程式設計 (一)

CH14. 鏈結串列

Ming-Hung Wang 王銘宏

tonymhwang@cs.ccu.edu.tw

Department of Computer Science and Information Engineering
National Chung Cheng University

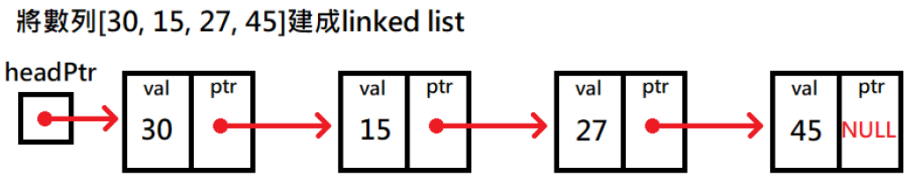
Last Semester, 2021

本章目錄

鏈結串列介紹

鏈結串列介紹

簡單來說，鏈結串列就是將一堆動態配置 的struct
使用指標 串聯成一條 list
如果 struct 是一個方形，鏈結串列會長得像下圖



鏈結串列相較於陣列的優缺點

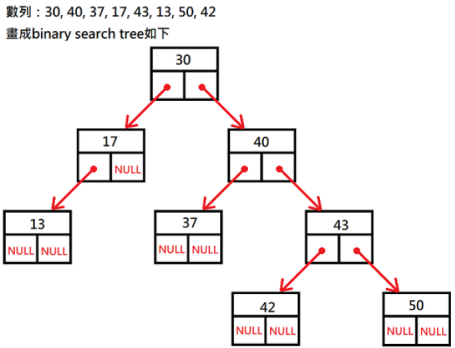
- 優點
  - 資料的插入、刪除、搬移方便快速
  - 容易實現資料結構的變形 (例如環狀鏈結串列、雙向鏈結串列等)
- 缺點
  - 資料的查找效率差、不可隨機存取
  - 在 C 語言裡，linked list 不是基本資料結構，需自行手動建立
- 其他特性
  - 鏈結串列並非使用連續記憶體

鏈結串列可實現的概念

- list (串列)
  - 可在任何位置插入或移除資料項目
- stack(堆疊)
  - 其資料項的插入和刪除操作只能夠在堆疊的頂部 (top) 進行
  - LIFO(Last In First Out)
- queue(佇列)
  - 其資料項的插入只能在佇列的尾端進行，刪除只能在佇列的頭端進行。
  - FIFO(First In First Out)

binary tree(二元樹)

常見的 tree 有 binary search tree、heap tree 等
是資料結構課程的重要章節



自我參考結構
(self-referential structure)

指一個 struct 含有 1 個或 1 個以上指向「與自身相同的 struct」的指標 (pointer) 的成員 (member)

自我參考結構

下面是一個自我參考結構的例子

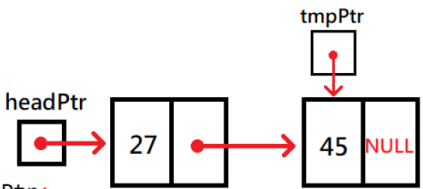
```
struct node{
    int value;
    struct node *nextPtr;
};
```

自我參考結構

試著動態配置 2 個結構，並將他們串在一起

```c
struct node{
    int value;
    struct node *nextPtr;
};

int main(){
    struct node *headPtr, *tmpPtr;
    headPtr = malloc(sizeof(struct node));
    headPtr->value = 27;
    tmpPtr = malloc(sizeof(struct node));
    tmpPtr->value = 45;
    headPtr->nextPtr = tmpPtr;
    tmpPtr->nextPtr = NULL;
}
```



使用 typedef 定義自我參考結構須注意

以下這個寫法是不被允許的

```
typedef struct{
    int value;
    Node *nextPtr; //ERROR 在這行之前尚未定義Node
} Node;
```

自我參考結構

這裡提供使用 typedef 定義自我參考結構的 2 種方法

```
//方法1
typedef struct node{ //在這裡加上struct tag
    int value;
    struct node *nextPtr;
} Node;
```

```
//方法2
typedef struct node Node; //先定義好Node是struct node
struct node{ //接著定義struct node
    int value;
    Node *nextPtr;
};
```

鏈結串列的建立/走訪

鏈結串列 (linked list) 是自我參考結構的線性集合，
每個物件稱為節點 (node)，它們透過指標鏈結 (link)
串起來，因此稱為「鏈結」串列。

鏈結串列的建立/走訪

- 鏈結串列通常由一個結構指標變數 (如下圖中的 headPtr) 來指向第一個節點，接下來的每一個節點的鏈結指標都會指向下一個節點。
- 依據慣例，串列最後一個節點的鏈結指標會設定為 NULL，代表此串列的結束。

鏈結串列的建立

建立一個鏈結串列，連續輸入整數直到輸入 0 為止

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct node{
    int value;
    struct node *nextPtr;
} Node;

int main(){
    Node *headPtr = NULL; //headptr會指向第一個node
    Node *currentPtr = NULL; /*建立串列時current會指向
                                已經建立的最後一個node*/
    int inputTmp; //使用者輸入的暫存
```

(程式碼續下一頁)

鏈結串列的建立/走訪

使用迴圈建立鏈結串列 (圖解這段程式碼)

```
14      while(scanf("%d", &inputTmp) && inputTmp != 0){
15          //動態配置新的node
16          if(headPtr == NULL){ //建立第一個node
17              headPtr = malloc(sizeof(Node));
18              currentPtr = headPtr;
19          }
20          else{ //建立第2個以後的node
21              currentPtr->nextPtr = malloc(sizeof(Node));
22              currentPtr = currentPtr->nextPtr; //遞移!
23          }
24          //指派值到新的node
25          currentPtr->value = inputTmp;
26          currentPtr->nextPtr = NULL;
27      }
```

(程式碼續下一頁)

鏈結串列的走訪

將建立的鏈結串列從頭到尾印出 (圖解這段程式碼)

```
28      currentPtr = headPtr;
29          while(currentPtr != NULL){
30                  printf("%d ", currentPtr->value);
31                  currentPtr = currentPtr->nextPtr; //遞移!
32      }
33  }
```

鏈結串列插入/刪除節點

鏈結串列的插入與刪除，跟陣列最大的不同處是不用將數據搬移。
如果我們要將陣列內的值做插入，必須將要插入的位置之後的值
全部往後移，而刪除則須全部往前移，若資料量一大，
這是一個非常大的工程。

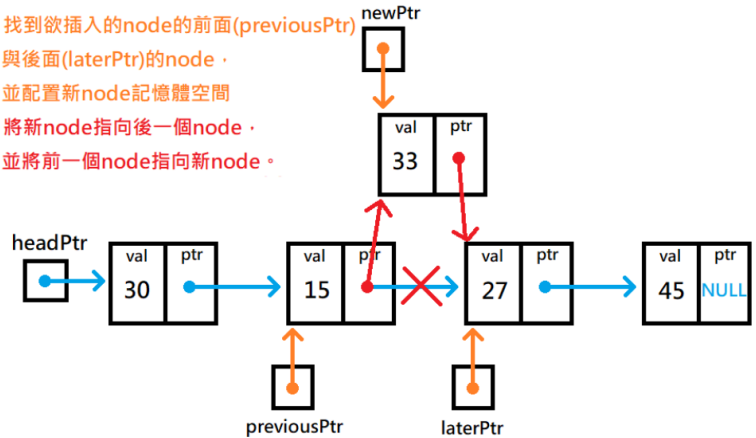
鏈結串列插入節點

一般來說，若要進行鏈結串列插入 node，就得先找到要插入的 node 的前一個與後一個 node，我們會在之後的程式碼中演示。下頁圖解找到要插入的 node 的前一個與後一個 node 後該如何插入新的 node。

鏈結串列插入/刪除節點

欲在 index 2 處插入數值 33

1. 找到欲插入的node的前面(previousPtr)
   與後面(laterPtr)的node，
   並配置新node記憶體空間
2. 將新node指向後一個node，
   並將前一個node指向新node。



插入節點範例副程式

(標頭檔、結構定義沿用前面 “建立鏈結串” 程式碼，故省略)

```
//插入val到鏈結串列*headPtrPtr的第index個，成功回傳1，失敗回傳0
int insert(Node **headPtrPtr, int index, int newVal){
    int i;
    Node *newPtr;       //要insert的node
    Node *previousPtr; //要insert的位置的上一個node
    Node *laterPtr;     //要insert的位置的下一個node
    //找到要insert的前一個node
    previousPtr = *headPtrPtr;
    for(i = 0; i < index - 1 && previousPtr->nextPtr != NULL;
        previousPtr = previousPtr->nextPtr;
    }
    if(i != index - 1 && index != 0){ //未找到index的位置
        return 0;
    }
```

(程式碼續下一頁)

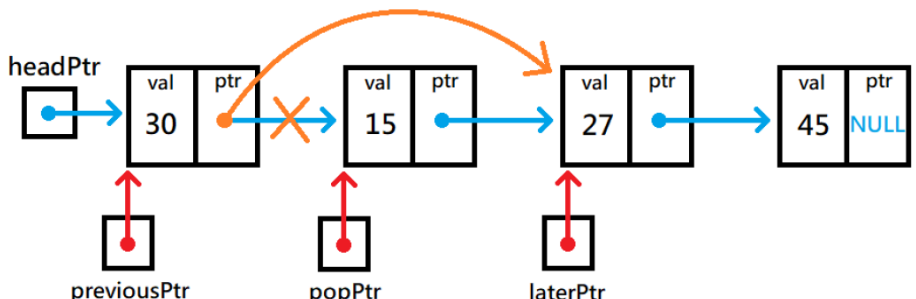
鏈結串列插入/刪除節點

```
    //配置要insert的node
    newPtr = malloc(sizeof(Node));
    if(index == 0){ //insert到最前面，原本的index 0會變index 1
        laterPtr = *headPtrPtr; //原本的index 0的指標
        *headPtrPtr = newPtr; //headPtrPtr指向新配置的node
    }
    else{
        laterPtr = previousPtr->nextPtr;
        previousPtr->nextPtr = newPtr;
    }
    newPtr->value = newVal;
    newPtr->nextPtr = laterPtr;
    return 1;
}
```

(圖解這段副程式)

鏈結串列刪除節點

基本上，刪除節點的概念與插入節點是相反動作



鏈結串列插入/刪除節點

刪除節點範例副程式

```
//將鏈結串列*headPtrPtr的第index個node刪除，回傳被刪除的值
int pop(Node **headPtrPtr, int index){
    int i, popInt;
    Node *popPtr;       //要pop的node
    Node *previousPtr; //要pop的位置的上一個node
    //找到要pop的前一個node
    previousPtr = *headPtrPtr;
    for(i = 0; i < index - 1 && previousPtr->nextPtr != NULL;
        previousPtr = previousPtr->nextPtr;
    }
    if(previousPtr == NULL || index < 0){ //未找到index的位置
        return 0;
    }
```

(程式碼續下一頁)

鏈結串列插入/刪除節點

```
    if(index == 0){ //pop最前面的，原本的index 1會變index 0
        popPtr = *headPtrPtr; //原本的index 0的指標
        *headPtrPtr = popPtr->nextPtr;
    }
    else{
        popPtr = previousPtr->nextPtr;
        if(popPtr == NULL) return 0;
        previousPtr->nextPtr = popPtr->nextPtr;
    }
    popInt = popPtr->value;
    free(popPtr);
    return popInt;
}
```

(圖解這段副程式)

参考資料：Deitel, H. M., & Deitel, P. J. (2015). C: How to program. Upper Saddle River, N.J: Prentice Hall.