

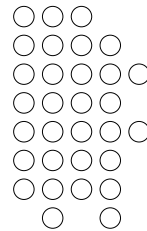
# 第十四章 動態記憶體配置與 鏈結串列

動態記憶體配置

鏈結串列

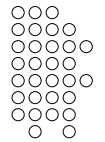
循序串列與鏈結串列的優缺點

實作鏈結串列



## 記憶體配置的方式

- 記憶體配置：
  - 靜態配置：
    - 在編譯時期（compile-time），編譯器就必須決定配置多少記憶空間給變數
    - 記憶體使用較無彈性，配置過多會浪費，配置過少會不夠使用
  - 動態配置：
    - 在程式執行時期（run-time），根據處理資料的需求，才向系統要求記憶空間
    - 可以有效地使用配置的記憶體



## 動態記憶體配置

- 程式設計者透過 malloc() 函數進行動態記憶體的配置

malloc() 的語法

指標變數 = (指標變數所指向的型態 \*) malloc(所需的記憶空間)

將 malloc() 所傳回的位址  
強制轉換成指標變數的型態

以位元組為單位

- 配置可存放3個整數的記憶空間：

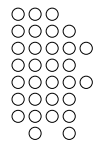
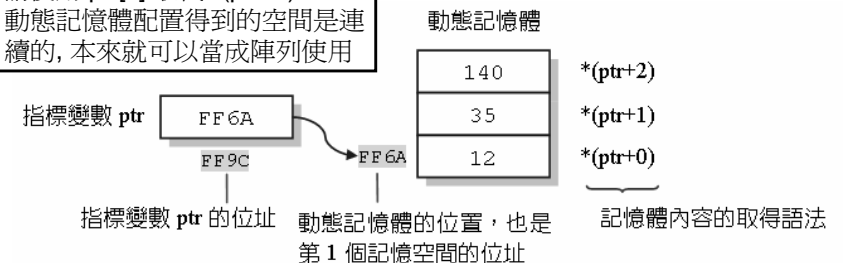
```
int *ptr; /* 宣告指向整數的指標 ptr */
ptr=(int *) malloc(12); /* 較不好的寫法 */
ptr=(int *) malloc(3*sizeof(int)); /* 較好的寫法 */
```

## 設定與取得記憶體的內容

- 第 k 個記憶空間的內容可藉由 \*(ptr+k) 來存取

```
int *ptr;
ptr=(int *) malloc(3*sizeof(int));
*ptr=12; /* 將 ptr 所指向的第 1 個記憶空間設值為 12 */
*(ptr+1)=35; /* 將第 2 個記憶空間設值為 35 */
*(ptr+2)=140; /* 將第 3 個記憶空間設值為 140 */
```

請使用 ptr[k] 取代 \*(ptr+k)  
動態記憶體配置得到的空間是連續的，本來就可以當成陣列使用





## 歸還記憶空間

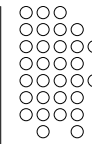
- 用 free() 函數把動態取得的記憶體歸還給系統

### free() 的語法

```
free(指標變數); /* 釋放由指標變數所指向的記憶空間 */
```

- 記憶體遺漏 (memory leakage)
  - 動態配置的記憶體沒有用 free() 歸還系統
- 記憶體區段存取失敗 (segmentation fault) 或非法記憶體存取 (illegal memory access)
  - 如果記憶空間已歸還了，卻還嘗試著去使用那塊記憶空間

5



## 動態記憶體實例一

```
01 /* prog14_1, 動態記憶體配置的範例 */ /* prog14_1 OUTPUT---
02 #include<stdio.h>
03 #include<stdlib.h> /* ptr+0=12
04 int main(void) /* ptr+1=35
05 { /* ptr+2=140
06     int *ptr,i; -----*/
07     ptr=(int *) malloc(3*sizeof(int)); /* 配置 3 個存放整數的空間 */
08
09     *ptr=12; /* 把配置之記憶空間的第 1 個位置設值為 12 */
10     *(ptr+1)=35; /* 把第 2 個位置設值為 35 */
11     *(ptr+2)=140; /* 把第 3 個位置設值為 140 */
12
13     for(i=0;i<3;i++)
14         printf("ptr+%d=%d\n",i,*(ptr+i)); /* 印出存放的值 */
15
16     free(ptr); /* 釋放由 ptr 所指向的記憶空間 */
17     system("pause");
18     return 0;
19 }
```

請使用 ptr[i] 取代 \*(ptr+i)

6



## 動態記憶體實例二 (1/2)

- 以 malloc() 配置記憶空間給結構變數

```
01 /* prog14_2, 配置記憶空間給結構變數 */
02 #include<stdio.h>
03 #include<stdlib.h>
04 int main(void)
05 {
06     int num,i;
07     struct student /* 定義結構 student */
08     {
09         char name[10];
10         int score;
11     } *ptr; /* 宣告指向結構 student 的指標 ptr */
12
13     printf("Number of student: ");
14     scanf("%d",&num);
15
16     ptr=(struct student *) malloc(num*sizeof(struct student));
17
```

7



## 動態記憶體實例二 (2/2)

```
18     for(i=0;i<num;i++)
19     {
20         fflush(stdin); /* 清空緩衝區的內容 */
21         printf("name for student %d: ",i+1);
22         gets((ptr+i)->name); /* 將鍵入的字串寫入 name 成員 */
23         printf("score for student %d: ",i+1);
24         scanf("%d",&(ptr+i)->score); /* 將鍵入的整數寫入 score 成員 */
25     }
26     for(i=0;i<num;i++)
27         printf("%s: score=%d\n",(ptr+i)->name,(ptr+i)->score);
28
29     free(ptr); /* prog14_2 OUTPUT-----
30
31     system("pause");
32     return 0;
33 }
```

請使用 ptr[i].name 取代 (ptr+i)->name  
ptr[i].score 取代 (ptr+i)->score

```
Number of student: 2
name for student 1: Jenny
score for student 1: 65
name for student 2: Teresa
score for student 2: 88
Jenny: score=65
Teresa: score=88
-----*/
```

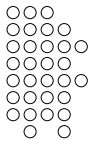
8



## 串列 (List)

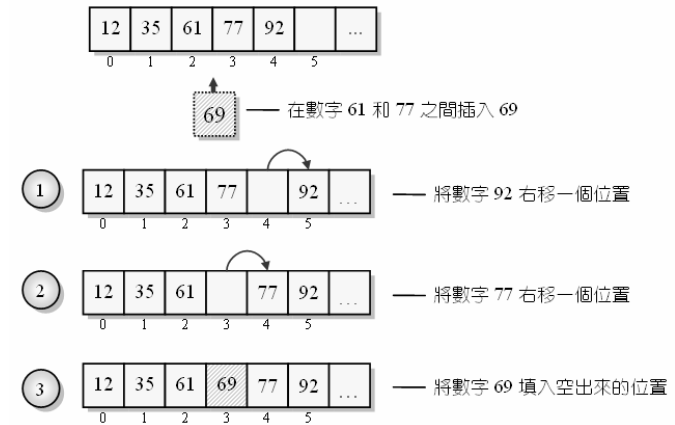
- 有次序的資料可組成一個串列 (list)
- 串列可分為循序串列與鏈結串列
  - 循序串列：存放串列的記憶體是循序的（即有先後次序）
    - 優點：存取方便 通常就叫做“陣列”了
    - 缺點：增加或刪除節點較麻煩，易造成記憶體空間不足或浪費
  - 鏈結串列：以指標將存放串列的記憶體鏈結起來
    - 優點：記憶體配置較有彈性
    - 缺點：搜尋某個元素時，可能會較為耗時

9

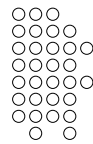


## 以陣列實作循序串列

- 以陣列實作循序串列，資料的移動會較為頻繁

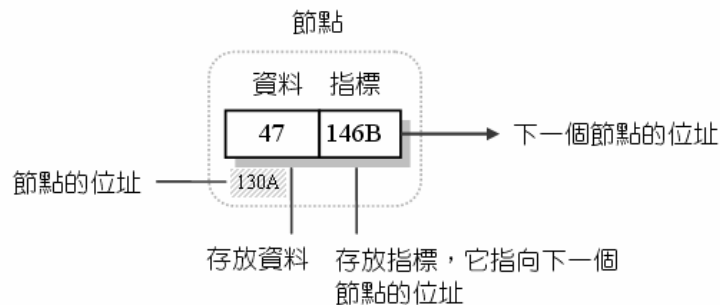


10

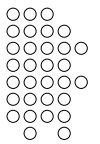


## 鏈結串列的表示法 (1/2)

- 每一節點包含兩個成員
  - 第一個成員是該節點所儲存的資料
  - 第二個成員是一個指標，它指向了下一個節點的位址

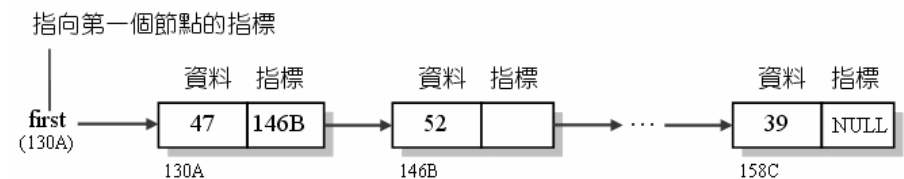


11



## 鏈結串列的表示法 (2/2)

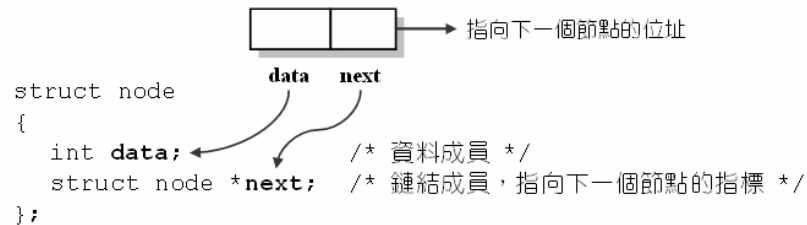
- 鏈結串列是由許多節點鏈結而成
  - 每一個節點均有一個指標指向下一個節點
  - 鏈結串列可由下圖來表示



12

## 以結構來實作鏈結串列

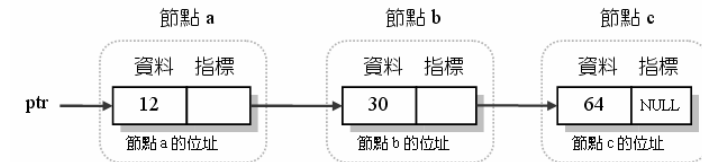
- 以結構實作鏈結串列：
  - 每一個節點有兩個成員
    - 第一個成員用來存放資料 (data)
    - 第二個成員用來存儲指向下一個節點的指標 (next)



13

## 鏈結串列的實作範例 (1/3)

- 下面的範例建立了如下圖的鏈結串列：



```

01 /* prog14_3, 建立 3 節點的鏈結串列 */
02 #include<stdio.h>
03 #include<stdlib.h>
04 struct node
05 {
06     int data; /* 資料成員 */
07     struct node *next; /* 鏈結成員，存放指向下一個節點的指標 */
08 };
09 #typedef struct node NODE; /* 將 struct node 定義成 NODE 型態 */
10
  
```

14

## 鏈結串列的實作範例 (2/3)

```

11 int main(void)
12 {
13     NODE a,b,c; /* 宣告 a,b,c 為 NODE 型態的變數 */
14     NODE *ptr=&a; /* 宣告 ptr，並将它指向節點 a */
15     a.data=12; /* 設定節點 a 的 data 成員為 12 */
16     a.next=&b; /* 將節點 a 的 next 成員指向下一個節點，即 b */
17     b.data=30;
18     b.next=&c;
19     c.data=64;
20     c.next=NULL; /* 將節點 c 的 next 成員設成 NULL */
21
22     while (ptr!=NULL) /* 當 ptr 不是 NULL 時，則執行下列敘述 */
23     {
24         printf("address=%p, ",ptr); /* 印出節點的地址 */
25         printf("data=%d, ",ptr->data); /* 印出節點的 data 成員 */
26         printf("next=%p\n",ptr->next); /* 印出下一個節點的地址 */
27         ptr=ptr->next; /* 將 ptr 指向下一個節點 */
28     }
29     system("pause");
30     return 0;
31 }
  
```

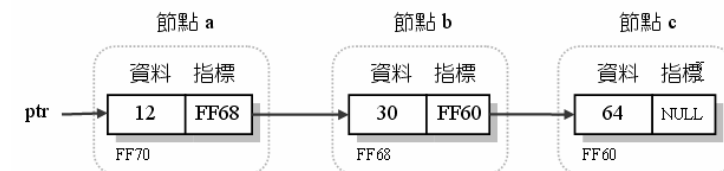
15

## 鏈結串列的實作範例 (3/3)

- prog14\_3 的執行結果

```

/* prog14_3 OUTPUT-----
address=0022FF70, data=12, next=0022FF68
address=0022FF68, data=30, next=0022FF60
address=0022FF60, data=64, next=00000000
-----*/
  
```



16

## 以動態記憶體配置鏈結串列 (1/3)

## ● 以動態記憶體配置鏈結串列的範例

```

01 /* prog14_4, 以動態記憶體配置鏈結串列 */
02 #include<stdio.h>
03 #include<stdlib.h>
04 struct node
05 {
06     int data;           /* 資料成員 */
07     struct node *next; /* 鏈結成員, 存放指向下一個節點的指標 */
08 };
09 typedef struct node NODE; /* 將 struct node 定義成 NODE 型態 */
10
11 int main(void)
12 {
13     int i,val,num;
14     NODE *first,*current,*previous; /* 建立 3 個指向 NODE 的指標 */
15     printf("Number of nodes: ");
16     scanf ("%d",&num); /* 輸入節點的個數 */

```

17

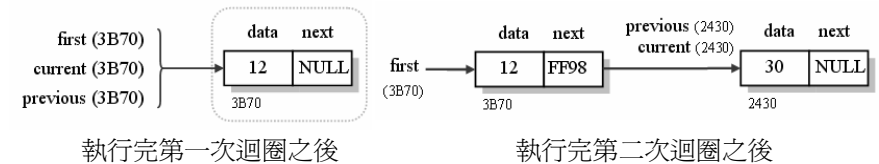


## 以動態記憶體配置鏈結串列 (2/3)

```

17     for(i=0;i<num;i++)
18     {
19         current=(NODE *) malloc(sizeof(NODE)); /* 建立新的節點 */
20         printf("Data for node %d: ",i+1);
21         scanf ("%d",&(current->data)); /* 輸入節點的 data 成員 */
22         if (i==0) /* 如果是第一個節點 */
23             first=current; /* 把指標 first 指向目前的節點 */
24         else
25             previous->next=current; /* 把前一個節點的 next 指向目前的節點 */
26         current->next=NULL; /* 把目前的節點的 next 指向 NULL */
27         previous=current; /* 把前一個節點設成目前的節點 */
28     }

```



18

## 以動態記憶體配置鏈結串列 (3/3)

```

29     current=first; /* 設定 current 為第一個節點 */
30     while (current!=NULL) /* 如果還沒有到串列末端, 則進行走訪的動作 */
31     {
32         printf("address=%p, ",current); /* 印出節點的位址 */
33         printf("data=%d, ",current->data); /* 印出節點的 data 成員 */
34         printf("next=%p\n",current->next); /* 印出節點的 next 成員 */
35         current=current->next; /* 設定 current 指向下一個節點 */
36     }
37     system("pause");
38     return 0; /* prog14_4 OUTPUT-----
39 }

```

```

Number of nodes: 3
Data for node 1: 12
Data for node 2: 30
Data for node 3: 64
address=003D3B70, data=12, next=003D2430
address=003D2430, data=30, next=003D2440
address=003D2440, data=64, next=00000000
-----*/

```

19



## 鏈結串列的基本操作 (1/4)

## ● 鏈結串列相關運算的標頭檔

```

01 /* linklist.h, 鏈結串列的標頭檔 */
02 struct node
03 {
04     int data; /* 資料成員 */
05     struct node *next; /* 鏈結成員, 存放指向下一個節點的指標 */
06 };
07 typedef struct node NODE; /* 將 struct node 定義成 NODE 型態 */
08
09 NODE *createList(int *, int); /* 串列建立函數 */
10 void printList(NODE *); /* 串列列印函數 */
11 void freeList(NODE *); /* 釋放串列記憶體空間函數 */
12 void insertNode(NODE *,int ); /* 插入節點函數 */
13 NODE *searchNode(NODE *, int ); /* 搜尋節點函數 */
14 NODE *deleteNode(NODE *, NODE *); /* 刪除節點函數 */

```

20





## 鏈結串列的基本操作 (2/4)

- 以陣列 arr 存放的資料建立鏈結串列

```

01 /* createList(), 串列建立函數 */
02 NODE *createList(int *arr, int len)
03 {
04     int i;
05     NODE *first,*current,*previous;
06     for(i=0;i<len;i++)
07     {
08         current=(NODE *) malloc(sizeof(NODE));
09         current->data=arr[i]; /* 設定節點的資料成員 */
10         if(i==0) /* 判別是否為第一個節點 */
11             first=current;
12         else
13             previous->next=current; /* 把前一個節點的 next 指向目前節點 */
14         current->next=NULL;
15         previous=current;
16     }
17     return first;
18 }

```

21



## 鏈結串列的基本操作 (3/4)

- 串列資料的列印函數：

```

01 /* printList(), 串列列印函數 */
02 void printList(NODE* first)
03 {
04     NODE* node=first; /* 將 node 指向第一個節點 */
05     if(first==NULL) /* 如果串列是空的，則印出 List is empty! */
06         printf("List is empty!\n");
07     else /* 否則走訪串列，並印出節點的 data 成員 */
08     {
09         while(node!=NULL)
10         {
11             printf("%3d", node->data);
12             node=node->next;
13         }
14         printf("\n");
15     }
16 }

```

22



## 鏈結串列的基本操作 (4/4)

- 釋放記憶體空間的函數：

```

01 /* freeList(), 釋放記憶體空間函數 */
02 void freeList(NODE* first)
03 {
04     NODE *current,*tmp;
05     current=first; /* 設定 current 指向第一個節點 */
06     while (current!=NULL)
07     {
08         tmp=current; /* 先暫存目前的節點 */
09         current=current->next; /* 將 current 指向下一個節點 */
10         free(tmp); /* 將暫存的節點釋放掉 */
11     }
12 }

```

23



## 鏈結串列的實例

- 以存放 {14,27,32,46} 的陣列建立鏈結串列：

```

01 /* prog14_5.c, 鏈結串列的建立、列印與記憶體的釋放 */
02 #include<stdio.h>
03 #include<stdlib.h>
04 #include "linklist.h" /* 含括標頭檔 linklist.h */
05
06 int main(void)
07 {
08     NODE *first;
09     int arr[]={14,27,32,46}; /* 建立陣列 arr[] */
10     first=createList(arr,4); /* 以陣列元素建立鏈結串列 */
11     printList(first); /* 印出鏈結串列的內容 */
12     freeList(first); /* 釋放記憶體空間 */
13     system("pause");
14     return 0;
15 }
16 /* 請將 createList() 函數放在此處 */
17 /* 請將 printList() 函數放在此處 */
18 /* 請將 freeList() 函數放在此處 */

```

**/\* prog14\_5 OUTPUT--**

14 27 32 46

-----\*

24

## 節點的搜尋

- 要插入一個節點，必須先知道節點的位址
- searchNode() 函數可用來找尋節點的位址

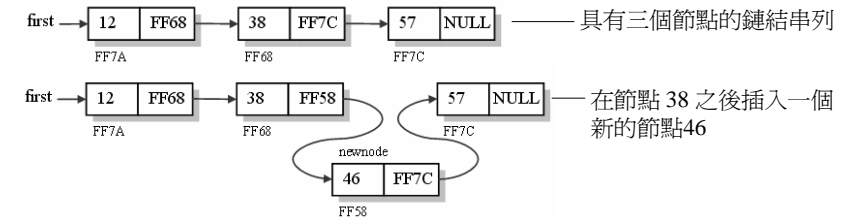
```
01 /* searchNode() 函數，可傳回第一個存放 item 之節點的位址 */
02 NODE* searchNode(NODE* first, int item)
03 {
04     NODE *node=first;
05     while (node!=NULL)
06     {
07         if (node->data==item) /* 如果 node 的 data 等於 item */
08             return node; /* 傳回 node，即該節點的位址 */
09         else
10             node=node->next; /* 否則將指標指向下一個節點 */
11     }
12     return NULL; /* 如果找不到符合的節點，則傳回 NULL */
13 }
```

25

## 節點的插入

- insertNode 可在指定節點之後插入一個新的節點

```
01 /* insertNode()，可在 node 之後加入一個新的節點 */
02 void insertNode(NODE *node,int item)
03 {
04     NODE *newnode;
05     newnode=(NODE *) malloc(sizeof(NODE)); /* 取得新節點的位址 */
06     newnode->data=item; /* 將新節點的 data 設為 item */
07     newnode->next=node->next; /* 將新節點的 next 設為原節點的 next */
08     node->next=newnode; /* 將原節點的 next 指向新節點 */
09 }
```

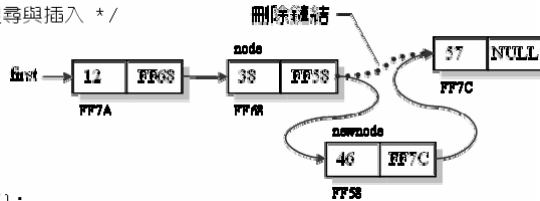


26

## 實例應用：節點搜尋與插入

- 節點搜尋與插入的範例：

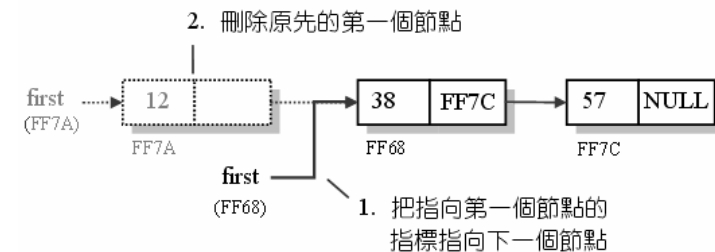
```
01 /* prog14_6.c, 節點的搜尋與插入 */
02 #include<stdio.h>
03 #include<stdlib.h>
04 #include "linklist.h"
05 int main(void)
06 {
07     NODE *first,*node;
08     int arr[]={12,38,57};
09     first=createList(arr,3); /* 建立鏈結串列 */
10     printList(first);
11
12     node=searchNode(first,38); /* 找出節點資料值為 38 的位址 */
13     insertNode(node,46); /* 將節點 46 連結在節點 38 之後 */
14     printList(first); /* 印出節點的內容 */
15     freeList(first);
16     system("pause");
17     return 0;
18 }
```



27

## 節點的刪除 (1/2)

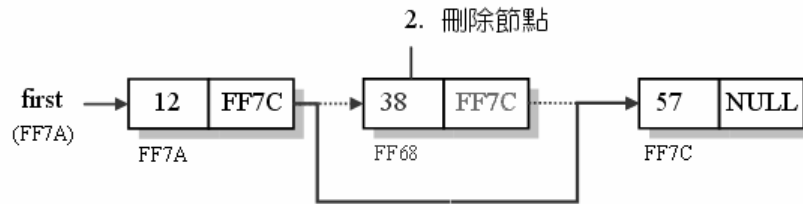
- 在刪除節點時，會有下列三種情況發生：
  - 如果串列為空串列時，則不必進行刪除的動作
  - 如果刪除的是第一個節點
    - 把指標 first 指向下一個節點，然後釋放第一個節點的記憶體空間



28

## 節點的刪除 (2/2)

- 刪除節點的第三種情況：
  - 刪除第一個節點以外的其它節點
    - 將指標指向要刪除之節點的下一個節點，然後釋放刪除之節點的記憶空間



- 把指向要刪除之節點的指標指向下一個節點

29

## 節點的刪除 — deleteNode() 函數

```

01 /* 刪掉 node，傳回刪掉 node 之後，串列第一個節點的位址 */
02 NODE* deleteNode(NODE *first, NODE *node)
03 {
04     NODE *ptr=first;
05     if(first==NULL) /* 如果串列是空的，則印出 Nothing to delete! */
06     {
07         printf("Nothing to delete!\n");
08         return NULL;
09     }
10     if(node==first) /* 如果刪除的是第一個節點 */
11         first=first->next; /* 把 first 指向下一個節點 */
12     else /* 如果刪除的是第一個節點以外的其它節點 */
13     {
14         while(ptr->next!=node) /* 找到要刪除之節點的前一個節點 */
15             ptr=ptr->next;
16         ptr->next=node->next; /* 重新設定 ptr 的 next 成員 */
17     }
18     free(node);
19     return first;
20 }

```

30

## 刪除節點的範例 (1/2)

- 刪除節點範例



```

01 /* prog14_7.c, 節點刪除的範例 */
02 #include<stdio.h>
03 #include<stdlib.h>
04 #include "linklist.h"
05 int main(void)
06 {
07     NODE *first,*node;
08     int arr[]={12,38,57};
09     first=createList(arr,3); /* 建立鏈結串列 */
10     printList(first);
11
12     node=searchNode(first,38); /* 找出節點資料值為 38 的位址 */
13     first=deleteNode(first,node); /* 將節點 38 刪除掉 */
14     printList(first); /* 印出節點的內容 */
15 }

```

/\* prog14\_7 OUTPUT-----

```

12 38 57 ----- 執行完第 10 行的結果
12 57 ----- 執行完第 14 行的結果
57 ----- 執行完第 17 行的結果
List is empty! - 執行完第 20 行的結果
-----*/

```

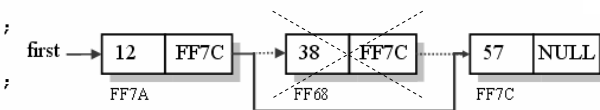
31

## 刪除節點的範例 (2/2)

```

16 first=deleteNode(first,first); /* 刪除掉第一個節點*/
17 printList(first); /* 印出節點的內容 */
18
19 first=deleteNode(first,first); /* 刪除掉第一個節點*/
20 printList(first); /* 印出節點的內容 */
21
22 freeList(first);
23
24 system("pause");
25 return 0;
26 }

```



/\* prog14\_7 OUTPUT-----

```

12 38 57 ----- 執行完第 10 行的結果
12 57 ----- 執行完第 14 行的結果
57 ----- 執行完第 17 行的結果
List is empty! - 執行完第 20 行的結果
-----*/

```

32