程式設計

第7章 結構
Struct

蘇維宗(Wei-Tsung Su)
suwt@au.edu.tw
564D

目標

認識結構

結構與函式

結構與陣列

結構與指標

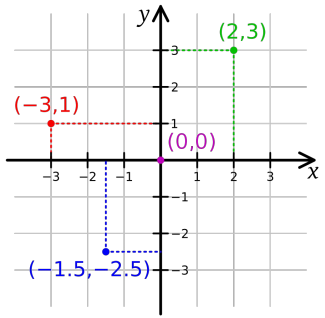
認識結構

Understanding Struct

為何需要使用結構?

現實生活的應用不會只有整數、實數、字元等基本型別。例如,

- 二維座標上的點包含
  - x值, y值



- 學生包含
  - 學號,姓名,國,英,數成績

結構的定義與使用

C語言中的**結構(struct)**就是用來定義現實生活中會使用到的衍生型別。

定義**二維座標點**衍生型別

```
1.   struct point {
2.     float x;
3.     float y;
4.   };
```

使用**二維座標點**衍生型別

```
5.   int main() {
6.     struct point src = {0, 0};
7.     struct point dst;
8.     dst.x = 4;
9.     dst.y = 3;
10.    printf("src:(%.2f,%.2f)\n", src.x, src.y);
11.    printf("dst:(%.2f,%.2f)\n", dst.x, dst.y);
12.  }
```

記憶體中的結構

變數名稱	記憶體位址	資料
src src.x	1008	0
src.y	1012	0

定義二維座標點衍生型別

```
1.  struct point {
2.    float x;
3.    float y;
4.  };
```

使用二維座標點衍生型別

```
5.  int main() {
6.    struct point src = {0, 0};
7.    struct point dst;
8.    dst.x = 4;
9.    dst.y = 3;
10.   printf("src:(%.2f,%.2f)\n", src.x, src.y);
11.   printf("dst:(%.2f,%.2f)\n", dst.x, dst.y);
12. }
```

記憶體中的結構

變數名稱	記憶體位址	資料
dst dst.x	1000	?
dst.y	1004	?
src src.x	1008	0
src.y	1012	0

定義二維座標點衍生型別

```
1.  struct point {
2.    float x;
3.    float y;
4.  };
```

使用二維座標點衍生型別

```
5.  int main() {
6.    struct point src = {0, 0};
7.    struct point dst;
8.    dst.x = 4;
9.    dst.y = 3;
10.   printf("src = (%.2f,%.2f)\n", src.x, src.y);
11.   printf("dst = (%.2f,%.2f)\n", dst.x, dst.y);
12. }
```

記憶體中的結構

變數名稱	記憶體位址	資料
dst dst.x	1000	**4**
dst.y	1004	?
src src.x	1008	0
src.y	1012	0

定義**二維座標點**衍生型別

```
1.  struct point {
2.    float x;
3.    float y;
4.  };
```

使用**二維座標點**衍生型別

```
5.  int main() {
6.    struct point src = {0, 0};
7.    struct point dst;
8.    dst.x = 4;
9.    dst.y = 3;
10.   printf("src:(%.2f,%.2f)\n", src.x, src.y);
11.   printf("dst:(%.2f,%.2f)\n", dst.x, dst.y);
12. }
```

記憶體中的結構

變數名稱	記憶體位址	資料
dst dst.x	1000	4
dst.y	1004	3
src src.x	1008	0
src.y	1012	0

定義二維座標點衍生型別

```
1.  struct point {
2.    float x;
3.    float y;
4.  };
```

使用二維座標點衍生型別

```
5.  int main() {
6.    struct point src = {0, 0};
7.    struct point dst;
8.    dst.x = 4;
9.    dst.y = 3;
10.   printf("src:(%.2f,%.2f)\n", src.x, src.y);
11.   printf("dst:(%.2f,%.2f)\n", dst.x, dst.y);
12. }
```

小技巧:以typedef定義與使用結構

在C語言中, typedef關鍵字可以用來為型別取別名(alias)

定義**二維座標點**衍生型別

```
1.   typedef struct point {
2.     float x;
3.     float y;
4.   } Point;
```

使用**二維座標點**衍生型別

```
5.   int main() {
6.     Point src = {0, 0};
7.     Point dst;
8.     dst.x = 4;
9.     dst.y = 3;
10.    printf("src:(%.2f,%.2f)\n", src.x, src.y);
11.    printf("dst:(%.2f,%.2f)\n", dst.x, dst.y);
12.  }
```

結構與函式

Struct and Function

函式傳入結構

範例, 計算歐幾里得距離

```
1.  float dist(Point src, Point dst) {
2.    float xx = pow(src.x - dst.x, 2);
3.    float yy = pow(src.y - dst.y, 2);
4.    return sqrt(xx + yy);
5.  }
```

函式傳回結構(錯誤)

範例，移動二維座標點

```
1.  void move(Point src, float dx, float dy) {
2.    src.x = src.x + dx;
3.    src.y = src.y + dy;
4.  }
```

請問這個程式有問題嗎?

函式傳回結構(正確)

範例，移動二維座標點(Call by Value)

```
1.  Point move(Point src, float dx, float dy) {
2.    struct point new_point;
3.    new_point.x = src.x + dx;
4.    new_point.y = src.y + dy;
5.    return new_point;
6.  }
```

練習：線

請設計一個線(Line)的衍生型別，包含線的兩個二維座標點。

實作計算歐幾里德距離的函式如下

```
float dist(Line line);
```

利用此衍生型別撰寫成數輸入兩個二維座標點後計算出歐幾里得距離。

限制條件:
使用結構

輸入	輸出
0 0 0 3	3
0 1 2 5	4.47

A few moments later …

結構與陣列

Struct and Array

結構與陣列

定義學生衍生型別

```
1.  typedef struct student {
2.    int id;
3.    char name[10];
4.    int chi; // Chinese
5.  } Student;
```

使用學生衍生型別陣列

```
6.  int main() {
7.    Student s[2] = {{0,"Bob",100},{1,"Alice",80}};
8.    printf("%s:%d\n", s[0].name, s[0].chi);
9.    printf("%s:%d\n", s[1].name, s[1].chi);
10. }
```

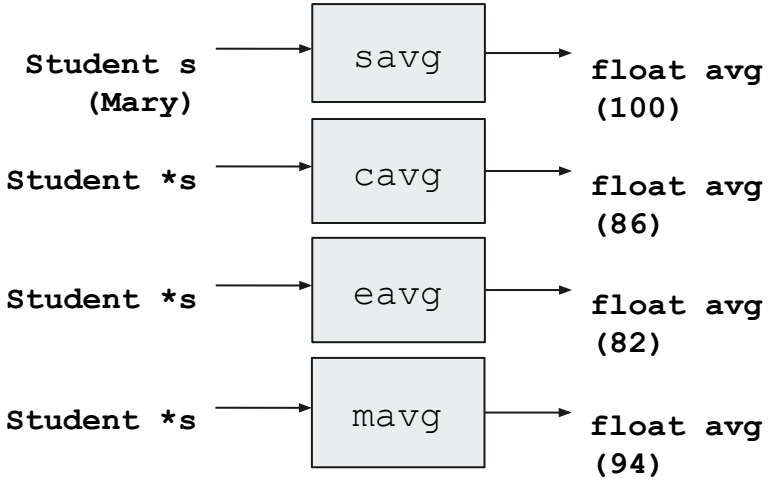
練習:學生

請設計一個學生的衍生型別，包含學生id(int)、學生姓名(char[10])、國文成績(int)、英文成績(int)、數學成績(int)。

初始化5個學生的資料，並撰寫計算平均成績的函式

限制條件:
使用結構

id	name	chi	eng	mat
0	Bob	100	70	90
1	Alice	90	90	100
2	John	70	80	100
3	Mary	100	100	100
4	Gorge	70	70	80



結構與指標

Struct and Point

結構與指標

定義學生衍生型別

```
1.   typedef struct student {
2.     int id;
3.     char name[10];
4.     int chi; // Chinese
5.   } Student;
```

使用學生衍生型別指標指向陣列元素

```
6.   int main() {
7.     Student s[2] = {{0,"Bob",100},{1,"Alice",80}};
8.     Student *ptr = s;
9.     printf("%s:%d\n", ptr->name, ptr->chi);
10.    ptr = ptr + 1;
11.    printf("%s:%d\n", ptr->name, ptr->chi);
12.  }
```

結構的動態記憶體配置

定義學生衍生型別

```
1.   typedef struct student {
2.     int id;
3.     char name[10];
4.     int chi; // Chinese
5.   } Student;
```

動態配置學生衍生型別陣列

```
6.   int main() {
7.     int n;
8.     scanf("%d", &n);
9.     Student *ptr = NULL;
10.    ptr = (Student*)malloc(sizeof(Student)*n);
11.    memset(ptr, 0, sizeof(Student)*n);
12.    ptr->id = 0;
13.    strcpy(ptr->name, "Bob");
14.    ptr->chi = 100;
15.    printf("%s:%d\n", ptr->name, ptr->chi);
16.  }
```

Q & A

Computer History Museum, Mt. View, CA

基礎資料結構

Preliminary on Data Struct

資料結構(Data Structure)

簡單來說，**資料結構**就是程式中儲存(store)資料的方式。

演算法(algorithm)則是程式中處理(process)資料的方式。

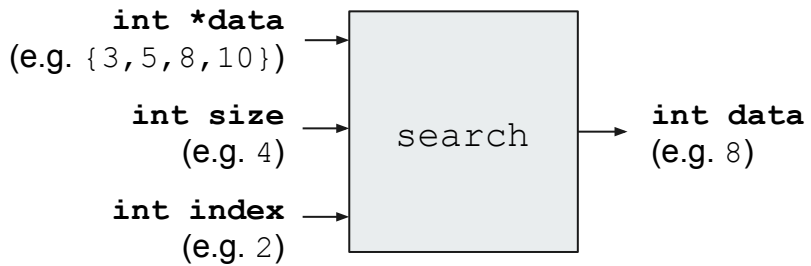
在適當的情況下，選擇適當的資料結構與演算法可以提高程式效率(速度更快或使用更少的記憶體)。

以上兩個議題都會在日後課程中各用滿滿的一學期來教你。

陣列(搜尋)

陣列適合搜尋(search)資料嗎?

```
1.   int data[5] = {10,34,56,78,100};
2.   printf("%d\n", search(data, 5, 3)); //印出78
```

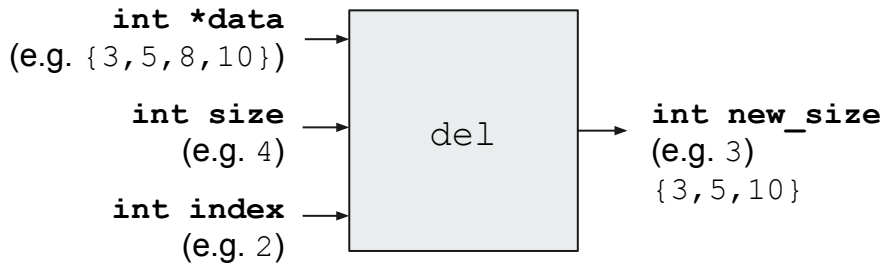


變數名稱	記憶體位址	資料
data	2000	10
	2004	34
	2008	56
	2012	78
	2016	100

陣列(刪除)

陣列適合刪除(delete)資料嗎?

```
1.   int data[5] = {10,34,56,78,100};
2.   del(data, 5, 2); //{10,34,78,100}
```

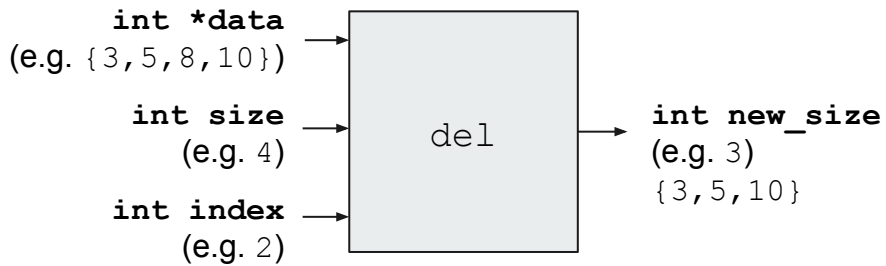


變數名稱	記憶體位址	資料
data	2000	10
	2004	34
	2008	56
	2012	78
	2016	100

陣列(刪除)

陣列適合刪除(delete)資料嗎?

```
1.   int data[5] = {10,34,56,78,100};
2.   del(ptr, 2); //{10,34,78,100}
```

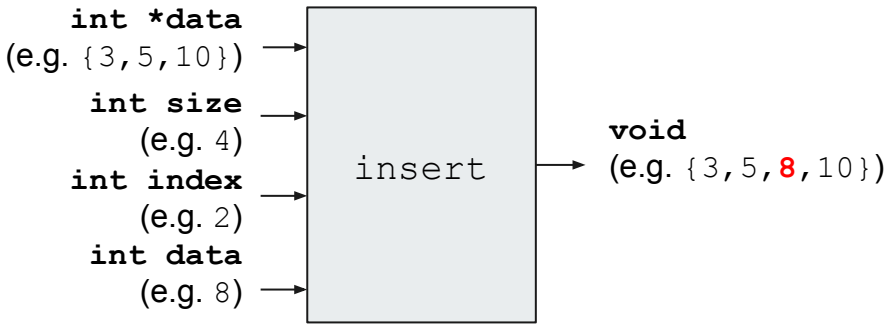


變數名稱	記憶體位址	資料
data	2000	10
	2004	34
	2008	78
	2012	100

陣列(插入)

陣列適合插入(inserft)資料嗎?

```
1.  int data[5] = {10,34,56,100};
2.  insert(data,4,3,60); //{10,34,56,60,100}
```

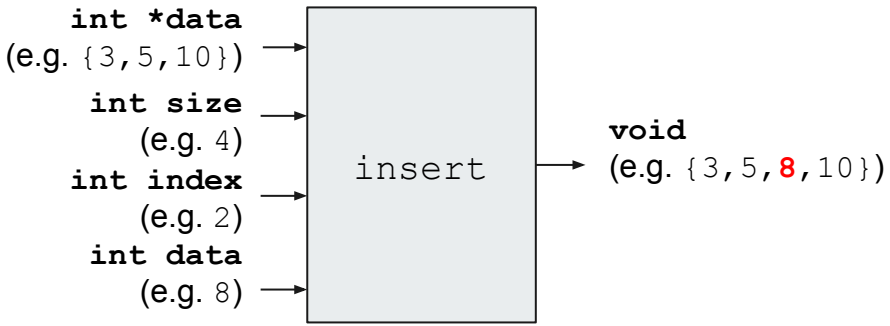


變數名稱	記憶體位址	資料
data	2000	10
	2004	34
	2008	56
	2012	100

陣列(插入)

陣列適合插入(inserft)資料嗎?

```
1.   int data[5] = {10,34,56,100};
2.   insert(data,4,3,60); //{10,34,56,60,100}
```



變數名稱	記憶體位址	資料
data	2000	10
	2004	34
	2008	56
	2012	60
	2016	100

練習:陣列操作

請完成前述陣列操作(搜尋、刪除、插入)
使用一個陣列來驗證，每依序執行上述操作。

```
int data[] = {10,20,30};
int size = 3;
```

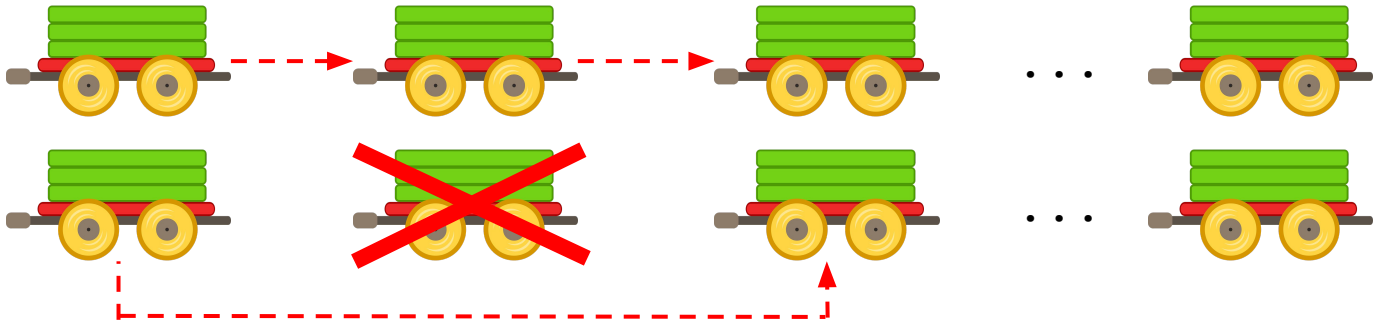
輸入	輸出
search(data, 1);	20
del(data, 1);	10 30
insert(data,1,15);	10 15 30

A few moments later …

鏈結串列(Linked List)

鏈結串列是一種將多筆相同型別的資料以**指標**串連在一起的資料結構。

因為資料是透過指標串連在一起(*就像是火車串連在一起*)，所以透過變更指標就可以很容易的刪除或插入資料。



新增Node通常會使用**動態配置記憶體** 如下

```
Node *new_node = (Node*)malloc(sizeof(Node));
```

所以相對的刪除Node時需要呼叫free(new_node)來釋放記憶體

鏈結串列

鏈結串列建立(與記憶體)

```
1.  typedef struct node {
2.    int data;
3.    struct node *next;
4.  } Node;
5.  ...
6.  Node n0 = {10, NULL};
7.  Node n1 = {20, NULL};
8.  Node n2 = {30, NULL};
9.  Node *head = &n0;
10. n0.next = &n1;
11. n1.next = &n2;
```

變數名稱	記憶體位址	資料
n0 data	3000	10
next	3004	NULL

鏈結串列(續)

鏈結串列建立(與記憶體)

```
1.   typedef struct node {
2.     int data;
3.     struct node *next;
4.   } Node;
5.   ...
6.   Node n0 = {10, NULL};
7.   Node n1 = {20, NULL};
8.   Node n2 = {30, NULL};
9.   Node *head = &n0;
10.  n0.next = &n1;
11.  n1.next = &n2;
```

變數名稱	記憶體位址	資料
n1 data	2000	20
next	2004	NULL
...	...	...
n0 data	3000	10
next	3004	NULL

鏈結串列(續)

鏈結串列建立(與記憶體)

```
1.   typedef struct node {
2.     int data;
3.     struct node *next;
4.   } Node;
5.   ...
6.   Node n0 = {10, NULL};
7.   Node n1 = {20, NULL};
8.   Node n2 = {30, NULL};
9.   Node *head = &n0;
10.  n0.next = &n1;
11.  n1.next = &n2;
```

變數名稱	記憶體位址	資料
n2 data	1000	30
next	1004	NULL
...	...	...
n1 data	2000	20
next	2004	NULL
...	...	...
n0 data	3000	10
next	3004	NULL

鏈結串列(續)

鏈結串列建立(與記憶體)

```
1.   typedef struct node {
2.     int data;
3.     struct node *next;
4.   } Node;
5.   ...
6.   Node n0 = {10, NULL};
7.   Node n1 = {20, NULL};
8.   Node n2 = {30, NULL};
9.   Node *head = &n0;
10.  n0.next = &n1;
11.  n1.next = &n2;
```

變數名稱	記憶體位址	資料
head	992	3000
n2	1000	30
	1004	NULL
...	...	...
n1	2000	20
	2004	NULL
...	...	...
n0	3000	10
	3004	2000

鏈結串列(續)

鏈結串列建立(與記憶體)

```
1.   typedef struct node {
2.     int data;
3.     struct node *next;
4.   } Node;
5.   ...
6.   Node n0 = {10, NULL};
7.   Node n1 = {20, NULL};
8.   Node n2 = {30, NULL};
9.   Node *head = &n0;
10.  n0.next = &n1;
11.  n1.next = &n2;
```

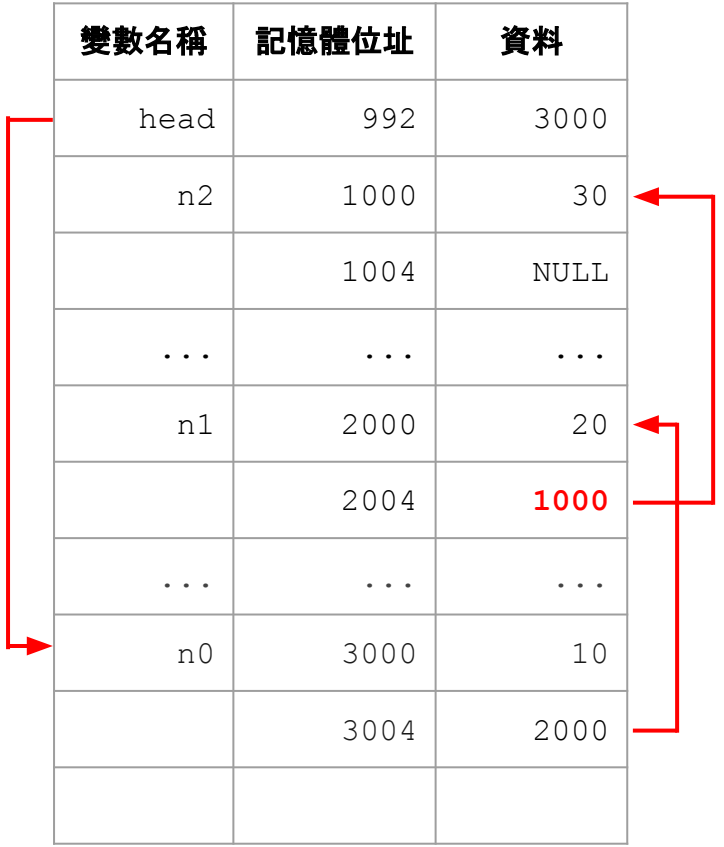
變數名稱	記憶體位址	資料
head	992	3000
n2	1000	30
	1004	NULL
...	...	...
n1	2000	20
	2004	NULL
...	...	...
n0	3000	10
	3004	2000

鏈結串列(續)

鏈結串列建立(與記憶體)

```
1.   typedef struct node {
2.     int data;
3.     struct node *next;
4.   } Node;
5.   ...
6.   Node n0 = {10, NULL};
7.   Node n1 = {20, NULL};
8.   Node n2 = {30, NULL};
9.   Node *head = &n0;
10.  n0.next = &n1;
11.  n1.next = &n2;
```

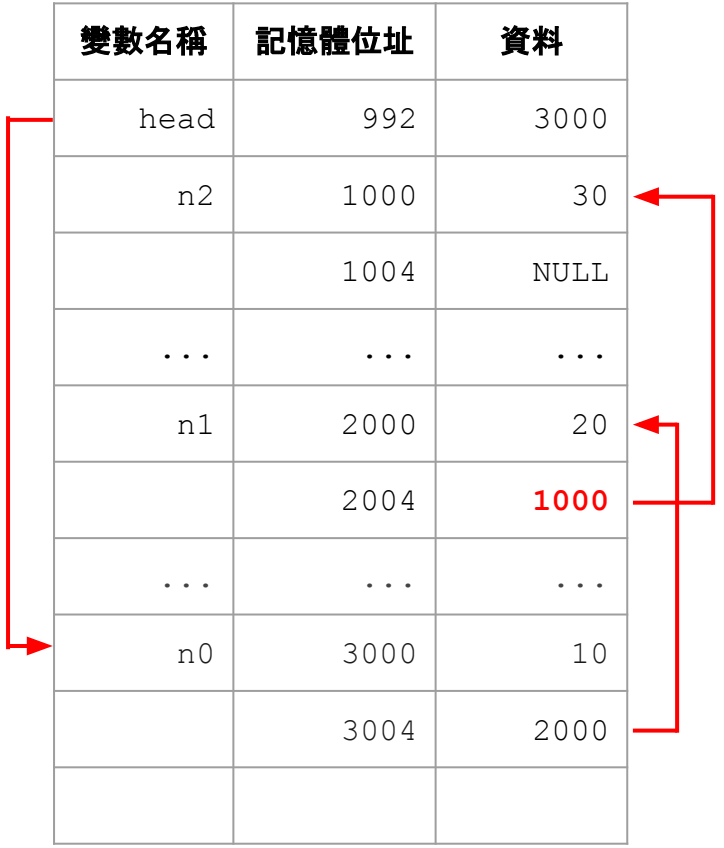
變數名稱	記憶體位址	資料
head	992	3000
n2	1000	30
	1004	NULL
...	...	...
n1	2000	20
	2004	**1000**
...	...	...
n0	3000	10
	3004	2000



鏈結串列(列印)

列印鏈結串列中的元素

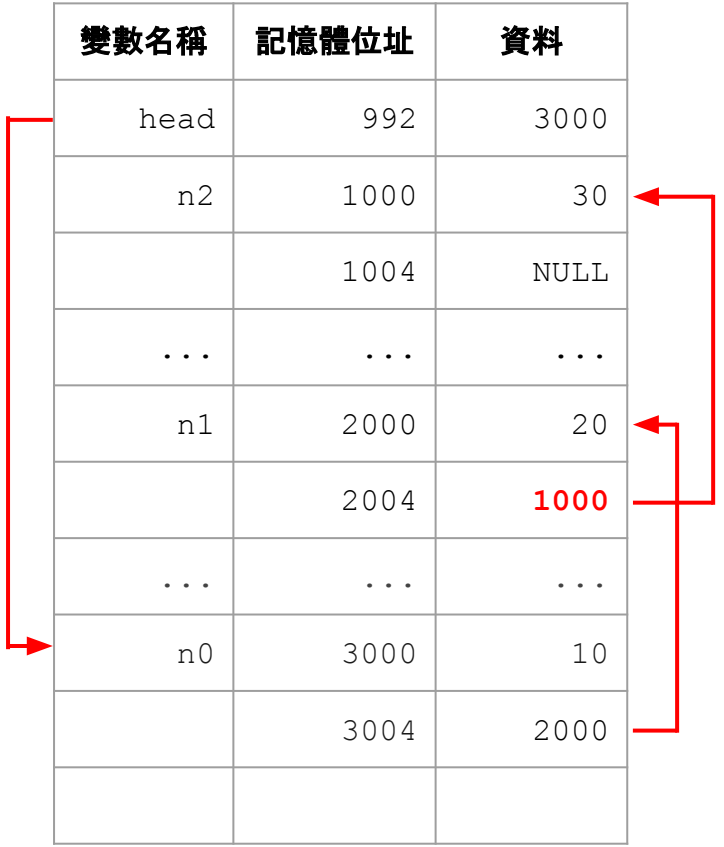
```
1.   Node n0 = {10, NULL};
2.   Node n1 = {20, NULL};
3.   Node n2 = {30, NULL};
4.   Node *head = &n0;
5.   n0.next = &n1;
6.   n1.next = &n2;
7.   ...
8.   Node *ptr = head;
9.   while(ptr != NULL) {
10.    printf("%d ", ptr->data);
11.    ptr = ptr->next;
12.  }
13.  print("\n");
```



變數名稱	記憶體位址	資料
head	992	3000
n2	1000	30
	1004	NULL
...	...	...
n1	2000	20
	2004	1000
...	...	...
n0	3000	10
	3004	2000

鏈結串列(刪除節點)

```
1.   Node n0 = {10, NULL};
2.   Node n1 = {20, NULL};
3.   Node n2 = {30, NULL};
4.   Node *head = &n0;
5.   n0.next = &n1;
6.   n1.next = &n2;
7.   ...
8.   Node *ptr = head;
9.   del(ptr, 1);
10.  while(ptr != NULL) {
11.    printf("%d ", ptr->data);
12.    ptr = ptr->next;
13.  }
14.  print("\n");
```



變數名稱	記憶體位址	資料
head	992	3000
n2	1000	30
	1004	NULL
...	...	...
n1	2000	20
	2004	1000
...	...	...
n0	3000	10
	3004	2000

鏈結串列(刪除節點)

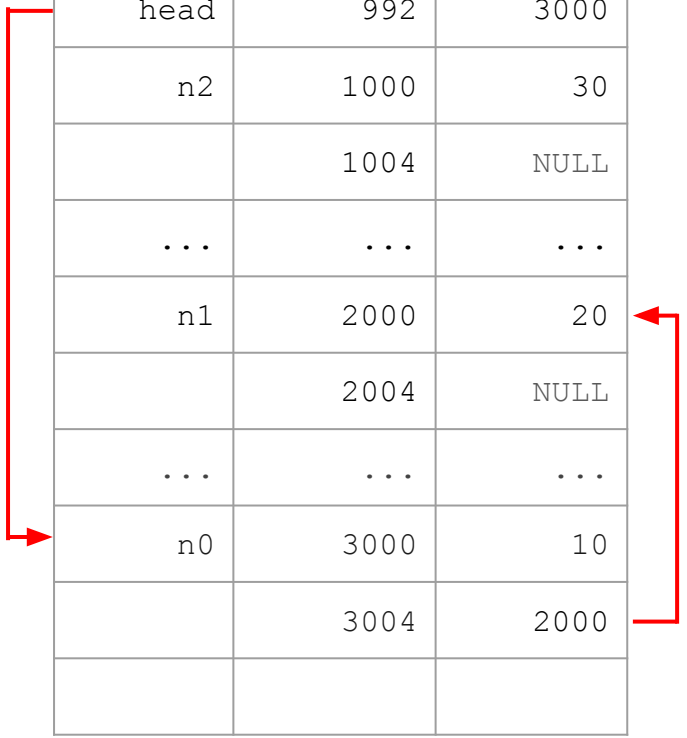
```
1.   Node n0 = {10, NULL};
2.   Node n1 = {20, NULL};
3.   Node n2 = {30, NULL};
4.   Node *head = &n0;
5.   n0.next = &n1;
6.   n1.next = &n2;
7.   ...
8.   Node *ptr = head;
9.   del(ptr, 1); //{10,30}
10.  while(ptr != NULL) {
11.    printf("%d ", ptr->data);
12.    ptr = ptr->next;
13.  }
14.  print("\n");
```

變數名稱	記憶體位址	資料
head	992	3000
n2	1000	30
	1004	NULL
...	...	...
n1	2000	20
	2004	1000
...	...	...
n0	3000	10
	3004	1000

鏈結串列(插入節點)

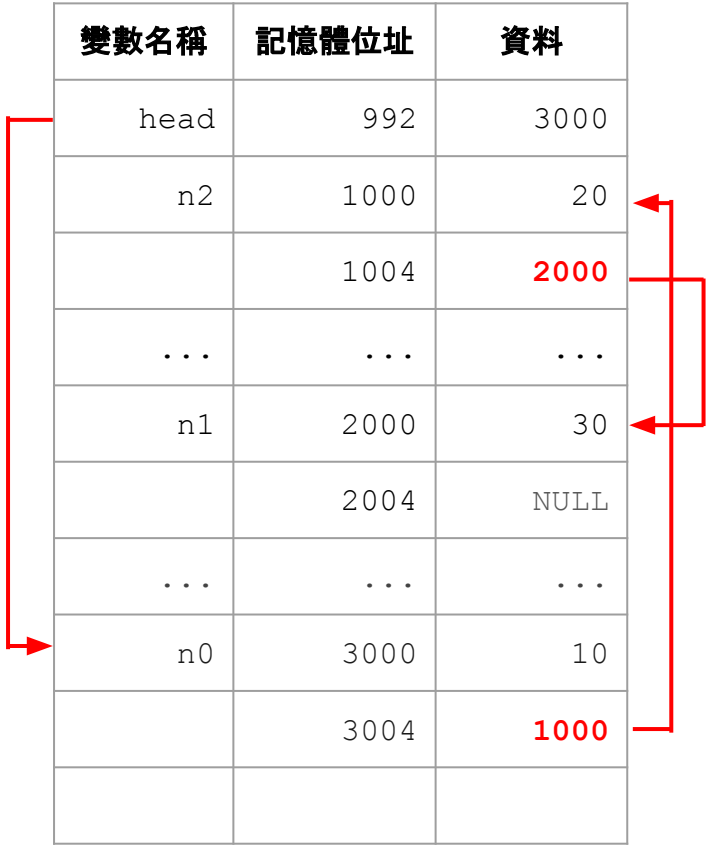
```
1.   Node n0 = {10, NULL};
2.   Node n1 = {30, NULL};
3.   Node n2 = {20, NULL};
4.   Node *head = &n0;
5.   n0.next = &n1;
6.   ...
7.   Node *ptr = head;
8.   insert(n0, 1, n2);
9.   while(ptr != NULL) {
10.    printf("%d ", ptr->data);
11.    ptr = ptr->next;
12.  }
13.  print("\n");
```

變數名稱	記憶體位址	資料
head	992	3000
n2	1000	30
	1004	NULL
...	...	...
n1	2000	20
	2004	NULL
...	...	...
n0	3000	10
	3004	2000



鏈結串列(插入節點)

```
1.   Node n0 = {10, NULL};
2.   Node n1 = {30, NULL};
3.   Node n2 = {20, NULL};
4.   Node *head = &n0;
5.   n0.next = &n1;
6.   ...
7.   Node *ptr = head;
8.   insert(n0, 1, n2);
9.   while(ptr != NULL) {
10.    printf("%d ", ptr->data);
11.    ptr = ptr->next;
12.  }
13.  print("\n");
```



變數名稱	記憶體位址	資料
head	992	3000
n2	1000	20
	1004	**2000**
...	...	...
n1	2000	30
	2004	NULL
...	...	...
n0	3000	10
	3004	**1000**

練習:鏈結串列操作

請完成前述鏈結串列操作(搜尋、刪除、插入)
使用一個鏈結串鏈來驗證，每依序執行上述操作。

```
Node n0 = {10, NULL};
Node n1 = {20, NULL};
Node n2 = {30, NULL};
Node *head = &n0;
n0.next = &n1;
```

輸入	輸出
`search(head, 1);`	`20`
`insert(head, 1, n2);`	`10 30 20`
`del(head, 1);`	`10 20`

A few moments later …

Q & A

Computer History Museum, Mt. View, CA