#資料結構 ch1

§遞迴

原理：問題越來越小，最後小到不見，就代表解決了。(divide and conquer)

優點：容易看懂，但是效率不一定較快。

遞迴常見解法：

1.factorial 階乘

2.greastest common divisor 最大公因數

3.search in array 搜尋

4.fibonacci series 費氏數列

5.combinatorial numbers 組合數

6.towers of hanoi 河內塔

遞迴思路：

1.define the problem in terms of smaller problems 遞迴定義

2.see if a recursive call decreases the problem size 問題簡化

3.find a complete set of base case 終止條件

4.every time it will always reach a base case 保證終止

- problem1：Given astring of characters, write it in reverse order.

  solution：字串長度每次減一

  base case：empty string

```
void writebackward(string s, int size){
        if(size>0){
                cout << s.substr(size-1,1); // 輸出最後一個字元
                writebackward(s,size-1); // 遞迴呼叫
        }
        // base case：size==0
}

void writearraybackward(const char anArray[], int first, int last){ // 改寫陣列
        if(first <= last){
                cout << anArray[last];
                wirtearraybackward(anrray, first, last-1) ;
        }
}
```

- practice1：Given two natural numbers a&b, where a>b, write a recursive funcation to compute the sum of all the integers from a to b.

```
//linear recursion
int sum(int a, int b){
        if(a==b) // base case!!
                return a ;
        else
                return sum(a, b+1)+ b ;
}

//binary recursion
int sumB(int a, int n){
        // assume n = b-a+1
        if(n==1)
                return a ;

        return sumB(a, n/2)+sumB(a+n/2, n-n/2);
}
```

- problem2：greastest common divisor(GCD)

solution1：

```
gcd1(x,y)=x                    (if y=0)
         =gcd1(x,y mod x) (if y>x)
         =gcd1(y,x mod y) (if x>y)

int gcd1(int x, int y){
        if(y==0) return x ;
        else if(y>x) return gcd1(x, y%x);
        else return gcd1(y, x%y) ;
}
```

solution2： //if x>y 較快，x<y 一樣快

```
gcd2(x,y)=y                    (if x mod y = 0)
         =gcd2(y,x mod y) (otherwise)

int gcd2(int x, int y){
        if (x%y == 0) return y ;
        else return gcd2(y, x%y);
}
```

- problem3：binary search with an array(二元搜尋)

```
int binarysearch(const int anArray[], int first, int last, int value){
// first 為陣列第一位,last 為陣列最後一位
        int index;
        if(first>last) index = -1 ;
        else{
                int mid = (first+last)/2;
                if(value == anArray[mid]) index = mid ;
                else if(value < anArray[mid])
                        index = binarysearch(anArray, first, mid-1, value); //找右半邊
                else // value > anArray[mid]
                        index = bimarysearch(anArray, mid+1, last, value); //找左半邊
        }
        return index;
}
```

- practice3：finding the largest item in an array

solution：一直切半找最大值互相比再回傳大的

// 這樣的情況遞迴很沒效率

```
int findmax(const int anArray[], int first, int last){
        if(first==last) return anArray[first];
        else {
                finadmax(anArray, first, last/2); //左半邊
                finadmax(anArray, first/2, last); //右半邊
        }
}
```

- practice3-2：finding the Kth smallest item in an array

solution：choose a pivot item(樞鈕) 將比 pivot item 大的放在 pivot item 右邊
　　　　　比 pivot item 小的放在 pivot item 左邊，再使用 binary search

```
//可以幫助排序
int ksmall(int k, const int anArray[], int first, int last){
        if(k==pivotindex-first+1) return anArray[k];
        else if(k<pivotindex-first+1) //左半邊
                return ksmall(k, anArray, first, pivotindex-1);
        else //右半邊
                return ksmall(k-(pivotindex-first+1), anArray,pivotindex+1, last) ;
}
```

. practice3-3：reverse an array
solution：

```
void reversearray( const int anArray[], int low, int high){
        if(low < high){
                swap(anArray,low, high); //交換頭和尾
                reversearray(anAray, low+1, high-1);
        }
        return ;
}
```

. problem4：towers of hanoi
solution：把 n 個盤子變成 n-1 個盤子， 最後變 1 個盤子。

```
alogorithm towers(numdisks, source, dest,auxiliary, step){
// numdisks:個數，source:起點，dest:終點，auxiliary:輔助
        print("towers: ", numdisks, source, dest, auxiliary);
        if(numdisks == 1)
                print("towers: ", numdisks, source, dest, auxiliary);
        else{
                towers(numdisks-1, source, auxiliary, dest, step);
                //將 n-1 個盤子從起點移到輔助
                print("move from" source "to" dest) ;
                towers(numdisks-1, auxiliary, dest, source, step);
                //將 n-1 個盤子從輔助移到終點
        }
}
```

. Binary Recursion

- problem1：畫刻度尺

  solution：

```
void drawonetick(int length, int label){
        for(int i = 0 ; i < length ; i++)
                cout << "-" ;

        if( label != -1 ) cout << " " << label << endl ;
        else cout << endl ;
}

void drawticks(int length) {
        if(length>0){
                drawticks(length-1);
                drawonetick(length, -1) ; //draw tick of the length
                drawticks(length-1);
        }
}

void drawruler( int inches, int majorlength ){
        drawonetick(majorlength, 0) ;
        for(int i = 0 ; i < inches ; i++ ){
                drawtick(majorlength-1) ;
                drawonetick(majorlength, i) ;
        }
}
```

- problem：multiplying rabbits(fibonacci sequence)

  assume：1.rabbits never die

  2.a rabbitreacher sexual maturity exactly two months after birth, that is, at the beginning of its third month of life.

  3.rabbits are always born in male-female pairs.

  solution：

  rabbit(n) = rabbit(n-1)+rabbit(n-2)

  rabbit(0)=0

  rabbit(1)=rabbit(2)=1

//1：較沒效率

```
int rabbit(int n){
        if(n <=2) return 1 ;
        else return rabbit(n-1)+rabbit(n-2);
}
```

//2：以空間換時間

使用動態規劃

//3：較有效率

```
Algorithm linearFibonacci(k){
    if(k==1) return (k,0);
    else{
        (i, j) = linearFibonacci(k-1);
        return(i+j,i);
     }
}
```

```
     5     →    4     →    3     →    2     →  1
(3+2,3)←(2+1,2)←(1+1,1)←(1+0,1)←(1,0)
```

- practice：算 x 的 n 次方

solution1：迴圈

```
double power1(double x, int n){
        double ans = 1 ;
        for(int i = 0 ; i<n ; i++){
            ans = ans*x ;
        }
        return ans ;
}
```

solution2：遞迴

```
double power2(double x, int n){
    if(n==1) return x ;
    else return power2(x, n-1)*x;
}
```

solution3：二元遞迴 //最快

```
double power3(double x, int n){
    if(n==0) return 1;
    else{
        double halfpower = power3(x, n/2);
        if(n%2==0) //n 是偶數
            return halfpower*halfpower;
        else //n 是奇數
            return x*halfpower&halfpower;
    }
}
```

- problem：organizeing a parade

solution：

1.F(n)：花車殿後

  F(n)= P(n-1)

2.B(n)：樂隊殿後

  B(n)= F(n-1)=P(n-2) //因為倒數第二個一定是花車

3.P(n)：F(n)+B(n)

  P(n)=P(n-1)+P(n-2)

→P(1)=2

P(2)=3

P(n)=P(n-1)+P(n-2) for n>2

- practice：找一整數的平方最接近且小於 n

```
int getvalue(int a, int b, int n){
    int returnvalue;
    cout << "enter: a= " << a << "b= " << b << endl; //進入時的狀態
    int c=(a+b)/2; //類似二元
    if( (c*c<=n)&&(n<((c+1)*(c+1))) )
        returnvalue = c ;
    else if( (c*c)>n )
        returnvalue = getvalue(a, c-1, n);
    else
        returnvalue = getvalue(c+1, b, n);

    cout << "leave: a= " << a << "b= " << b << endl; //離開時的狀態
    return returnvalue ;
}
```

． problem：c n 取 k
c(n,k) = c(n-1,k-1)+c(n-1,k)
base case：

！遞迴參數含遞迴電腦很容易爆掉！

#資料結構 ch3

§鏈結串列

． 為何要使用 link list?
→因為記憶體是有限的，link list 可以幫助節省資源。

． pointer　// 指標 = 門牌
宣告：int *p;　// =(int *)p; //此時 p 裡面還沒有房子
如果 int x = 10;　// 房子 x 的門牌 =500 裡面存放 10
　　p = &x; //&x = x 的門牌 = 500
　　p = new int ; //配置了房子 //動態配置

！std::bad_alloc 代表記憶體不足！

若要清除房子：delete p; //歸還房子
　　　　　　　p = NULL; //忘記該門牌

． pointer 指令：
(a)指向別人
1.int *p ;
　int x;
　→declaring pointer variables
2.p = &x; //p 是位址
　→pointing to statically allocated memory
3.*p = 6; //*p 是位址裡的內容
　→assigned a value
(b)新建天地

1.p = new int ;

→allocating memory dynamically

2.*p = 7 ;

→assigning a value

3.delete p; //一定要記得 delete 避免記憶體不足

p = NULL;

→deallocating memory

．動態陣列

(a)陣列之動態陣列

1.int arraysize = 50 ;

double *anarray = new double[arraysize];

2.anarray[2] = *(anarray+2) ;

3.double *oldarray = anarray ;

anarray = new double[3*arraysize] ;

4.double *oldarray = anarray ;

anarray = mew double[3*arraysize];

if 清空：delete [] oldarray ; //[]是要告訴 cpu 是清空整個 array

．存檔&讀檔

```
#include<iostream>
#include<string>
#include<cstdio>

#define SID_LEN 12
#define SR_NUM 5
using namespace std ;

typedef struct student{
        char sid[SID_LEN];
        int score;
} studentType;

void savefile(FILE *fp, studentTypedA[], int no){
        for(int i = 0 ; i<no ;i++){
                fwrite( &dA[i].sizeof(dA[i]),1,fp);
                cout << dA[i].sid<<","<<dA[i].score<<endl;
```

```
    }
    fclose(fp);    //close the file
}

//一般
int main(void){
    FILE *outfile = NULL; //在 cstdio 裡面的宣告  去找檔案位址
    string fileName = "DSsample1.dat";
    studentType allS[SR_NUM]={
    {"10027113",60},{"10127102",70},{"10027213",90},
    {"10127256",80},{"10227108",100}                    };

    outfile = fopen(fileName.c_str(),"a"); //open a file to write
    if(outfile!=NULL)
        savefile(outfile,allS,SR_NUM);

    return 0 ;
}

//動態規劃
int main(void){
    FILE *infile = NULL, *outfile = NULL;
    string fileName = "DSsample.dat" ;
    studentType *bufS;
    int studentNo = 0 ;

    infile=fopen(fileName.c_str(),"r");
    if(infile!=NULL){
        fseek(infile,0,SEEK_END);
        studentNo=ftell(infile)/sizeof(studentType); //total number of students
        rewind(infile);

        bufS = new studentType[studentNo];
        for(int i = 0, i<studentNo;i++)
            fread(&bufS[i],sizeof(studentType),1,infile); //read data.oe by one

        fileName=fileName.substr(0,8)+"2,dat"; //change the file name
        outfile=fopen(fileName.c_str(),"a"); //open a file to write
```

```
            if(oufile!=NULL)
                savefile(outfile, bufS,studentNo);

            delete [] bufS; //release the space
        }

        fclose(infile) ;
        return 0 ;
}
```

. linked list

(a)宣告

```
struct Node{
   int item;
    Node *next;
};

Node *p ;
p = new Node ;
```

(b)走訪

```
for(Node *cur = head; cur!=NULL;cur = cur->next)
    cout << cur->item << endl;
```

(c)刪除

```
cur->next = NULL;
delete cur;
cur =NULL;
//順序不可變
```

(d)新增

```
nextPtr->next = cur ;
pre->next = nextPtr ;
//加在中間,順序可變

newPtr->next =head;
head = newPtr ;
//加在最前面,順序不可變
```

```
newPtr->next =NULL;
pre->next = nextPtr ;
//加在最後面,順序可變
```

- ADT

1.constructor //建構

2.destuctor //解構

```
List::~List(){
        while(!isEmpty())
              remove(1);
 }
```

3.shallow copy

//新建一個 head 指向舊 head 的 head->next

4.deep copy

//複製成兩條