

# C++資料結構與程式設計

## 資料結構概論

NTU CSIE

# Outline

資料結構概論

C語言的結構 (struct)

結構化的資料

常見的資料結構簡介

# 從一個例子開始...

算出班上十位同學成績之總分與平均

```
#include <stdio.h>
int main()
{
    //宣告變數與資料內容
    int a0=80, a1=90, a2=70, a3=66, a4=56;
    int a5=99, a6=88, a7=50, a8=60, a9=77;
    int sum;
    double average;

    //計算
    sum = a0+a1+a2+a3+a4+a5+a6+a7+a8+a9;
    average = (double)sum/10;

    //輸出結果
    printf("總分: %d\n", sum);
    printf("平均: %.2lf\n", average);
    return 0;
}
```

# 從一個例子開始...

改用陣列(資料儲存方式) 與控制 (程式行為)

```
#include <stdio.h>
int main()
{
    //宣告變數與資料內容
    int a[10]={80, 90, 70, 66, 56, 99, 88, 50, 60, 77}, i;
    int sum;
    double average;

    //計算
    sum=0;
    for(i=0; i<10; i++)
        sum+=a[i];
    average = (double)sum/10;

    //輸出結果
    printf("總分: %d\n", sum);
    printf("平均: %.2lf\n", average);
    return 0;
}
```

# 換一個例子...

算出班上N位同學成績之總分與平均-變成自訂N與自行輸入  
但陣列的大小卻仍然是固定的 (100)

```
#include <stdio.h>
int main()
{
    //宣告變數與資料內容
    int a[100];
    int sum, i, n;
    double average;

    //輸入
    printf("輸入班上同學人數: ");
    scanf("%d", &n);
```

```
    for(i=0; i<n; i++)
    {
        printf("%d號: ", i+1);
        scanf("%d", &a[i]);
    }

    //計算
    sum=0;
    for(i=0; i<n; i++)
        sum+=a[i];
    average = (double)sum/n;

    //輸出結果
    printf("總分: %d\n", sum);
    printf("平均: %.2lf\n", average);
    return 0;
```

```
}
```

# 換一個例子...

使用動態記憶體配置減少不必要的浪費與避免資料不夠用

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    //宣告變數與資料內容
    int *a;
    int sum, i, n;
    double average;

    //輸入
    printf("輸入班上同學人數: ");
    scanf("%d", &n);
    a = (int *)malloc(sizeof(int)*n);
```

```
    for(i=0; i<n; i++)
    {
        printf("%d號: ", i+1);
        scanf("%d", &a[i]);
    }

    //計算
    sum=0;
    for(i=0; i<n; i++)
        sum+=a[i];
    average = (double)sum/n;

    //輸出結果
    printf("總分: %d\n", sum);
    printf("平均: %.2lf\n", average);
    return 0;
```

```
}
```

# 資料結構與演算法

## 資料儲存方式→資料結構

- 資料如何擺放
- 空間夠不夠用
- 會不會浪費空間

## 程式行為→演算法

- 資料如何儲存、搜尋、刪除
- 執行速度

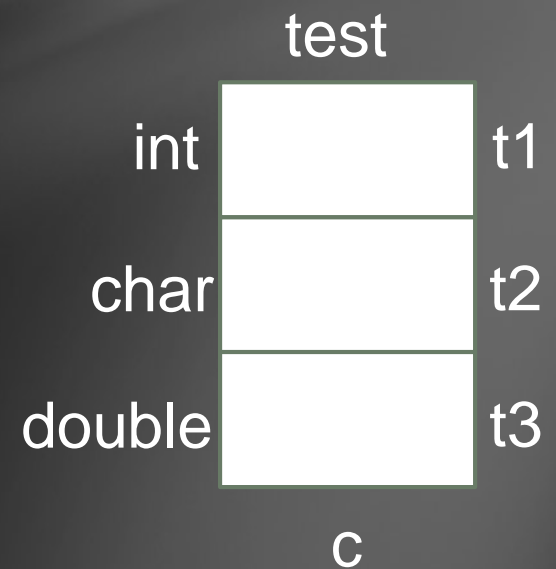
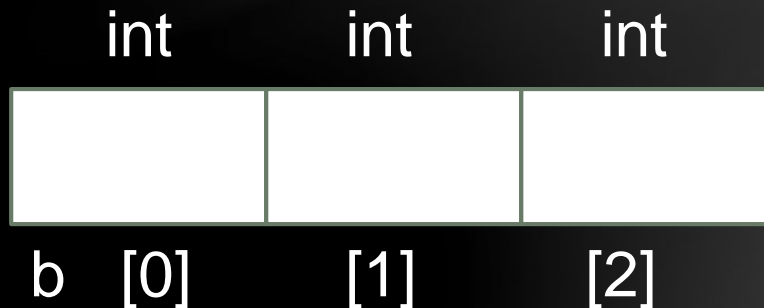
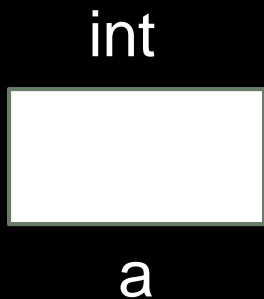
# 資料的儲存

變數

陣列

結構

```
#include<stdio.h>
struct _test
{
    int t1;
    char t2;
    double t3;
};
typedef struct _test test;
int main()
{
    int a;
    int b[3];
    test c;
    return 0;
}
```





# 思考

- 你想設計一個通訊錄程式
- 用來紀錄使用者好友資料(姓名, 電話, Email)
- 資料該如何儲存?
- 程式該如何設計?
  - 結構陣列? (資料可能浪費或不夠用)
  - 結構指標+動態記憶體配置? (請輸入你的好友個數???)
  - 其他?

# Outline

資料結構概論

C語言的結構 (struct)

結構化的資料

常見的資料結構簡介

# 結構 ( Struct )

通常一個簡單之變數或陣列不足以用來儲存複雜之記錄

C語言中有結構體之架構，允許使用者宣告資料實體將不同形式之元素儲存一起

結構是一種由使用者自訂之資料型態

# 結構 ( Struct )

**struct** 結構名稱標籤

```
{  
    資料型態 資料變數元素1;  
    資料型態 資料變數元素2;  
    . . . . .  
};
```

typedef struct 結構名稱標籤 命名結構名稱標籤

EX:

**struct** \_Person

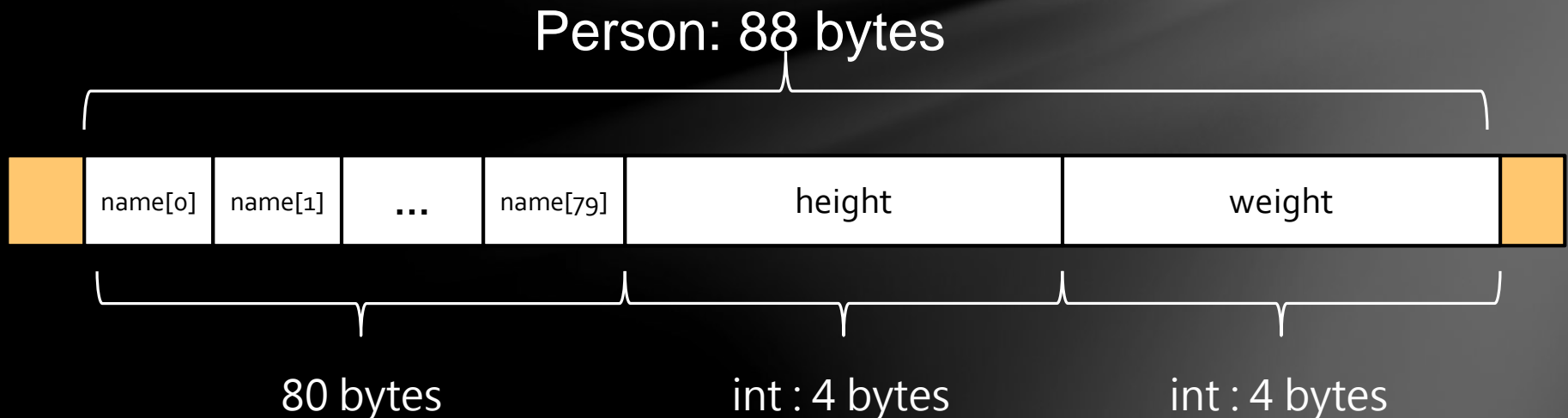
```
{  
    char name[80];  
    int height;  
    int weight;  
};
```

typedef **struct** \_Person **Person**

# 結構 ( Struct )

EX:

- struct \_Person  
{  
    char name[80];  
    int height;  
    int weight;  
};
- typedef struct \_Person Person



# 結構 ( Struct )

結構被宣告後即可定義任何變數。由上述宣告的例子來說明, **Person**為此結構的名稱,又稱為標籤 ( tag )。

結構的成員(元素)其資料型態可以使用 **int**, **float** 及 **char**。  
也可用陣列與指標變數。

使用結構定義之變數,要存取該成員可透過 **.** 來直接存取;  
使用結構定義之指標,要存取該成員可透過 **->** 來間接存取

# 結構指標

在本課程中,我們將經常使用結構指標來解決各式各樣的問題,所以我們先從以下範例來了解結構指標使用的原理:

```
#include <stdio.h>

struct _Person
{
    char name[80];
    int height;
    int weight;
};
typedef struct _Person Person;

int main()
{
    Person p1;
    Person *p2;
    p2 = &p1;
    gets(p2->name);
    scanf("%d",&p2->height);
    scanf("%d",&p2->weight);
    return 0;
}
```

# 結構陣列

```
#include<stdio.h>
struct _student
{
    int math;
    int english;
    int computer;
};
typedef struct _student student;
int main()
{
    student s[5]; //結構陣列
    return 0;
}
```

	student	student	student	student	student
math					
english					
computer					
	s [0]	[1]	[2]	[3]	[4]



# 小練習 (chap01\_ex1.c)

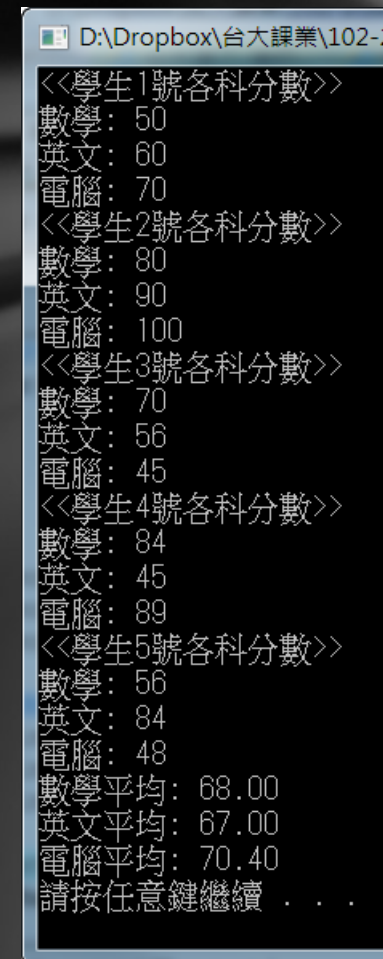
寫一個可以輸入並儲存班上五位同學的數學、英文、電腦成績之程式，並印出各科平均。

使用結構陣列

```
struct _student
{
    int math;
    int english;
    int computer;
};
typedef struct _student student;

student s[N];

sum_m+=s[i].math;
sum_e+=s[i].english;
sum_c+=s[i].computer;
```



```
D:\Dropbox\台大課業\102-1
<<學生1號各科分數>>
數學: 50
英文: 60
電腦: 70
<<學生2號各科分數>>
數學: 80
英文: 90
電腦: 100
<<學生3號各科分數>>
數學: 70
英文: 56
電腦: 45
<<學生4號各科分數>>
數學: 84
英文: 45
電腦: 89
<<學生5號各科分數>>
數學: 56
英文: 84
電腦: 48
數學平均: 68.00
英文平均: 67.00
電腦平均: 70.40
請按任意鍵繼續 . . .
```

```
#include<stdio.h>
#define N 5
struct _student
{
    int math;
    int english;
    int computer;
};
typedef struct _student student;
int main()
{
    // 宣告資料
    student s[N];
    int i;
    int sum_m=0;
    int sum_e=0;
    int sum_c=0;
    double aver_m;
    double aver_e;
    double aver_c;
```

//輸入

```
for(i=0;i<N;i++)    {  
    printf("<<學生%d號各科分數>>\n", i+1);  
    printf("數學: ");  
    scanf("%d", &s[i].math);  
    printf("英文: ");  
    scanf("%d", &s[i].english);  
    printf("電腦: ");  
    scanf("%d", &s[i].computer);  
}
```

//計算

```
for(i=0;i<N;i++)    {  
    sum_m+=s[i].math;  
    sum_e+=s[i].english;  
    sum_c+=s[i].computer;  
}
```

```
aver_m = (double)sum_m/N;  
aver_e = (double)sum_e/N;  
aver_c = (double)sum_c/N;
```

//輸出

```
printf("數學平均: %.2lf\n", aver_m);  
printf("英文平均: %.2lf\n", aver_e);  
printf("電腦平均: %.2lf\n", aver_c);
```

```
system("pause");
```

```
return 0;
```

```
}
```

# Outline

資料結構概論

C語言的結構 (struct)

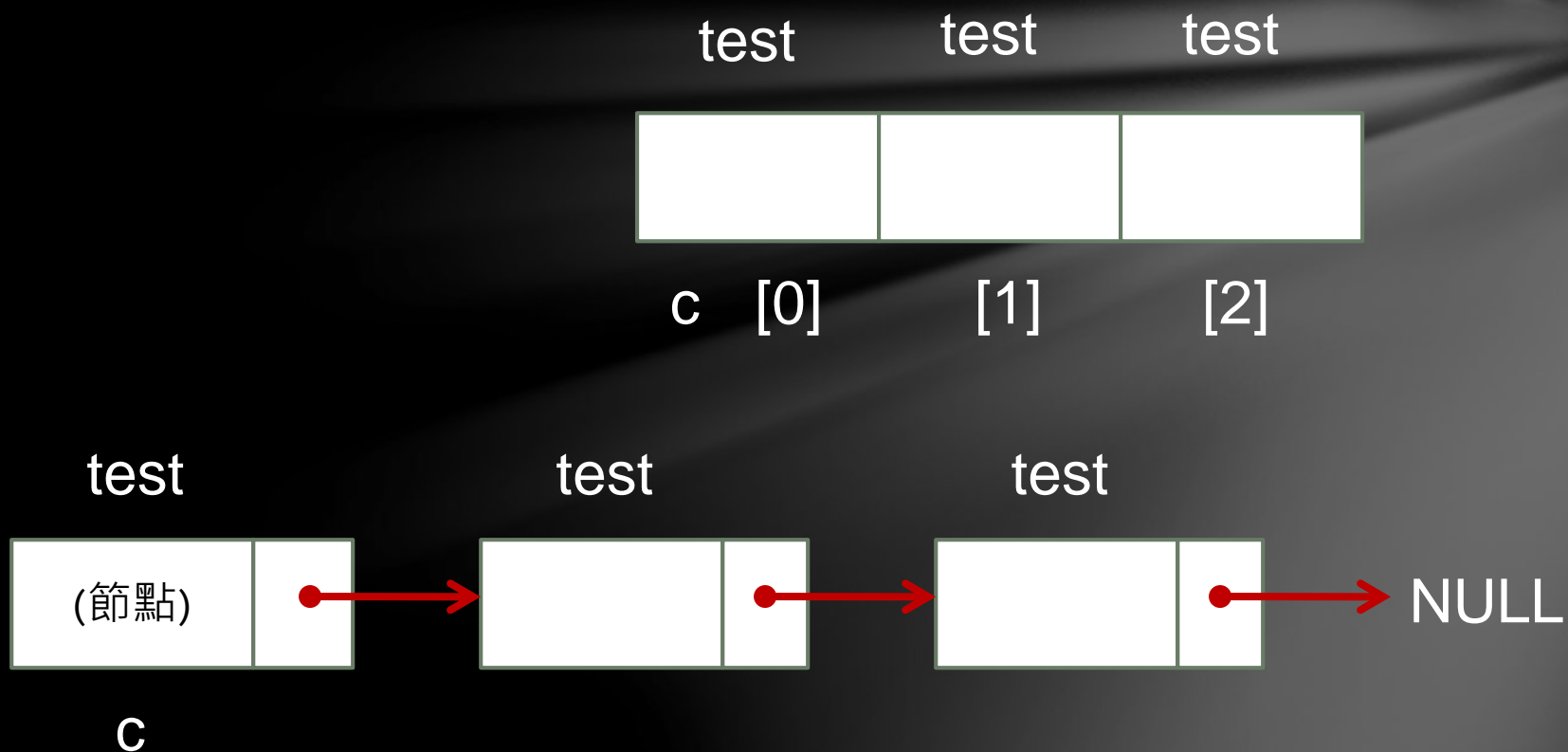
結構化的資料

常見的資料結構簡介

# 結構化的資料

靜態配置 (ex. 結構陣列)

動態配置 (ex. 鏈結串列)



# 結構化的資料:靜態配置

```
#include<stdio.h>
struct _student
{
    int math;
    int english;
    int computer;
};
typedef struct _student student;
int main()
{
    student s[5]; //結構陣列
    return 0;
}
```

	student	student	student	student	student
math					
english					
computer					
s	[0]	[1]	[2]	[3]	[4]

# 結構化的資料:動態配置

```
#include<stdio.h>
#include<stdlib.h>
struct _student
{
    int math;
    int english;
    int computer;
    struct _student *next;
};
typedef struct _student student;
int main()
{
    student *s;
    s = (student *)malloc(sizeof(student));
    s->next = (student *)malloc(sizeof(student));
    s->next->next = (student *)malloc(sizeof(student));
    s->next->next->next = (student *)malloc(sizeof(student));
    s->next->next->next->next = (student *)malloc(sizeof(student));
    s->next->next->next->next->next = NULL;
    return 0;
}
```

student \*



s

math  
english  
computer  
next

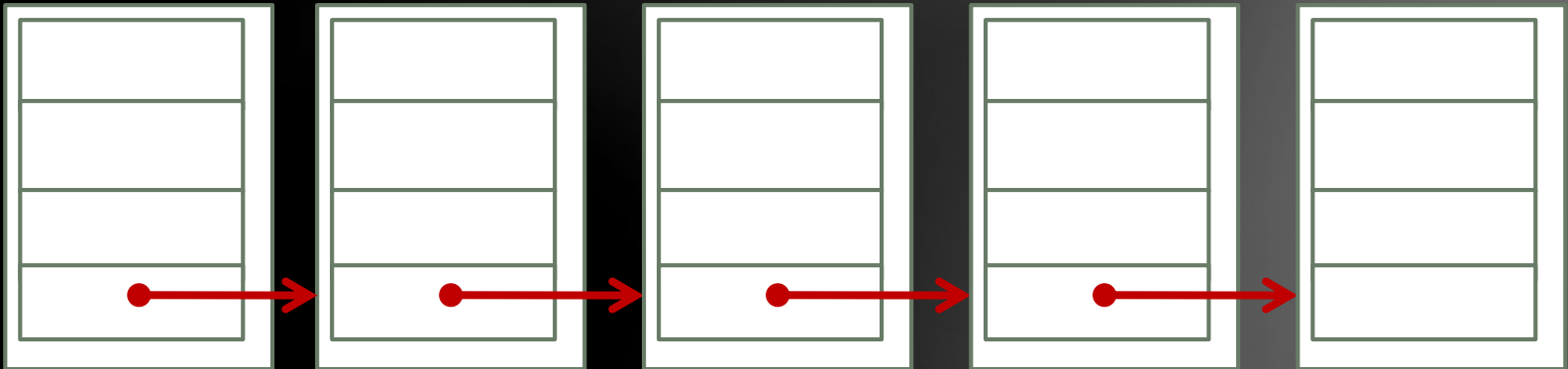
student

student

student

student

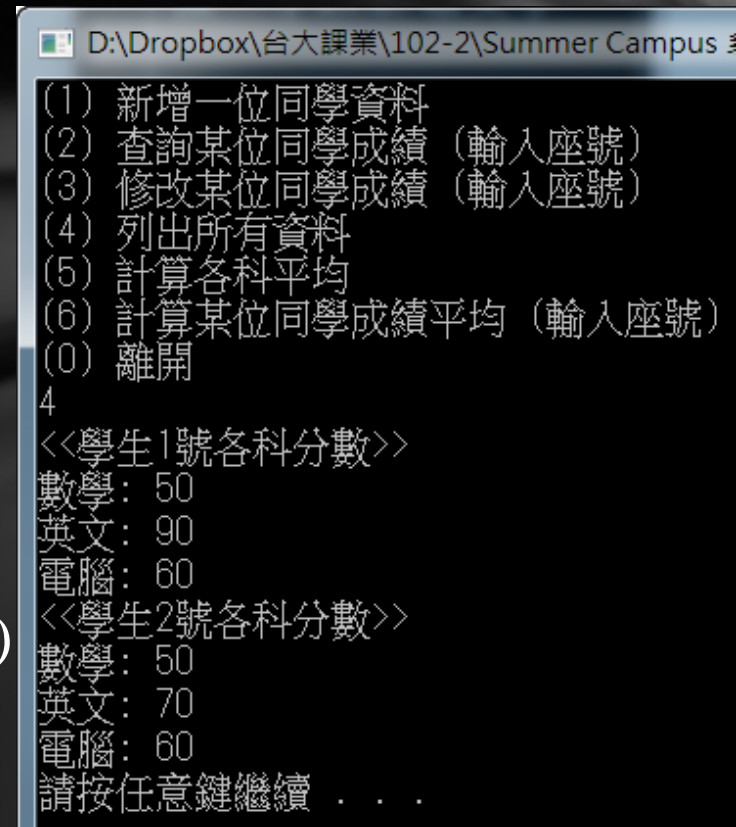
student



# 小練習 (chap01\_ex2.c)

用結構陣列寫一個可以輸入並儲存班上同學(人數上限5人)的數學、英文、電腦成績之程式，並提供以下功能：

- (1) 新增一位同學資料
- (2) 查詢某位同學成績 (輸入座號)
- (3) 修改某位同學成績 (輸入座號)
- (4) 列出所有資料
- (5) 計算各科平均
- (6) 計算某位同學成績平均 (輸入座號)
- 提示：可用switch case



```
D:\Dropbox\台大課業\102-2\Summer Campus
(1) 新增一位同學資料
(2) 查詢某位同學成績 (輸入座號)
(3) 修改某位同學成績 (輸入座號)
(4) 列出所有資料
(5) 計算各科平均
(6) 計算某位同學成績平均 (輸入座號)
(0) 離開
4
<<學生1號各科分數>>
數學: 50
英文: 90
電腦: 60
<<學生2號各科分數>>
數學: 50
英文: 70
電腦: 60
請按任意鍵繼續 . . .
```



# Outline

資料結構概論

C語言的結構 (struct)

結構化的資料

常見的資料結構簡介

# 動態資料結構

在C/C++中利用指標配合結構/類別可使電腦記憶體重複的使用

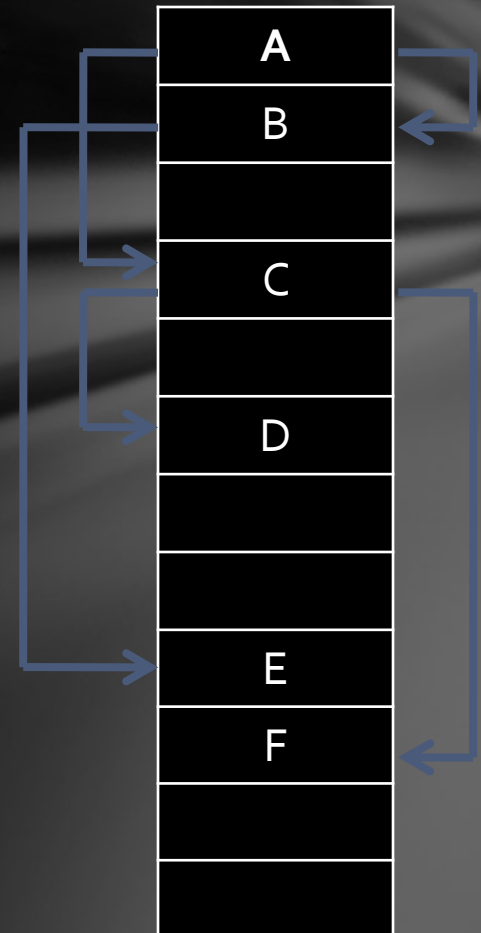
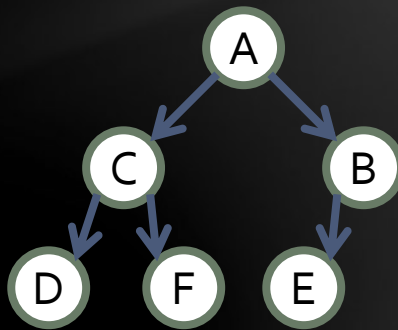
常見的資料結構

- 鏈結串列(Linked List)
- 堆疊(Stack)
- 佇列(Queue)
- 樹狀結構(Tree)
- 圖形結構(Graph)

演算法

- 新增 (Insert)
- 刪除 (Delete)
- 搜尋 (Search)
- 排序 (Sort)

Example: 樹狀結構



# 資料結構

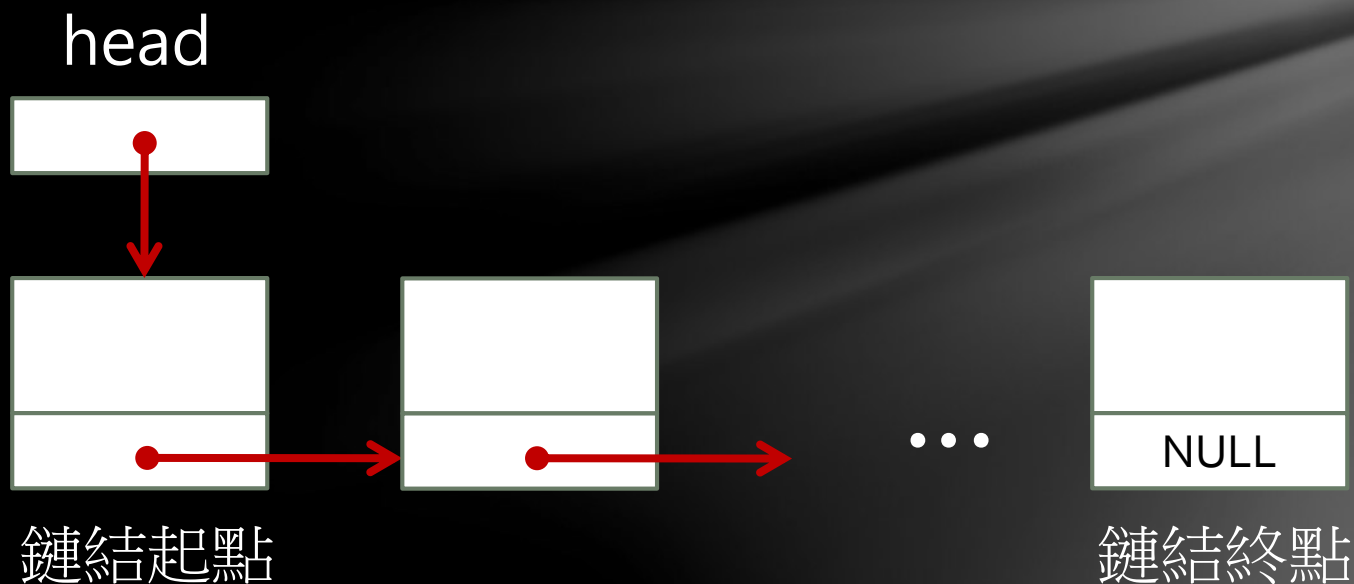
## 了解資料結構的三個重點

- 理論：這個結構用來描述什麼問題
- 程式：這個結構如何用程式來描述
- 應用：這個結構可以用在什麼地方

# 單向鏈結串列

單向鏈結串列之結構如下圖所示

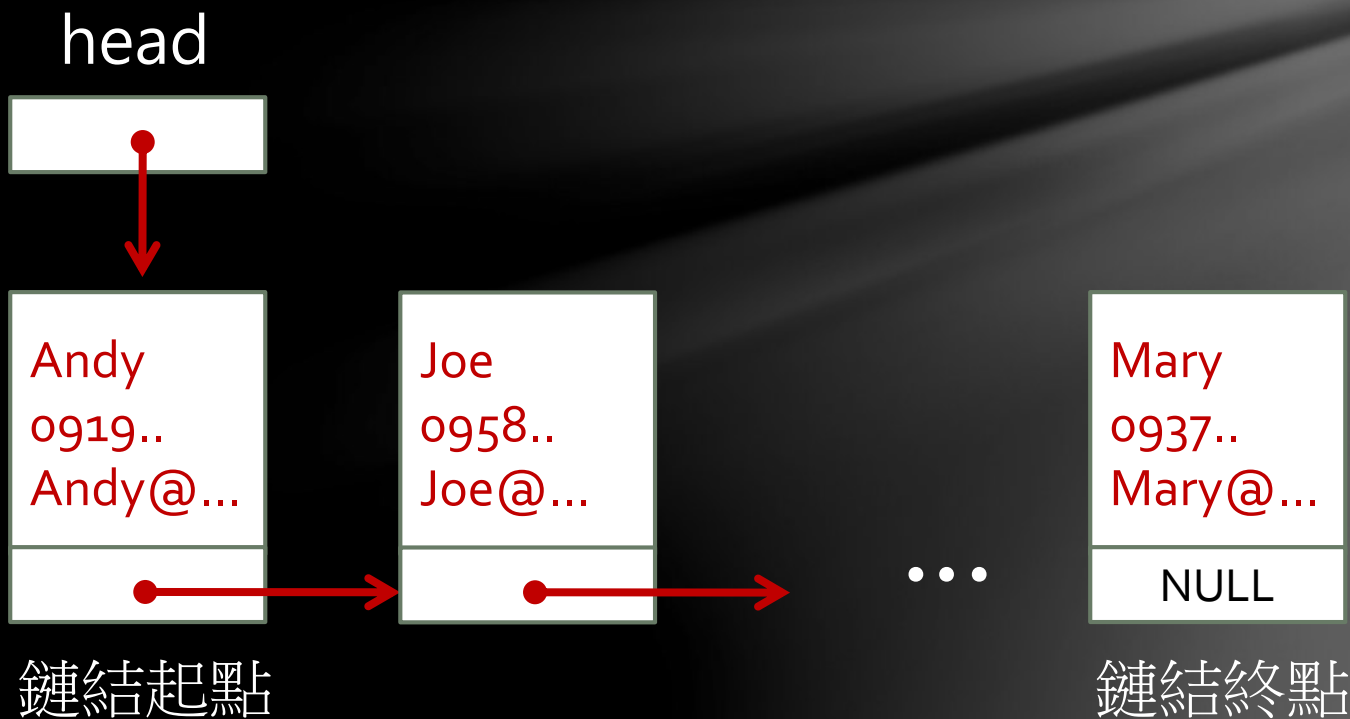
- head：指向串列前端之指標



# 鏈結串列 - 通訊錄程式實作

我們可以使用鏈結串列來實作個有效率且能有效利用記憶體空間之通訊錄程式

- head：指向串列前端之指標



# 堆疊與佇列

## 堆疊(Stack)

- 加入(push)與刪除(pop)於同一端
- 後進先出(LIFO)
- 例子：疊盤子、發牌、走迷宮



## 佇列(Queue)

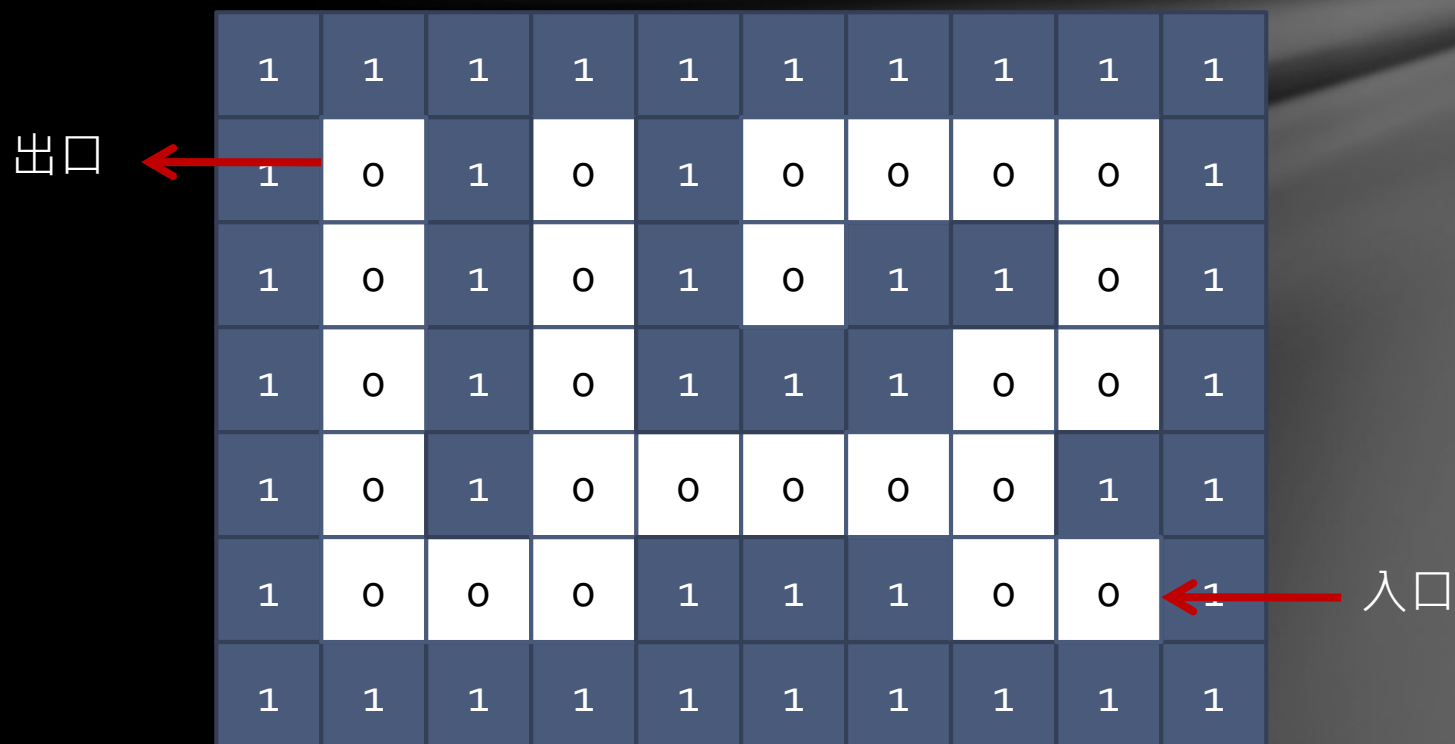
- 加入(enqueue)與刪除(dequeue)於不同端(front & rear)
- 先進先出(FIFO)
- 例子：排隊買票、坐公車、網路伺服器實作



# 堆疊 - 自動走迷宮程式

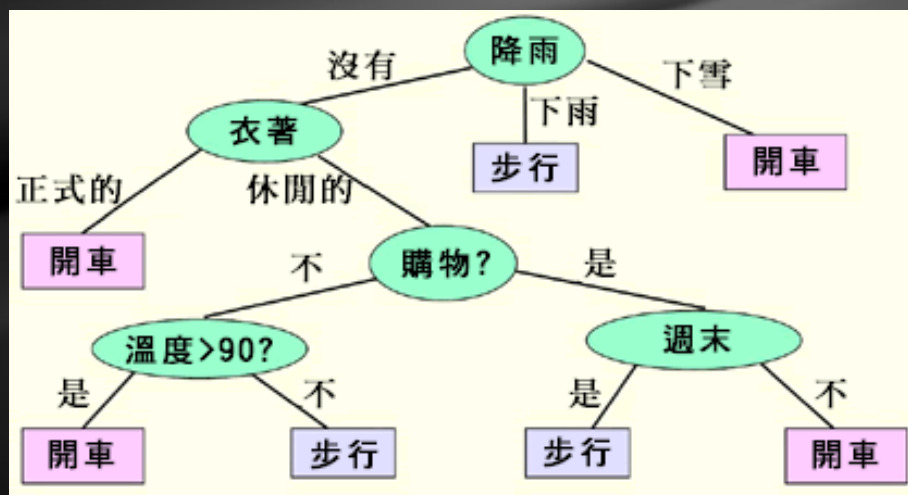
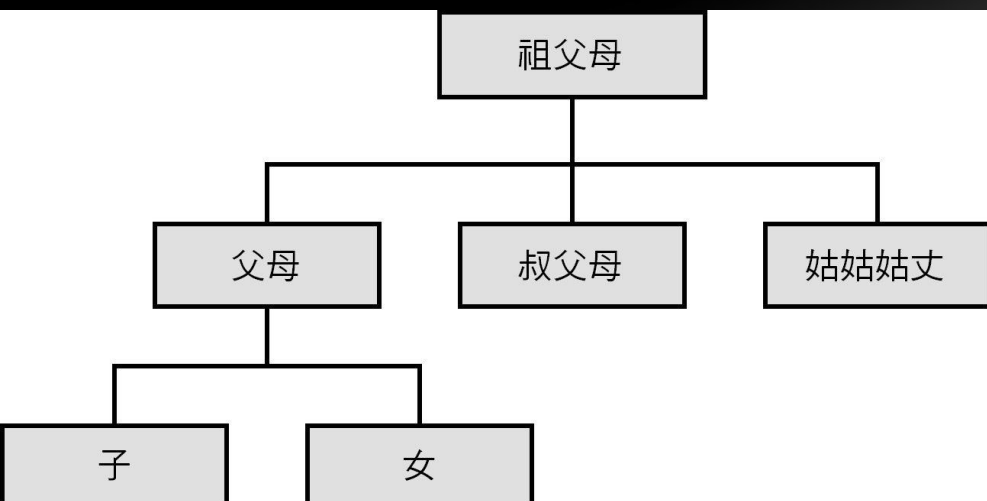
## ▶ 使用堆疊結構實作走迷宮問題

- 走法: 每次都把目前位置存到堆疊, 然後走下一步
- 下一步順序: 上, 下, 左, 右
- 無路可走: 從堆疊中取出上一位置, 看看有沒路走



# 樹狀結構－「樹」(Tree)

- 是一種模擬現實生活中樹幹和樹枝的資料結構，屬於一種階層架構的非線性資料結構
- 例如：家族族譜, 決策模型





# 樹狀結構 - 二元搜尋樹

如何利用樹狀結構實作一個快速的搜尋法？

以下的數都只有左右兩個分支, 左分支的數字一定比右分支大, 在一堆數字當中, 如果你要找20這個數字, 該如何找? 為什麼比較快呢?

試著跟一般陣列或鏈結串列比較

