Linked List 鏈結串列

Yu-Hsuan Chen

2019 Spring, Introduction to Programming

複習 struct 語法

```
struct [結構名稱]

{

      [結構成員]

};
```

C裡面使用需要加上Struct識別字
C++不需要
(直接使用 Student S;)

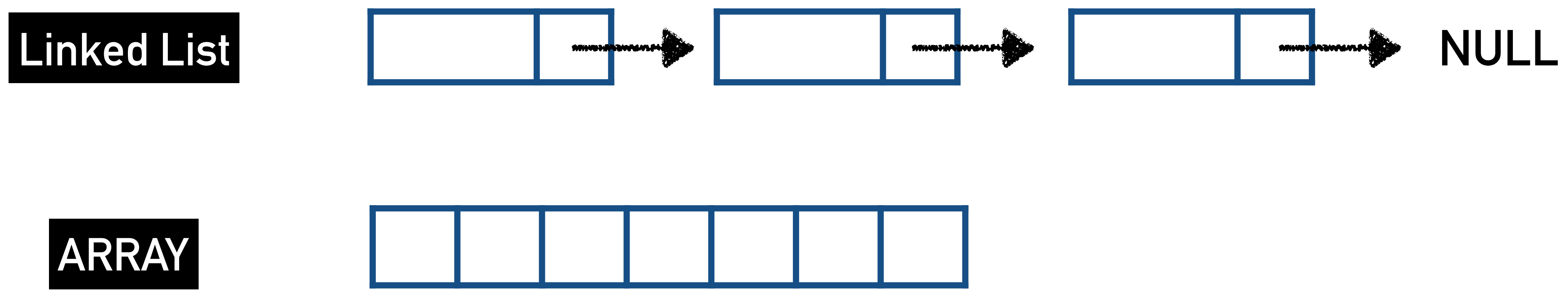
舉個實際例子：

```
struct Student
{
    string name;
    string id;
    int score;
};

int main()
{
    struct Student S;
    struct Student ST[3];
}
```

struct也存在陣列的型態

Linked List介紹



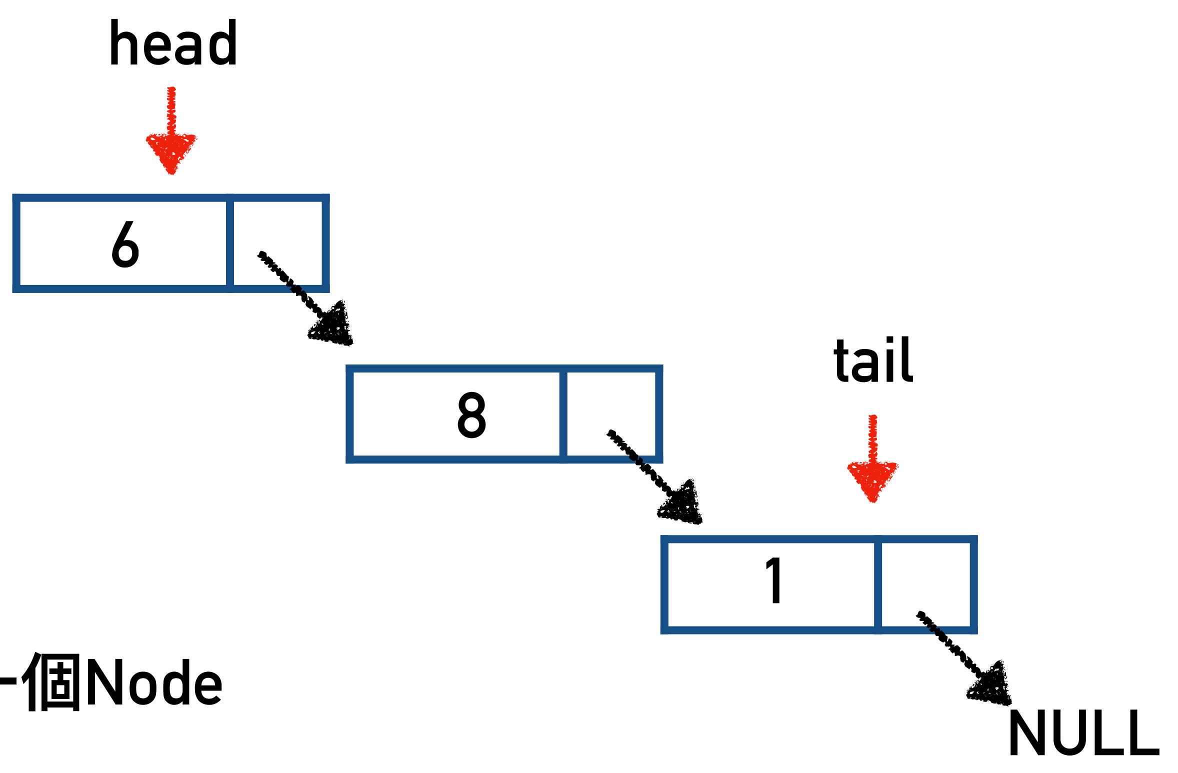
- Linked List，鏈結串列，是使用Node(節點)來記錄資料，每個Node都會記錄下一個Node的位址，最終會串成一個長鍊。

- Linked List是使用struct來實作

Linked List as Struct

```
struct Node
{
    int value;
    Node *next;
};
```

next負責指向下一個Node



```
int main()
{
    Node * head, tail;
    …
}
```

List會需要一個head指標記錄這個串列的起始
這個head如果被改掉了 那麼整個List也就失去控制權了
tail指標則是可有可無的

Linked List v.s. Array

Linked List	Array
新增/刪除資料容易 調整Pointer即可串起新資料	新增/刪除資料麻煩 需要移動大量的陣列內容
大小隨意調整	宣告固定大小之後就無法修改
因為只有head指標 存取內容較慢	可以任意存取陣列的元素

Linked List的基本操作

without TAIL

- 插入Node在List最後方
- 刪除List中的任何Node
- 拜訪整個List

插入Node在List後方

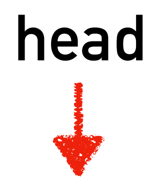
```
Node *head, *term;
…
insert(&head, term);
```

```
void insert(Node **headPtr, Node *term)
{
    if(*headPtr == NULL)
        *headPtr = term;
    else
    {
        Node *temp = *headPtr;
        while(temp->next != NULL)
            temp = temp->next;
        temp->next = term;
    }
    term->next = NULL;
}
```

插入Node在List後方

```
Node *head, *term;
…
insert(&head, term);
```

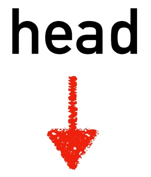
```
void insert(Node **headPtr, Node *term)
{
    if(*headPtr == NULL)
        *headPtr = term;
    else
    {
        Node *temp = *headPtr;
        while(temp->next != NULL)
            temp = temp->next;
        temp->next = term;
    }
    term->next = NULL;
}
```



插入Node在List後方

```
Node *head, *term;
…
insert(&head, term);
```

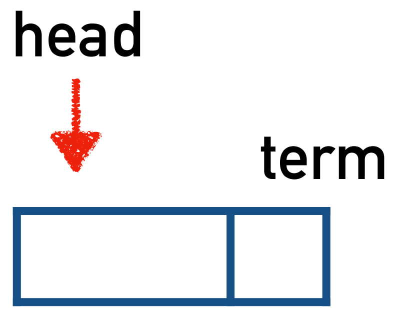
```
void insert(Node **headPtr, Node *term)
{
    if(*headPtr == NULL)
        *headPtr = term;
    else
    {
        Node *temp = *headPtr;
        while(temp->next != NULL)
            temp = temp->next;
        temp->next = term;
    }
    term->next = NULL;
}
```



插入Node在List後方

```
Node *head, *term;
…
insert(&head, term);
```

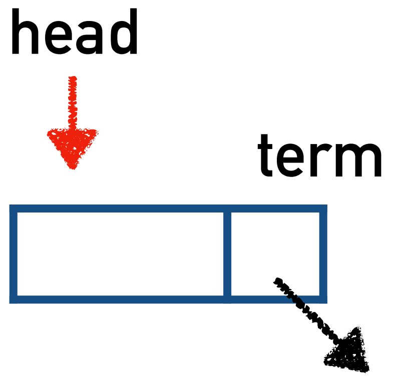
```
void insert(Node **headPtr, Node *term)
{
    if(*headPtr == NULL)
        *headPtr = term;
    else
    {
        Node *temp = *headPtr;
        while(temp->next != NULL)
            temp = temp->next;
        temp->next = term;
    }
    term->next = NULL;
}
```



插入Node在List後方

```
Node *head, *term;
…
insert(&head, term);
```

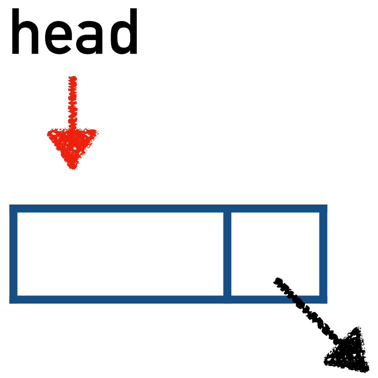
```
void insert(Node **headPtr, Node *term)
{
    if(*headPtr == NULL)
        *headPtr = term;
    else
    {
        Node *temp = *headPtr;
        while(temp->next != NULL)
            temp = temp->next;
        temp->next = term;
    }
    term->next = NULL;
}
```



插入Node在List後方

```
Node *head, *term;
…
insert(&head, term);
```

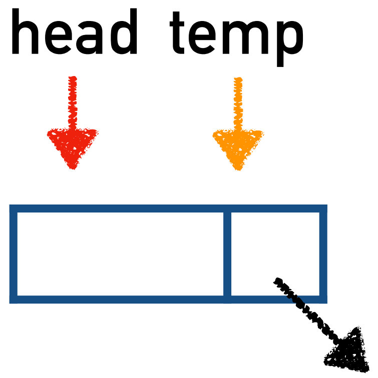
```
void insert(Node **headPtr, Node *term)
{
    if(*headPtr == NULL)
        *headPtr = term;
    else
    {
        Node *temp = *headPtr;
        while(temp->next != NULL)
            temp = temp->next;
        temp->next = term;
    }
    term->next = NULL;
}
```



插入Node在List後方

```
Node *head, *term;
…
insert(&head, term);
```

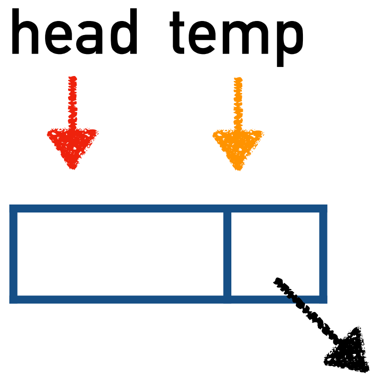
```
void insert(Node **headPtr, Node *term)
{
    if(*headPtr == NULL)
        *headPtr = term;
    else
    {
        Node *temp = *headPtr;
        while(temp->next != NULL)
            temp = temp->next;
        temp->next = term;
    }
    term->next = NULL;
}
```



插入Node在List後方

```
Node *head, *term;
…
insert(&head, term);
```

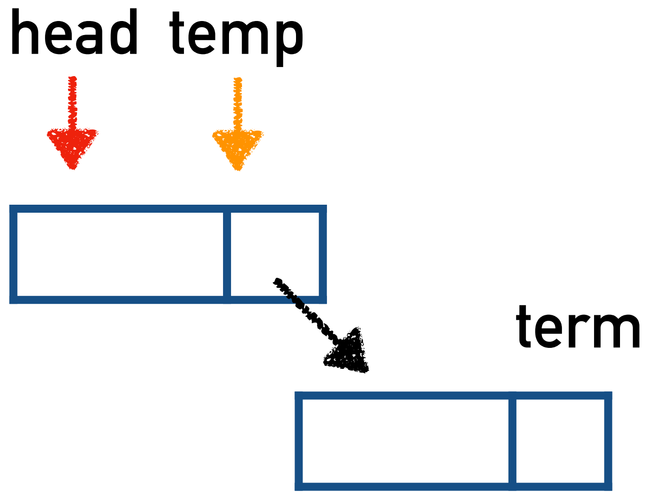
```
void insert(Node **headPtr, Node *term)
{
    if(*headPtr == NULL)
        *headPtr = term;
    else
    {
        Node *temp = *headPtr;
        while(temp->next != NULL)
            temp = temp->next;
        temp->next = term;
    }
    term->next = NULL;
}
```



插入Node在List後方

```
Node *head, *term;
…
insert(&head, term);
```

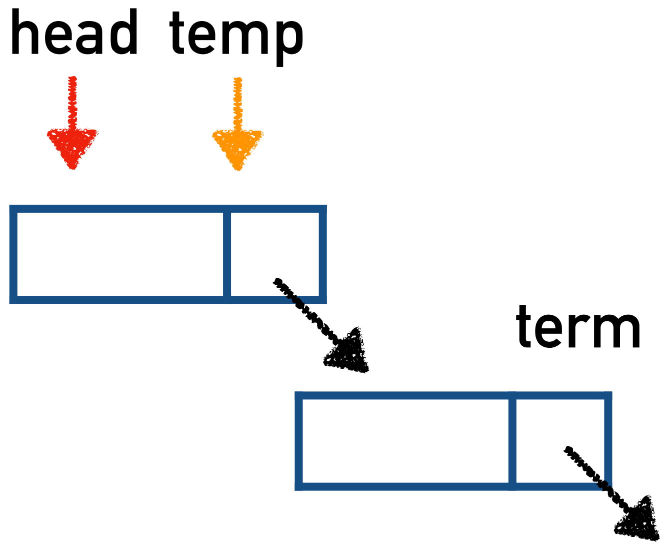
```
void insert(Node **headPtr, Node *term)
{
    if(*headPtr == NULL)
        *headPtr = term;
    else
    {
        Node *temp = *headPtr;
        while(temp->next != NULL)
            temp = temp->next;
  →     temp->next = term;
    }
    term->next = NULL;
}
```



插入Node在List後方

```
Node *head, *term;
…
insert(&head, term);
```

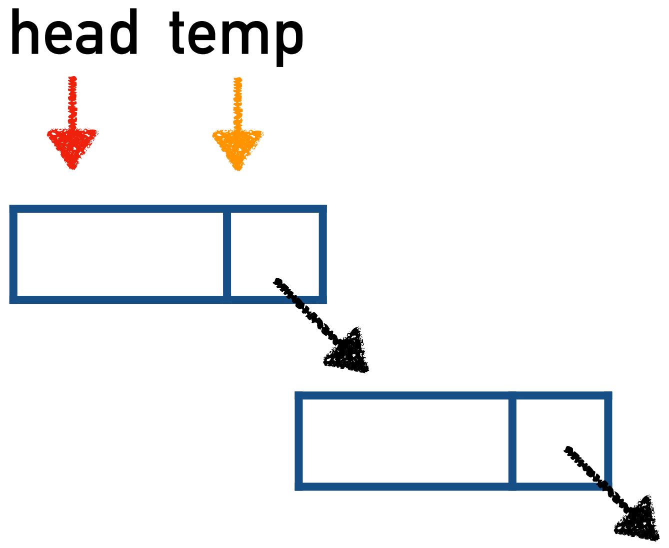
```
void insert(Node **headPtr, Node *term)
{
    if(*headPtr == NULL)
        *headPtr = term;
    else
    {
        Node *temp = *headPtr;
        while(temp->next != NULL)
            temp = temp->next;
        temp->next = term;
    }
    term->next = NULL;
}
```



插入Node在List後方

```
Node *head, *term;
…
insert(&head, term);
```

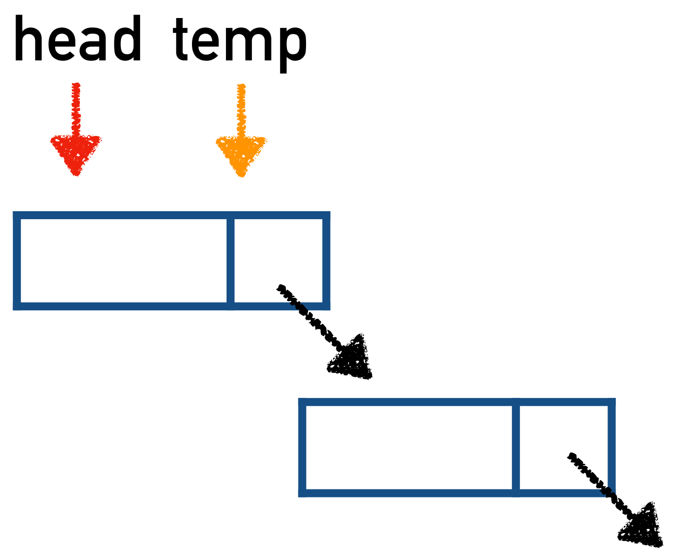
```
void insert(Node **headPtr, Node *term)
{
    if(*headPtr == NULL)
        *headPtr = term;
    else
    {
        Node *temp = *headPtr;
        while(temp->next != NULL)
            temp = temp->next;
        temp->next = term;
    }
    term->next = NULL;
}
```



插入Node在List後方

```
Node *head, *term;
…
insert(&head, term);
```

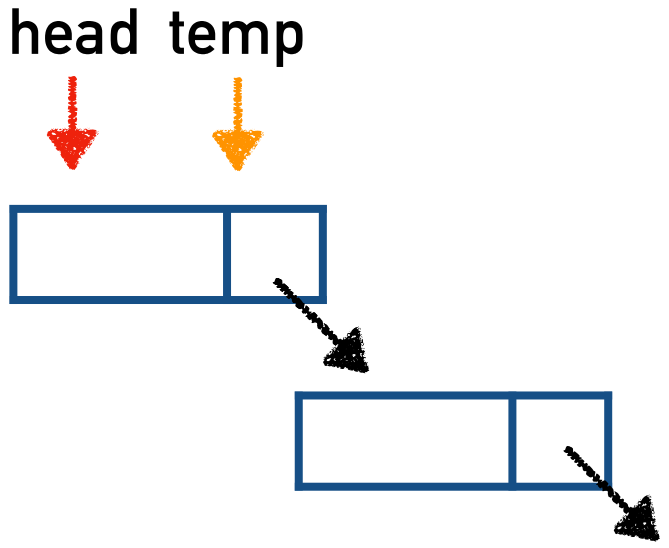
```
void insert(Node **headPtr, Node *term)
{
    if(*headPtr == NULL)
        *headPtr = term;
    else
    {
        Node *temp = *headPtr;
        while(temp->next != NULL)
            temp = temp->next;
        temp->next = term;
    }
    term->next = NULL;
}
```



插入Node在List後方

```
Node *head, *term;
…
insert(&head, term);
```

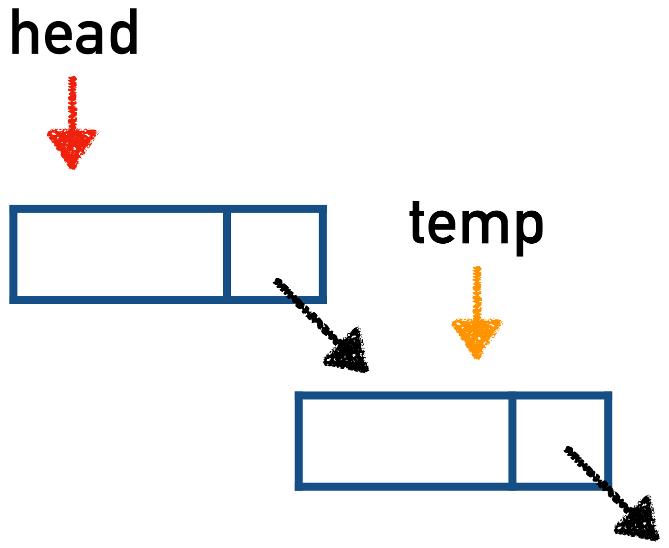
```
void insert(Node **headPtr, Node *term)
{
    if(*headPtr == NULL)
        *headPtr = term;
    else
    {
        Node *temp = *headPtr;
        while(temp->next != NULL)
            temp = temp->next;
        temp->next = term;
    }
    term->next = NULL;
}
```



插入Node在List後方

```
Node *head, *term;
…
insert(&head, term);
```

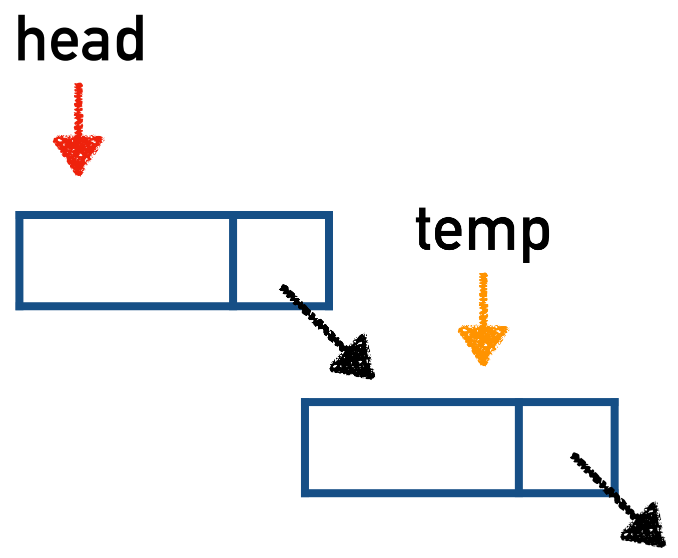
```
void insert(Node **headPtr, Node *term)
{
    if(*headPtr == NULL)
        *headPtr = term;
    else
    {
        Node *temp = *headPtr;
        while(temp->next != NULL)
        →   temp = temp->next;
        temp->next = term;
    }
    term->next = NULL;
}
```



插入Node在List後方

```
Node *head, *term;
…
insert(&head, term);
```

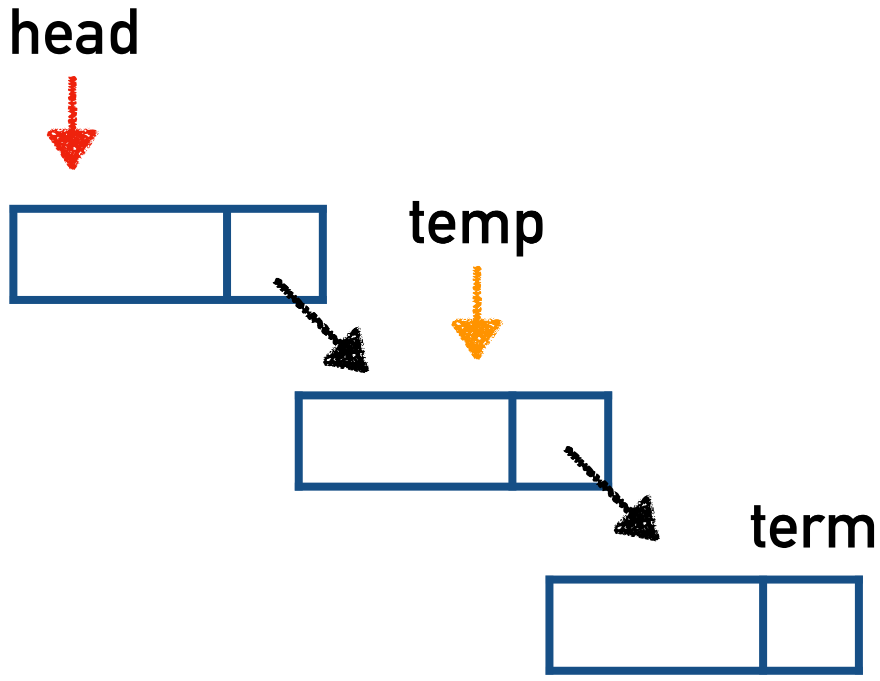
```
void insert(Node **headPtr, Node *term)
{
    if(*headPtr == NULL)
        *headPtr = term;
    else
    {
        Node *temp = *headPtr;
        while(temp->next != NULL)
            temp = temp->next;
        temp->next = term;
    }
    term->next = NULL;
}
```



插入Node在List後方

```
Node *head, *term;
…
insert(&head, term);
```

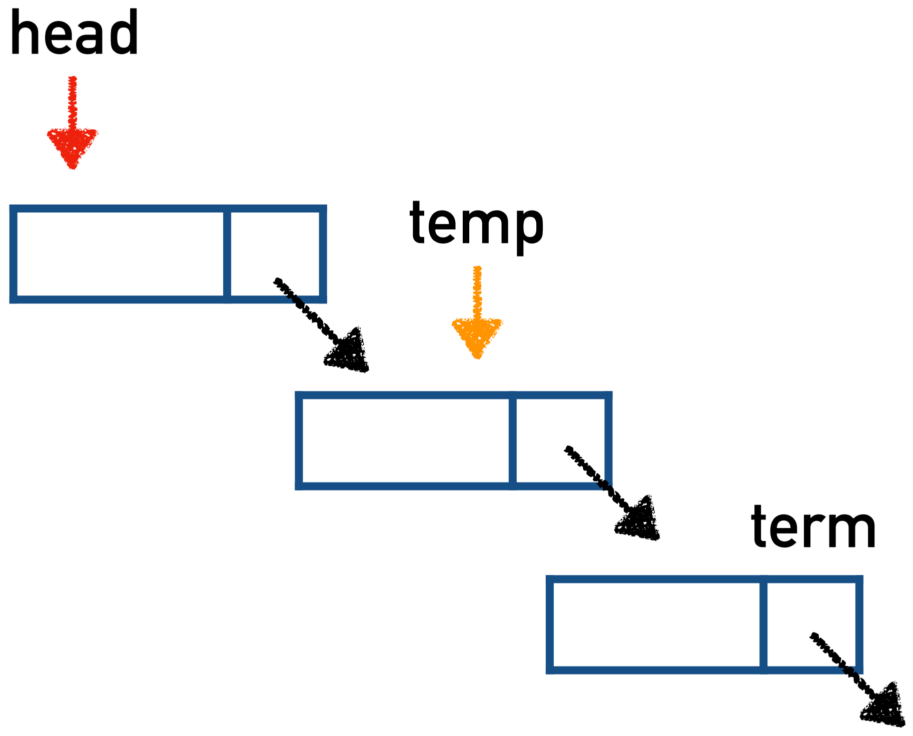
```
void insert(Node **headPtr, Node *term)
{
    if(*headPtr == NULL)
        *headPtr = term;
    else
    {
        Node *temp = *headPtr;
        while(temp->next != NULL)
            temp = temp->next;
        temp->next = term;
    }
    term->next = NULL;
}
```



插入Node在List後方

```
Node *head, *term;
…
insert(&head, term);
```

```
void insert(Node **headPtr, Node *term)
{
    if(*headPtr == NULL)
        *headPtr = term;
    else
    {
        Node *temp = *headPtr;
        while(temp->next != NULL)
            temp = temp->next;
        temp->next = term;
    }
    term->next = NULL;
}
```

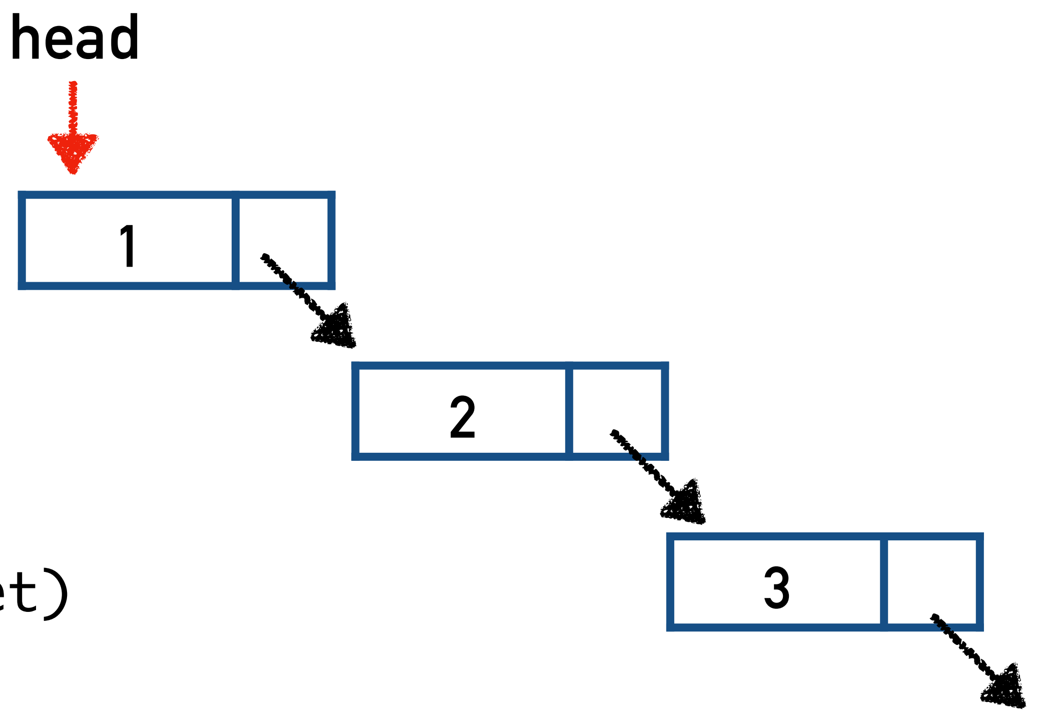


刪除特定的Node

示範：先刪除第二個Node，再刪除Head Node

```
void delete(Node **head, string target)
{
    if((*head)->name == target)
    {
        Node *temp = *head;
        (*head) = (*head)->next;
    }
    else
    {
        Node *current = *head;
        while(current->next !=NULL)
            if(current->next->name != target)
                current = current->next;
        Node *temp = current->next;
        current->next = temp->next;
    }
    delete temp;
}
```

```
Node *head;
string target;
…
delete(&head, target);
```

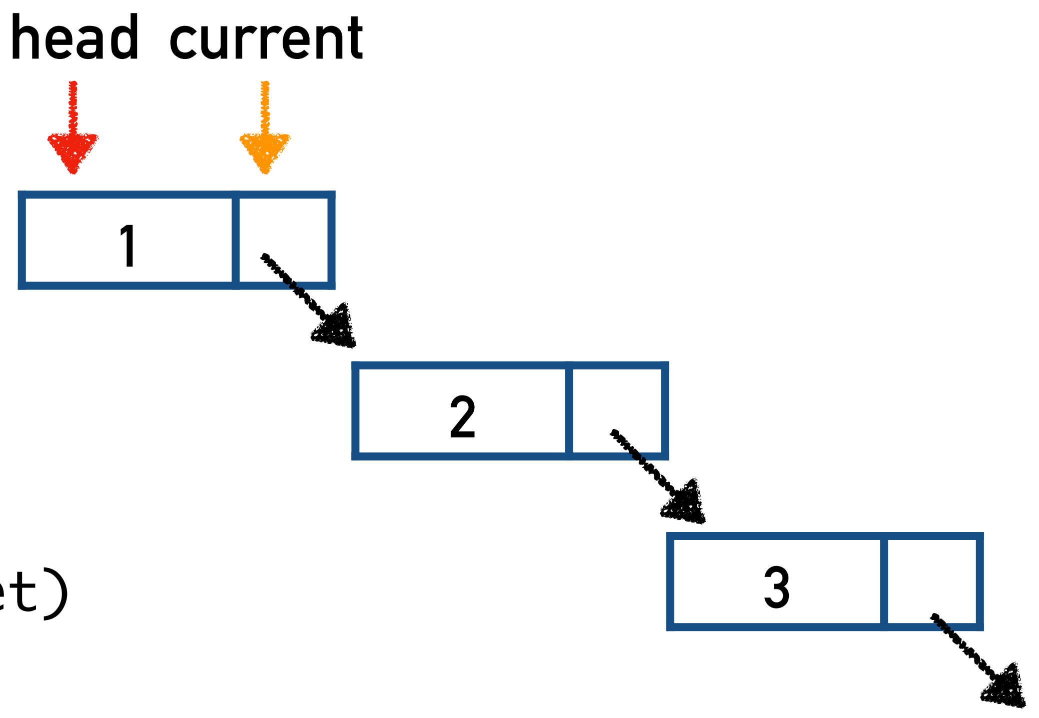


刪除特定的Node

示範：先刪除第二個Node，再刪除Head Node

```
void delete(Node **head, string target)
{
    if((*head)->name == target)
    {
        Node *temp = *head;
        (*head) = (*head)->next;
    }
    else
    {
        Node *current = *head;
        while(current->next !=NULL)
            if(current->next->name != target)
                current = current->next;
        Node *temp = current->next;
        current->next = temp->next;
    }
    delete temp;
}
```

```
Node *head;
string target;
…
delete(&head, target);
```

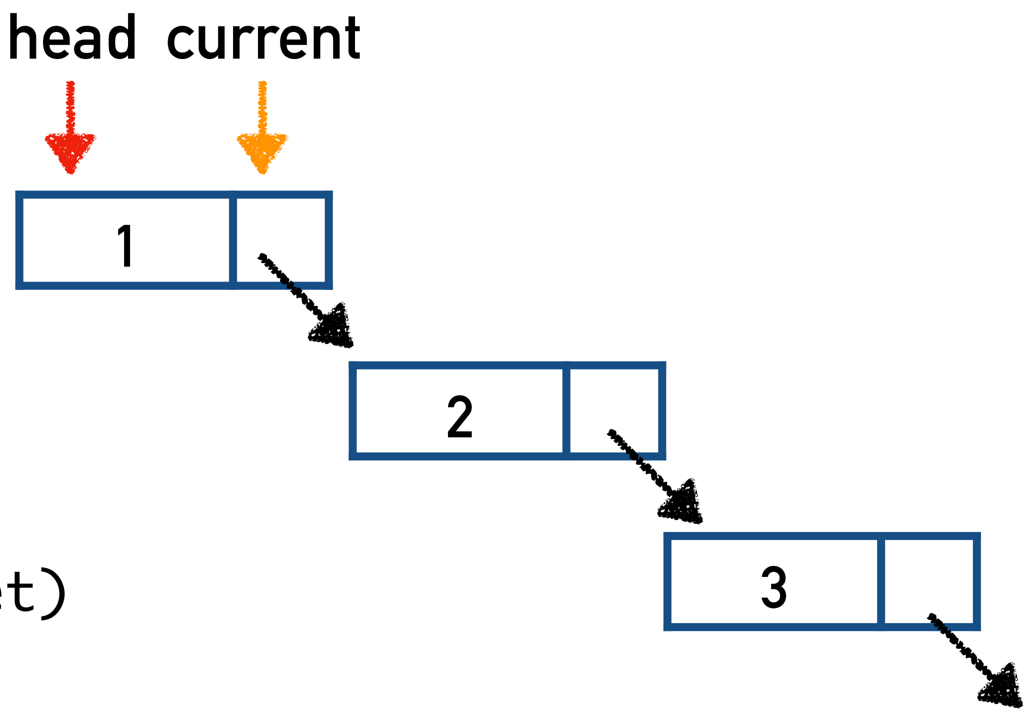


刪除特定的Node

示範：先刪除第二個Node，再刪除Head Node

```
void delete(Node **head, string target)
{
    if((*head)->name == target)
    {
        Node *temp = *head;
        (*head) = (*head)->next;
    }
    else
    {
        Node *current = *head;
        while(current->next !=NULL)
            if(current->next->name != target)
                current = current->next;
        Node *temp = current->next;
        current->next = temp->next;
    }
    delete temp;
}
```

```
Node *head;
string target;
…
delete(&head, target);
```

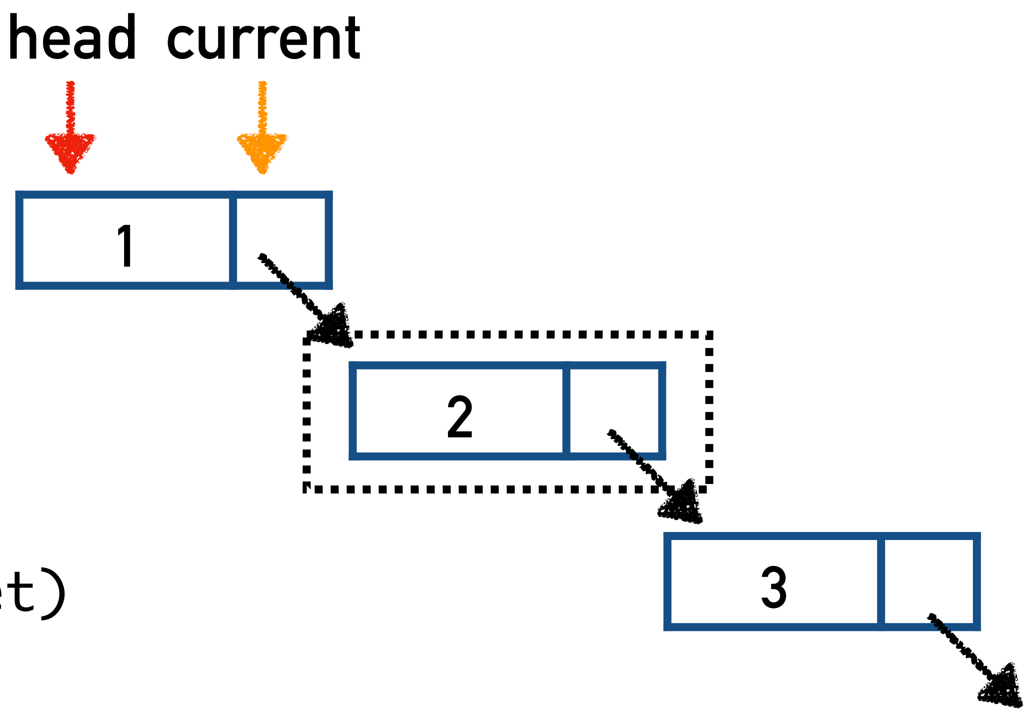


刪除特定的Node

示範：先刪除第二個Node，再刪除Head Node

```
void delete(Node **head, string target)
{
    if((*head)->name == target)
    {
        Node *temp = *head;
        (*head) = (*head)->next;
    }
    else
    {
        Node *current = *head;
        while(current->next !=NULL)
        if(current->next->name != target)
            current = current->next;
        Node *temp = current->next;
        current->next = temp->next;
    }
    delete temp;
}
```

```
Node *head;
string target;
…
delete(&head, target);
```

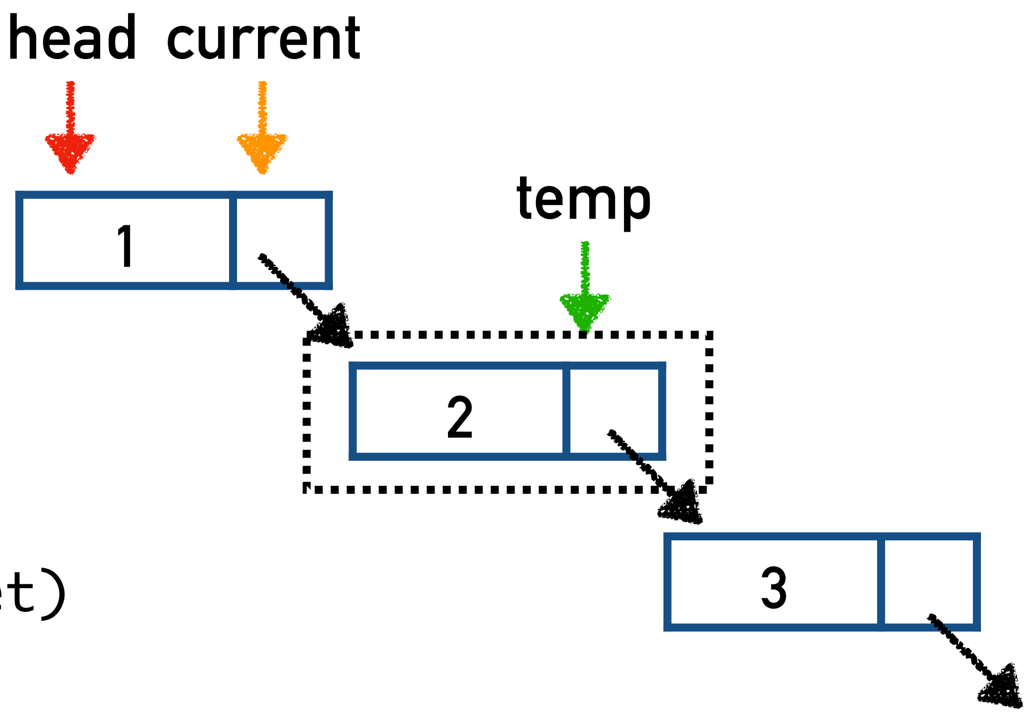


刪除特定的Node

示範：先刪除第二個Node，再刪除Head Node

```
void delete(Node **head, string target)
{
    if((*head)->name == target)
    {
        Node *temp = *head;
        (*head) = (*head)->next;
    }
    else
    {
        Node *current = *head;
        while(current->next !=NULL)
            if(current->next->name != target)
                current = current->next;
        Node *temp = current->next;
        current->next = temp->next;
    }
    delete temp;
}
```

```
Node *head;
string target;
…
delete(&head, target);
```

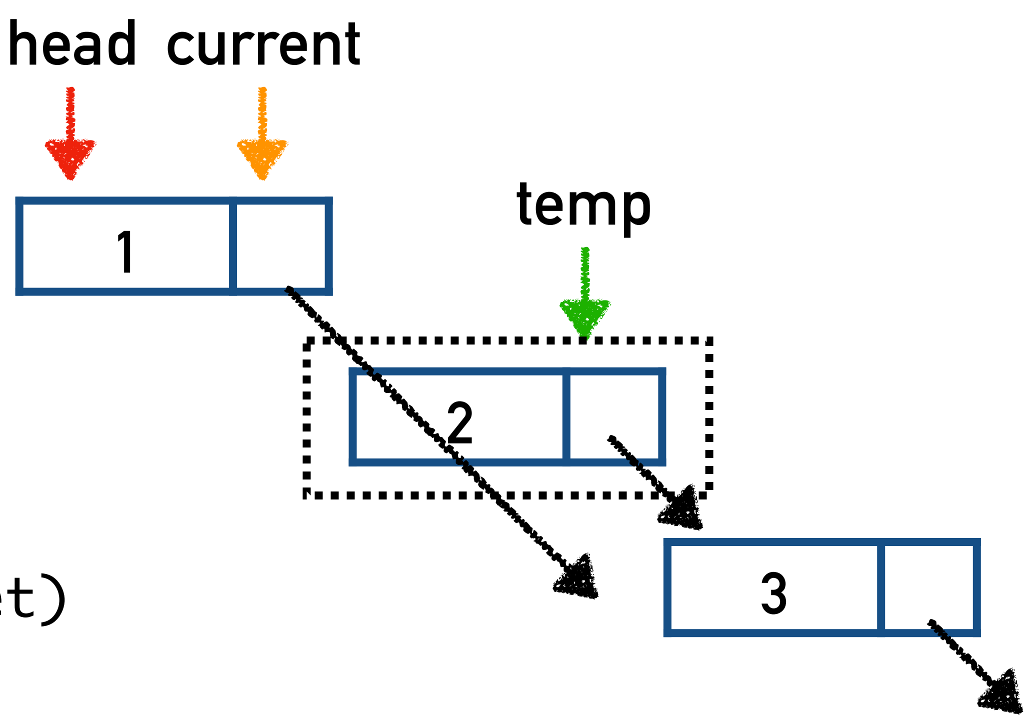


刪除特定的Node

示範：先刪除第二個Node，再刪除Head Node

```
void delete(Node **head, string target)
{
    if((*head)->name == target)
    {
        Node *temp = *head;
        (*head) = (*head)->next;
    }
    else
    {
        Node *current = *head;
        while(current->next !=NULL)
            if(current->next->name != target)
                current = current->next;
        Node *temp = current->next;
        current->next = temp->next;
    }
    delete temp;
}
```

```
Node *head;
string target;
…
delete(&head, target);
```

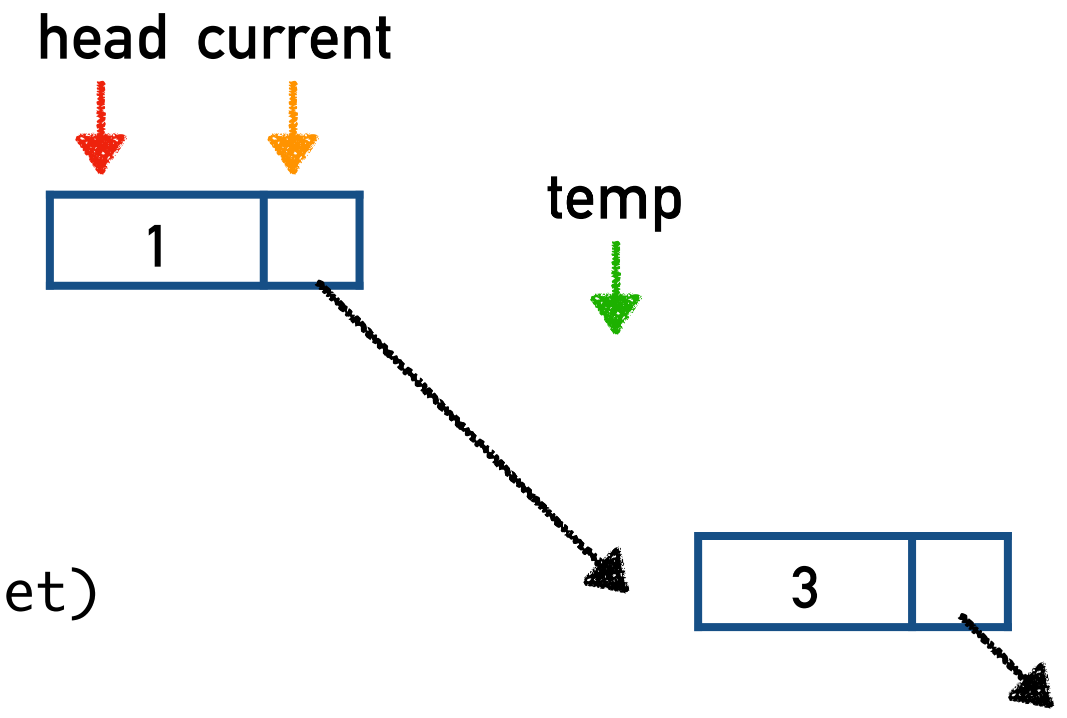


刪除特定的Node

示範：先刪除第二個Node，再刪除Head Node

```
void delete(Node **head, string target)
{
    if((*head)->name == target)
    {
        Node *temp = *head;
        (*head) = (*head)->next;
    }
    else
    {
        Node *current = *head;
        while(current->next !=NULL)
            if(current->next->name != target)
                current = current->next;
        Node *temp = current->next;
        current->next = temp->next;
    }
    delete temp;
}
```

```
Node *head;
string target;
…
delete(&head, target);
```

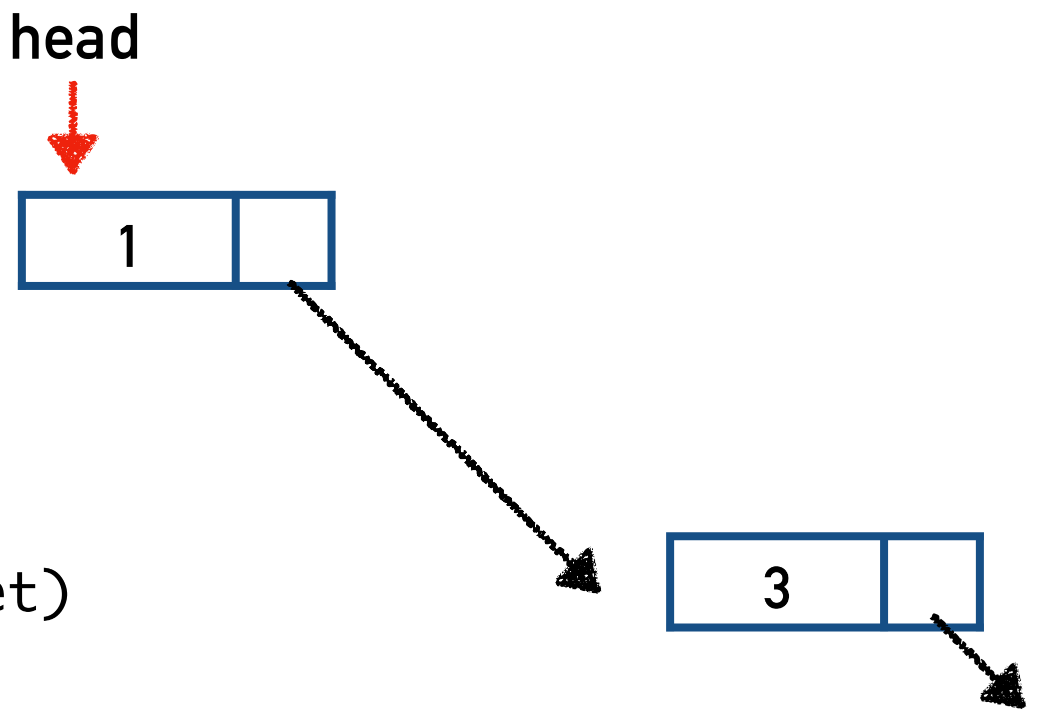


刪除特定的Node

示範：先刪除第二個Node，再刪除Head Node

```
void delete(Node **head, string target)
{
    if((*head)->name == target)
    {
        Node *temp = *head;
        (*head) = (*head)->next;
    }
    else
    {
        Node *current = *head;
        while(current->next !=NULL)
            if(current->next->name != target)
                current = current->next;
        Node *temp = current->next;
        current->next = temp->next;
    }
    delete temp;
}
```

```
Node *head;
string target;
…
delete(&head, target);
```

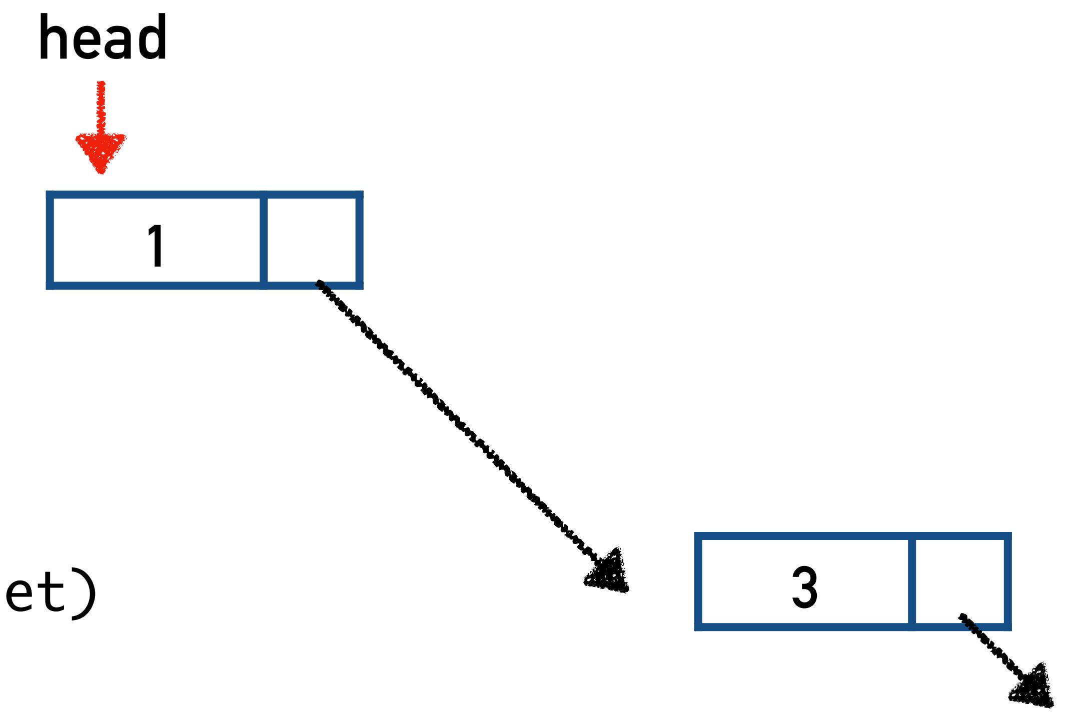


刪除特定的Node

示範：先刪除第二個Node，再刪除Head Node

```
void delete(Node **head, string target)
{
    if((*head)->name == target)
    {
        Node *temp = *head;
        (*head) = (*head)->next;
    }
    else
    {
        Node *current = *head;
        while(current->next !=NULL)
            if(current->next->name != target)
                current = current->next;
        Node *temp = current->next;
        current->next = temp->next;
    }
    delete temp;
}
```

```
Node *head;
string target;
…
delete(&head, target);
```

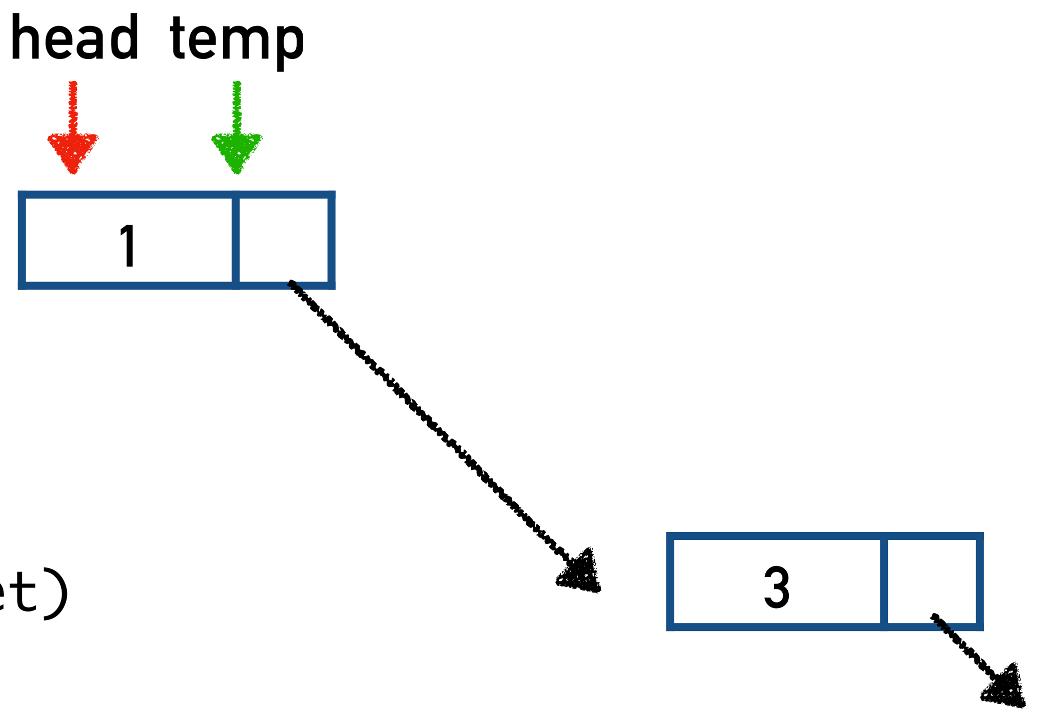


刪除特定的Node

示範：先刪除第二個Node，再刪除Head Node

```
void delete(Node **head, string target)
{
    if((*head)->name == target)
    {
        Node *temp = *head;
        (*head) = (*head)->next;
    }
    else
    {
        Node *current = *head;
        while(current->next !=NULL)
            if(current->next->name != target)
                current = current->next;
        Node *temp = current->next;
        current->next = temp->next;
    }
    delete temp;
}
```

```
Node *head;
string target;
…
delete(&head, target);
```

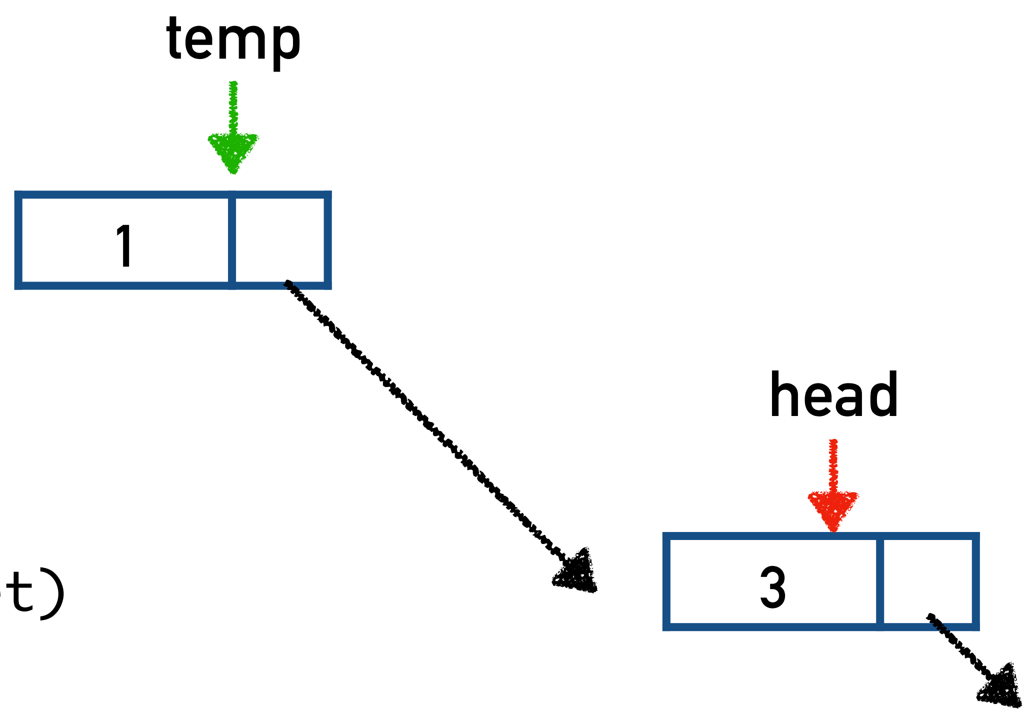


刪除特定的Node

示範：先刪除第二個Node，再刪除Head Node

```
void delete(Node **head, string target)
{
    if((*head)->name == target)
    {
        Node *temp = *head;
        (*head) = (*head)->next;
    }
    else
    {
        Node *current = *head;
        while(current->next !=NULL)
            if(current->next->name != target)
                current = current->next;
        Node *temp = current->next;
        current->next = temp->next;
    }
    delete temp;
}
```

```
Node *head;
string target;
…
delete(&head, target);
```

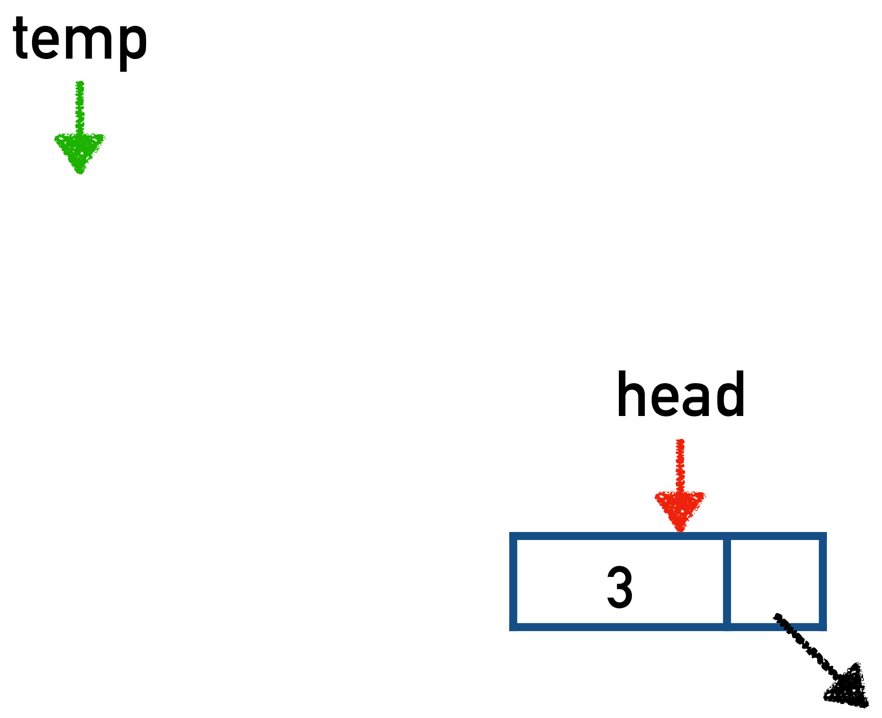


刪除特定的Node

示範：先刪除第二個Node，再刪除Head Node

```
void delete(Node **head, string target)
{
    if((*head)->name == target)
    {
        Node *temp = *head;
        (*head) = (*head)->next;
    }
    else
    {
        Node *current = *head;
        while(current->next !=NULL)
            if(current->next->name != target)
                current = current->next;
        Node *temp = current->next;
        current->next = temp->next;
    }
    delete temp;
}
```

```
Node *head;
string target;
…
delete(&head, target);
```

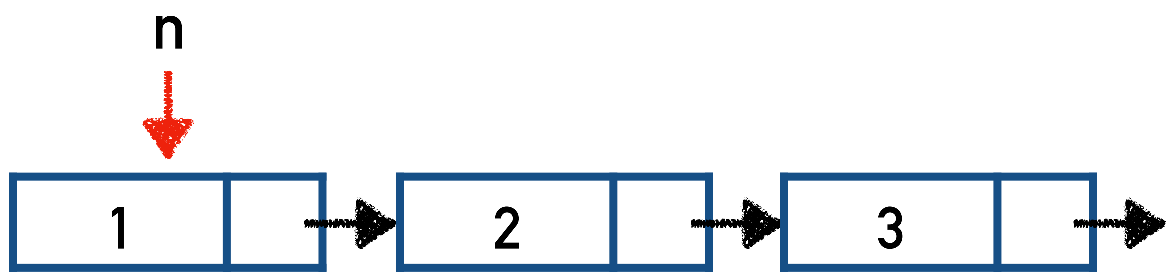


拜訪整個List

```
Node *head=NULL;
…
printList(head);
```

```
void printList(Node *n)
{
    while(n != NULL)
    {
        cout << n->value << endl;
        n = n->next;
    }
}
```

```
OUTPUT
```

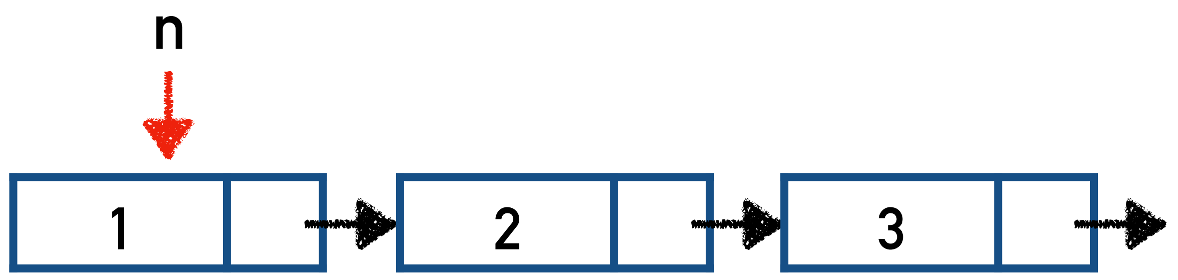


拜訪整個List

```
Node *head=NULL;
…
printList(head);
```

```
void printList(Node *n)
{
    while(n != NULL)
    {
        cout << n->value << endl;
        n = n->next;
    }
}
```

OUTPUT

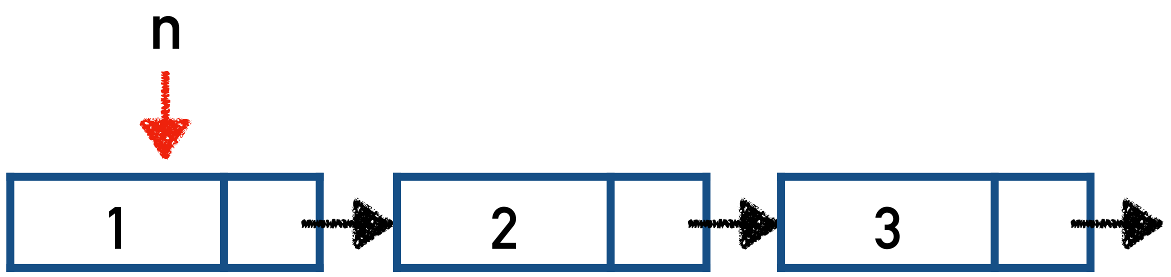


拜訪整個List

```
Node *head=NULL;
…
printList(head);
```

```
void printList(Node *n)
{
    while(n != NULL)
    {
        cout << n->value << endl;
        n = n->next;
    }
}
```

```
OUTPUT
 1
```

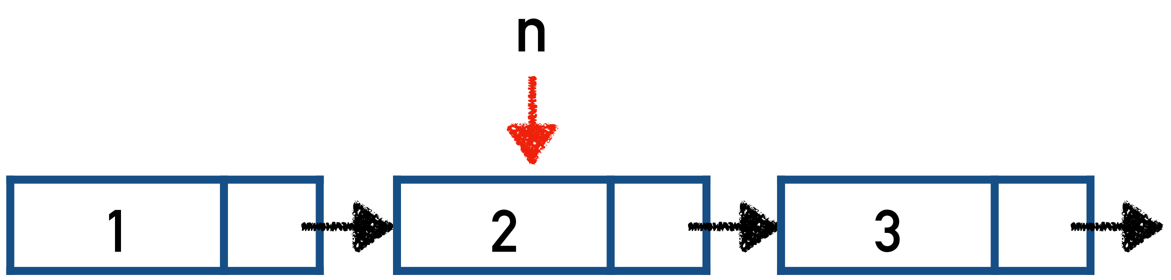


拜訪整個List

```
Node *head=NULL;
…
printList(head);
```

```
void printList(Node *n)
{
    while(n != NULL)
    {
        cout << n->value << endl;
        n = n->next;
    }
}
```

```
OUTPUT
 1
```

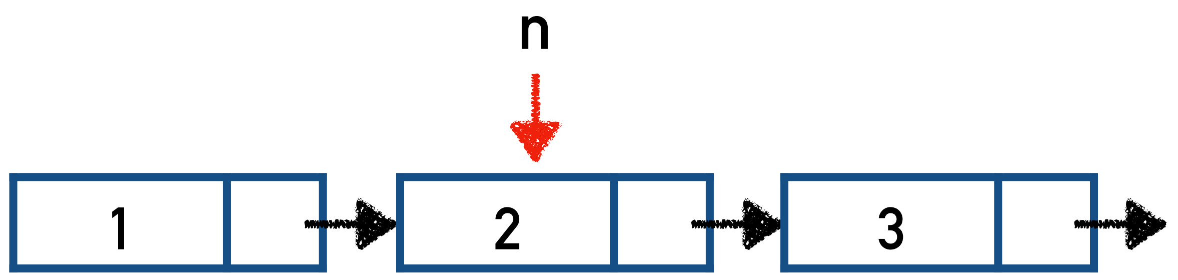


拜訪整個List

```
Node *head=NULL;
…
printList(head);
```

```
void printList(Node *n)
{
    while(n != NULL)
    {
        cout << n->value << endl;
        n = n->next;
    }
}
```

```
OUTPUT
 1
```



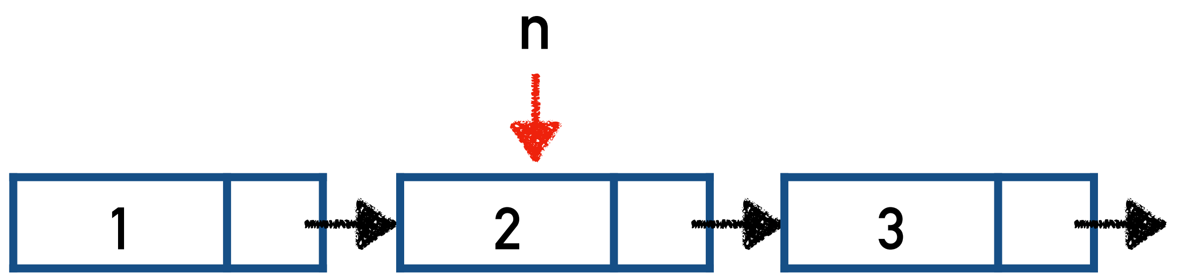
拜訪整個List

```
Node *head=NULL;
…
printList(head);
```

```
void printList(Node *n)
{
    while(n != NULL)
    {
        cout << n->value << endl;
        n = n->next;
    }
}
```

```
OUTPUT

 1
 2
```



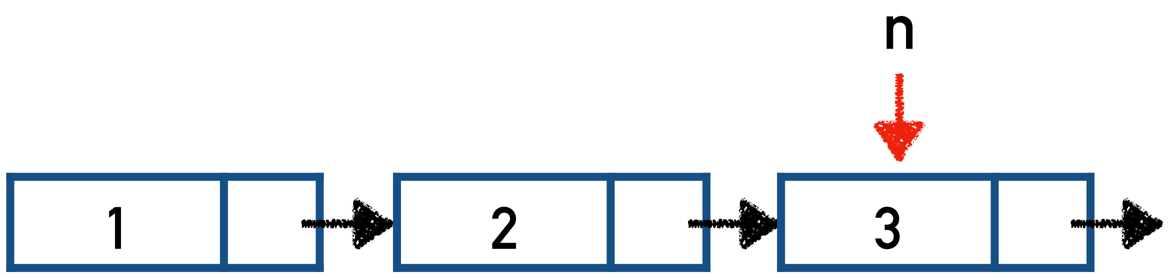
拜訪整個List

```
Node *head=NULL;
…
printList(head);
```

```
void printList(Node *n)
{
    while(n != NULL)
    {
        cout << n->value << endl;
        n = n->next;
    }
}
```

```
OUTPUT

 1
 2
```



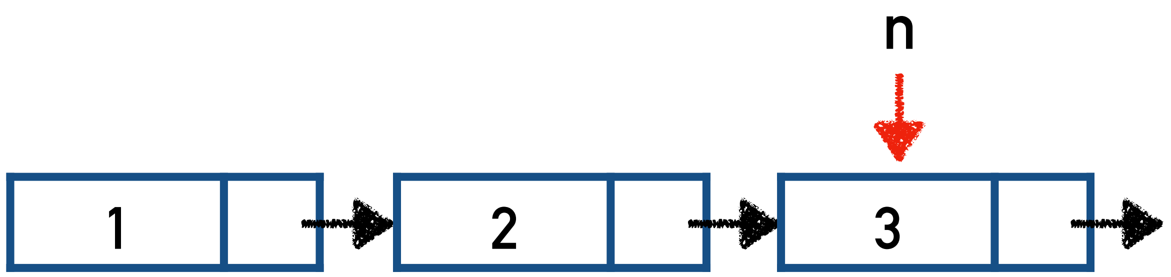
拜訪整個List

```
Node *head=NULL;
…
printList(head);
```

```
void printList(Node *n)
{
    while(n != NULL)
    {
        cout << n->value << endl;
        n = n->next;
    }
}
```

```
OUTPUT

 1
 2
```



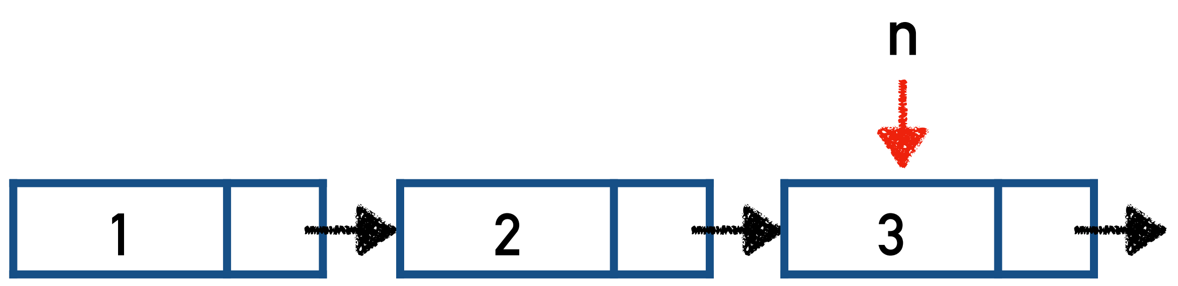
拜訪整個List

```
Node *head=NULL;
…
printList(head);
```

```
void printList(Node *n)
{
    while(n != NULL)
    {
        cout << n->value << endl;
        n = n->next;
    }
}
```

```
OUTPUT

 1
 2
 3
```



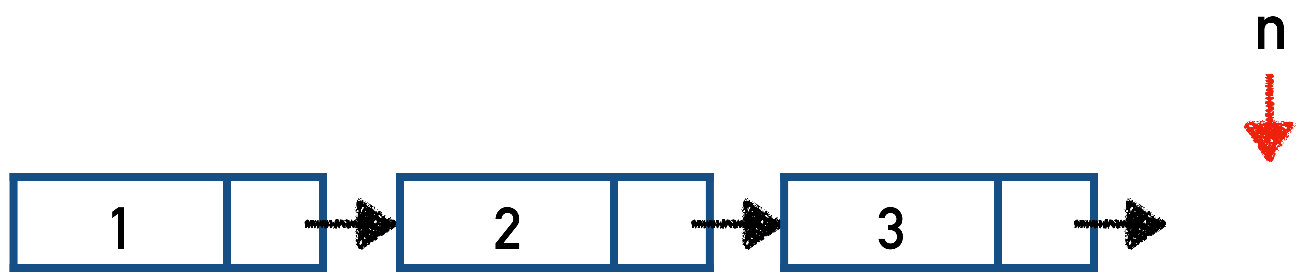
拜訪整個List

```
Node *head=NULL;
…
printList(head);
```

```
void printList(Node *n)
{
    while(n != NULL)
    {
        cout << n->value << endl;
        n = n->next;
    }
}
```

```
OUTPUT

 1
 2
 3
```



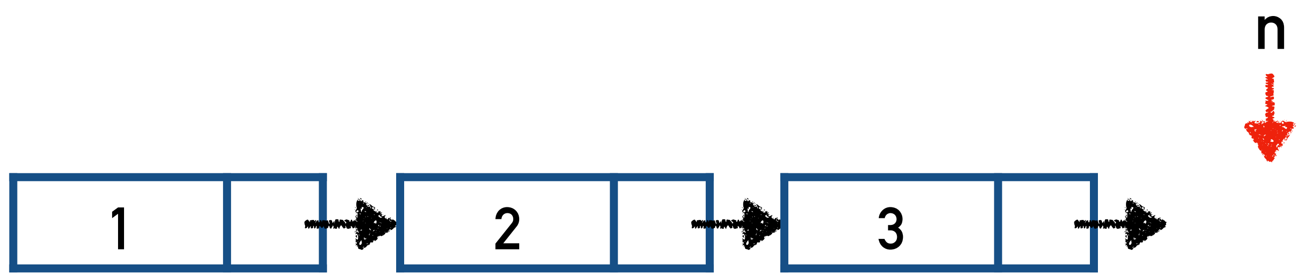
拜訪整個List

```
Node *head=NULL;
…
printList(head);
```

```
void printList(Node *n)
{
    while(n != NULL)
    {
        cout << n->value << endl;
        n = n->next;
    }
}
```

```
OUTPUT

 1
 2
 3
```

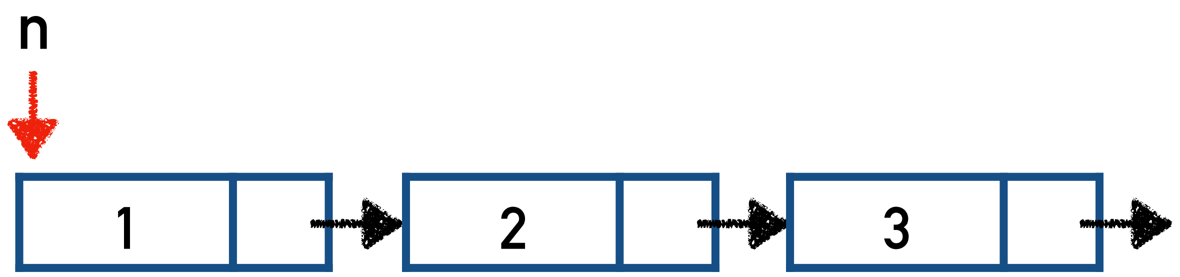


回收整個List

用完的記憶體是要還的

```
Node *head=NULL;
…
deleteList(head);
```

```
void deleteList(Node *n)
{
    while(n != NULL)
    {
        Node *temp = n;
        n = n->next;
        delete temp;
    }
}
```

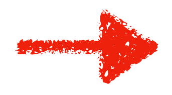


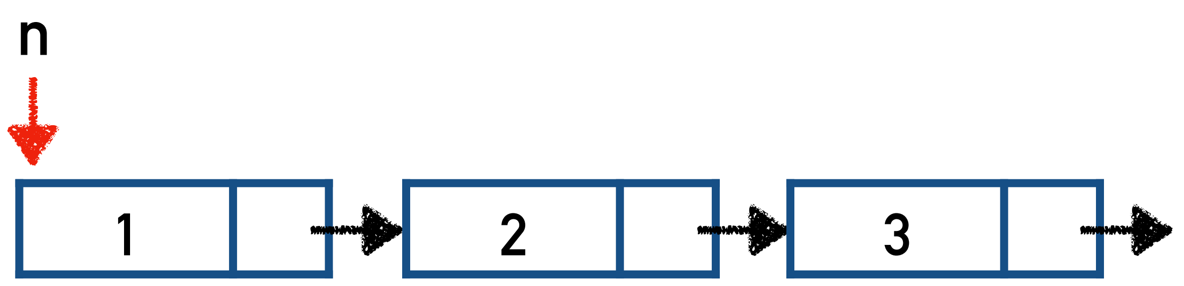
回收整個List

用完的記憶體是要還的

```
Node *head=NULL;
…
deleteList(head);
```

```
void deleteList(Node *n)
{
    while(n != NULL)
    {
        Node *temp = n;
        n = n->next;
        delete temp;
    }
}
```



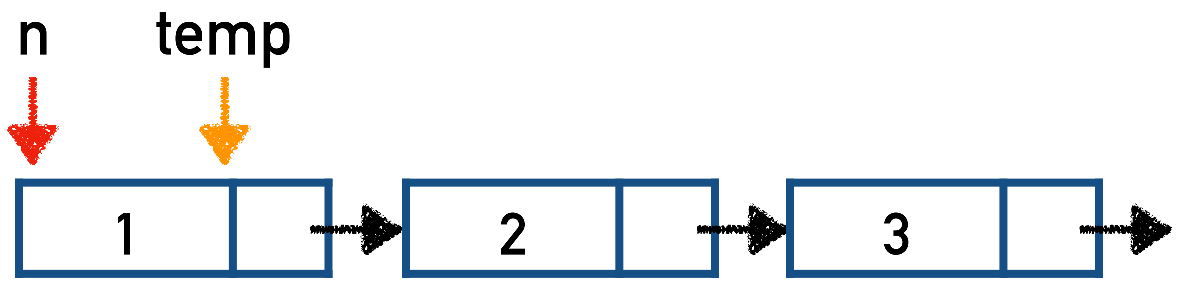


回收整個List

用完的記憶體是要還的

```
Node *head=NULL;
…
deleteList(head);
```

```
void deleteList(Node *n)
{
    while(n != NULL)
    {
        Node *temp = n;
        n = n->next;
        delete temp;
    }
}
```

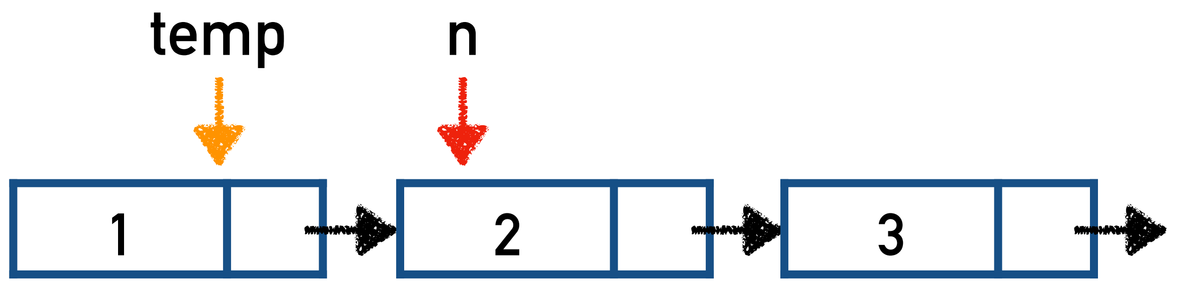


回收整個List

用完的記憶體是要還的

```
Node *head=NULL;
…
deleteList(head);
```

```
void deleteList(Node *n)
{
    while(n != NULL)
    {
        Node *temp = n;
        n = n->next;
        delete temp;
    }
}
```

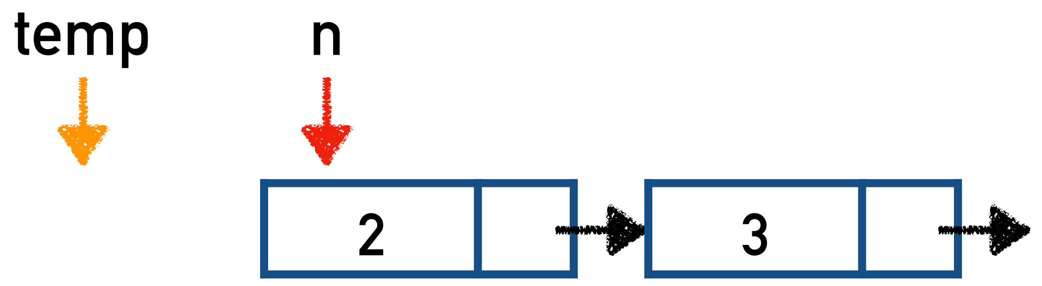


回收整個List

用完的記憶體是要還的

```
Node *head=NULL;
…
deleteList(head);
```

```
void deleteList(Node *n)
{
    while(n != NULL)
    {
        Node *temp = n;
        n = n->next;
        delete temp;
    }
}
```

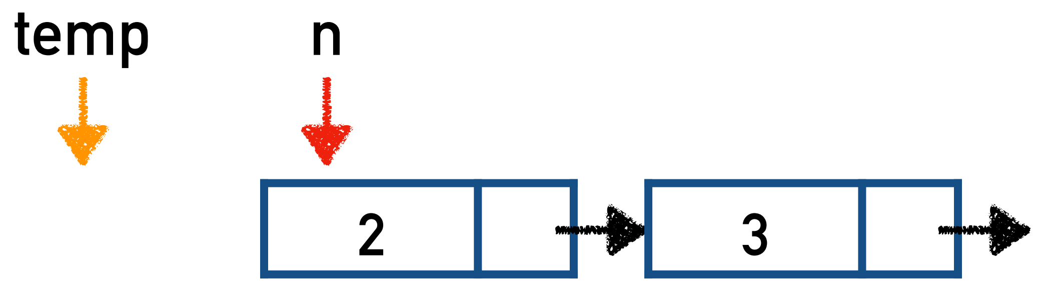


回收整個List

用完的記憶體是要還的

```
Node *head=NULL;
…
deleteList(head);
```

```
void deleteList(Node *n)
{
    while(n != NULL)
    {
        Node *temp = n;
        n = n->next;
        delete temp;
    }
}
```

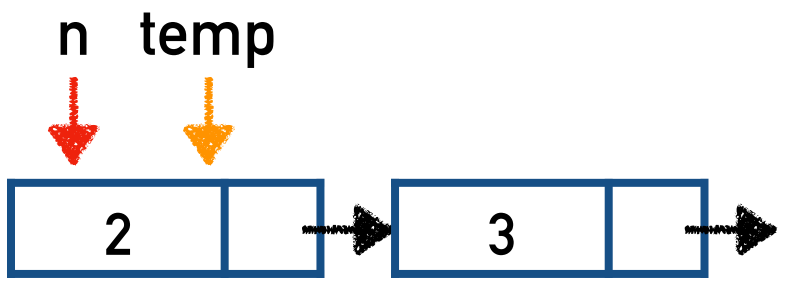


回收整個List

用完的記憶體是要還的

```
Node *head=NULL;
…
deleteList(head);
```

```
void deleteList(Node *n)
{
    while(n != NULL)
    {
        Node *temp = n;
        n = n->next;
        delete temp;
    }
}
```

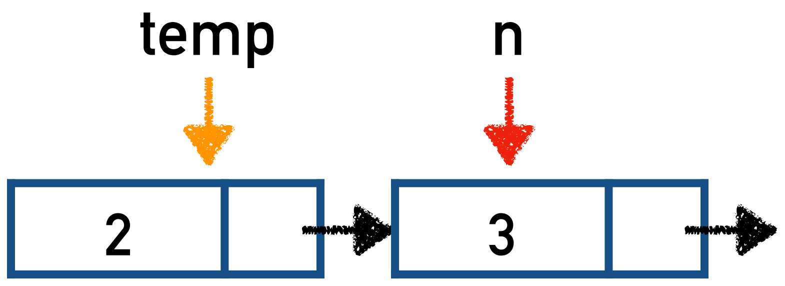


回收整個List

用完的記憶體是要還的

```
Node *head=NULL;
…
deleteList(head);
```

```
void deleteList(Node *n)
{
    while(n != NULL)
    {
        Node *temp = n;
        n = n->next;
        delete temp;
    }
}
```

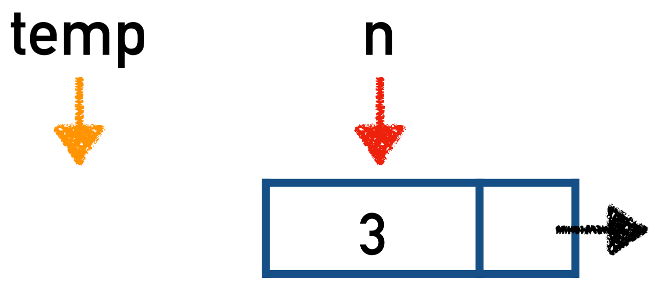


回收整個List

用完的記憶體是要還的

```
Node *head=NULL;
…
deleteList(head);
```

```
void deleteList(Node *n)
{
    while(n != NULL)
    {
        Node *temp = n;
        n = n->next;
        delete temp;
    }
}
```

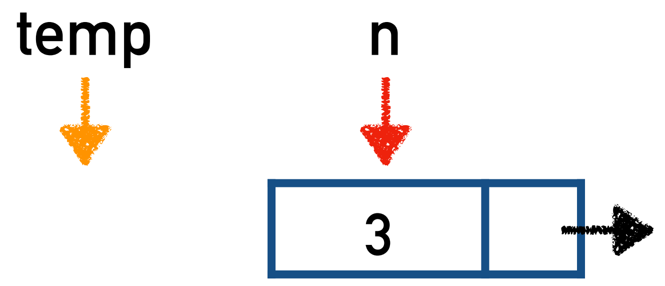


回收整個List

用完的記憶體是要還的

```
Node *head=NULL;
…
deleteList(head);
```

```
void deleteList(Node *n)
{
    while(n != NULL)
    {
        Node *temp = n;
        n = n->next;
        delete temp;
    }
}
```

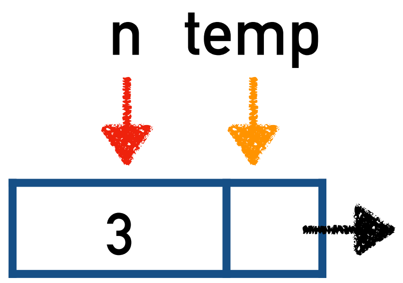


回收整個List

用完的記憶體是要還的

```
Node *head=NULL;
…
deleteList(head);
```

```
void deleteList(Node *n)
{
    while(n != NULL)
    {
        Node *temp = n;
        n = n->next;
        delete temp;
    }
}
```

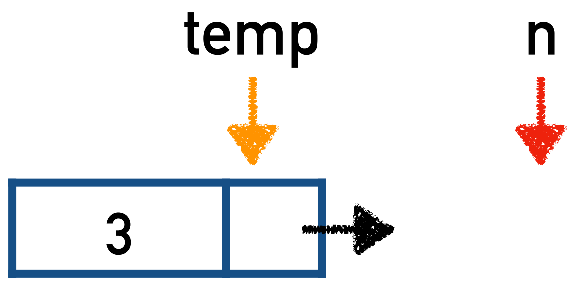


回收整個List

用完的記憶體是要還的

```
Node *head=NULL;
…
deleteList(head);
```

```
void deleteList(Node *n)
{
    while(n != NULL)
    {
        Node *temp = n;
        n = n->next;
        delete temp;
    }
}
```



回收整個List

用完的記憶體是要還的

```
Node *head=NULL;
…
deleteList(head);
```

```
void deleteList(Node *n)
{
    while(n != NULL)
    {
        Node *temp = n;
        n = n->next;
  →     delete temp;
    }
}
```

回收整個List

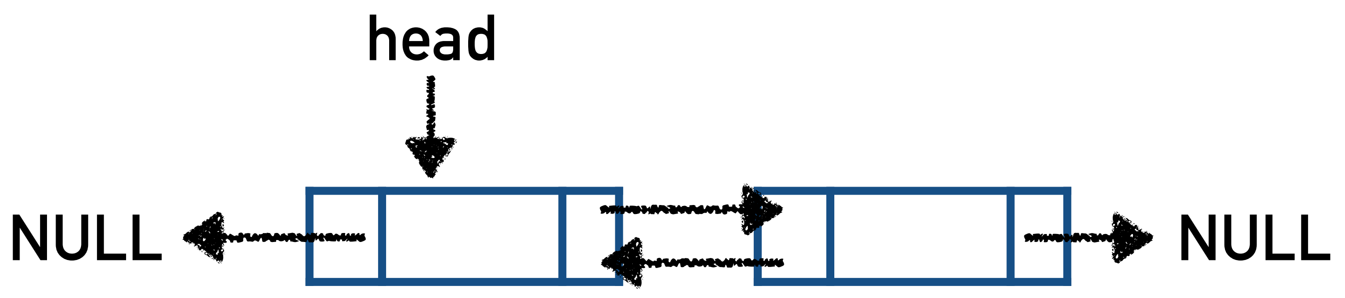
用完的記憶體是要還的

```
Node *head=NULL;
…
deleteList(head);
```

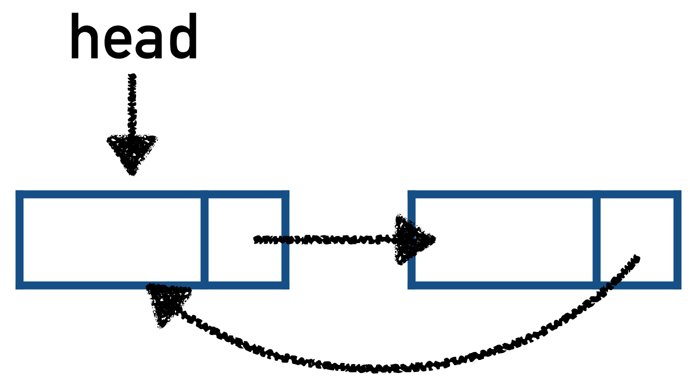
```
void deleteList(Node *n)
{
    while(n != NULL)
    {
        Node *temp = n;
        n = n->next;
        delete temp;
    }
}
```

Linked List的其他變形

- Doubly Linked List（雙向鏈結串列）
- Circular Linked List（環狀串列）



Doubly Linked List



Circular Linked List