

## 指定

```
int i, j;
i = j = 4.8;
```

\* '=' is an operator in C, so  $j=0$  will return the value of  $j$   
right associative  $\Rightarrow j$  will be 4,  $i$  will be 4

## 除法

```
int i, j;
i = 13 / 5;
j = -13 / 5;
```

$\Rightarrow 2$   
 $\Rightarrow -2$

看取商數

## 位元

```
int i;
printf("%ld", sizeof(i));
```

$\Rightarrow 4$  單位為位元組, 4 bytes = 4.8 bits  
一個 int  $\in [-2^{31}, 2^{31}-1]$

若超過此範圍, 則產生「溢位」

## 快捷運算

\* 把重要的條件放在前面

```
int k = 3, l = 4, i = 1, j = 2;
if ((k = i) > 0 || (l = j) > 0)
    printf("%d", k);
    printf("%d", l);
```

$\Rightarrow 1$   
 $\Rightarrow 4$

$\because k$  變成 1,  $> 0$ , 為 True,  $\therefore l=j$  就沒做  
使得  $++l$  成立  $\therefore l=j$  就沒做  
\*  $*p++ \Rightarrow$  先取  $P$  指向的值, 再  $P++$

## 判斷式值

\*  $-i = j$  are not allowed  
 $+i--$

```
int i = 1, j = 2, k = 3;
int max = (i > j)? i : j;
if (k > max)
    max = k;
```

三者最大值

\* return 也能用  
此語法

OPERATOR	TYPE	ASSOCIATIVITY
0 1 . ->		left-to-right
++ -- +- ! ~ (type) * & sizeof	Unary Operator	right-to-left
* / %	Arithmetic Operator	left-to-right
+ -	Arithmetic Operator	left-to-right
<< >>	Shift Operator	left-to-right
< <= > >=	Relational Operator	left-to-right
== !=	Relational Operator	left-to-right
&	Bitwise AND Operator	left-to-right
^	Bitwise EX-OR Operator	left-to-right
	Bitwise OR Operator	left-to-right
&&	Logical AND Operator	left-to-right
	Logical OR Operator	left-to-right
:	Ternary Conditional Operator	right-to-left
= += -= *= /= %= &= ^=  = <<= >>=	Assignment Operator	right-to-left

## switch

```
switch (num){  
case 1: case 3:  
    /*code*/  
    break;  
case 2:  
    /*code*/  
    break;  
default:  
    /*code*/  
}
```

\* 若符合 case 1，則就會開始執行下一個 tab，沒有 break，就會一直執行下去  
 \* 可用在 case 後增加程式可讀性  
 \* 適合用在會一直重複出現的值，直接改 define 很方便  
 ex: #define F (5.0f/9.0f)

## goto

```
goto Label1;  
Label1:  
    ...
```

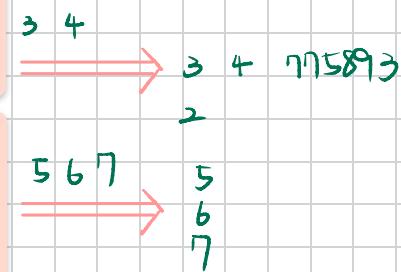
會直接跳到標籤的位置

## scanf

```
int a, b, c;  
int n = scanf("%d%d%d", &a, &b, &c);  
printf("%d %d %d\n%d", a, b, c, n);
```

```
int k;  
while (scanf("%d", &k) != EOF)  
    printf("%d\n", k);
```

此時仍有讀到資料，n 是 EOF



## for

```
for (int i = 1; /*code*/ && /*code*/; i *= 2)  
    /*code*/;
```

此為 NULL statement  
→ (不做任何事)

→ 不要用 //code，:// 會把 // 去掉

Note：在迴圈中宣告的變數無法在迴圈外使用

## assert

assert(條件);

若為 false, 則結束程式

## exit

\* `include <stdlib.h>

exit(0);

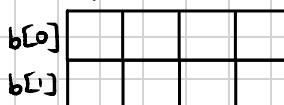
直接結束程式

## 陣列位址

a 陣列



b 陣列



\* 記憶体位址使用十之進位, 格式為 %?

① a, &a, &a[0] 告為此陣列的起始位址

② &a[2] 與 a+2 為同一位址

\* a[2] 是變數, : 它的值不會是位址 (一維陣列下)

&b[1][2], b+1x4+2, b[1]+2 為同一位址

\* 局部變數: 離開函式或迴圈後, 記憶體會被釋放, 意味著記憶體位址消失

## 陣列初始化

int a[7][6] = {{1, 2}, {4}};

int b[7][6] = {{1, 2, 0}, {4, 0, 0}, {0, 0}};

int c[10][10] = {};

a, b 兩陣列元素情形一樣

c 的所有元素為 0。

## 布林陣列

bool a[7] = {};

bool b[7][7] = {};

Designated

int a[5] = { [1] = 2, [3] = 4};

→ {0, 2, 0, 4, 0} ?

皆可設陣列中所有元素為 false

## 全域 & 區域 變數

```
main() {
    int a = 1;
    int b = 1;
    {
        int b = 2;
        {
            int a = 3;
            cout << a << b;
        }
        {
            int b = 4;
            cout << a << b;
        }
        cout << a << b;
    }
}
```

B<sub>2</sub>有一群區域變數

Figure 1.10: Blocks in a C++ program

## 陣列輸入

```
int a[3][4];
for (int i = 0; i < 3; i++)
    for (int j = 0; j < 4; j++)
        scanf("%d", &a[i][j]);
```

## 陣列大小

只能在 main 用

```
bool key[] = {2, 5, 7, 8, 10};
int size = sizeof(key)/sizeof(bool);
```

4

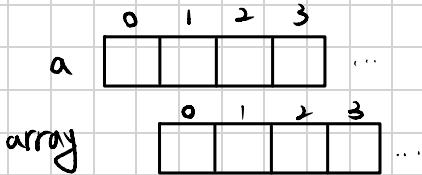
5

## 陣列傳遞

```
int add(int array[]);
int main()
{
    int a[10];
    add(a+1);
}
```

參數傳遞的是「值」,

∴ 陣列傳遞利用「記憶體位址」



array 被視為指標, ∴ 設定 array 的大小無意義

∴ 也不能用 sizeof(array) 計算陣列大小

但若 a[3][4], 而 array [4][3], 就靠記憶體了

## 類別轉換

### 強制轉換

(type) expression

Note: a / 2.0 → a 也會轉成 float

### 被動轉換

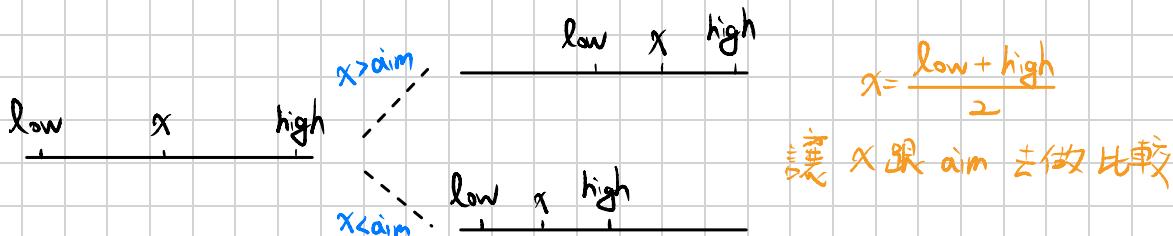
```
int a; float b, c;
c = a / b;
```

∴ float 的等級 > a 的

∴ a 會轉成 float 再運算

## 二分逼近

※ ① low ~ high 需呈連續遞增 ② 反利用指數爆發



⇒ 請回位元

## 指標

```
int i, k, *iptr;
iptr = &k;
*iptr = i;
```

意思一樣

int i;

int \*iptr = &i;

此時 iptr 指向 i

iptr 與 &i 不同，且值也不相等

↓  
i 的記憶体位址 → iptr 本身的記憶体位址

```
int v = 2; int *p = &v;
int **p2 = (&p);
```

指向指標的指標變數要用 \*\* 壓告

## 指標應用

```
void apply(int *p1)
{
...
}
```

```
int main()
{
    int i, *iptr = &i;
    apply(iptr);
}
```

此時 iptr 及 p1  
皆指向 i，但  
iptr 與 p1 仍處在  
不同記憶體位址，  
只有 i 動才互有影響

# 指標與陣列

```
int a[] = {1,2,3,4,5}, *p = a;  
for (int i = 0; i < 5; i++, p++)  
    printf("%d ", *p);
```

⇒ 1 2 3 4 5

→ 以記憶體位址看，就是到了  $1 \times 4$

若為 int 陣列，則此時 p++ 的單位是 sizeof(int) = 4 bytes

“ double 陣列，

“ 的單位是 sizeof(double) = 8 bytes

① 差異在於決定 p+1  
實際上會 shift 多 4 bytes

② 若 p1 指向 int 陣列，

p2 指向 int，則 p1=p2

→ 以記憶體位址看，就是到了  $2 \times 4$

會報錯

## NULL

C 語言中，NULL 為空指標 0x00000000

```
int a[2];
```

```
for (int *p = a; *p != NULL; p++)  
    //code
```

① 會報錯，： \*p 是 int 但 p 超過陣列不會  
實際上是一維陣列 是 NULL

\* 若 a 是二維陣列，則 a[i] 型別是 int\*，指向第二列的首元素

&a[i] 型別是 int (&a[i])，指向整個第二列

a[i]

→ 等同於 a+i

arr[i][j]

\*(&arr+i)+j)

\* 指標陣列的尾端是 NULL

## 相對路徑

\* 以當前目錄為基準點

text.c

./text.c

同一層目錄下

的 text.c 檔

../NCKU/a.txt

上一層目錄下 NCKU 子目錄的 a 檔

```
int foo(int arr[]){
    printf("%d", sizeof(arr));
```

若陣列作為參數傳遞，則此時 arr 是 pointer  
⇒ 8 → 在 32 位元平臺會是 4

```
int(int *P[]);
```

```
int(int **P);
```

注意這是指標陣列嘛，元素是指標，∴用 \*[] or \*\*

## 字元

Note: 可以 char c = 255; printf("%d", c);

\* '\n' 是「一個字元」，'2' 轉成 int 為 50 (+48)

① 佔 8 bits，表示能存 -2<sup>7</sup> ~ 2<sup>7</sup> 之間的整數

② 0~127 的數字皆有字元對應 (one-to-one)，

此時這些數字或字元可用 int 或 char 變數存取，

輸出時用 %d, %c 就分別以 int, char 型態輸出

```
char ch = 65;  
printf("%c %c", ch, ++ch);
```

\* '\' can be used to  
join two or more lines  
into a single line.

## 字元函式

\* include <ctype.h>

isXXX(c)

"是" 就回傳 true

"否" 就回傳 false

tolower('a')

會回傳 A

↳ 非英文字母就回傳一樣東西

函式名稱	分類
isalnum	英文字母或數字
isalpha	英文字母
islower	小寫英文字母
isupper	大寫英文字母
isdigit	數字
isxdigit	十六進位數字
isprint	可顯示字元 (包含空白)
isgraph	可顯示字元 (不包含空白)
isspace	空白
ispunct	標點符號
iscntrl	控制字元
tolower	轉成小寫英文字母
toupper	轉成大寫英文字母

## 字串初始化

```
char w[] = {'m', 'a'};  
char s[100] = {'m', 'a'};
```

\* 空白是一個字元，不是 '\0'

w[2] 是 '\0'

s[2] ~ s[99] 都是 '\0' s 的大小為 100 bytes

\* "abc\n" 为 abcdef ('\' 可跨兩行合併)

## 字串讀取

\* %8s 表讀入  
最多 8 個 character

\* %.6s 表輸出

## scanf

(printf) 前 6 個 character



```
char string[100];  
scanf("%s", string);  
printf("%s", string);
```

scanf 用 %s 讀取時，讀不到 space

## 字元指標

```
char *ptr = "coding";
```

可初始化，但 ↗ 指向的是 c，且無法 printf("%s", \*ptr);

sizeof(ptr) 仍是 8    \*'a' is an integer, "a" is a pointer

## 指標陣列

```
char *table[] = {"ab", "bc", "ca"};
```

table[] 為 "ab" 的記憶體位址

table[] 中元素本身為 數字，不像字串陣列的元素是字元陣列，  
可以做到 任意更換數值 運算 指定為其他變數的值 比大小

Note：每次用函式盡量都要初始化值，否則可能會用上次函式使用的  
記憶體位址中的值

\* fgets() 會讀取結尾的 '\n'，gets() 不會

\* gets(string) has similar functions  
**fgets** but it may cause overflows

```
char s[100];  
fgets(string, sizeof(s), stdin);
```

讀到 '\n' 或字元上限為止；若前面

有用 scanf，則要用 getchar() 吃掉 '\n'

讀到的值不一定要賦值給誰 ↗

# memory allocation

## malloc

malloc(sizeof(data));

向系統要求指定  
大小的記憶體

void \*malloc ( )  
↑  
ex: (struct node \*)

- ① 回傳值為無固定類別的指標，需再轉型
- ② 要求的記憶體不會隨 function 結束而收回
- ③ 沒有用完必須 free 掉

calloc + 初始化為 0

free

free(ptr);

釋放記憶體

realloc 調整已分配的記憶體大小 (原本的值仍在)

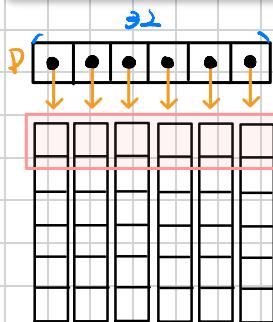
```
int *p = (int*)malloc(3*sizeof(int));
p[0] = 1; p[1] = 2; p[3] = 3;
p = (int*)realloc(p, 5*sizeof(int));
p[4] = 4; p[5] = 5;
```

→ int \*p = calloc(3, sizeof(int))

## ※ 二維以上陣列宣告方法

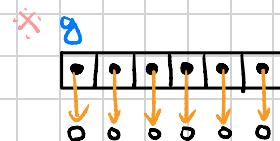
```
Node ***P = malloc(32*sizeof(Node**));
for (int i = 0; i < 32; i++)
    P[i] = malloc(256*sizeof(Node*));
```

此陣列中每個元素  
皆為 pointer to Node



記憶體位址  
是上面陣列  
元素的值

每個元素又是  
a pointer



- ① 其中每個元素的值都是 Linkedlist 的 head
- ② head 的位址是連續的

## 字串函式

必引入 <string.h> 必先宣告的變數 記憶體位址較大

strlen(string);

計算至遇到第一個 '\0' 為止 (不含 '\0')

strcpy(destination, source);

把 des 的字換掉, 以 source 替代, 直到遇到 source 裡的 '\0'

strncpy(destination, source, n);

strncat(destination, source, n);

strcat(destination, source);

把 source 接在 des 後面

以 des 的第一個 '\0' 開始, 覆蓋 source 長度的字串 (接續), 回傳 des 的位址

strcmp(string1, string2);

從首字元開始比較 ASCII 碼大小, 如果相同,  
就比下個字元, 若 string1 > string2, 則回傳正數;  
若 string1 < string2, 則回傳負數;  
若 string1 = string2, 則回傳 0。

strchr(string, c);

從首字元開始找字元 c, 若有找到, 則回傳第一個 c  
的記憶體位址; 若無, 則回傳 NULL

strrchr(string, c);

從末字元 ...

記憶體位址



後宣告 ←

先宣告 ←

若  $p1 = \text{strchr}(\text{string1}, 'c')$ ,  
則  $\text{strchr}(p1, 'c')$  結果為 NULL  
⇒ 還是會在同字串中搜尋

## 字串函式

strstr(string1, string2);

從首字元開始在string1中找string2，有就回傳string2出現的起始位址；反之，回傳NULL

strspn(string, chars); 回傳從string的首字元開始有多少個連續字元在chars中

strspn ("acbbef", "bac") 為 4

strcspn(string, chars);

... 有幾個連續字元沒出現在chars中

strcspn ("acbbef", "fe") 為 4

advanced ↳ 可列出所有英文字母，得出有幾個連續非英文字母

```
char *start = strtok(string, delimeters);
while (start != NULL){
    /*process sting at start*/
    start = strtok(NULL, delimeters);
}
```

## char \*p v.s. char s[]

```
char *p = "abc";
char s[] = "def";
```

- ① "abc"的位址有可能在不可改寫的地方，...又稱讀
- ② "def"在此被視為初始化的東西
- ③ P要指向一個有名寫的字串才能改寫

其中 delimeters 為字串

string                    delimeters  
" acbbef"              " bf"

strtok ↓

start 依次為 "ac", "e"

且 strtok() 會幫 start 末端補 '\0'

## 宣告修飾

### int

```
short i; printf("%hd", i);
```

→ 最小範圍

$-2^{16} < i < 2^{16}-1$

: 每台機器規格不同

```
long j; printf("%ld", j);
```

$-2^32 < j < 2^{32}-1$

```
long long k; printf("%lld", k);
```

$-2^{64} < k < 2^{64}-1$

Note: sizeof() 的回傳值型態為 long int

### const

```
const double pi = 3.1415926;
```

pi 的值若變了，編譯器會提示

以此類推（含陣列、參數）

## 指標

```
int k;  
const int *p1;  
int *const p2;  
const int *const p3;
```

(\*p1)++; 會出錯，但用 \*p1 以外的方式改 k 值可以

p2 不能改

\*p3 和 p3 都不能改

## static

※ 若將 global variable 加上 static，則該 variable 是會 visible 在

```
void foo()  
{  
    static int k = 0;  
    k++;  
    printf("%d ", k);  
}
```

Note: static 不可用在參數列

這個檔案

呼叫 foo() 五次

→ 1 2 3 4 5

static 變數

① 可存在至整個程式結束

② 只可被初始化一次

③ 若在 block 內，它還是局部變數

Note: Most Significant Bit / Least Significant Bit

## 位元運算

char value = 49, t1 = 8;  
char num = 11, t2 = 127;

value		7 6 5 4 3 2 1 0
t1		
num		
t2		MSB                    LSB

回動作檢查用

value & t1

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

=0

value | t1

0	0	1	1	1	0	0	1
---	---	---	---	---	---	---	---

test 有1,value 沒1的就槓掉

value ^ num

0	0	1	1	1	0	0	1
---	---	---	---	---	---	---	---

位元一樣就0, 相異就1

num ^ t2

0	1	1	1	0	1	0	0
---	---	---	---	---	---	---	---

t2中1的地方反轉,  
0的地方保留

.....

\* 在8bit integer中, 10000000是-128

負數數值算法(0當成1) (負數才會+1)

↓    -7

MSB: 0表正數 1表負數	反轉	+1	-7
-------------------	----	----	----

要知道值的話先-1

## 數字移位

左移

LSB一律補0

↕ 也就是  $\times 2$

$a \ll= 1;$

右移

正數之MSB補0, 負數之MSB補1

$\gg 1$

$a \gg= 1;$

$\gg 1$

$\rightarrow$  不是  $\div 2$

## xor-swap

\*自己跟自己

xor會變0喔

$*x \wedge = *y;$

$*y \wedge = *x;$

$*x \wedge = *y;$

Let  $t = x \wedge y$

$\therefore \wedge$  擁有對稱性

$t \wedge x = y$

$t \wedge y = x$

# struct

## 定義

```
struct file {
    char name[10];
    int id;
    float grade[3];
};
```

(不用 'file')  
Ann 放在 ; 前宣告也行  
↑

## 宣告

- ① 沒初始化就補 0
- ② 也能用 designated initializer

```
struct file Ann =
```

```
{.id = 45511, .grade = {3.1, 2.0, 1.1}};
```

struct file 相當於是個資料類別

## offsetof

\* include <stddef.h> ↗ Ann 大小會多出 2 位元組，因 grade 的  
起始記憶體位址為 4 的倍數

```
printf("%ld", offsetof(struct file, name));
printf("%ld", offsetof(struct file, id));
printf("%ld", offsetof(struct file, grade));
```

⇒ 0 12 16

回傳各欄位離起點累占的位元組數

## 副函式

```
struct file function (struct file *Andrew, struct file Jessica)
{ ... return Ann; }
```

- ① 回傳值類別可為 struct
- ② 呼叫 function 時，將值複製進 Jessica 中，因 Jessica 的記憶體位址與其不同
- ③ 複製 struct 很費時，建議用指標參數

## 欄位

沿用<副函式>

Andrew->id

Jessica.id

↳ 一般 & 指標  
兩用

↳ 一般用

## 特性

\* struct 裡放 array 也行哦

```
struct file a, b, temp;
temp = a;
a = b;
b = temp;
```

struct 資料  
是一個值，  
具賦予功能

\* '==' '!= ' 等等不能用

# typedef

```
typedef struct number {  
    int value;  
    int bit;  
} Number;  
  
typedef struct numberSet {  
    Number nums[100];  
    int count;  
} NumberSet;
```

- ① 把 struct number 簡化成 Number
- ② struct 裡也能有 struct
- ③ 也能宣告 struct 陣列

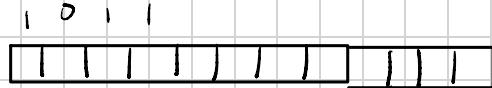
# define

```
#define SUB(x, y) (x - y)  
#define SWAP(x, y) int tmp = x; x = y; y = tmp;  
#define MULT(x, y) (x) * (y)
```

\* #define SWAP(x,y) \  
int tmp=x; \  
x=y; \  
y=tmp; \  
    >反斜線  
        可換行

↳ : 可能遇到 MULT(1+1, 2+2) 寫被視為 (1+1) · (2+2), 沒有()就gg了

```
#ifndef BOOLEAN_H  
#define BOOLEAN_H  
#define TRUE 1
```



# union

① struct 的修訂版 ④ 目的為節省空間

```
union {  
    char title[10];  
    char content[10];  
};
```

title 和 content  
會佔據相同  
位置的記憶體

```
union {  
    int i;  
    float f;  
    double d;  
} u;
```

d  
-----  
j (f)  
u.f = 3.0; int j = u.i;  
此時 j 的 bit pattern 並非  
f 作為 float 時的 bit pattern

## enum

enum {

A, B = 8,

C = 3, D

} tag;

\* enum 裡的 tag 可以被 debugger 看到  
但 define 就不行

- ① A 預設為 0
- ④ 跟 int 是完全相通的
- ② D 為 C + 1
- ③ 只能用在 int

switch (tag){  
case A: case B  
}

跟 switch  
很搭

## 九宮格排列

\* portal: permutation.c

每個數字都排過每一位, ∵ 有  $9!$  ?

也可以用實際 count 的方式, 來判斷到底是不是對的

## Random

\* include <stdlib.h> include <time.h>

srand((unsigned)time(NULL));  
int a = rand() % 12 + 1;

- ① 隨機生成 1 ~ 12
- ③ 不放 NULL, 也得放 0
- ② srand(seed): seed 也可以自己輸入

## Quicksort

13 8 2 11 9 7 7 → 8 2 9 7 11 13 9  
 (pivot)  
 以 13 為基準      比 13 小的排左邊  
 比 13 大的排右邊

⇒ 左、右側分別做一樣的事，直到全部的兩邊皆剩 1 個數字

\* 排左、右側的方法：填滿從 0 開始到小於 pivot 的總數 - 1 的索引

## Extern

① .c/.h 檔皆可放 ② 聲明某個在某個.c 檔內宣告的變數是全域變數

↳ extern 聲明不能初始化

## Static

① 若放在 .h 檔，則聲明該變數給每個 .c 檔獨立使用



static 聲明可初始化

## printf

\* 注意 printf() 輸出 %s 只到第一個 '\0' 而已 \* 先 printf 在左邊!

```
int d = 23, j;
int a = printf("d = %05d\n", d, &j);
printf("%d %d", j, a);
```

$\Rightarrow \begin{array}{r} d = 00023 \\ 9 \ 10 \end{array}$

j 的值為 %0n 前面輸出的字元個數  $\Rightarrow$  這個我用怪怪的  
a 的值為 該 printf 輸出的字元個數

printf("%(-)m.pX", ..);      m specifies the minimum number of characters  
                                        p specifies the 預設

%e : 將數字表示成科學記號，若 p=0，則小數點後有 6 位小數

%g : Choose either %e or %f to output the variable

\t : 讓字對齊上面那行

printf("Item\tUnit\tPurchase\n\tPrice\tDate");

$\Rightarrow \begin{array}{l} \text{Item} \quad \text{Unit} \quad \text{Purchase} \\ \text{Price} \quad \text{Date} \end{array}$

\b : 使光標退回前一格，若其後還有輸出，則會吃掉輸出會佔據的字元

printf("Hello\b\bW");  $\Rightarrow \text{He}\text{W}$

## scanf

\* %e.%f.%g are interchangeable when used with scanf

scanf ignores white-space characters (' ', '\t', '\v', '\f', '\n', '\r')

\* scanf 讀 %s 時，會 skip 空白字元

ex: 100004 0 1000  
          ——  ——

scanf("%d/%d", &i, &j);  $\begin{array}{r} 5 / 9 \\ \downarrow \end{array}$

讀到 5 後一定要讀 '/'  $\downarrow \hookrightarrow$  下一個要讀 integer，而 '/' 為 white-space

$\downarrow \hookrightarrow$  \* scanf 讀 %c 時，不會 skip 空白字元 character

## sscanf

```
sscanf(str, "%d %*d %f", &a, &b);  
printf("%d %f", a, b);
```

2 5 3.14  
→ 2 3.14

- ① 從字符串中提取指定格式數值 ② \*可忽略該項的讀取 ③ return成功讀取數量

## sprintf

```
s(n)printf(str, (n), "Name: %s", "Alan");
```

- ① 以格式化字符串覆蓋(從%開始) str, 並自動添加'\0', 後面沒覆蓋到的字元不會管  
② return 輸出的字元數

# Integer

int must not be shorter than short int

long int must not be shorter than int 2's complement 有把負數往後平移

MSB [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] LSB (16-bit)

↳ sign(0:+, 1:-) 剩下15格能表示的 Max =  $2^0 + 2^1 + 2^2 + \dots + 2^{14} = 2^{15} - 1$   
總共能表示 2<sup>16</sup> 種組合, ∵ min =  $-[2^{16} - (2^{15} - 1) - 1] = -2^{15}$

\* unsigned 就是連 sign 那格都拿去表示數值 (格式為 %u)

\* int 轉 unsigned int 是直接將位元組合複製過去 ex: -12345 → 53191

```
int n = 10; unsigned int a = 1;  
for (int i = n - 1; i - a >= 0; i--)  
    printf("i: %x\n", i);
```

會陷入無窮迴圈

∴ i-a >= 0 全程會以 unsigned int 的角度去處理, ∵ 恒 > 0

machine	range		
	short	int	long
16-bit	$[2^{-15}, 2^{15}-1]$	$[2^{-15}, 2^{15}-1]$	$[2^{-32}, 2^{32}-1]$
32-bit	$[2^{-15}, 2^{15}-1]$	$[2^{-32}, 2^{32}-1]$	$[2^{-32}, 2^{32}-1]$
64-bit	$[2^{-15}, 2^{15}-1]$	$[2^{-32}, 2^{32}-1]$	$[2^{-64}, 2^{64}-1]$

## Octal

\* %o 是用在指標

① 以 0~7 表示 ② 開頭一定是 0 ③ 格式 %o ex: 017  $\Rightarrow 7 \times 8^0 + 1 \times 8^1 = 7 + 8 = 15$

## Hexadecimal

\* decimal + 進位, 格式 %u

① 以 0~9、A~F 表示 (大小寫沒差) ② 開頭一定是 0x ③ 格式 %x

## letter

對整數後端加上 L, 視為 long; ... 加上 U, 視為 unsigned ex: 0xffffUL

# Float

第n格

	Sign	Exponent	Fraction	Bias
float	32	31~24 8	23~1 23	127
double	64	63~53 11	52~1 52	1023

Stored exponent

$$= \text{real exponent} + \text{bias}$$

※ bias 的意義在於，使 exponent 為正數

而 float 比大小時可以直接比較

... (負數二進位不好比)

0 0111011 1001100100 110011001101

expression of 0.1  $\Rightarrow 1.1001\dots \times 2^{-4}$

$2^n$	n	第一個小於 1 的數是 $2^{-4}$ ,
0.5	-1 0	由 -4 往下
0.25	-2 0	
0.125	-3 0	∴ 一定要是 1...
0.0625	-4 1	↓
0.03125	-5 1	: 這個
0.015625	-6 0	-一定存在
0.0078125	-7 0	: 被省略
0.00390625	-8 1	

expression of 100

$$100 = 2^6 + 2^5 + 2^2$$

$$\Rightarrow 1.1001 \times 2^6$$

有效位數  
↑

Range

Precision

	float	double
	$1.17 \dots \times 10^{-38} \sim 3.40 \dots \times 10^{38}$	$2.22 \dots \times 10^{-38} \sim 1.79 \dots \times 10^{38}$
	6	15

$$\times 23 \times 0.301 = 6.923, 52 \times 0.301 = 15.652$$

資料型態	格式化符號	描述
<b>整數類型 (Integer Types)</b>		
int	%d 或 %i	有符號十進位整數
unsigned int	%u	無符號十進位整數
short	%hd	有符號短整數
unsigned short	%hu	無符號短整數
long	%ld	有符號長整數
unsigned long	%lu	無符號長整數
long long	%lld	有符號長長整數
unsigned long long	%llu	無符號長長整數
int8_t (來自 <stdint.h>)	%d	8 位元整數
int16_t (來自 <stdint.h>)	%d	16 位元整數
int32_t (來自 <stdint.h>)	%d	32 位元整數
int64_t (來自 <stdint.h>)	%lld	64 位元整數
uint8_t (來自 <stdint.h>)	%u	8 位元無符號整數
uint16_t (來自 <stdint.h>)	%u	16 位元無符號整數
uint32_t (來自 <stdint.h>)	%u	32 位元無符號整數
uint64_t (來自 <stdint.h>)	%llu	64 位元無符號整數
<b>浮點數類型 (Floating-Point Types)</b>		
float	%f	十進位浮點數 (預設小數點後 6 位數字)
double	%lf 或 %f	十進位浮點數
long double	%Lf	長浮點數
float / double	%e 或 %E	科學記數法表示法
float / double	%g 或 %G	根據數值大小自動選擇 %f 或 %e
<b>字元與字串類型 (Character and String Types)</b>		
char	%c	單一字元
char[]	%s	字元陣列 (字串)
<b>指標類型 (Pointer Types)</b>		
void*	%p	指標 (以十六進位格式輸出)
<b>十六進位與八進位 (Hexadecimal and Octal)</b>		
int 或其他整數	%x	無符號十六進位 (小寫字母 a-f)
int 或其他整數	%X	無符號十六進位 (大寫字母 A-F)
int 或其他整數	%o	無符號八進位
<b>其他控制符</b>		
%%	%%	輸出 % 字符
字元數計數	%n	將目前輸出的字元數存入變數 (指標)

# file

## open / close

FILE \*fp;

fp = fopen(filepath, option);  
fclose(fp);

① 記得關檔案

② a為從檔案尾  
往後寫

r+不會清除內容，而是覆蓋內容

開檔模式字串	動作
"r"	從檔案頭讀取檔案，檔案必須存在
"r+"	從檔案頭讀取或寫入檔案，檔案必須存在
"w"	從檔案頭寫入檔案，檔案存在則將其內容清除，否則建立新檔案
"w+"	從檔案頭讀取或寫入檔案，檔案存在則將其內容清除，否則建立新檔案
"a"	從檔案尾端寫入檔案，檔案存在則保持其內容，否則建立新檔案
"a+"	從檔案尾端讀取或寫入檔案，檔案存在則保持其內容，否則建立新檔案

## fgetc / fputc

除非

n = fgetc(fp);  
fputc(n, fp);

fgetc() 讀取 fp 的一個字元，下一次讀取為下一個字元，  
close 後再 open，才會再從第一個開始；

↳ n 為 int

fputc 具寫入功能

\* 檔案中的資料有型態區分，最終 fgetc() 的回傳值必為 int

↳ ex: 若讀到 '2', n=50; 若讀到 2, n=2

## fgets / fputs

fgets(string, num, fp);  
fputs(string, fp);

fgets() 將讀取的資料存入 string, num 為  
放入 string 的最大字元數（含 '\0'），並回傳 string

## fprintf / fscanf

fprintf(fp, format, ...);  
fscanf(fp, format, ...);

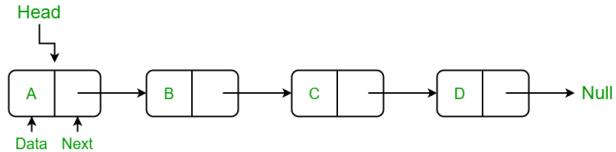
寫入特定格式的字串

讀取特定格式的字串

↳ 仍有 scanf 的性質 ex: skip white-space characters

## Linked List

必須記得沒事的節點都要初始化成NULL



\* Next 指向的是下一個 struct

必須知道 head 很重要

## Generate Node

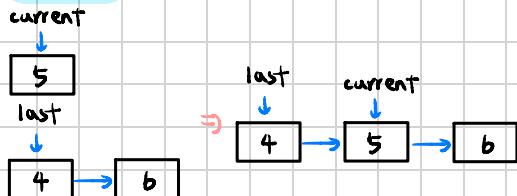
```
ListNode *genNode(int data, ListNode *next){  
    ListNode *node = (ListNode *)malloc(sizeof(ListNode));  
    assert(node != NULL);  
    node->data = data;  
    node->next = next;  
    return node;  
}
```

## 保留節點

```
for(ptr = head; ptr != NULL; previous = ptr, ptr = ptr->next);
```

insert

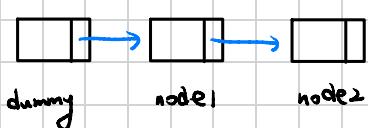
## 文基楚作法



```
current->next = last->next;  
last->next = current;
```

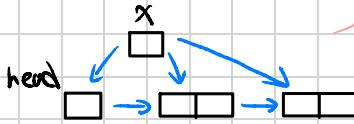
必须要建好 Linked list 好像是 generate() + insert()

Dummy



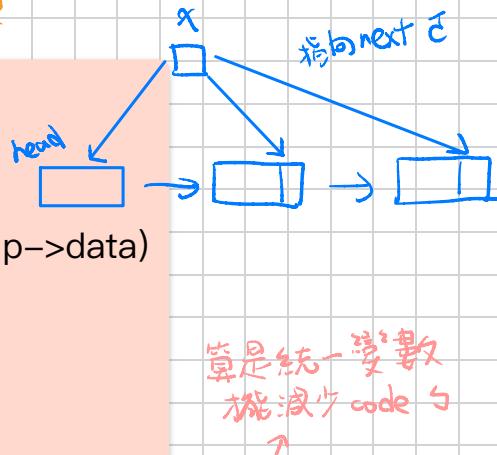
- ① dummy 始終是 Linkedlist 的起始點
  - ② 避免  $p == \text{head}$  的 special case
  - ③ dummy 可以存東西

## insert



太過複雜作法  
透過一個媒介  
可省下很多麻煩

```
void insert(node **head, node *p){
    node **x = head;
    if (p == NULL) return;
    while ((*x) != NULL && (*x)->data < p->data)
        x = &((*x)->next);
    p->next = (*x);
    (*x) = p;
}
```



$\because x$  指向 node 而不是指向 pointer 同， $\therefore$  也要有 head 的 address  
且有  $\exists$  head 的 address，可以直接改 Linked list 的 head  
但注意一下，在 function 中  $\&head$  會是區域變數的 address

## delete

```
node *delete(node **head, int d){
    node *t = *head, *prev = NULL;
    while(t != NULL && t->data != d){
        if (t->data > d) return NULL;
        prev = t; t = t->next;
    }
    if (t == NULL) return NULL;
    if (prev == NULL){*head = t->next; t->next = NULL; return t;}
    prev->next = t->next; t->next = NULL; return t;
}
```

## Warning

- ① 若  $p$  為 NULL，則 ①  $*p$  ②  $p \rightarrow \text{next}$  皆會 result in Segmentation fault
- ②  $\text{for } (a; b; c)$  的執行順序為  $a, b, c, b, c, b, c$  以此類推，  
∴ 注意最後做的  $c$  可能因為不符合條件而使程式報錯  
這也是與 while 的不同 (while 錄靠 if-else 決定  $c$  什麼時候執行)
- ③ 若  $p$  尚未指向變數，則  $*p$  會 result in Segmentation fault
- ④  $p$  和  $*p$  的資料型別差很多喔！
- ⑤ 注意是否為區域變數會不會影響

## Pointer to Function

```
int (*pf) (double);    void (*file_cmd[9]) (void);
```

pf is a pointer to a function

file\_cmd is a pointer array

```
int f (double a);  
pf = f; pf(2.0);
```

指令也是在 memory 裡，`func` 是 function f 的起始位址  
`pf` 指向 `func`，可以將呼叫

```
typedef int (*pfdd) (double);
```

此時 `pfdd` 是一個 type (a pointer to function)

```
typedef double dbl;
```

此時 `db` 是一個 `type` (`double`)

## Qsort in <stdlib.h>

\* 請記得有 `qsort` 能用

```
void qsort(void *base, size_t nitems, size_t size,  
int (*compar)(const void *, const void*))
```

**base** is the start memory of the data

**n\_items** is the number of elements to be sorted

**size** is the size of an element

**compare** is a pointer to the comparison function

为什么要自己寫？ ↳ 專門比 string, int & float, struct 等等的 functions

cdabbbab

i=2

$$y = x + 2$$

$$y = 5x + 9$$

10

$$10.549 - 359$$