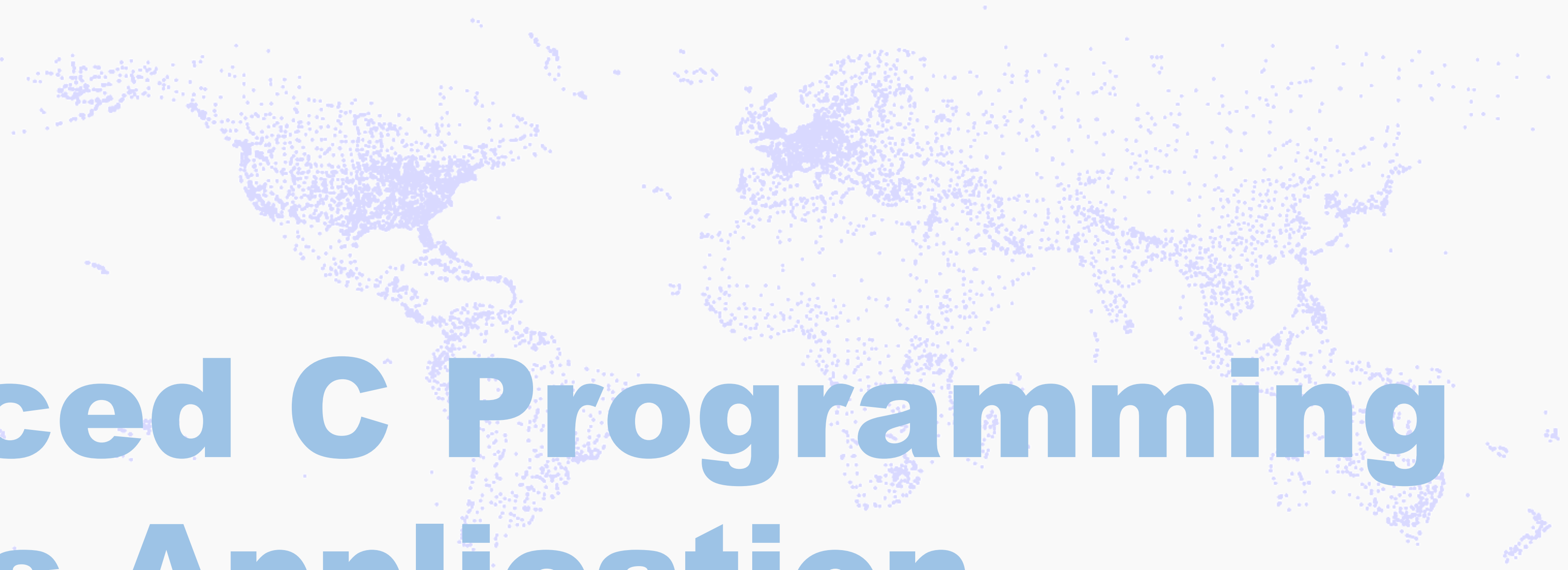


Advanced C Programming And It’s Application

Linked List Part. I

Assistant Prof. Chan, Chun-Hsiang

Department of Artificial Intelligence, Tamkang University

Feb. 9, 2022

<Outline/>

大綱

Part I.

[1] Concept

[2] Define a linked list

[3] Search

[4] Insert

[5] Delete

[6] Add with DMA

Part II.

[7] Stack

[8] Push

[9] Pop

[10] Release

[11] Insertion in Order

[12] Delete

[13] Assignments

[14] References

</Outline>

Concept of Linked List

之前學過struct，這次要學的就是如何把struct彼此串起來，就跟串貢丸一樣或是串珠一樣。如果大家還記得的話，之前我們教過struct array就可以做到類似的效果，那為甚麼還需要用到鏈結呢? 讓我們先看下方這個漫畫...

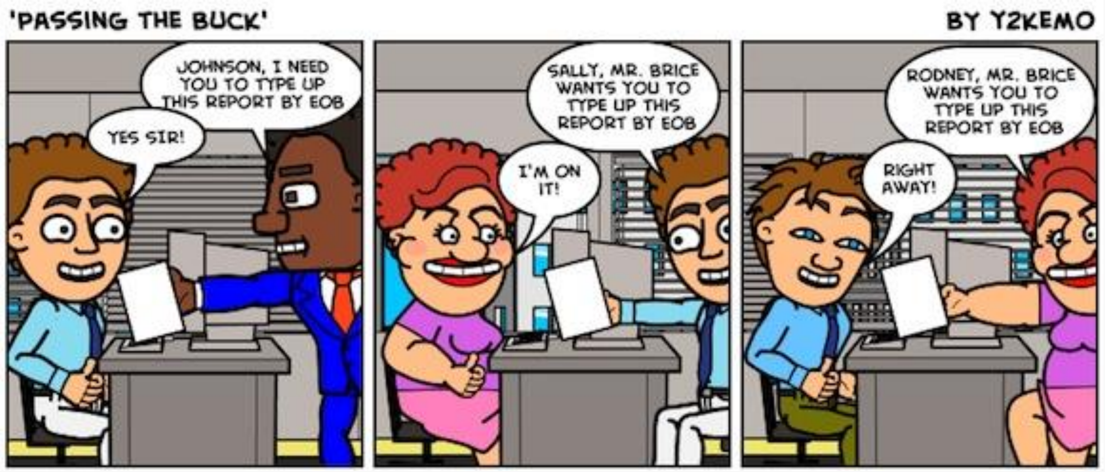

'PASSING THE BUCK'
BY Y2KEMO
JOHNSON, I NEED YOU TO TYPE UP THIS REPORT BY EOB
YES SIR!
SALLY, MR. BRICE WANTS YOU TO TYPE UP THIS REPORT BY EOB
I'M ON IT!
RODNEY, MR. BRICE WANTS YOU TO TYPE UP THIS REPORT BY EOB
RIGHT AWAY!

Photo credit: https://learn.co/lessons/linked-lists-reading

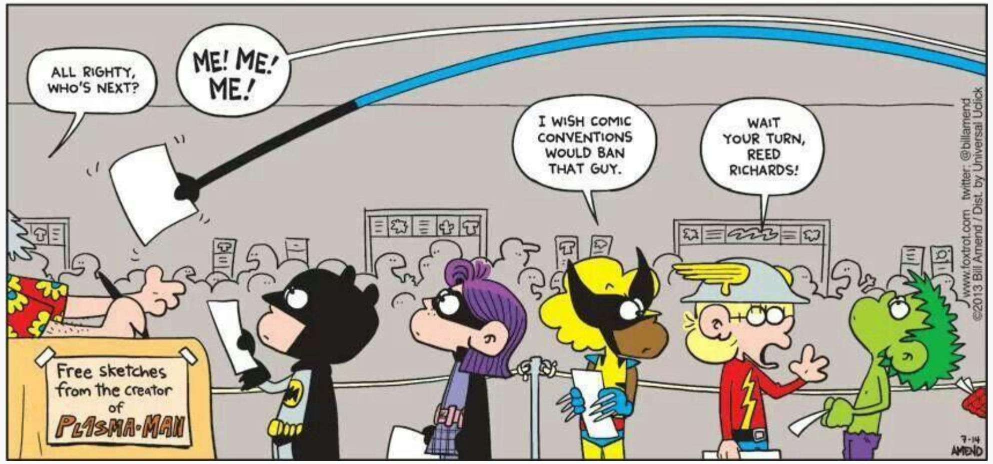

ALL RIGHTY, WHO'S NEXT?
ME! ME! ME!
I WISH COMIC CONVENTIONS WOULD BAN THAT GUY.
WAIT YOUR TURN, REED RICHARDS!
Free sketches from the creator of PLASMA-MAN
www.foxtrot.com twitter: @billamend
©2013 Bill Amend / Dist. by Universal Uclick
7-14 AMEND

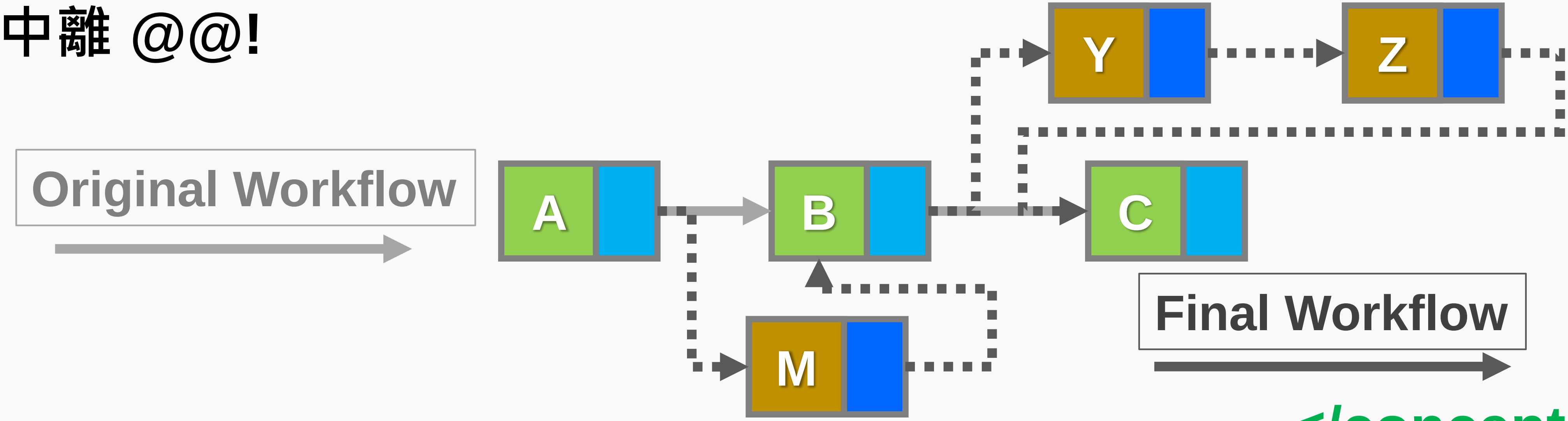
Photo credit: https://medium.com/@luc.highwalker/skip-the-nodes-dcb2fb542aa0

Concept of Linked List

正常的情況就像是我們上面左邊看到的漫畫一樣，問題是現實生活中就會像是上面右邊的一樣。如果以一個專案為例，我們永遠不會知道到底需要多少人才能某一件任務。有可會出現以下兩個情況(但不只...):

(1) 做到一半突然有不會的地方，需要請求支援。

(2) 有人中離 @@!



Concept of Linked List

所以這個時候我們就要用到鏈結(linked list)，通常在資料結構與演算法的課程會再仔細介紹他的精神與應用(很多很多...)。

一般來說，鏈結分為單向(single)與雙向鏈結(doubly linked list)兩種。又可以用資料I/O順序分為: 後進先出(Last In, First Out; LIFO) 、先進先出(First In, First Out; FIFO)。

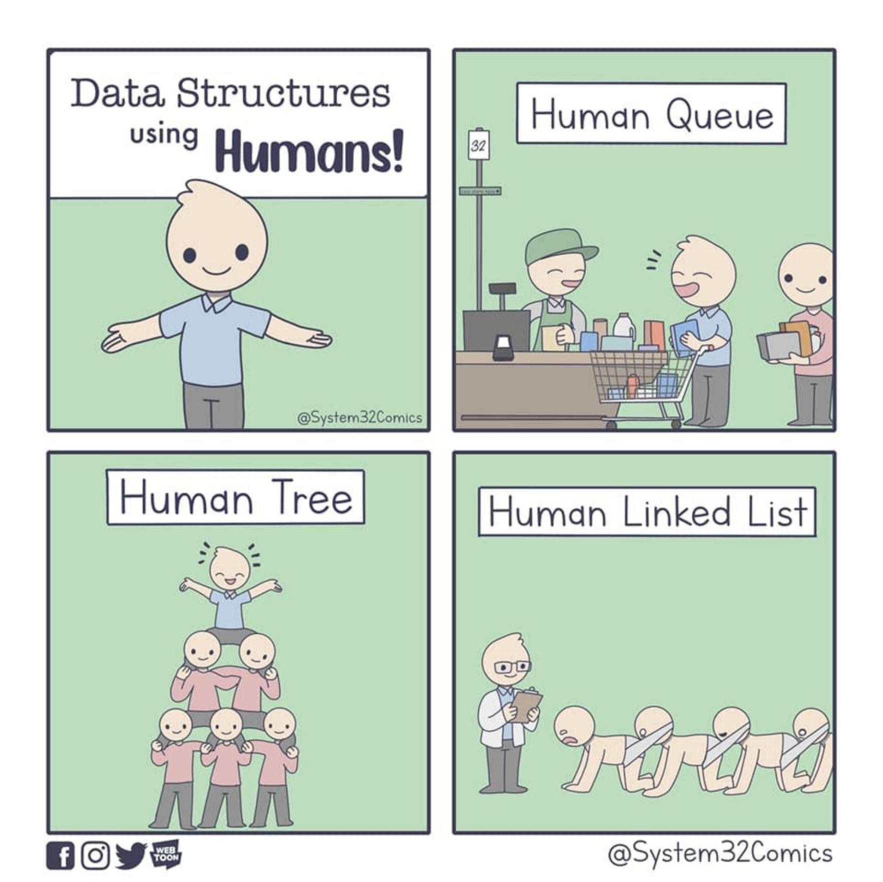


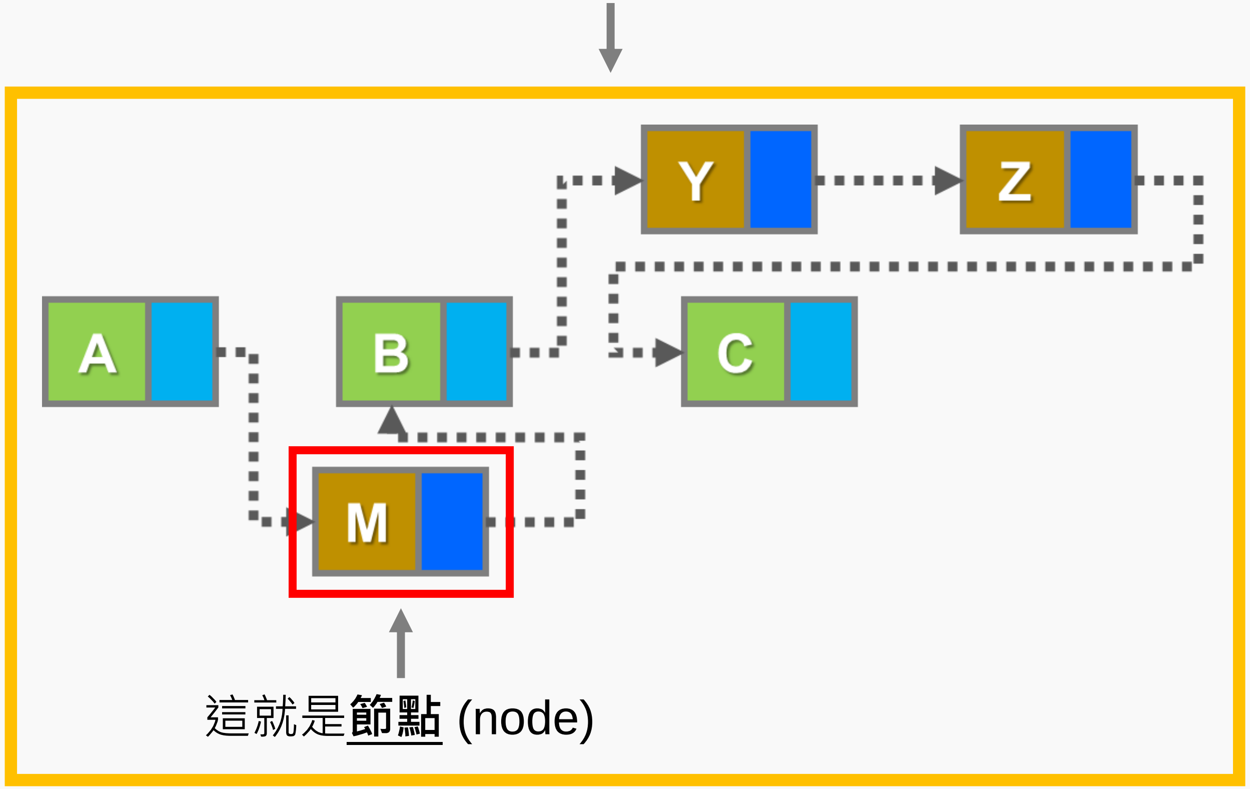
Photo credit: https://www.facebook.com/System32ComicsAdvanced/

Define a Linked List

在學習怎麼建立linked list之前，我們要先知道如何建立節點(node)，也就是linked list上每一個元素。

像是右圖，在這個linked list中，我們有六個nodes。

這就是**鏈結** (linked list)



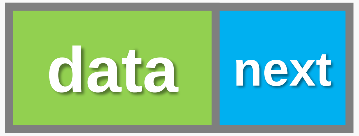
這就是**節點** (node)

Define a Node

```
#include <stdio.h>
#include <string.h>
typedef struct flight{
        char flightNo[10];
        char airline[30];
        char origin[4], destination[4];
        int frequency, sitCapacity;
        double duration;
} Flight;
typedef struct node{
        Flight data;
        struct node *next;
} Node;
```

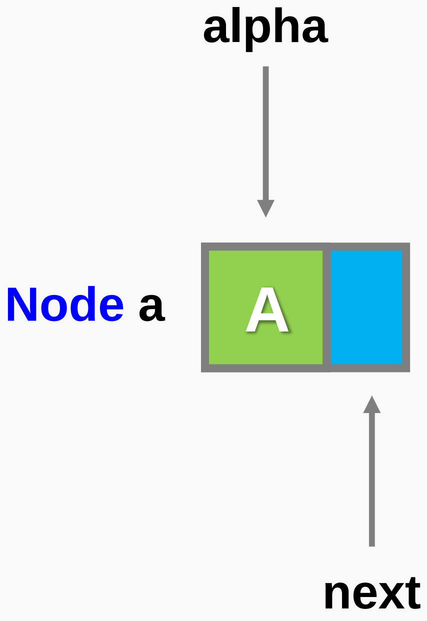
```
int main(){
        /*Ex 14-1: define a node of linked list */
        return 0;
}
```

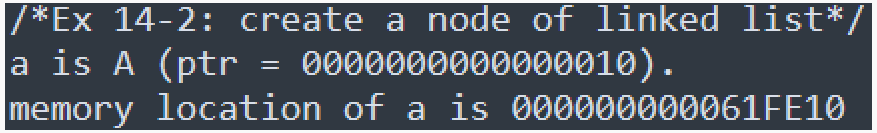
A node …



Create a Simple Node

```
#include <stdio.h>
typedef struct node{
        char alpha;
        struct node *next;
} Node;
int main(){
        /*Ex 14-2: create a node of linked list */
        printf("/*Ex 14-2: create a node of linked list*/\n");
        Node a;
        a.alpha = 'A';
        printf("a is %c (ptr = %p).\n", a.alpha, a.next);
        printf("memory location of a is %p\n", &a);
}
```

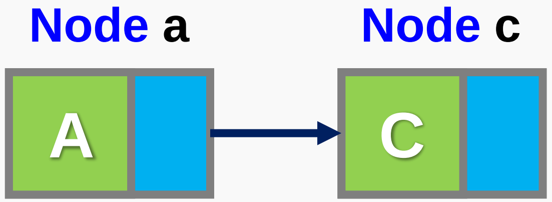


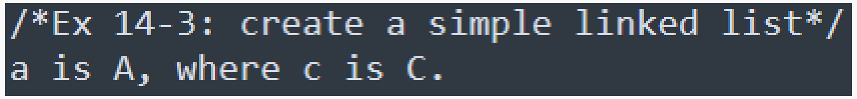


```
/*Ex 14-2: create a node of linked list*/
a is A (ptr = 0000000000000010).
memory location of a is 000000000061FE10
```

Create a Simple Linked List

```
#include <stdio.h>
typedef struct node{…SKIP…} Node;
int main(){
        /*Ex 14-3: create a simple linked list*/
        printf("/*Ex 14-3: create a simple linked list*/\n");
        Node a, c;
        a.alpha = 'A';
        a.next = &c;
        // c.alpha = 'C'
        a.next -> alpha = 'C';
        printf("a is %c, where c is %c.\n", a.alpha, c.alpha);
}
```

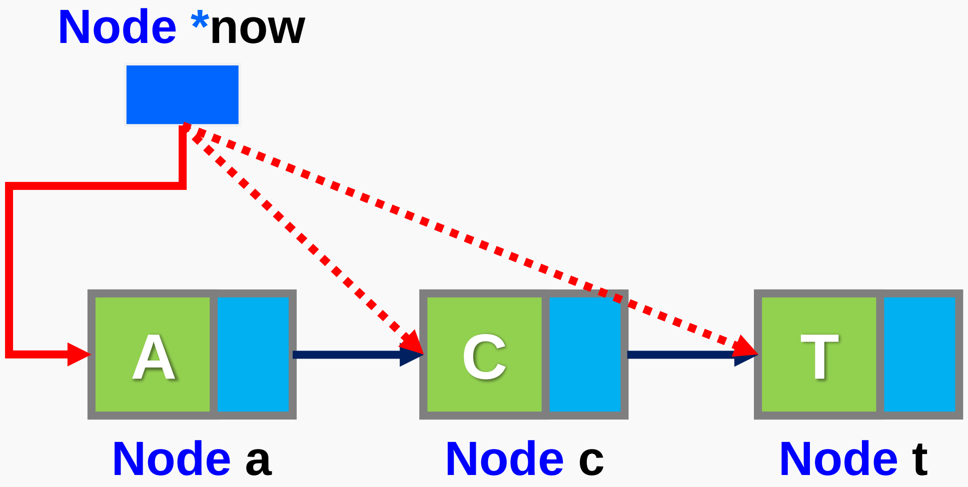


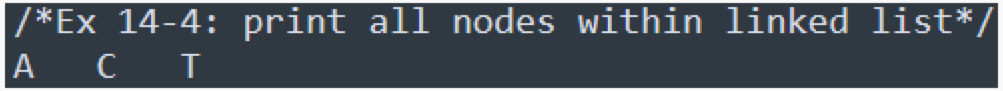


```
/*Ex 14-3: create a simple linked list*/
a is A, where c is C.
```

Print the data of a Node

```
#include <stdio.h>
typedef struct node{…SKIP…} Node;
int main(){
        /*Ex 14-4: print all nodes within linked list*/
        printf("/*Ex 14-4: print all nodes within linked list*/\n");
        Node a, c, t;
        a.alpha = 'A';
        a.next = &c;
        a.next -> alpha = 'C';
        a.next -> next = &t;
        a.next -> next -> alpha = 'T'; // t.alpha = 'T';
        a.next -> next -> next = 0;
        Node *now = &a;
        while(now){ // now != 0
                printf("%c\t", now->alpha);
                now  = now -> next;
        }
        putchar('\n');
}
```

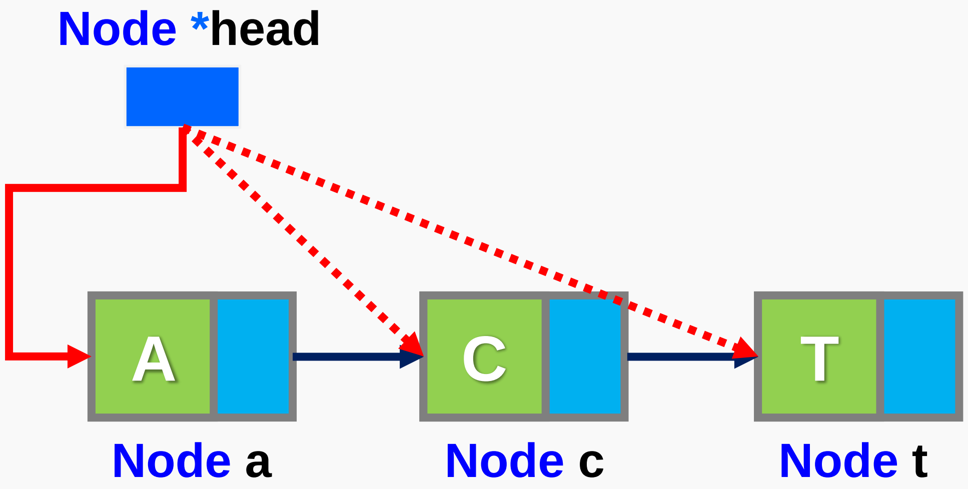




```
/*Ex 14-4: print all nodes within linked list*/
A	C	T
```

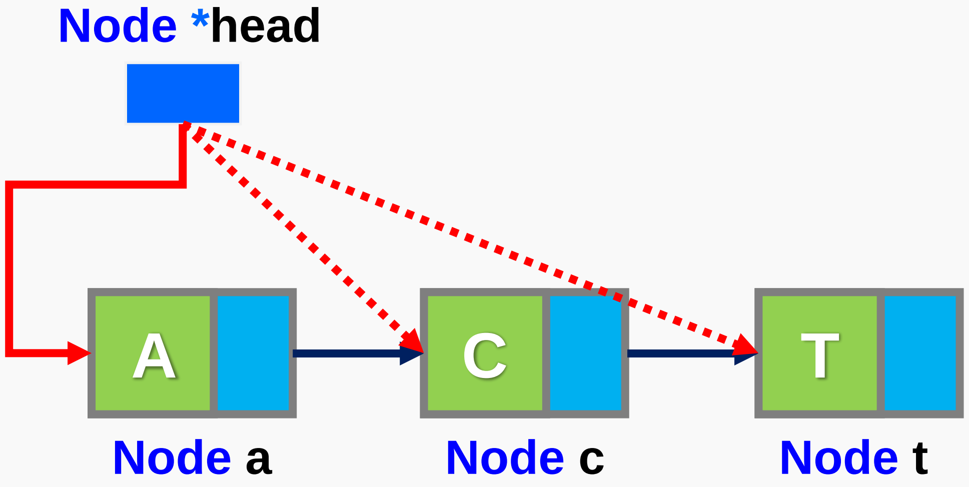
Print all Nodes Information by Func

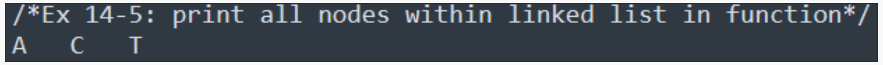
```
#include <stdio.h>
#include <string.h>

typedef struct node{…SKIP…} Node;
void printNode(const Node *head){
        while(head){ // head != 0
                printf("%c\t", head->alpha);
                head  = head -> next;
        }
        putchar('\n');
}
…
```



Print all Nodes Information by Func

```
…
int main(){
        /*Ex 14-5: print all nodes within linked list in function*/
        printf("/*Ex 14-5: print all nodes within linked list in function*/\n");
        Node a, c, t;
        a.alpha = 'A';
        a.next = &c;
        a.next -> alpha = 'C';
        a.next -> next = &t;
        a.next -> next -> alpha = 'T';
        a.next -> next -> next = 0;
        printNode(&a);
}
```



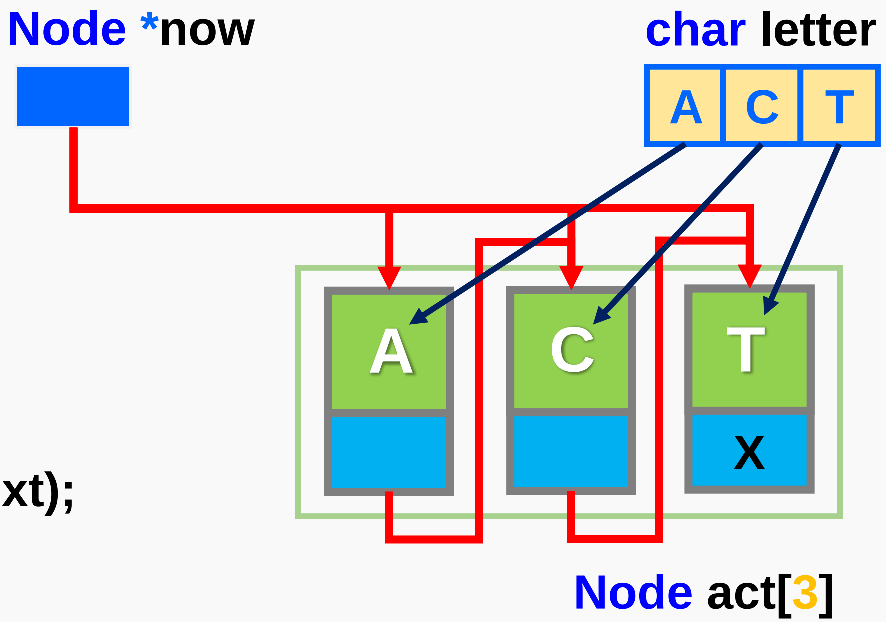


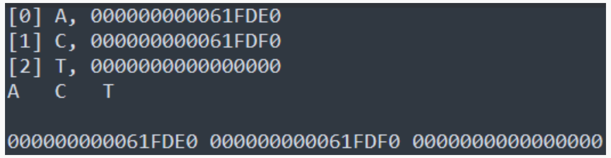
```
/*Ex 14-5: print all nodes within linked list in function*/
A   C   T
```

```
#include <stdio.h>
#include <string.h>
typedef struct node{…SKIP…} Node;
void printNode(const Node *head){…SKIP…}
```

Build Linked List by Loop

```
int main(){
        /*Ex 14-6: build linked list by loop*/
        printf("/*Ex 14-6: build linked list by loop*/\n");
        int i; char letter[4] = {'A','C','T'};
        Node act[3];
        Node *now = &act[0];
        for (i=0; i<3; i++){
                now->alpha = letter[i];
                if (i==2){
                        now->next = 0;
                }else{
                        now->next = &act[i+1];}
                printf("[%d] %c, %p\n", i, now->alpha, now->next);
                now = now -> next;
        }
        printNode(&act[0]);
        putchar('\n');
        printf("%p %p %p\n", act[0].next, act[1].next, act[2].next);}
```





```
[0] A, 000000000061FDE0
[1] C, 000000000061FDF0
[2] T, 0000000000000000
A   C   T

000000000061FDE0 000000000061FDF0 0000000000000000
```

Build Linked List by Loop in Func

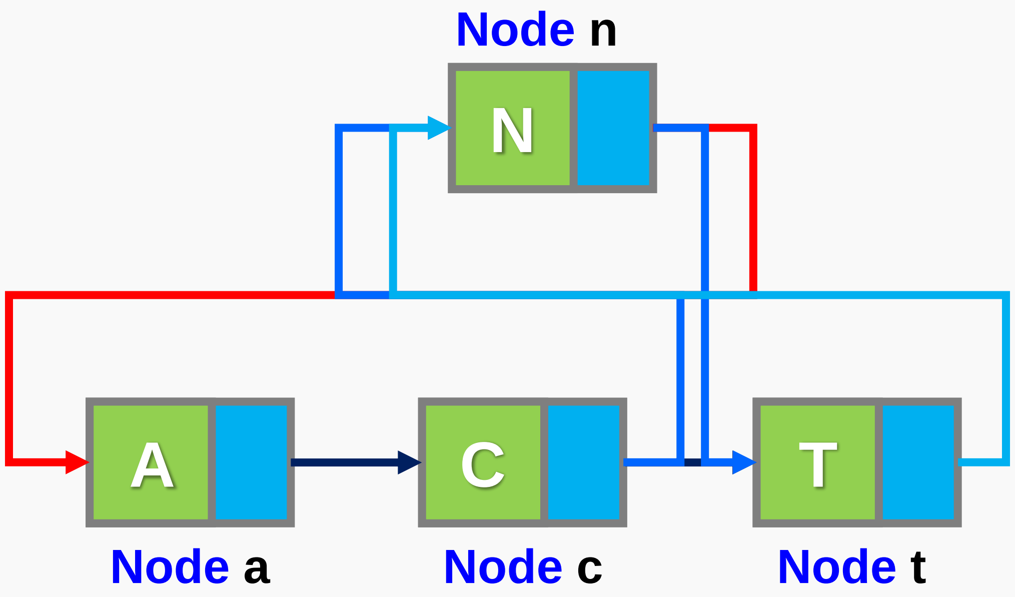
Lab 14-1:

請用上一個範例(EX14-6)，宣告一個函數**bulitLLByLoop(字元陣列、節點陣列)**，再利用for loop or while loop將每個node串起來，並且把字元放入其中。

Search, Insert, and Delete in a Linked List

講了那麼多，都還沒開始提到如何插入與刪除節點。
這些問題大概可以分為三個階段:

(1) 搜尋指定的節點 ((尋找插入點
(2) 插入指定的節點
(3) 刪除指定的節點

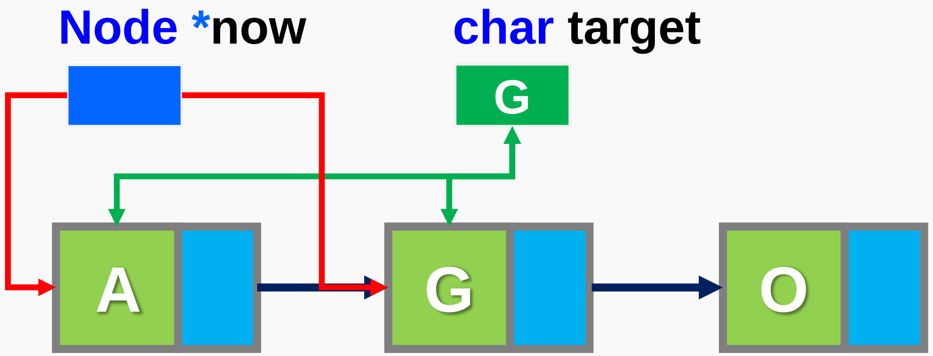


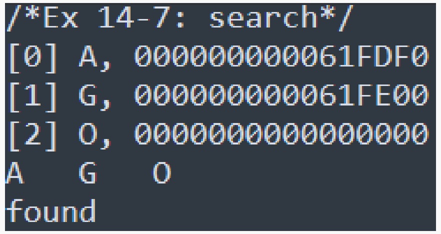
<search in LL/>

Search in a Linked List

```
#include <stdio.h>
#include <string.h>
typedef struct node{…SKIP…} Node;
void printNode(const Node *head){…SKIP…}
void bulitLLByLoop(const char letter[], Node act[]){…SKIP…}
```

```
int main(){
        /*Ex 14-7: search*/
        printf("/*Ex 14-7: search*/\n");
        // build a linked list
        char letter[4] = {'A','G','O'};
        char target = 'G';
        Node act[3], *now = &act[0];
        bulitLLByLoop(letter, act);
        printNode(&act[0]);
        // search position
        while(now){
                if(now->alpha == target){
                        printf("found\n");
                        break;
                }
                now = now -> next;}
        if(now == 0){
                        printf("cannot find\n");
                }
}
```





```
/*Ex 14-7: search*/
[0] A, 000000000061FDF0
[1] G, 000000000061FE00
[2] O, 0000000000000000
A    G    O
found
```

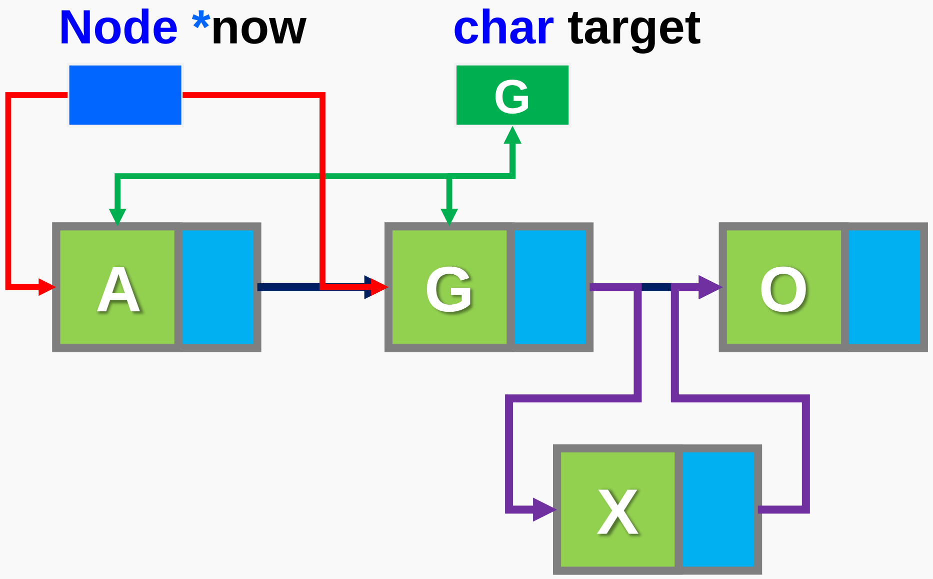
</search in LL>

<insert in LL/>

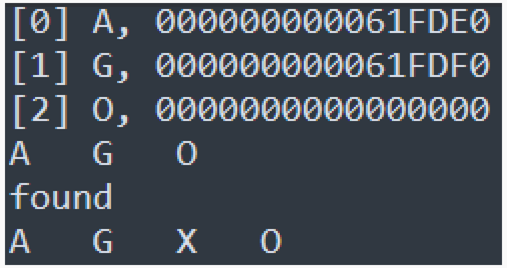
Insert in a Linked List

```
#include <stdio.h>
#include <string.h>
typedef struct node{…SKIP…} Node;
void printNode(const Node *head){…SKIP…}
void bulitLLByLoop(const char letter[], Node act[]){…SKIP…}
```

```
int main(){
        /*Ex 14-8: insert*/
        printf("/*Ex 14-8: insert*/\n");
        // build a linked list
        /*…SKIP…*/
```



```
// search position for insertion
while(now){
        if(now->alpha == target){
                printf("found\n");
                // copy the memory location
                Node *loc = now->next;
                x.alpha = letter4insert;
                x.next = loc;
                // reconnect to the original linked list
                now->next = &x;
                break;
        }
        now = now -> next;
}
if(now == 0){
        printf("cannot find\n");
}
printNode(&act[0]);}
```



```
[0] A, 000000000061FDE0
[1] G, 000000000061FDF0
[2] O, 0000000000000000
A   G   O
found
A   G   X   O
```

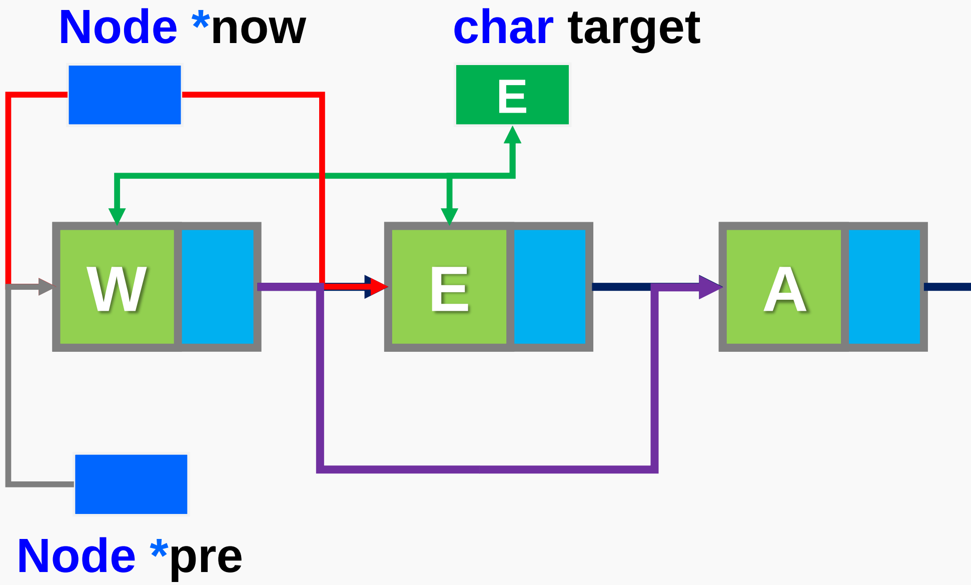
</insert in LL>

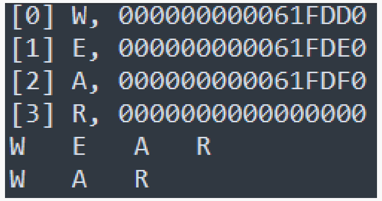
<delete in LL/>

```
#include <stdio.h>
#include <string.h>
typedef struct node{…SKIP…} Node;
void printNode(const Node *head){…SKIP…}
void bulitLLByLoop(const char letter[], Node act[]){…SKIP…}
```

Delete in a Linked List

```
// build a linked list
char letter[5] = {'W','E','A','R'};
char target = 'E';
Node act[4], *now = &act[0], *pre = &act[0], x;
bulitLLByLoop(letter, act);
printNode(&act[0]);
```





```
[0] W, 000000000061FDD0
[1] E, 000000000061FDE0
[2] A, 000000000061FDF0
[3] R, 0000000000000000
W    E    A    R
W    A    R
```

```
int main(){
        /*Ex 14-9: delete*/
        printf("/*Ex 14-9: delete*/\n");
        // build a linked list
        /*…SKIP…*/
        // search position for deletion
        while(now){
                if(now->alpha == target){
                        pre->next = now->next;
                        break;
                }
                pre = now;
                now = now -> next;
        }
        if(now == 0){
                printf("cannot find\n");
        }
        printNode(&act[0]);}
```

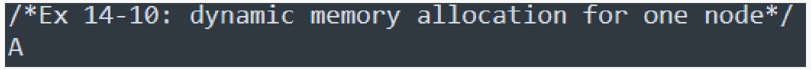
</delete in LL>

<add with DMA/>

Add with Dynamic Memory Allocation

除了可以做新增刪除節點的事情之後，我們希望可以在配置記憶體空間更有效率，這時候就會提到我們之前所學的動態記憶體配置。

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
typedef struct node{…SKIP…} Node;
void printNode(const Node *head){…SKIP…}
int main(){
          /*Ex 14-10: dynamic memory allocation for one node*/
          printf("/*Ex 14-10: dynamic memory allocation for one node*/\n");
          Node *head = 0, *now = 0;
          // declare a memory space for a node by DMA
          now = (Node*) malloc (sizeof(Node));
          now->alpha = 'A';
          now->next = 0;
          // add to a linked list
          head = now;
          printNode(head);
          // free memory space
          free(head);}
```



```
/*Ex 14-10: dynamic memory allocation for one node*/
A
```

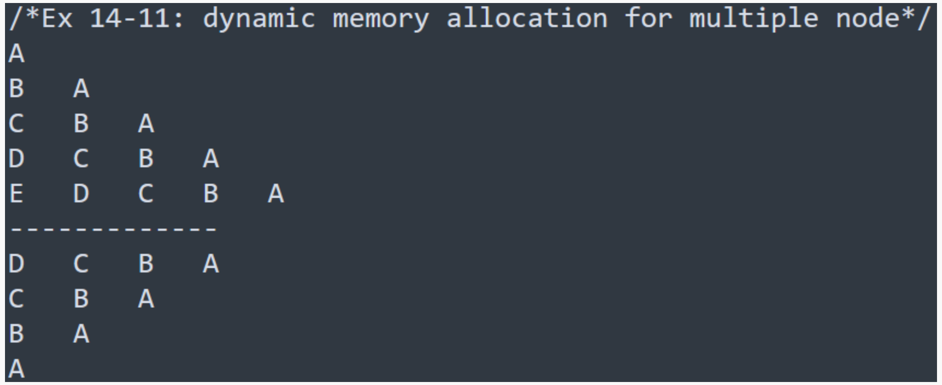
</add with DMA>

<add with DMA/>

Add with Dynamic Memory Allocation

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
typedef struct node{…SKIP…} Node;
void printNode(const Node *head){…SKIP…}
int main(){
        /*Ex 14-11: dynamic memory allocation for multiple node*/
        printf("/*Ex 14-11: dynamic memory allocation for multiple node*/\n");
        int i; Node *head = 0, *now = 0;
        for (i=0; i<5; i++){
                // declare a memory space for a node by DMA
                now = (Node*) malloc (sizeof(Node));
                now->alpha = 'A'+i;
                now->next = 0;
                // add to a linked list
                now->next = head;
                head = now;
                printNode(head);}
```

```
// free memory space
while(head){
        Node *del = head;
        head = head->next;
        printNode(head);
        free(del);
}
printNode(head);}
```



```
/*Ex 14-11: dynamic memory allocation for multiple node*/
A
B   A
C   B   A
D   C   B   A
E   D   C   B   A
------------
D   C   B   A
C   B   A
B   A
A
```

</add with DMA>

參考資料

1. 堆疊(stack) 資料結構
2. Data Structure - Doubly Linked List
3. [資料結構] 雙向鏈結串列教學[1]: 新增與印出
4. Queue: Intro(簡介)，並以Linked list實作
5. 以連結串列 (Linked List) 為基礎的佇列 (Queue)
6. Stack Data Structure (Introduction and Program)
7. C 語言：鏈結串列(Linked List)的建立與刪除
8. [資料結構]Stack — 堆疊和Queue — 佇列
9. Linked List: 新增資料、刪除資料、反轉
10. 蔣宗哲教授講義