程式設計
(Programming)

真理大學 資訊工程系 吳汶涓老師

CH12
C資料結構

本章綱要

12-1 簡介
12-2 自我參考結構
12-3 動態記憶體配置
12-4 鏈結串列 (linked list)
12-5 堆疊 (stack)
12-6 佇列 (queue)
12-7 樹 (tree)

12.1 簡介

- 固定大小的資料結構
  - 一維、二維陣列
  - 結構 (struct)
- 動態資料結構
  - 它的大小會在執行期間變大或縮小
  - **鏈結串列 (linked list)**
  - **堆疊 (stack)**
  - **佇列 (queue)**
  - **樹 (tree)**

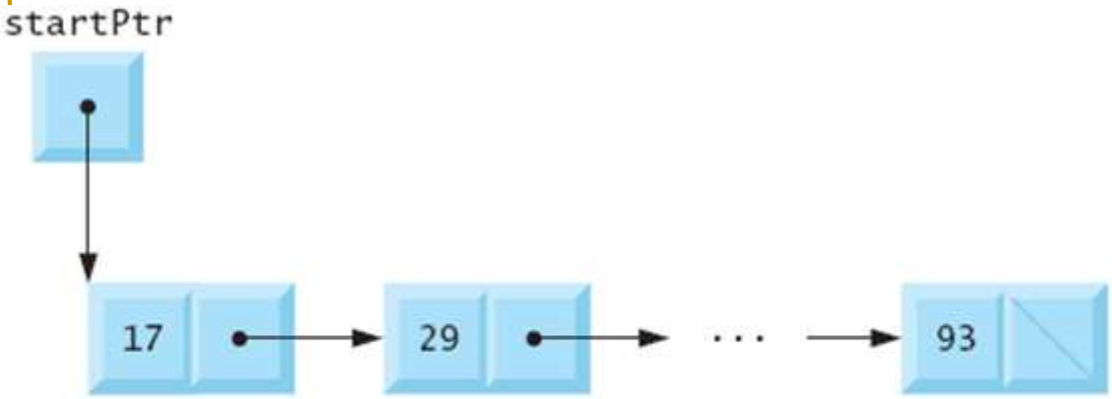
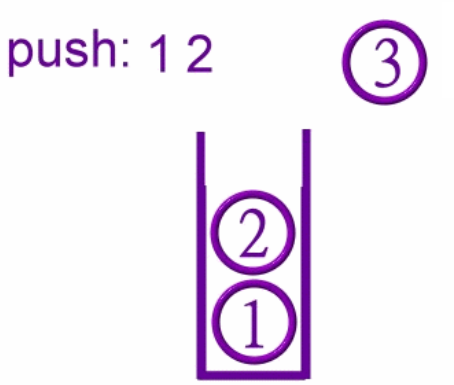
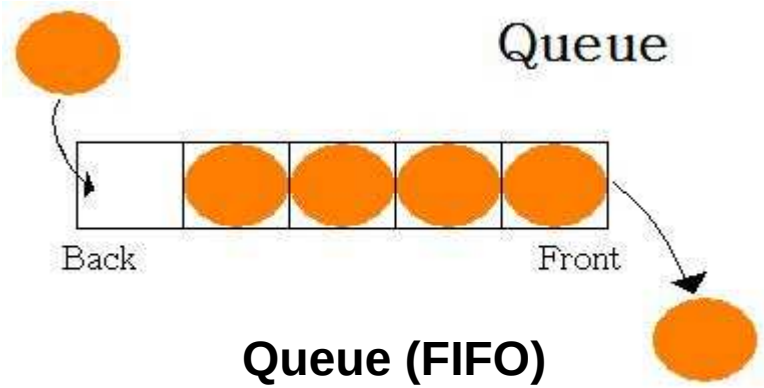


圖 12.2　鏈結串列的圖形表示法



Stack (LIFO)



Queue (FIFO)

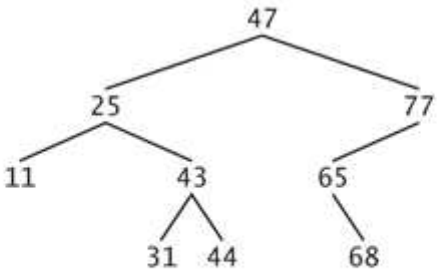


圖 12.18　二元搜尋樹 (binary search tree)

12.2 自我參考結構

- Self-referential structure
  - 含有一個指向相同型態結構的指標成員

```
struct node{
    int data;
    struct node *nextPtr;
};
```

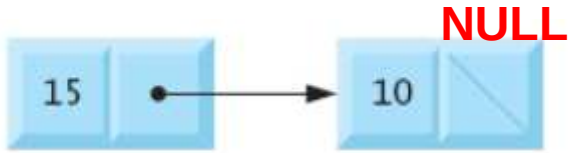


圖 12.1　鏈結在一起的自我參考結構

- 上述結構具有兩個成員：整數成員data以及指標成員nextPtr
- 其中，nextPtr指向相同型別的另一結構，此稱爲 “自我參考結構”
- 透過nextPtr相連(link)，自我參考結構可以鏈結起來，形成有用的資料結構
- 最後的指標記得要指向**NULL**，即代表資料結構的結束

常見的程式設計錯誤 12.1

沒有將串列最後一個節點的鏈結設定爲 NULL，可能導致執行時期錯誤

12.3 動態記憶體配置

- Dynamic memory allocation
  - 讓程式在執行期間可取得更多的記憶體空間來儲存新的節點
  - 能釋回不再需要的記憶體空間
  - 指令：
    - malloc – 配置所需要的記憶體空間 (若沒有可用空間，則malloc會回傳NULL指標)
    - free – 釋回不用的記憶體空間
    - sizeof – 計算某型態的大小

```
newPtr = malloc( sizeof(struct node) );
free(newPtr);
```

常見的程式設計錯誤 12.3

當不再需要使用動態配置的記憶體空間時，沒有將它釋放可能會導致系統提前耗光記憶體。這有時稱爲「記憶體外漏」

12.4 鏈結串列

- Linked list
  - 透過指標來**鏈結(link)**每個節點資料(node)
  - 可用**指標**指向串列第一個節點，用以存取整個串列
  - 串列最後一個節點指標須**設定爲NULL**，表串列結束
  - 例如:

```
insert ?17
insert ?29
insert ?23
insert ?93
insert ?0
List is 17      29      23      93      請按任意鍵繼續 .
```

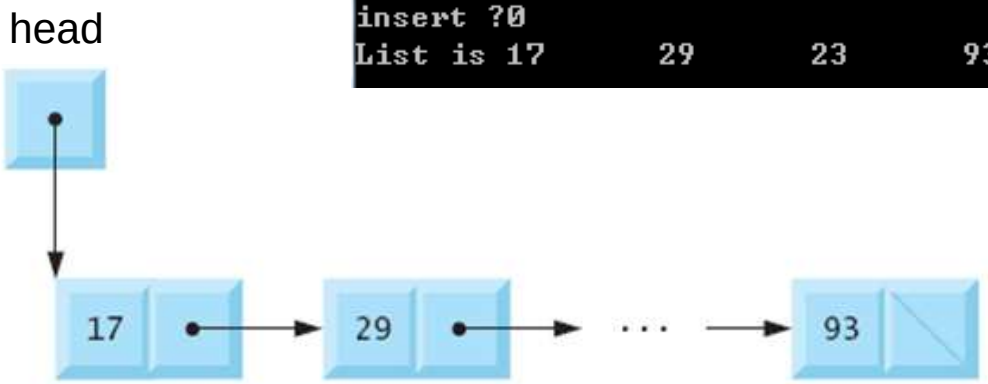
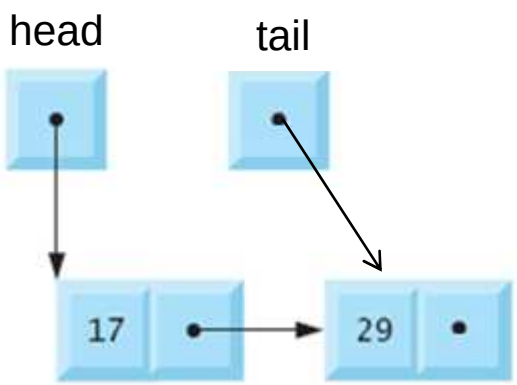


圖 12.2　鏈結串列的圖形表示法

```
4 struct node{
5     int value;
6     struct node *nextPtr;
7 };
```

- 鏈結資料新增




```
9  int main(void){
10     struct node *head=NULL;
11
12     int num=0;
13     printf("insert ?");
14     scanf("%d",&num);
15
16     struct node *tail=NULL;
17     while(num!=0){
18         struct node *newPtr;
19         newPtr=malloc(sizeof(struct node));
20         newPtr->value=num;
21         newPtr->nextPtr=NULL;
22         if(head==NULL){
23             head =newPtr;
24             tail = newPtr;
25         }
26         else{
27             tail->nextPtr=newPtr;
28             tail=newPtr;
29         }
30         printf("insert ?");
31         scanf("%d",&num);
32     }
```

- 鏈結資料顯示

```
34      //output
35      if(head==NULL)
36          printf("List is empty");
37      else{
38          printf("List is ");
39          tail=head;
40          while(tail!=NULL){
41              printf("%d \t",tail->value);
42              tail=tail->nextPtr;
43          }
44      }
```

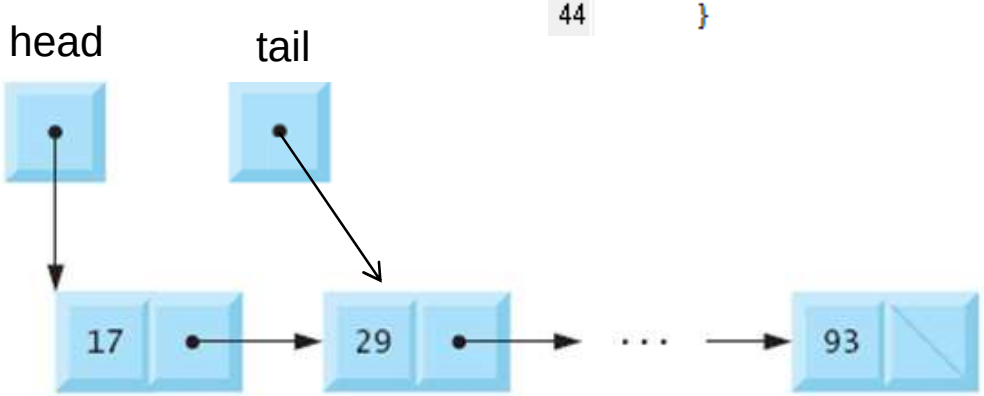
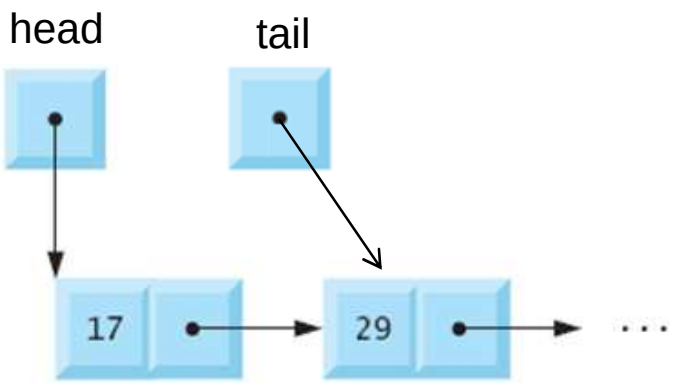


圖 12.2　鏈結串列的圖形表示法

- 鏈結資料刪除




```
45      //delete
46      int flag=0;
47      struct node *tempPtr;
48      printf("\ndelete ?");
49      scanf("%d",&num);
50      if(head==NULL)
51          printf("List is empty and data not found\n");
52      else{
53          tail=head;
54          if(tail->value==num){
55              printf("data %d found\n",tail->value);
56              flag=1;
57              head=tail->nextPtr;
58              free(tail);
59          }
60          else{
61              tempPtr=tail->nextPtr;
62              while(tempPtr->nextPtr!=NULL){
63                  if(tempPtr->value==num){
64                      printf("data %d found\n",tempPtr->value);
65                      flag=1;
66                      tail->nextPtr=tempPtr->nextPtr;
67                      free(tempPtr);
68                      break;
69                  }
70                  else{
71                      tempPtr=tempPtr->nextPtr;
72                      tail=tail->nextPtr;
73                  }
74              }
75              if(tempPtr->value==num){
76                  printf("data %d found\n",tempPtr->value);
77                  flag=1;
78                  tail->nextPtr=NULL;
79                  free(tempPtr);
80              }
81              if(flag==0)
82                  printf("data %d not found\n", num);
83          }
84
85      }
```

- 鏈結串列的優點
  - 當資料項個數無法事先得知時，可動態增加或減少鏈結串列的節點
  - 資料的新增、刪除比陣列的使用還要快速、方便

增進效能的小技巧 12.1

陣列可宣告更多的元素，但會浪費記憶體。鏈結串列提供較佳的記憶體使用率

增進效能的小技巧 12.2

在排序過的陣列中加入和刪除元素都很耗時間，因爲排在加入和刪除元素之後的所有元素都要移動

增進效能的小技巧 12.3

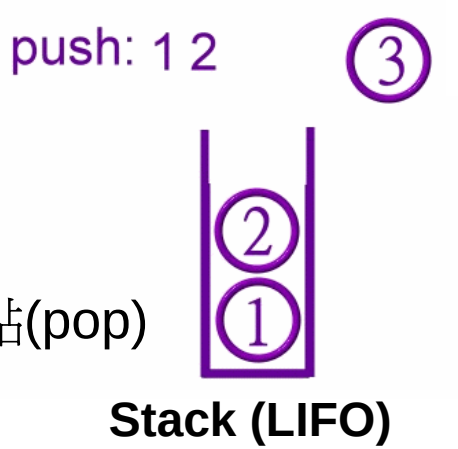
陣列的所有元素會連續存放在記憶體中。因爲任何陣列元素的位址，都可依據它相對於陣列的起始位置直接算出來，所以程式可直接存取任何陣列元素。鏈結串列無法立即存取他們的元素

練習

- 將上述的範例程式修改成選單式功能，有“新增”、“修改”、“刪除”、“顯示”，並將每個功能改成函式(副程式)呼叫。(類似課本圖12.4範例)

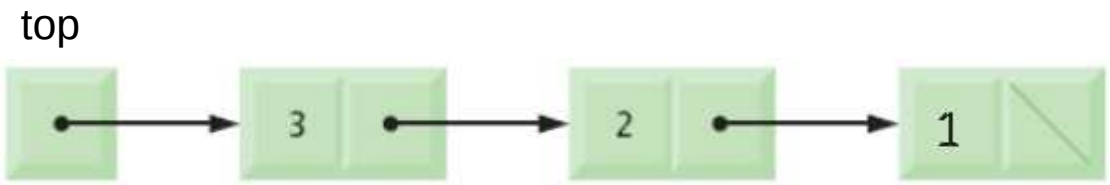
```
Enter your choice:
   1 to insert an element into the list.
   2 to delete an element from the list.
   3 to end.
? 1
Enter a character: B
The list is:
B --> NULL

? 1
Enter a character: A

The list is:
A --> B --> NULL

? 1
Enter a character: C
The list is:
A --> B --> C --> NULL

? 2
Enter character to be deleted: D
D not found.
```

12.5 堆疊



Stack (LIFO)

- Stack
  - 堆疊是一種有限制的鏈結串列
  - 只能由堆疊的頂端加入新節點(push)或刪除節點(pop)
  - 又稱爲後進先出 (last-in first-out, LIFO)
  - 堆疊會透過一個指向頂端 (top) 的指標來進行存取
  - 堆疊底部節點的鏈結 (link) 會設定爲NULL

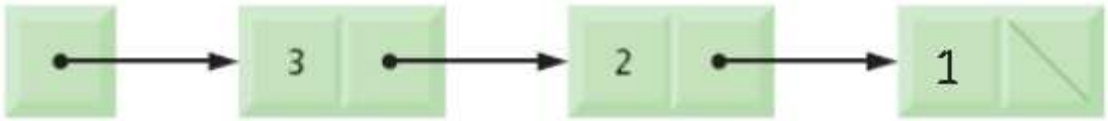


堆疊的圖形表示法

- 堆疊資料新增

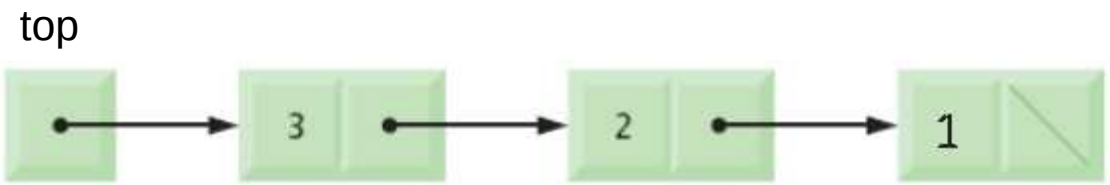
```
9  int main(void){
10     struct node *top=NULL;
11
12     int num=0;
13     printf("insert ?");
14     scanf("%d",&num);
15
16     while(num!=0){
17         struct node *newPtr;
18         newPtr=malloc(sizeof(struct node));
19         newPtr->value=num;
20         if(top==NULL){
21             newPtr->nextPtr=NULL;
22         }
23         else{
24             newPtr->nextPtr=top;
25         }
26         top=newPtr;
27         printf("insert ?");
28         scanf("%d",&num);
29     }
```

top



- 堆疊資料顯示

```
31      //output
32      if(top==NULL)
33          printf("List is empty");
34      else{
35          printf("List is ");
36          while(top->nextPtr!=NULL){
37              printf("%d \t",top->value);
38              top=top->nextPtr;
39          }
40          printf("%d \t",top->value);
41      }
```



練習

- 請將上述的堆疊結構加入刪除功能，並將功能(新增、修改、刪除、顯示)選單