Linked Lists
鏈結串列

林劭原老師

Linked Lists

- 4.1 singly linked list(又叫做chain)
- 4.4 circular linked list
- 4.10 doubly linked list

Singly Linked Lists(also called Chains)

- 我們之前學的資料結構(array,stack,queue,matrix…)用的是**sequential representation**(循序表示法)，特性是：successive elements of a list are located a fixed distance apart.例如：
- 1. If a_{ij} is stored at location L_{ij}, then $a_{i,j+1}$ is stored at location $L_{i,j+1}$.
- 2. If the i-*th* element in a queue is stored at location L_i, then the $(i+1)$-*th* element is stored at location $(L_i+1)\%n$ for circular representation.
- 3. If the topmost element of a stack is at location L_T, then the element beneath it is at location L_T-1.

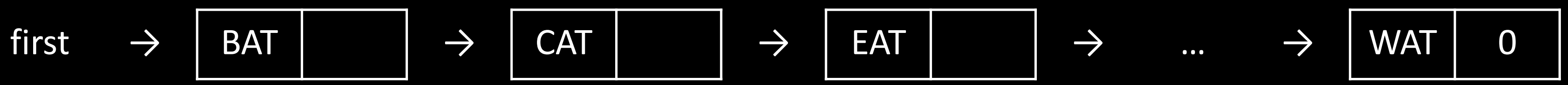
Singly Linked Lists(also called Chains)

- For sequential representation, insertion and deletion of arbitrary elements become expensive(可能要做很多data movements).
- For example：
- BAT,CAT,EAT,FAT,HAT,JAT,LAT,MAT,OAT,PAT,RAT,SAT,VAT,WAT
- Insert GAT：
- Delete LAT：
- **In sequential representation, physical order is the same as logical order.**

Singly Linked Lists(also called Chains)

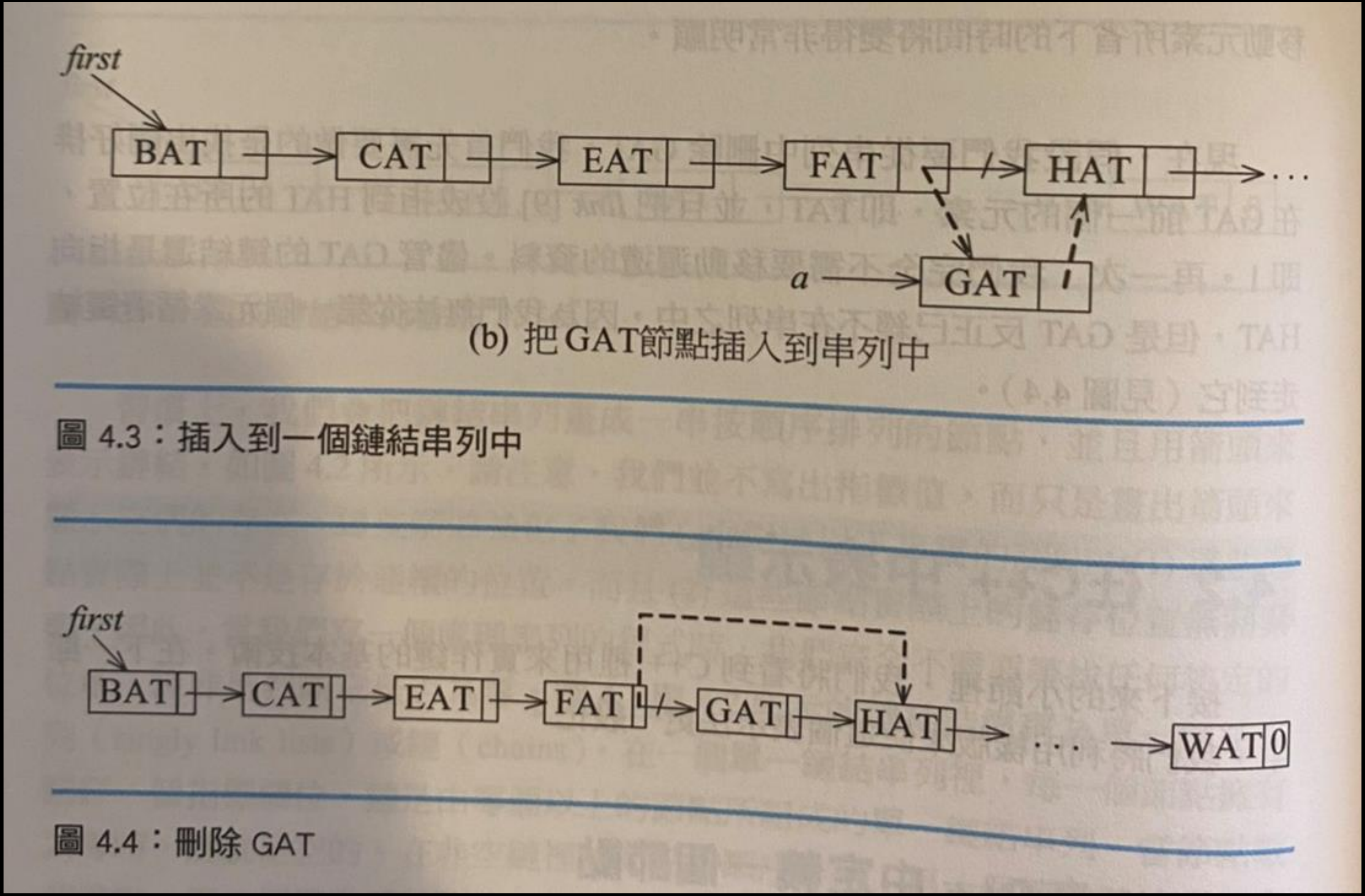
- Chapter 4 要學的資料結構用的是 **linked representation(鏈結表示法)**.
- Linked representation 解決了 sequential representation 怕 insert 和 delete 的問題.
- 它的特性是：successive elements of a list may be placed anywhere in memory.

	data	link
1	HAT	15
2		
3	CAT	4
4	EAT	9
5		
6		
7	WAT	0
8	BAT	3
9	FAT	1
10		
11	VAT	7
	…	…



first → BAT → CAT → EAT → … → WAT 0

Singly Linked Lists(also called Chains)



(b) 把GAT節點插入到串列中

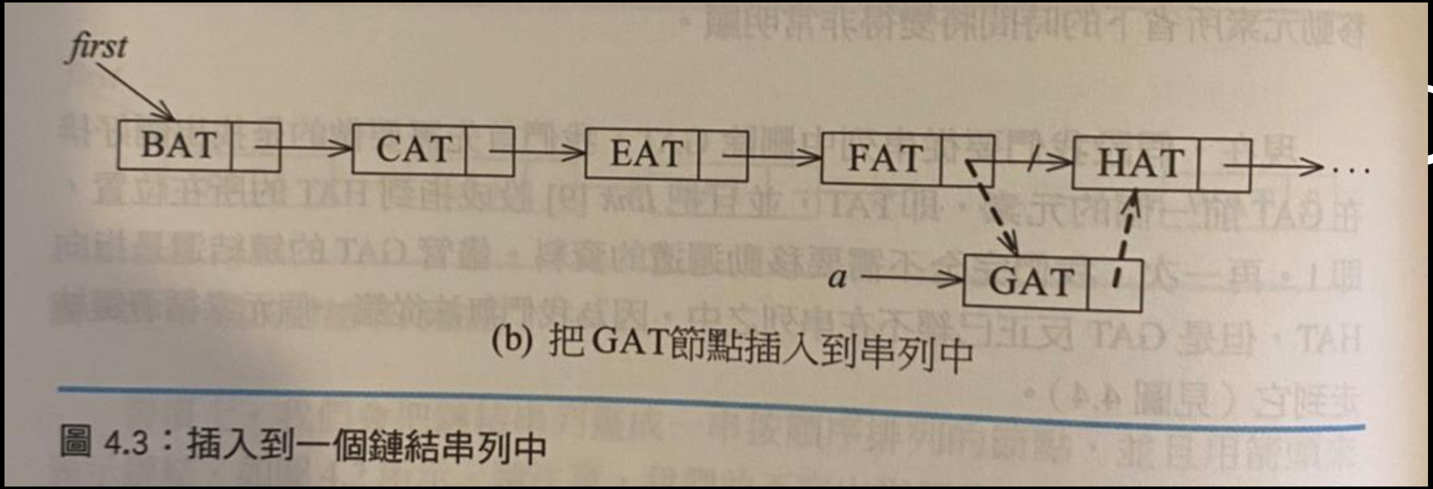
圖 4.3：插入到一個鏈結串列中

圖 4.4：刪除 GAT

Singly Linked Lists(also called Chains)

- In linked representation, physical order may not be the same as logical order.
- Linked representation 也有其缺點:怕 random access, 同時, 存 links 也要額外花 memory.

Singly Chains)



(b) 把GAT節點插入到串列中

圖 4.3：插入到一個鏈結串列中

- To insert the data item GAT between FAT and HAT, the following steps are adequate:
- 1. Get a node (say,a) that is currently unused.
- 2. Set data field of a to GAT.
- 3. Set link field of a to point to the node after FAT, which contains HAT.(先)
- 4. Set link field of the node containing FAT to a.(後)
- 可發現：沒有 data movements

Singly Linked Lists(also called Chains)

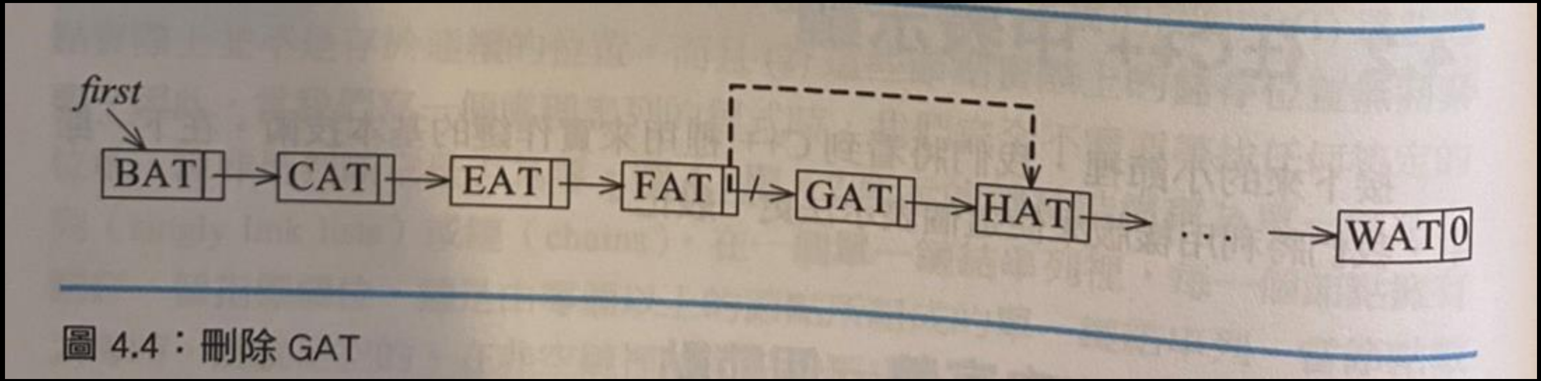


圖 4.4：刪除 GAT

- delete:
- 雖然 GAT 的 node 仍指向 HAT，但 GAT 的 node 卻已不屬於此 lists 中了

- 可再次發現:沒有data movements

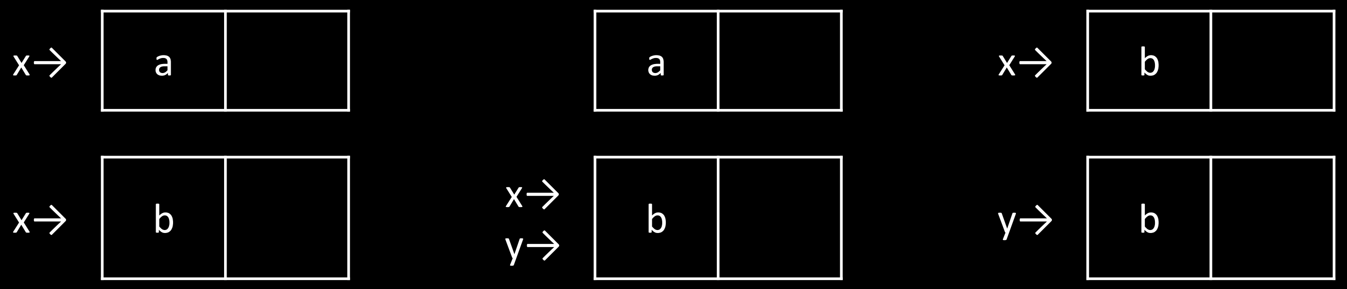
Representing Chains in C++

- class Chain; // forward declaration
- class ChainNode{
  friend class Chain;
  private:
  ```
          char data[3];
          ChainNode *link;
  }
  ```
- Class Chain{
  public:
  // chain operations
  private:
  ```
          ChainNode *first;
  }
  ```

- 由於 ChainNode 形成 Chain ，因此，以 class ChainNode 和 class Chain 來完成。
- 在觀念上，Chain 中有 ChainNode
- 但是我們不知道 Chain 中有多少個 ChainNode?因此實作上，只在 Chain 中存first.
- 在這個解法中，宣告兩個classes。其中 Chain 為 ChainNode 的 friend，因此，Chain 可讀取 ChainNode 的 private data members.

Pointer Manipulation in C++(C++裡的指標處理)

- If x is a pointer variable, then the expression x+1 is valid, but may have no logical meaning.
- If x and y are pointer variables of the same type, then x == y, x != y, x == 0, x != 0 are all valid.
- x = y; 和 *x = *y ; 並不相同:



(a) (b) x = y (c) *x = *y

Chain Manipulation Operations

- 介紹 3 個 chains 的 functions, 目的: create a chain with two nodes, insert a node into the chain, delete a node from the chain. 這裡的做法沒有加上 a header node.

Chain Manipulation Operations

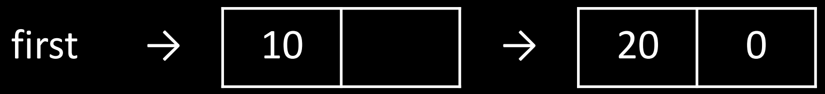
-
  ```
  class Chain;
  ```

-
  ```
  class ChainNode{
  friend class Chain;
  public:
          ChainNode(int element = 0,ChainNode *next = 0){
                  data = element; link = next;
          }
  private:
          int data;
          ChainNode *link;
  };
  ```

-
  ```
  Class Chain{
  public:
          void Create2();  //create a chain with two nodes
          void Insert50(ChainNode *x); //insert 50 到 x 所指的 node 後面
          void Delete(ChainNode *x, ChainNode *y); // delete x 所指的節點，設 x 的 predecessor 是 y
  private:
          ChainNode *first;
  };
  ```

Chain Manipulation Operations

- Create a chain with two nodes as follows



first → 10 → 20 0

- void Chain::Create2(){

```
        ChainNode *second = new ChainNode(20,0);  //2nd node
        first = new ChainNode(10,second);               //1st node
  }
```

Chain Manipulation Operations

- 加入50，有兩種可能:
  (1)原本的chain是空的 (2)原本的chian非空 ，insert 50到x所指的node後
- void Chain::Insert50(ChainNode *x){

```
void Chain::Insert50(ChainNode *x){
        if(first) x->link = new ChainNode(50,x->link); //當first非NULL時，(first)得true
        else first = new ChainNode(50,0);
}
```

Chain Manipulation Operations

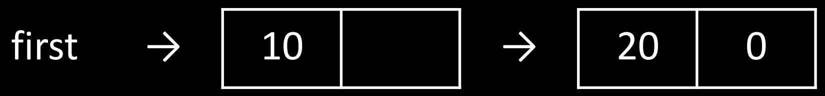
- 去掉 x 所指的節點，有兩種可能:
  (1)x是list的第一個節點，這是first是x (2)x有predecessor(前者)是y
-
  ```
  void Chain::Delete(ChainNode *x, ChainNode *y){
          if(x == first) first = first->link; //當x是list的第一個節點
          else y->link = x -> link;
  }
  ```

Chain Manipulation Operations

- 上面的做法沒有加上a header node, insert和delete都有(1)(2)兩情況要處理。下面的做法加上 a header node，為方便，假設 header node 的初始 data 是-1，但不取用它。

Chain Manipulation Operations with a Header Node

- Create a chain with two nodes as follows



- void Chain::Create2(){

```
void Chain::Create2(){
        ChainNode *temp = new ChainNode(20,0);
        temp = new ChainNode(10,temp);
        first = new ChainNode(-1,temp);
}
```

- void Chain::Insert50(ChainNode *x){

```
void Chain::Insert50(ChainNode *x){
        x->link = new ChainNode(50, x->link); }
```

- Void Chain::Delete(ChainNode *x,ChainNode *y){

```
Void Chain::Delete(ChainNode *x,ChainNode *y){
        y->link = x->link; }
```

Chain Manipulation Operations

- 常做的動作有 insertAtFront, insertAtBack, deleteFromFront, deleteFromBack
- 分別表示：由chain的前方insert, 後方insert, 前方delete, 後方delete.
- 後面的做法除了加上a header node, 並且加上 last 指向最後一個節點

- class Chain;

-
  ```
  class ChainNode{
  friend class Chain;
  public:
          ChainNode(int element = 0,ChainNode *next = 0){ data = element; link = next; }
  private:
          int data;
          ChainNode *link;
  };
  ```

-
  ```
  Class Chain{
  public:
          Chain();                                  // constructor
          void insertAtFront(int element);          // insert element at the front
          void insertAtBack(int element);           // insert element at the back
          void deleteFromFront(int &element);       // delete element at the front
          void deleteFromBack(int &element);        // delete element at the back
  private:
          ChainNode *first;   // 多一個header node, first指在它上面
          ChainNode *last;    // last 指向最後一個節點。注意:ChainNode *first, last;是錯的
  };
  ```

-
  ```
  Chain::Chain(){
          first = new ChainNode(-1,0);   // header node
          last = first;
  }
  ```

-
  ```
  void Chain::insertAtFront(int element){
          ChainNode *temp = new ChainNode(element, first->link); // 新增 a node 要放在最前面
          first->link = temp; // the front node 變了，first->link 改為指向它
          if(first == last) last = temp;
  }
  ```

-
  ```
  void Chain::insertAtBack(int element){
          last->link = new ChainNode(element,0)  //新增 a node 要放在最後面
          last = last->link; // the last node 變了，last 改為指向它

  }
  ```

-
  ```
  Chain::deleteFromFront(int &element){
          ChainNode *temp = first->link; // temp 指向 the front node
          element = temp->data;           // 取出 the front node的data並存入element
          first->link = temp->link;       // 移除 the front node(是指在chain中移除它)
          delete temp;                    // 歸還 the front node所指向的記憶體
  }
  ```

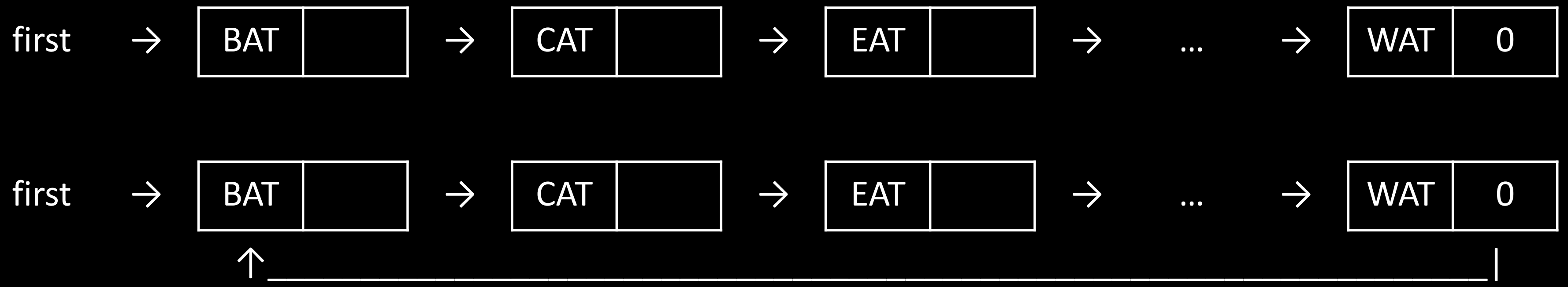
-
  ```
  Chain::deleteFromBack(int &element){
          ChainNode *temp = first;        // 和 deleteFromFront 不同，這裡需要找出last的前一點
          while(temp->link != last) temp = temp->link;   // temp指向last的前一點
          element = last->data;           // 取出 the last node的data並存入element
          temp->link = last->link;        // 移除 the last node(是指在chain中移除它)，也可以寫成temp->link = 0
          delete last;                    // 歸還 last 所指向的記憶體
          last = temp;                    // last變了，它是原本 last 的前一點，亦即 temp
  }
  ```

Chain Manipulation Operations

- 加上模板
- 走訪整個 chain
- 合併 2 chains
- 反轉 a chain

Circular Lists 環狀的鏈結串列

- 將a singly-linked list 的最後節點(the last node)指向第一個節點(the first node)，就得到 a circular list.



first → BAT | → CAT | → EAT | → … → WAT | 0

first → BAT | → CAT | → EAT | → … → WAT | 0

↑____|

Circular Lists 環狀的鏈結串列

- 對 circular list 而言，若只有 first 指標，則不論是insertAtFront或insertAtBack都會花很多時間，其原因是link的改變要知道the last node，而找出the last node要走訪整個list.
- 附帶一提:如何知道目前的節點(假設叫做current)是否為the last node? 可用 current->link == first
- 對circular list而言，會多存last指標，或著就只存last指標，那麼不論是insertAtFront或insertAtBack都可以在O(1)time完成。

Doubly Linked Lists 雙鏈結串列

- 對 singly linked list 而言，要刪除 x 就需要知道 x 的 predecessor，但是對 singly linked list 而言，要知道一個節點的 predecessor 並不容易，要從頭慢慢走訪。
- 對有些問題而言，雙向的(doubly) linked list是較好的選擇。
- 對 doubly linked list 而言，幾乎都會加上 header node 而且變成 circular.
- doubly linked list 的節點至少有3個欄位(fields),書上取為data,left,right.
- 若 x 指向某節點，則 x equals to x->left->right equals to x->right->left

作業4:實作 Radix sort

- 說明：吾人想利用 radix sort 進行PR值的排序，其中每個人的PR值為一個小數，整數部份介於0到99之間，小數部份只有一位。每次排序先輸入欲排序的人數(<100000)，而後輸入每個人的PR值進行排序。(100分)
- 舉例：
  輸入：10 27.1 9.3 3.3 98.4 5.5 30.6 20 17.9 85.9 0.9
  輸出：0.9 3.3 5.5 9.3 17.9 20 27.1 30.6 85.9 98.4
- 限制：需使用 linked list 儲存資料、函式庫只能使用 iostream
- 加分：重複執行直到使用者輸入0人(10分)，將所有變數和函式都包在class內(10分)