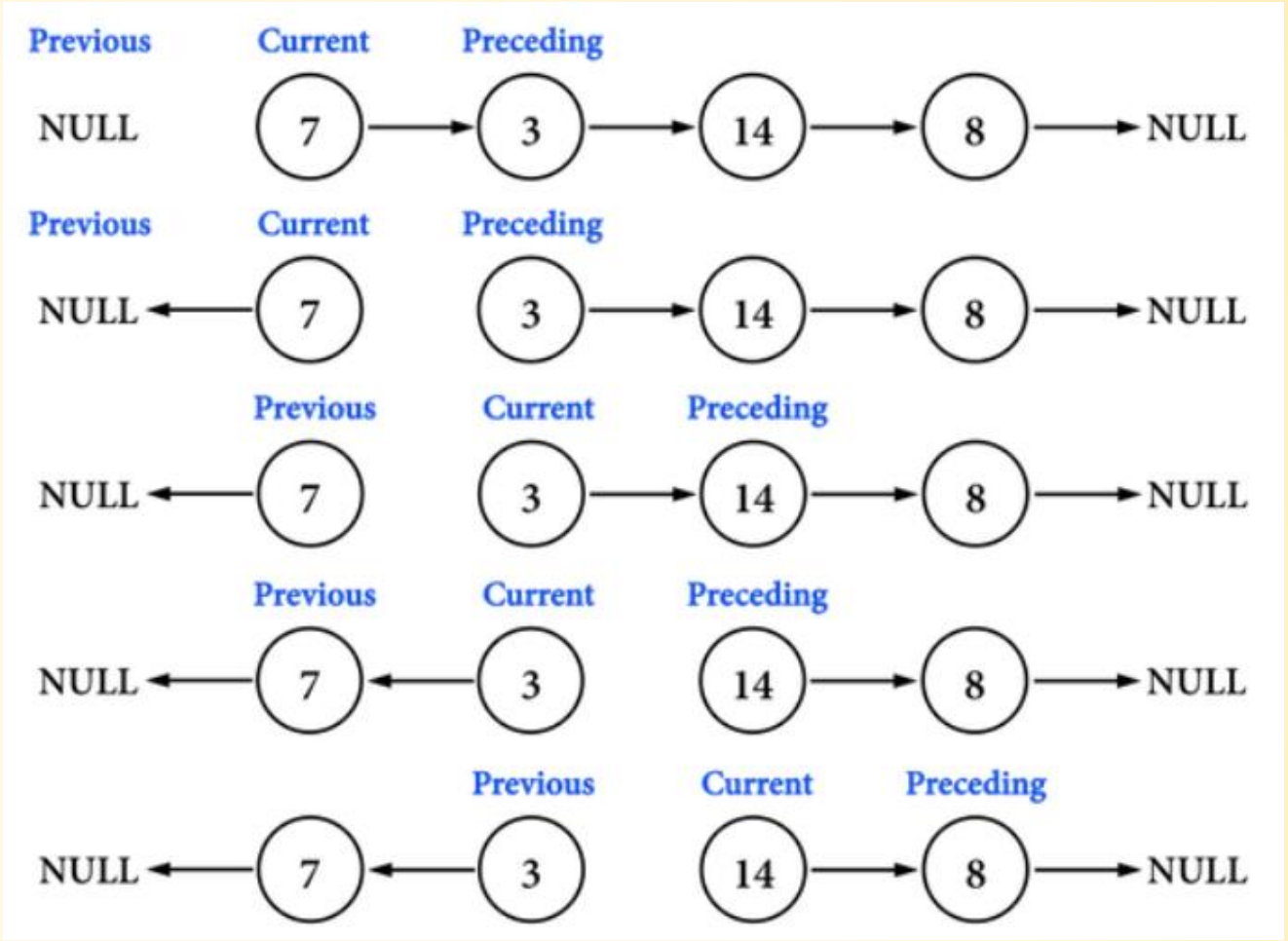
```
Node pointer: cur          //To travel the whole list
While(first != null){
        cur = first          // store a node into cur so we can delete it later
        first = first->next // the node, which cur points to, is now removed from List
        delete cur
}
```

Question 4

Write a pseudo code to reverse the linked list A into B as illustrated below. (Your code should manipulate pointers to achieve the goal.)



(Question 4) Ans:



Can we use only two pointers?

- https://stackoverflow.com/questions/1801549/how-to-reverse-a-singly-linked-list-using-only-two-pointers
- https://www.geeksforgeeks.org/iteratively-reverse-a-linked-list-using-only-2-pointers/

(Question 4) Ans:

```
Node pointers: cur, prev, prec                  // prec stores the preceding nodes of cur.
```

(cur is the node being reversed; prev is the head of the new reversed list; prec is the head of the remaining list.)

```
If(first or first->next is null) return         // List contains only 0 or 1 node
prev=null, cur=first, prec=first->next          //Start to scan the old list
While(prec != null){
        cur->next = prev                        // reverse cur
        prev = cur                              // point the head of the reversed list to the newly inserted cur
        cur = prec                              // Switch cur to the next target in the old list
        prec = prec->next                       // Switch prec to the next node of the new target
}
cur->next = prev                                // Reverse the last node in the old list
fisrt = cur                                     // Assign the head of the list to the last node
```

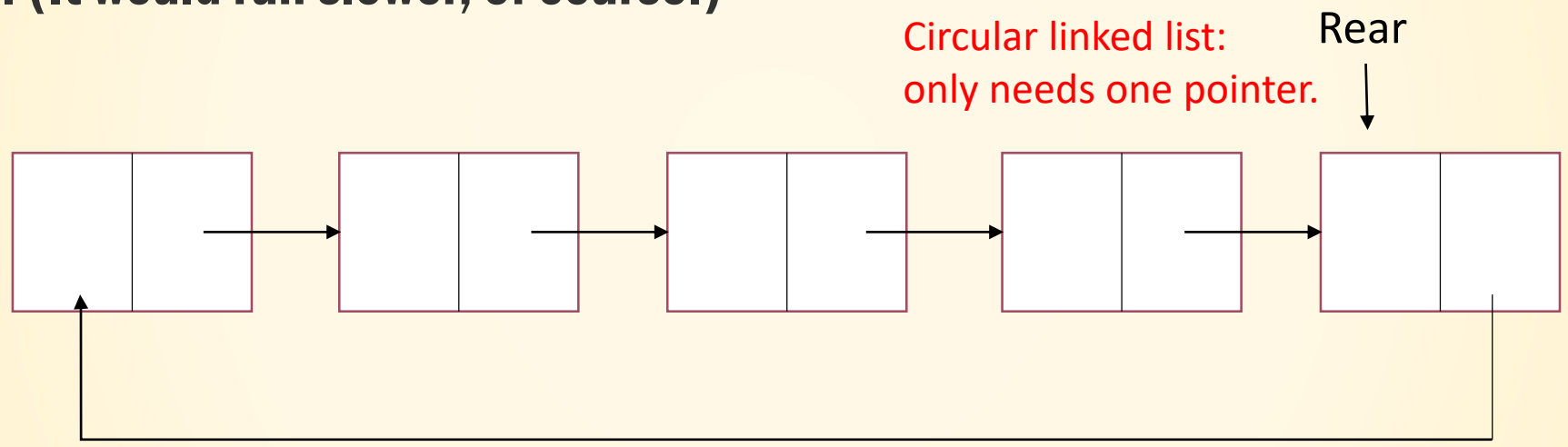
Question 5

Can one use a linked list to implement a circular queue? If yes, please describe how to do it.

(Question 5) Ans:

Front(): Get the first item from queue.
Rear(): Get the last item from queue.
enqueue(value): insert value into the queue at Rear.
dequeue(): delete from the queue at Front.

Implementing a circular queue with linked lists is more convenient than with array. (It would run slower, of course.)

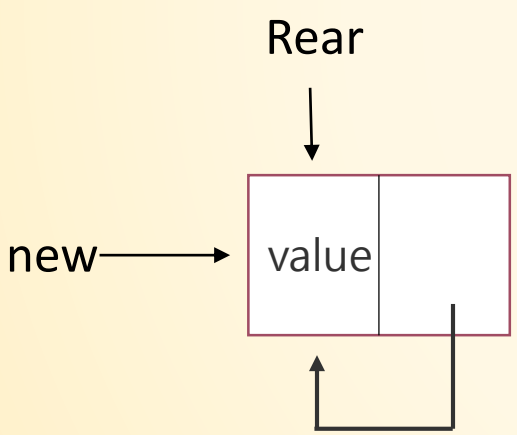


Using linked list to implement a “circular” queue is a tricked question. The origin of a circular queue is to use an “circular “ array to implement the queue to make queues more memory friendly.

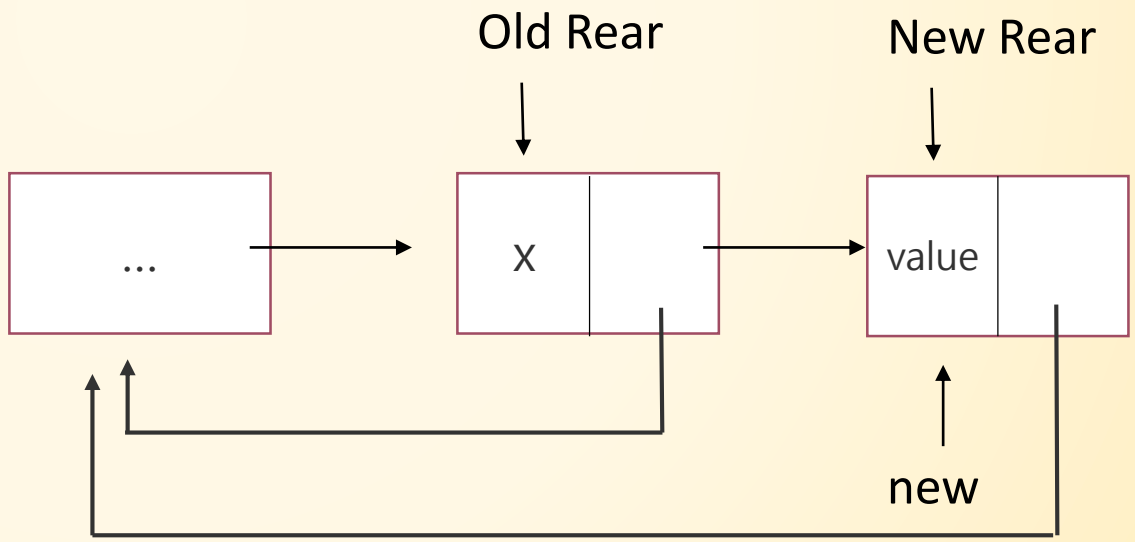
(Question 5) enqueue(value)

New a node whose data is value.

If(queue is empty)



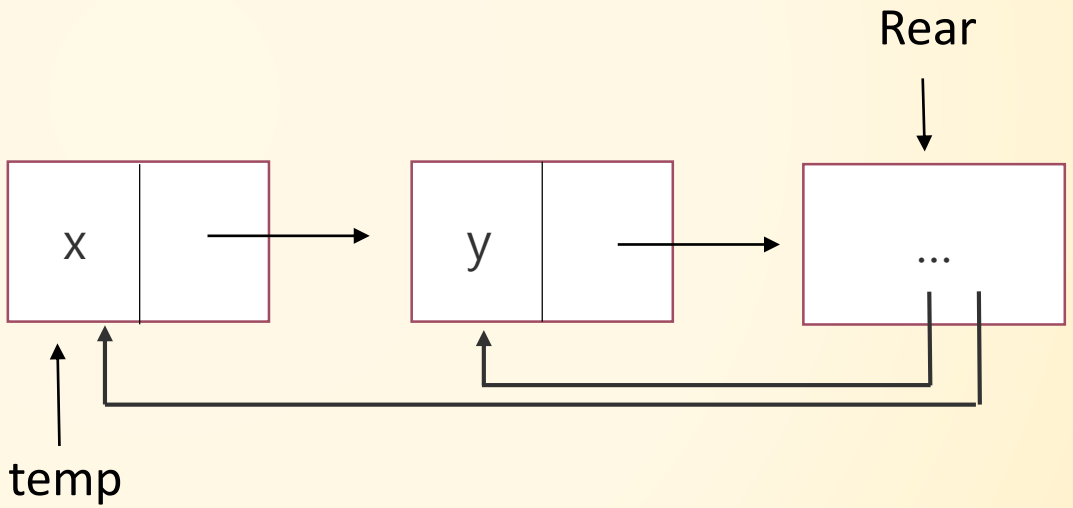
If(queue is not empty)



(Question 5) dequeue()

If(queue is empty)

If(queue is not empty)



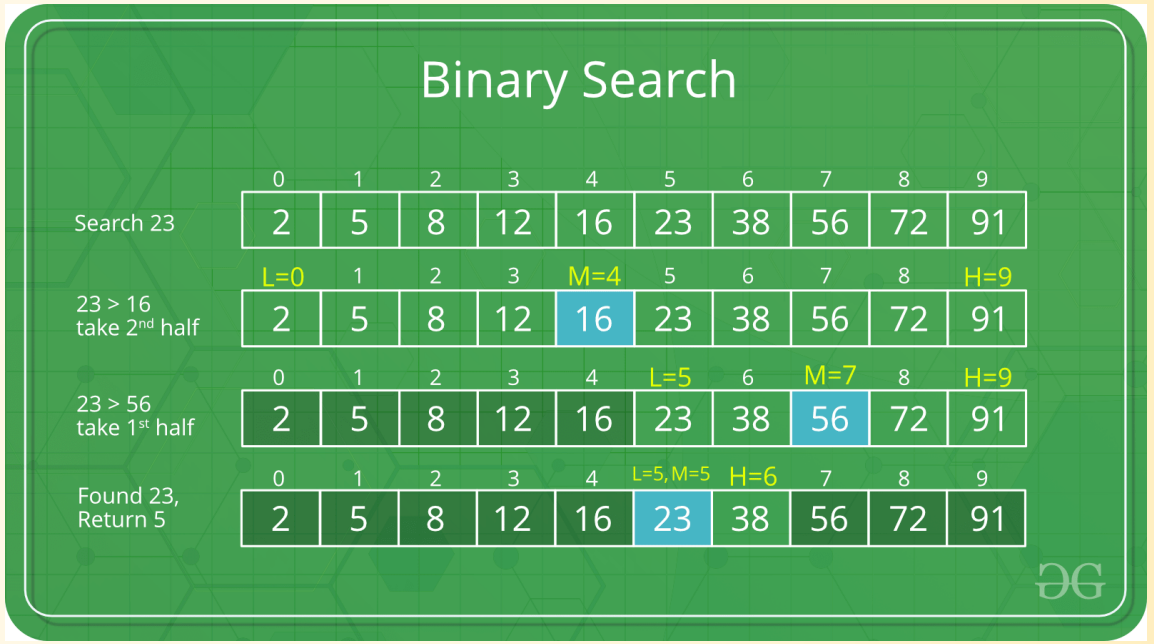
(Question 5) front() or rear()

rear(): return value of the rear pointer
front(): return value of the rear->next pointer

Question 6

https://www.geeksforgeeks.org/searching-algorithms/

Please analyze the time complexities of the search function for a linked list and an array.

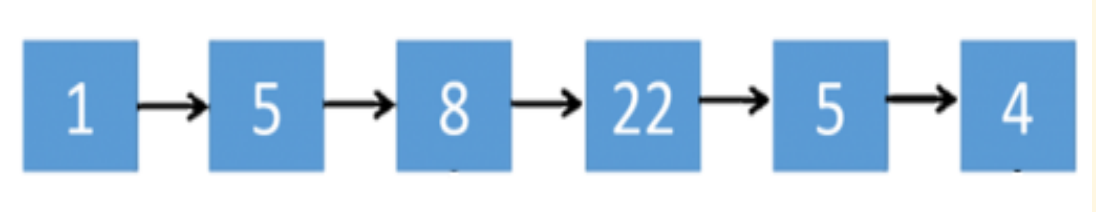


- **Any input data (sequential search)**
  - **Array: O(n), scan one by one**
  - **Linked list: O(n), scan one by one**

- **Sorted data**
  - **Array: O(log n) by binary search,**
    Since we can random access an array.
  - **Linked List: Even you apply binary search, You still need to travel through the list one by one.**
    O(n)

Question 7

Please analyze the time complexity of the “insert” and “delete” function of a linked list. Consider cases when the mentioned functions occur at the head, middle and the end of a linked-list.



(Question 7) Ans:

Suppose we only maintain a "first" pointer for the list.
(Of course, we can maintain more pointers like "last" or even "middle" pointer to accelerate certain operations.)

Head: only take constant manipulation of pointers, so O(1).

Middle: needs to travel half of the list to get to middle + constant manipulations of pointers. So $\frac{n}{2} + O(1) = O(n)$.

End: needs to travel all the way through the list + constant manipulations of pointers. so $n + O(1) = O(n)$.

Question 8

1. Print all the elements into doubly linked list.
2. Push a new element in the last of doubly linked list.

- DoublyLinkedList
  - Node *firstNode;
  - Node *lastNode;
- Node
  - int data;
  - Node *pre;
  - Node *next;

(Question 8) Ans:

1. 因為**Node**的**member**皆為**private**，故在處理**Node**物件時要用程式給的**function**不能用**pointer->next**等直接存取的寫法。

```
void printAll(){
    if(firstNode == 0){ // 若firstNode指向null，表示空list
        cout << "List is empty." << endl;
        return;
    }

    Node *current = firstNode;          // current用來travel整個list
    while (current != 0) {
        cout << current->getData() << " "; // 依序輸出
        current = current->getNext();       // current指向下一個node
    }
    cout << endl;
}
```

(Question 8) Ans:

2. Linked list沒特別要求是否**circular**，且同時有儲存**last**和**first**。故以**non-circular**想法寫。

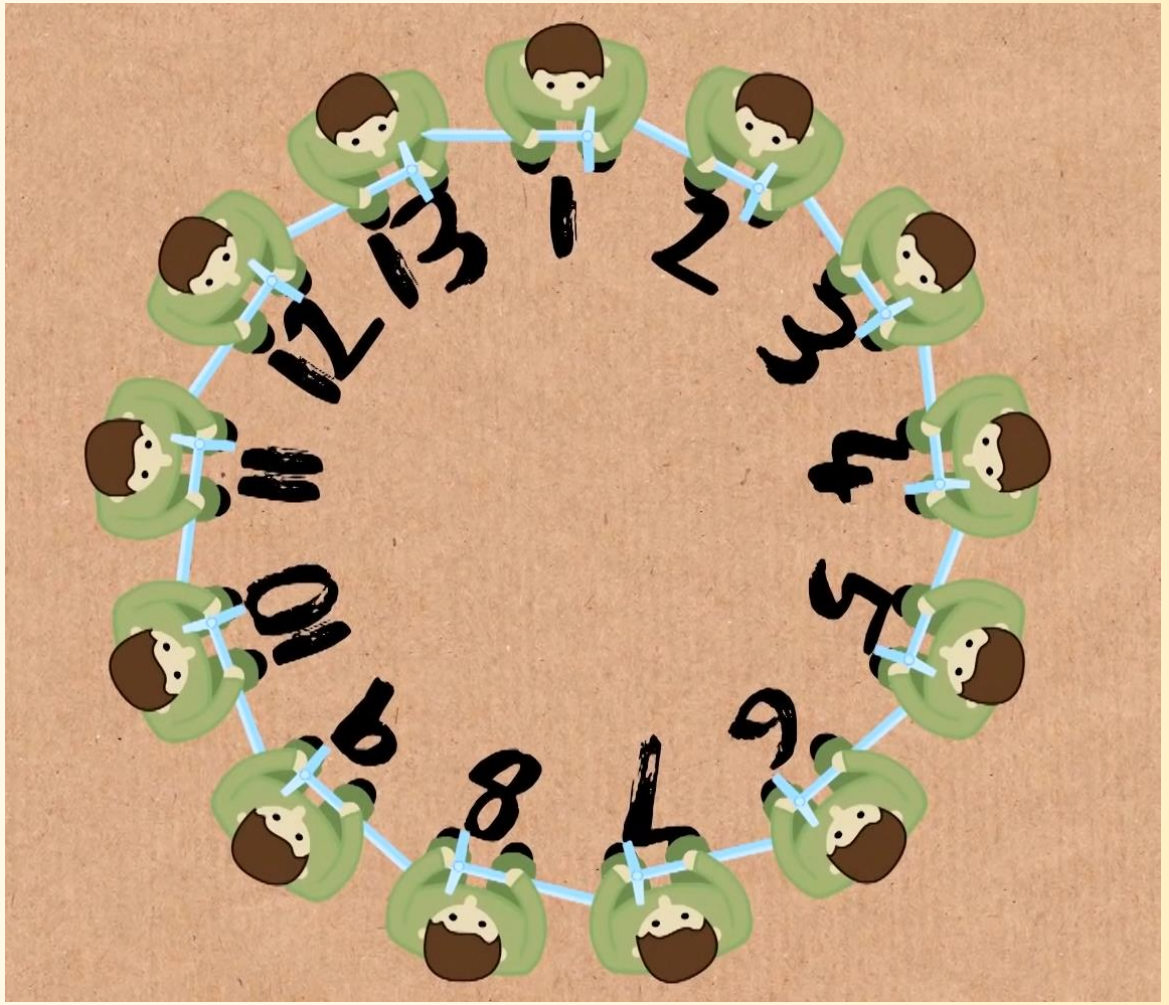
```
void push_back(Node *node){
    if(lastNode == 0){   // list為empty，lastNode和firstNode皆為empty
        lastNode = node;
        firstNode = node; //last和first皆為此新的node
        node->setPre(0);
        node->setNext(0); //此新的node上一個和下一個皆為null

        /*
            有些實作會把first的(或是last)另外做個empty node(不是null，其data不重要)
            所以firstNode指標(lastNode)和真正的第一個node的pre(真正最後一個的next)會指向同一個node(也就是該empty node)
            且codind時就不用特別分要處理first/last和處理非first/last的不同狀況。
        */
    }
    else{ // list不為empty，故lastNode非null
        lastNode->setNext(node);   //原本的lastNode的下一個node要指向新push進去的這個node
        node->setPre(lastNode);    //新push進去的這個node的上一個node要指向原本的lastNode
        node->setNext(0);          //新push進去的這個node會成為新的lastNode，故其下一個node要指向null
        lastNode = node;           //一切處理完後將lastNode指向最後push進去的這個新node
    }
}
```

Question 9

https://www.youtube.com/watch?v=uCsD3ZGzMgE

Use a circular linked list to figure out the last survivor of the game (The Josephus Problem) from the video.



(Question 9) Ans:

Build a circular linked list: L (with N nodes of value 1~N)

Node pointer: cur = first node, temp

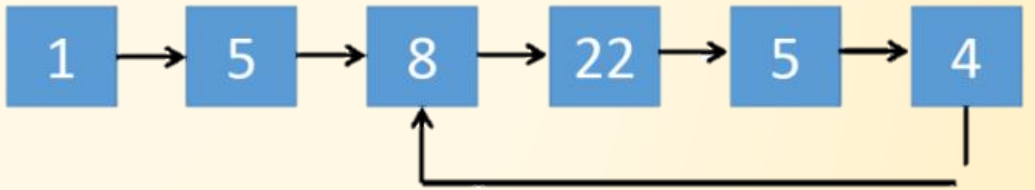
```
while(L has more than one node){
        temp = cur->next;          // 抓出將被executed的人
        cur->next = temp->next;    // 維護L的完整連接性
        delete(temp);              // "execute"被抓出來的人
        cur = cur->next;           // 把刀給下一個活著的人
}
```

Return the last node in L.

Question 10

Given the following Node structure of a singly linked list and an example of a “circular” (if there is any cycle in the list, it is considered circular here) linked-list, please give an efficient pseudo code about how to determine whether a singly linked list is circular or not.

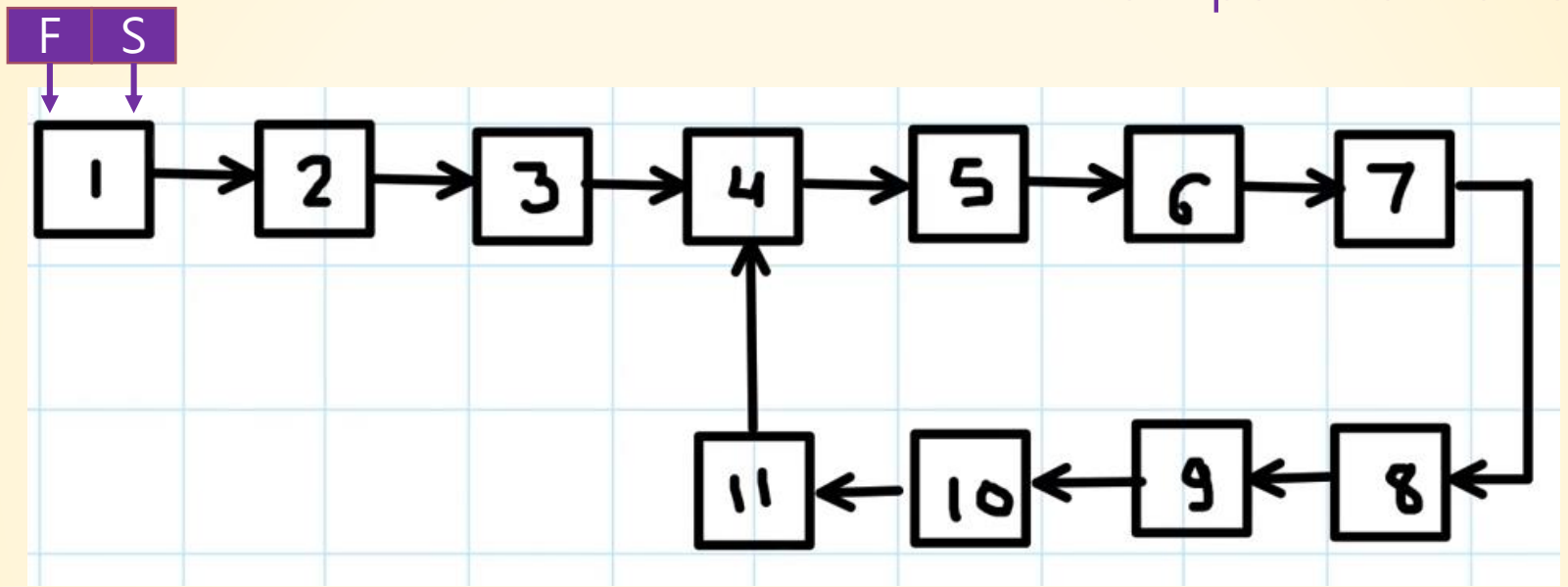
```
public class ListNode {
        ListNode *next;
        int val;
        ListNode(int value) { val = value; }
}
```



https://www.youtube.com/watch?v=Cs3KwAsqqn4

Floyd Cycle detection (for a linked list):

F: move 2 steps at a time
S: move 1 step at a time
Both point to the head of the list.



- Cycle detection?

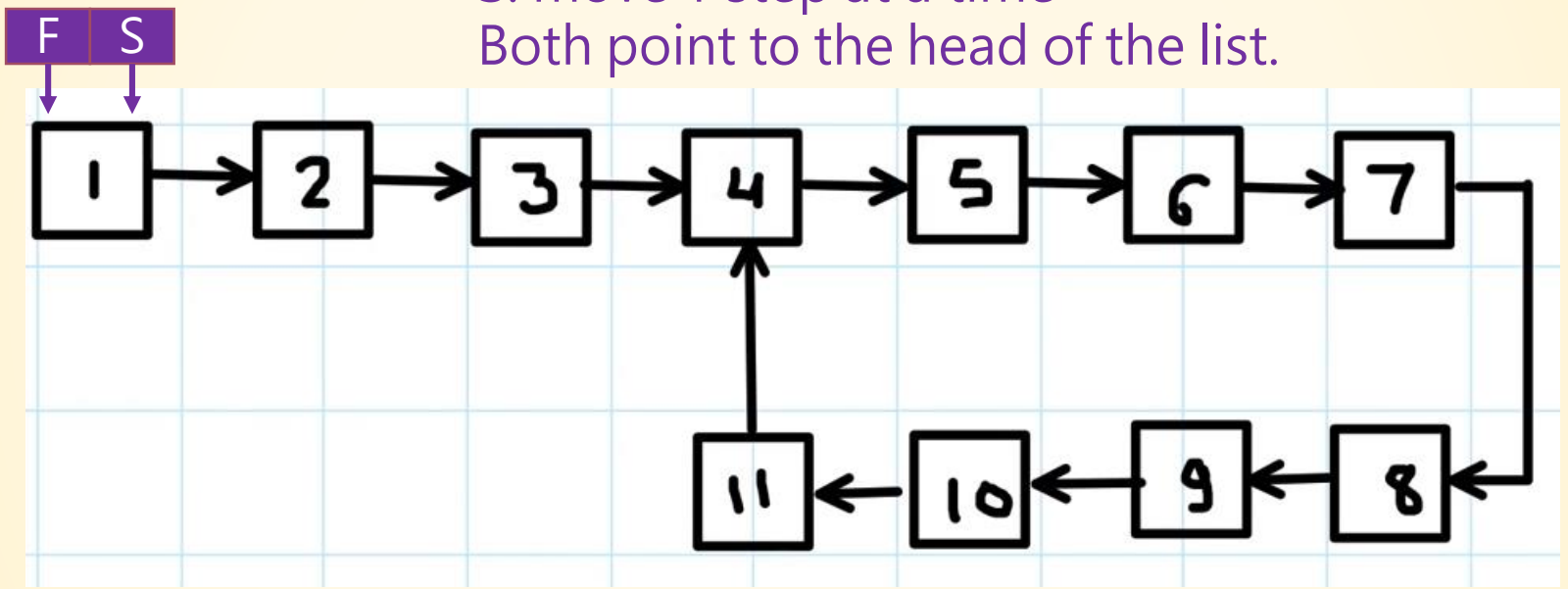
F, S meet at some node **A**→ cycle!
F reaches the end itself → no cycle!

- Where is the starting node of the loop?

1. F starts at node **A**, S starts from starting node of list.
2. Both move 1 step at a time.
3. When they meet at some node B, B is the starting node of the loop.

Floyd Cycle detection (for a linked list):

F: move 2 steps at a time
S: move 1 step at a time
Both point to the head of the list.

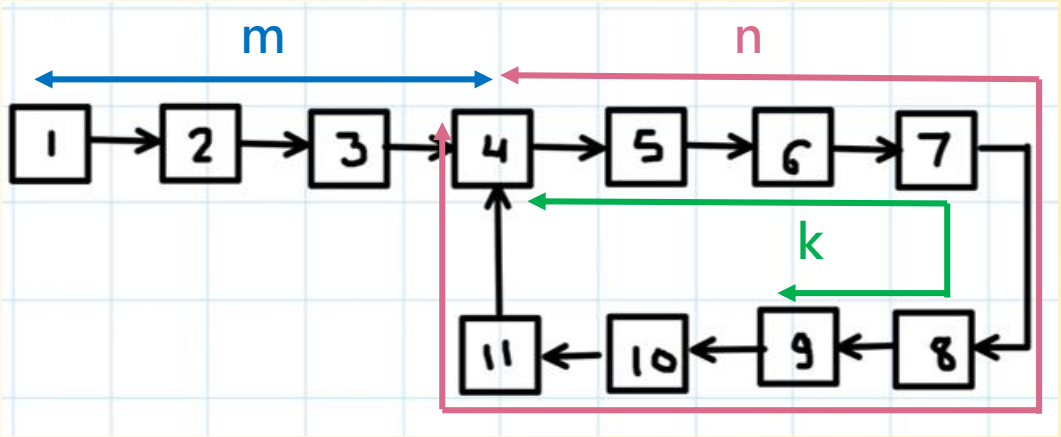


- What is the length of the loop?

1. S starts from A, F stays at A.
2. S move 1 step at a time.
3. When S reaches node A again, the total steps S travel is the length of the loop.

Why does Floyd Algorithm work?

m: Steps from "start of the list" to "start of the loop".
n: Steps in the loop.
k: Steps into the loop where the two pointers meet.

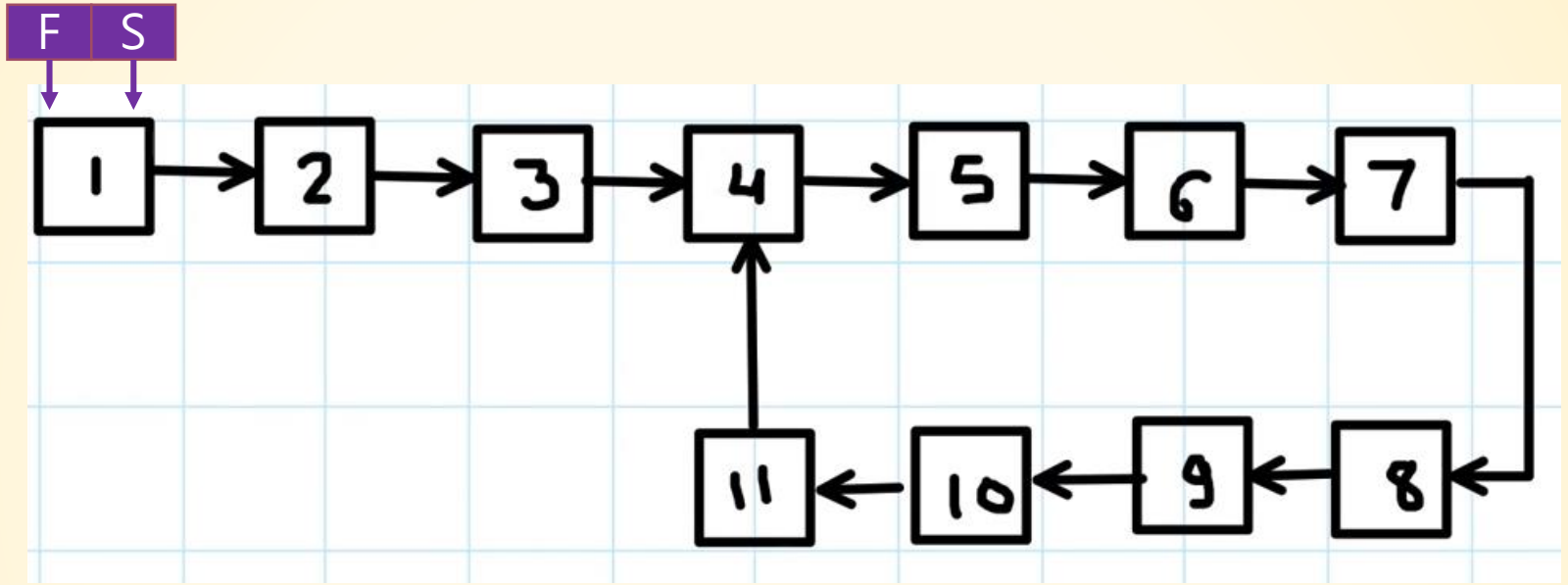


When F and S meets at some node (in the cycle), $2 * (steps\ of\ S) = steps\ of\ F$
Total step of S: $m + k$, Total step of F: $m + cn + k$ for some c.
$2(m + k) = m + cn + k$
$m + k = cn$
For a fixed m and n, we can always find some k and c to satisfy this equation. So this cycle detection works!
Time complexity: O(m+n), Space complexity: O(1) (space for the list excluded)

- What about the explanation for "the starting node of the loop"? $hint$: $m = cn - k$

https://blog.csdn.net/u011221820/article/details/78821464

Brent's Cycle detection (for a linked list)



Idea: 讓S趕快進入loop中，所以會有個"S直接移到F的位置上的"指令。

How to find "length of the loop" right away?
Hint: When they meet, Step of S = m + cn + k, Step of F = (m + cn + k) + n