递迴：

1. 递回定义
2. 问题简化
3. 终止条件
4. 保证终止.

★提取字元指令（substr）

e.g. s.substr(size-1, 1);

输出字串 → 简化 → 提取字元 → 迴圈停止条件

求最大公因数 GCD

① 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);
}

② int gcd2(int x, int y){
    if!(x%y) return y;
    else return gcd2(y, x%y);
}

①会多做一次递回呼叫
∴②更有效率.

若 x<y.
递回次数相同.

二分法：

看有没有找到目标.

没有 → 分成两半，分开递回查找.（两边都要找）

找第k小的数值.

用二分法做.
用一个枢钮（pivot item）分成两部分
一次只找一边.
边找边排序.
不需要排所有的数就能找到.

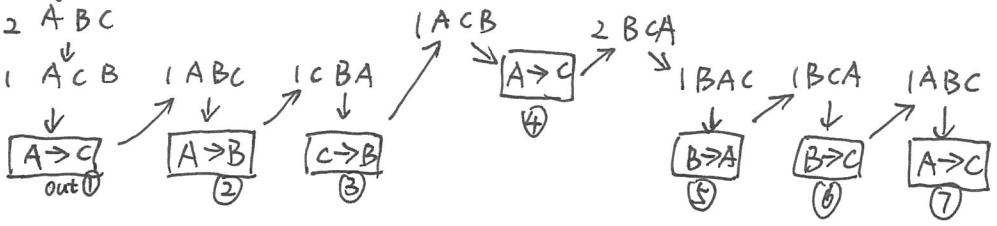
河内塔/汉诺塔 递回解决.

```
void SolveTowers(int count, char source, char destination, char spare)
{
  if(count==1){
    cout<<"Moves top disk from pole"<<source<<"to pole"<<destination<<endl;
  }
  else
  {
        2 tobe next
```

（count：个数；source：起点；destination：终点；spare：辅助）

```
SolveTowers(count-1, source, spare, destination);   (终点辅助互换)
SolveTowers(1, source, destination, spare);          (out)
SolveTowers(count-1, spare, destination, source);    (起点辅助互换).
}
}
```

递回深度	输入
1	3 A C B
2	2 A B C
3	1 A C B
4	A→C (out ①)



刻度尺问题.

```
void drawTicks(int tickLength){
  if(tickLength > 0){
    drawTicks(tickLength-1);
    ~~drawTicks~~
    drawOneTick(tickLength, -1);
    drawTicks(tickLenght-1);
  }
}
```

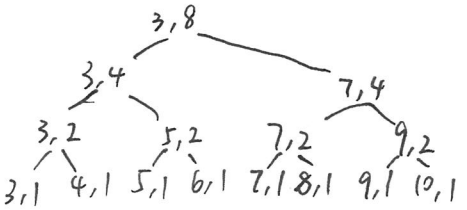
从 a 加到 b.

① `int sumA(int a, int b){ //a<b`　　8次递回呼叫

```
  if(a==b)
    return a;
  return a + sumA(a+1, b);
}
```

② `int sumB(int a, int n){ //n=b-a+1`

```
  if(n==1)
    return a;
  return sumB(a, n/2) + sumB(a+n/2, n-n/2);
}
```



15次递回呼叫

二分法求 a^n

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

	乘法次数.		递归呼叫次数	
	9^{32}	9^{19}	9^{32}	9^{19}
普通做法	32	19	32	19
power3	7	8	7	6

尾端递归.

函式的最后一个动作是递回呼叫.

可被很方便的改写成迴圈.

可用 Dev C++ -O2优化解决栈溢出.

CH 2

A Class combines

Attributes (characteristics) of object of a single type
- Typically data
- Called data members

属性.

Behaviors (operations)
- Typically operate on the data
- Called methods or member functions

运算

封装. 继承. 多型.

Cohesion - modules perform single well-defined tasks
- highly cohesive modules desired 高内聚

Coupling - measure of dependence among modules
- Loosely coupled modules desired 低耦合.

ADT List Operations

建构．解构．是否为空．计算个数

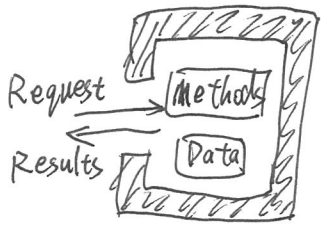
插入．删除．检索．

c++ class

①封装

②成员

③私密公开



Constructors

- Create and initialize new instances of a class
- Have the same name as the class
- Have no return type, not even void

C++ 的继承.

父类别：Sphere　　class ColorSphere : public Sphere {

子类别：ColorSphere　　...

　　}

CH3

- 指标
- 链结串列
- 环状链结串列
- 双向链结串列

阵列：需要移动资料

链结串列：不需要移动资料

指标 = 门牌

(int *)(x);

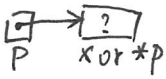
&x = 房子的x的门牌.

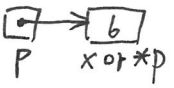
delete p; 归还房子

p=NULL; 忘记门牌.

a) int *p, *q;　　  　申请空白门牌

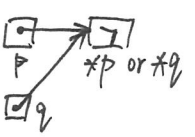
　int x;

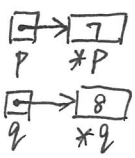
b) p=&x;　　  　抄写别人的门牌

c) *p=6;　　  　据占茑巢

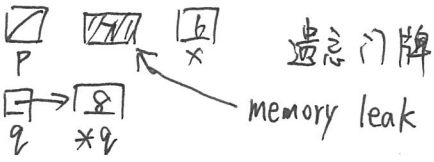
d) p=new int;　　  　紧急配置

e) *p=7;　　  　堆放家当.

f) q=p;　　  　抄写至另一张门牌

g) q=new int;　　  　紧急配置并堆放家当

　*q=8;

h) p=NULL;　　 

动态(配置)阵列

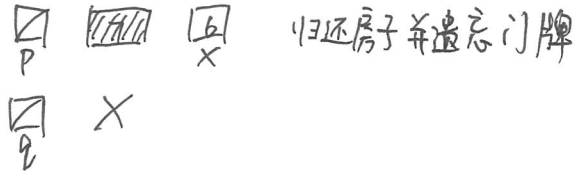
```
double *anArray = new double[50];
anArray[2] = *(anArray+2)
```

搬家用 for 一个一个搬

```
delete[] anArray;
```

归还旧社区

```
delete q;
q = NULL;
```



归还房子并遗忘门牌

Save/Copy a File

```
#include <cstdio>
FILE *outfile = NULL;  //文档指标
outfile = fopen(fileName.c_str(), "a");  ← 只写文档.
```

若 outfile == NULL 则开启失败.

```
fclose(outfile);
```

关闭文档.

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

梦结构.

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

空白门牌

新房子.

```
delete p;
p = NULL;
```

Shallow copy (浅层复制): 只复制地址, 不复制内容.

Deep Copy (深层复制) 复制内容.