數據類型Data Type

int 整數 占用4位元組Byte或32位元bit，只可以表示-2^31~2^31-1，大約-20億~20億

char 字符 1 Byte，例如a ，只可以表示-128~127

float 浮點數或實數 4 Bytes，可以直觀地看作小數

double 雙精度浮點數 8 Bytes

unsigned 無符號數 只能表示非負數

unsigned int 表示0~2^32-1

const 常數 程序永遠不能更改的值

const int 一般名字是大寫

#include <cs50.h>

bool 布林 1 byte，只有兩種值，通常是真true和假false

long整數 8 bytes

string 字串 字串實際上只是字符陣列，以空字符'\ 0'結束，即”hi”需要3 Bytes

字串 = 一維字符陣列

string names;

names = "hi";

等同

char names[2];

names[0] = "h";

names[1] = "i";

字串陣列 =二維字符陣列

string names[4];

names[0] = "EMMA";

names[1] = "RODRIGO";

names[2] = "BRIAN";

names[3] = "DAVID";

printf("%s\n", names[0]);

// EMMA

printf("%c%c%c%c\n", names[0][0], names[0][1], names[0][2], names[0][3]);

// E M M A

類型Type

Void 空 代表不需要任何參數

變量Variable=數據類型Data Type 名字Name;

int number；

char letter；

可用同一行數據類型，創造多個相同數據類型之變量

int hight，width；

float sqrt2，sqrt3，pi；

只需聲明一次變量之數據類型

int number；

number=17；

char letter；

letter='h'；

可在變量聲明數據類型同時賦予值

int number=17；

char letter='h'；

運算符Operators

數值運算符

+ 加

x=x+1；等於x+=1；等於x++；

- 減

x=x-1；等於x-=1；等於x--；

\* 乘

x=x\*5；等於x\*=5；

/ 除

% 餘數

int x=13%4； //x等於1

邏輯運算符

&& 且

(x&&y) //x和y需皆為真true，才為真true

|| 或

(x||y) //x或y其中只要一個為真true，就為真true

! 不

x=true

!x=false

關係運算符

>大於

>=大於或等於

<小於

<=小於或等於

==等於

!=不等於

布爾表達式Boolean expressions

在C語言中0為假false，其餘皆為真true

條件語句Conditional Statements

IF語法

執行或不執行括弧中內容

if (Boolean expression) //如果布爾表達式為真，執行括弧中內容

{

}

執行第一個或第二個括弧中內容(2選1執行)

if (Boolean expression) //如果布爾表達式為真，執行第一個括弧中內容

{

}

else //不然執行第二個括弧中內容

{

}

執行第一個或第二個或第三個或第四個括弧中內容(4選1執行)

if (Boolean expression1) //如果布爾表達式1為真，執行第一個括弧中內容

{

}

else if (Boolean expression2) //不然如果布爾表達式2為真，執行第二個括弧中內容

{

}

else if (Boolean expression3) //不然如果布爾表達式3為真，執行第三個括弧中內容

{

}

else //不然執行第四個括弧中內容

{

}

執行或不執行第一個括弧中內容(獨立判斷)

執行或不執行第二個括弧中內容(獨立判斷)

執行第三個括弧中內容或執行第四個括弧中內容(2選1執行)

if (Boolean expression1) //如果布爾表達式1為真，執行第一個括弧中內容

{

}

if (Boolean expression2) //如果布爾表達式2為真，執行第二個括弧中內容

{

}

if (Boolean expression3) //如果布爾表達式3為真，執行第三個括弧中內容

{

}

else //不然執行第四個括弧中內容

{

}

SWITCH語法

int x=get\_int();

switch(x)

{

case1:

printf("one/n")； //x=1執行

break； //結束

case2:

printf("two/n")； //x=2執行

break； //結束

default:

printf("sorry/n")；

}

?:三元運算符

int x=(Boolean expression) ?5:6；

等同

int x；

if (Boolean expression)

{

x=5；

}

else

{

x=6；

}

循環Loops

WHILE語法

先判斷後執行，常用於重複進行

while (Boolean expression) //如果布爾表達式為真，重複執行括弧中內容，直到布爾表達式為假

{

}

DO WHILE語法

先執行後判斷，至少執行一次，常用於使用者輸入

do

{

}

while (Boolean expression)； //如果布爾表達式為真，重複執行括弧中內容，直到布爾表達式為假

FOR語法

常用於重複一定次數

for (int i = 0; i < 10; i++) //i一開始為0，如果i < 10為真，執行括弧中內容，且i++，直到i < 10為假

{

}

函式Function=返回類型Return type 名稱Name(參數列表Argument list);

float mult\_two\_reals(float x, float y); //聲明函式

int main(void)

{

float a=1,b=2,c;

c=mult\_two\_reals(a, b);

}

float mult\_two\_reals(float x, float y) //定義函式功能

{

float product=x\*y; //或return x\*y;

return product;

}

陣列Array=類型type 名稱name[大小size];

int numbers[40];

有40個空間，但第一個空間是numbers[0]，第四十個空間是numbers[39]

bool logic[3];

logic[0]=false;

logic[1]=true;

logic[2] =true;

或bool logic[3]={false, true,true};

或bool logic[]={false, true,true};，因為依據後面三個值，可知大小為三，故可不宣告

int numbers[10] [10];

10\*10的二維陣列

int foo[3] ={1, 2,3};

int bar[3];

for (int i = 0; i < 3; i++)

{

foo[i]= bar[i];

}

在C中數列必須一個個元素進行，不行整個數列進行，如foo= bar;

命令列引數Command Line Arguments

int main(int argc,string argv[])

{

}

argc是引數計數，紀錄命令列引數數量，argv[]是引數數列，紀錄命令列引數內容，每個引數之間以空白作區隔

prog hello.c -o -c project.p

argc 是5

argv[0] 是 "prog"、 argv[1] 是 "hello.c"、argv[2] 是 "-o"、argv[3] 是 "-c"、argv[4] 是 "project.p"

argv[0]是第一個命令列引數，argv[argc-1]是最後一個命令列引數

變量Variables and 作用域Scope

局部變量Local Variables

在函式中創建，只能在該函式中使用

全域變數Global Variables

在函式外創建，可以在任意函式中使用

線性搜索Linear Search O(n)、Ω(1)

按一定的順序檢查陣列中每一個元素，直到找到所要尋找的特定值為止

二分搜尋Binary Search O(log n)、Ω(1)

限制條件:有序陣列

從陣列的中間元素開始，如果中間元素正好是要搜尋的元素，則搜尋過程結束；如果某一特定元素大於或者小於中間元素，則在陣列大於或小於中間元素的那一半中搜尋，使每一次搜尋範圍縮小一半，重複至找到要搜尋的元素或某一步驟陣列為空，代表找不到

選擇排序Selection Sort O(n^2)、Ω(n^2)

在未排序陣列中找到最小值元素，並與未排序陣列中第一個值交換，同時標示為已排序陣列元素，持續對越來越少的元素重複上面的步驟，直到所有元素均已排序，最後得到一個由小至大的有序陣列

冒泡排序Bubble Sort O(n^2)、Ω(n)

從第一對開始比較每一對相鄰的元素，如果第一個比第二個大，就交換，直到最後一對，最後的元素會是最大的數，剔除最後一個並持續對越來越少的元素重複上面的步驟，直到沒有任何一對數字需要交換，最後得到一個由小至大的有序陣列

插入排序Insertion Sort O(n^2)、Ω(n)

從第一個元素開始，第一個元素可以認為已經被排序，對於未排序資料，在已排序陣列中從後向前掃描，找到相應位置並插入已排序陣列，直到最後一個元素，最後得到一個有序陣列

遞迴Recursion

通過重複將問題分解為同類的子問題而解決問題的方法，可以利用函式function呼叫自身來進行遞迴，以階乘n!為例

n!=1×2×3×...×n

以遞迴方式定義

1!=1

n!=(n-1)!×n

在C中呈現為

int fact(int n)

{

if (n == 1)

{

return 1;

}

else

{

return n \* fact(n - 1);

}

}

合併排序Merge Sort O(n log n)、Ω(n log n)

遞迴地把當前陣列平均分割成兩半，直到皆為一元素子陣列，在保持元素順序的同時將上一步得到的子陣列合併，直到為一陣列

結構體Structs

可用來創建自定義數據類型Data Type

typedef struct

{

string name;

string number;

}

person;

int main(void)

{

person people[2];

people[0].name = "EMMA";

people[0].number = "617–555–0100";

people[1].name = "RODRIGO";

people[1].number = "617–555–0101";

}

十六進制Hexadecimal

為了簡化2進位數字，我們採用十六進制，將1到16用數字0到9和字母A到F表示， A到F相當於十進位的10到15，因為4個位元bit可以很簡單以一個16進位數字表示，但是這種混合表示法容易令人混淆，因此在C中以0x表示為16進位數字

二進制0000 = 十進制0 = 十六進制0x0

1001 = 9 = 0x9

1111 = 15 = 0xF

0x397

=16^2\*3+16^1\*9+16^0\*7

=256\*3+16\*9+1\*7

=919

0XADC

=16^2\*10+16^1\*13+16^0\*12

=256\*10+16\*13+1\*12

=2780

指標Pointer

用一個值來儲存另一個值的位址

&代表獲取此變量的地址get address，\*代表去go to

如果沒使用的指標須為空指標NULL Pointer

#include <stdio.h>

int main(void)

{

int n = 50;

//0x7ffe00b3adbc

printf("%p\n", &n);

//50

printf("%i\n", \*&n);

int \*p = &n;

//0x7ffe00b3adbc

printf("%p\n", p);

//50

printf("%i\n", \*p);

}

陣列的名字即是指向陣列中第一個元素的指標

array1[10]中array1紀錄了第一個元素array1[0]的位址

CS50庫定義了一個新類型string，作為指向字串中第一個字符的指標，string s中的s紀錄了第一個字符s[0]的位址，藉由printf函式中的%s和 \0來指示字符串的結尾，來讀取整個字串

#include <cs50.h>

#include <stdio.h>

int main(void)

{

string a = "EMMA";

// EMMA

printf("%s\n", a);

char \*b = "EMMA";

// EMMA

printf("%s\n", b);

}

傳值pass by value

代表將舊變數的數據複製一份副本，放進一塊新的記憶體，並讓新變數指向新的記憶體位址，存放的是兩份獨立的資料，其中一份改變，不會影響另一份

在C函式中都是傳值pass by value

傳址Pass by reference

代表新變數會直接指向舊變數的記憶體位址，等於新舊變數共用同一個位址的資料，其中一份改變，另一份也會跟著改變

動態記憶體分配Dynamic Memory Allocation

在C中可以用malloc和free函式進行動態記憶體分配

malloc函式

找到所需相連大小內存並回傳位址，或找不到內存回傳NULL

//一個int需要四個相連的Bytes

int \*px = malloc(4);

或

int \*px = malloc(sizeof(int));

free函式

釋放佔用內存

char \*word = malloc(50 \* sizeof(char));

free(word);

內存(RAM)由上至下分別為

程式碼text

存放的是可執行的 CPU 指令，例如編譯程序後的二進制代碼，即機器碼machine code，當我們運行程序時，該代碼將被加載到內存的頂部

初始化靜態變數initialized data

儲存的是一些已經初始化的靜態變數，例如有經過初始化的C語言的全域變數global variables以及靜態變數

未初始化靜態變數uninitialized data

儲存尚未被初始化的靜態變數，而這些變數在程式執行之前會被系統初始化為0或是null

棧stack

用於儲存函式的區域變數，以及各種函式呼叫時需要儲存的資訊，例如函式返回的記憶體位址還有呼叫者函式的狀態等，每一次的函式呼叫就會在堆疊區段建立一個 stack frame，儲存該次呼叫的所有變數與狀態，這樣一來同一個函式重複被呼叫時就會有不同的 stack frame

堆heap

用於儲存動態配置的變數，在C中可以用malloc得到空的記憶體，在程序中使用

命令列引數Command Line Argument與環境變數Environment Variable

儲存一些命令列參數與環境變數，這部分會跟系統有關

呼叫堆疊Call Stacks

儲存有關正在執行的子程式的訊息的堆疊，以存放子程式的返回位址為例，在呼叫任何子程式時，主程式都必須暫存子程式執行完畢後應該返回的位址，如果被呼叫的子程式還要再呼叫其他的子程式，其自身的返回位址就必須存入呼叫堆疊，在其自身執行完畢後再行取回。在遞迴程式中，每一層次遞迴都必須在呼叫堆疊上增加一條位址，因此如果程式出現無限遞迴或僅僅是過多的遞迴層次，呼叫堆疊就會產生堆疊溢位

文件指標File Pointers

fopen函式

打開一個文件並給予指向該文件之文件指標

r=read=讀取模式

w=write=寫入模式，如果打開文件中有內容，將重頭複寫蓋過原本內容

a=append=附加模式，如果打開文件中有內容，將從原本內容之後追加內容

//建立新文件並打開

//文件名字file1.txt執行動作r

FILE \*ptr = fopen(“file1.txt”,”r”);

//建立多個新文件並打開 文件位址dictionary執行動作r

//sprintf函式打印內容成為字串

//儲存位址filename 000.jpg-999.jpg

for(int i = 0; i < 1000; i++)

{

sprintf(filename, " %03i.jpg", i);

FILE \* fileptr = fopen(filename, ”w”);

}

fclose函式

藉由文件指標關閉一個文件

//文件指標ptr

fclose(ptr);

fgetc函式

讀取並返回文件中下一個字符

char ch;

//文件指標ptr

// EOF = End Of File

//印出整份文件內容

while((ch = fgetc(ptr)) != EOF)

{

printf(“%c”,ch)

}

fputc函式

寫入一個字符到文件中

char ch;

//文件指標ptr ptr2

//文件指標ptr 指向的文件複製整份文件內容至文件指標ptr2 指向的文件

while((ch = fgetc(ptr)) != EOF)

{

fputc(ch,ptr2)

}

fread函式

從文件讀取Bytes

int arr[10];

//儲存信息地指標arr

//信息單元大小sizeof(int)

//信息單元數量10

//讀取信息地指標 ptr

fread(arr, sizeof(int), 10, ptr);

//儲存信息地指標&c

char c;

fread(&c, sizeof(char), 1, ptr);

//動態記憶體分配

double \*arr2 = malloc(sizeof(double) \* 80);

fread(arr2, sizeof(double), 80, ptr);

fwrite函式

將Bytes寫入文件

int arr[10];

//讀取信息地指標arr

//信息單元大小sizeof(int)

//信息單元數量10

//儲存信息地指標 ptr

fwrite (arr, sizeof(int), 10, ptr);

//讀取信息地指標&c

char c;

fwrite (&c, sizeof(char), 1, ptr);

//動態記憶體分配

double \*arr2 = malloc(sizeof(double) \* 80);

fwrite (arr2, sizeof(double), 80, ptr);

fscanf函式

從文件獲取輸入

fgets函式

從文件讀取整個字串

fputs函式

寫入整個字串到文件中

fpintf函式

寫入整個有格式的字串到文件中

fseek函式

在文件中前進或後退

ftell函式

告訴你在文件中的位置

feof函式

告訴你是否在文件末尾

連結串列Linked Lists

由一連串節點組成，每個節點包含任意的實例資料data fields和一或兩個用來指向上一個或下一個節點的位置的連結links，常用於組織檢索較少，而刪除、添加、遍歷較多的資料，由於不必須按順序儲存，連結串列在插入為O(1)，但尋找一個節點則為O(n)

連結串列為線性表，即由n ( n ≥ 0 )個元素(節點)組成的有限序列，存在一個唯一沒有前驅的元素和一個唯一沒有後繼的元素，此外，每一個元素皆有一個前驅和一個後繼元素

陣列為順序表，即用一組位址連續的儲存單元依次儲存資料元素的線性結構

優點:

克服陣列需要預先知道資料大小的缺點

實現靈活的記憶體動態管理

缺點:

失去了陣列隨機讀取的優點，即陣列可以直接訪問特定元素，因為陣列依序排序

增加了結點的指標域，空間開銷比較大

單向Singly 連結串列Linked Lists

由一連串節點組成，每個節點被分成兩個部分，第一個部分儲存資訊data，第二個部分儲存下一個節點的位址pointer，單向連結串列只可向一個方向遍歷

1.創建新的數據類型Data Type節點

typedef struct tmpname

{

int number; //data

//使用正式數據類型Data Type名稱struct tmpname

struct tmpname \*next; //pointer

}

//此行數據類型Data Type名稱的簡稱node才宣告

node;

2.創建指向第一個節點的指標list

node \*list = NULL;

3.創建指向新節點的指標n

//malloc找到所需相連大小內存並回傳位址

node \*n = malloc(sizeof(node));

//檢查是否有空間，如果malloc找不到內存空間返回NULL

if (n != NULL)

{

// 數據類型node的指標n指向的節點中的number = 2

//等同(\*n).number = 2;

n->number = 2;

//n指向的節點中的指標next = NULL，即n指向的節點為最後一個節點

n->next = NULL;

}

4.在連結串列最前方插入一個新節點

將list指向新節點前，需要使新節點指向原本第一個節點，否則，我們將失去其餘節點，即內存洩漏

//如果list尚未指向節點，即為第一個節點

if (list == NULL)

{

// list更新為指向第一個節點

list = n;

}

else

{

// n指向的新節點中的指標next =指向原本第一個節點的指標list

n->next = list;

//指向原本第一個節點的指標list更新為指向新的第一個節點

list = n;

}

5. 釋放連結串列佔用內存

//cursor和tmp和指向第一個節點

node \* cursor = list;

node \*tmp = list;

//直到cursor = NULL，即連結串列最後一個節點的next

while(cursor != NULL)

{

// cursor指向tmp指向的後一個節點，防止內存洩漏

cursor = cursor->next

free(tmp);

tmp = cursor

}

二元搜尋樹

若任意節點的左子樹不空，則左子樹上所有節點的值均小於它的根節點的值；若任意節點的右子樹不空，則右子樹上所有節點的值均大於或等於它的根節點的值；任意節點的左、右子樹也分別為二元搜尋樹

尋找、插入的時間複雜度較低，為O(log n)

哈希表Hash table

通過哈希函式hash function，將數據data產生相對應的哈希碼hash code，並將數據儲存在哈希表Hash table中，哈希碼hash code即數據在哈希表中儲存的位置，加快了查找速度

例如為了找電話簿中某人的名字，可以創建一個按照首字母排序的哈希表Hash table

hash function (Peter) = 80 //哈希函式hash function，用來取首字母的ASCII

80 //哈希碼hash code

Hash table [80] = Peter //哈希表Hash table在此為一陣列

衝突collision

不同data卻有相同的hash code

hash function (Peter) = 80

hash function (Paul) = 80

衝突解決方法:線性探測Linear probing

逐個查找哈希表中離衝突單元最近的空單元，並且把新的值插入這個空閒單元

Hash table [80] = Peter

Hash table [81] = Paul

線性探測導致聚集Cluster

哈希函式的結果不均勻地占據Hash table的單元，形成區塊，例如很多P…，形成一P區塊，造成像Q = 81的名字儲存偏離原本位置

Hash table [80] = Peter

…

Hash table [98] = Paul

Hash table [99] = Que

聚集解決方法: 單獨鍊表法Chain

將同一個hash code的所有元素保存在一個鍊表Linked Lists中

Hash table [0] = Linked Lists A

…

Hash table [25] = Linked Lists Z

hash function:使A至Z變成0-25的hash code

Hash table:Linked Lists中指向第一個節點的指標

字典樹Tries

鍵不是直接儲存在節點中，而是由節點在樹中的位置決定，一個節點的所有子孫都有相同的字首，也就是這個節點對應的字串，而根root節點對應空字串，可以快速插入、尋找、刪除，但占很大空間

例如apple

root

→a

→ap

→app

→appl

→apple

鍵值key數據value

陣列:

array[12] = 156

key為12 value為156

哈希表:

Hash table [80] = Peter

key為hash code = 80 value為Peter

資料結構 Data structure

陣列array

插入在最後一個元素(缺點)

刪除中間元素會導致空缺(缺點)

查找可以隨機訪問(優點)

排序容易(優點)

空間小(優點)，只占用宣告之固定大小(缺點)

連結串列Linked Lists

插入在第一個元素(優點)

刪除中間元素容易，因為無排序(優點)

查找必須線性搜索Linear Search(缺點)

排序困難(缺點)

空間小，只陣列array大一點(優點)

哈希表Hash table

插入容易，只需用hash function從data得到hash code並加入data(優點)

刪除容易(優點)

查找優於連結串列Linked Lists，雖然依舊需要線性搜索Linear Search，但基於多個Linked Lists，使長度縮短(優點)

排序困難，不建議用Hash table(缺點)

無固定大小，取決於資料量(優點)

字典樹Tries

插入只占用固定時間(優點)，但人為設計部分結構很複雜(缺點)

刪除容易(優點)

查找時間極為接近陣列array(優點)

排序在儲存時已完成(優點)

空間極大，即使只儲存少量資料(缺點)

抽象資料型別Abstract Data Type ADT

抽象資料型別的強處在於對使用者隱藏了實作細節，僅公開其介面，這表示抽象資料型別可以用各種方法來實作，只要遵循其介面，就不會影響到使用者

在抽象資料型別ADT和資料結構Data structure之間，有一個實作上的微妙差別，例如堆棧stack可以用一維陣列或連結串列來實現

隊列queue

只允許在後端rear進行插入操作，在前端front進行刪除操作，是先進先出FIFO，即First In First Out

堆棧stack

只允許在有序的線性資料集合的一端，即堆疊頂端top進行加入資料push和移除資料pop，按照後進先出LIFO，即Last In First Out