计算机程序设计

Computer Programming

函数

主讲：李卫海

目录
CONTENTS

函数的作用

```
#include <stdio.h>
int main() {
    printf("Hello, world!\n");
    return(0);
}
```

- printf() 体现了函数的重要性：
  - 可以完成特定功能（按指定格式打印字符串）
  - 屏蔽功能的实现细节（如果没有printf函数可用，必须直接跟操作系统甚至硬件打交道）
  - 可以被任意程序反复调用，减少编程工作量
  - 便于模块化、结构化程序设计，简化程序调试
  - 支持任务分工，使大规模软件开发成为可能
  - ……

◎ 函数的作用

- 一般的C程序包含一个main函数及多个其它函数，其中的函数可以是：
  - 库函数：由程序开发环境提供，或从第三方获得，常为.lib文件（需要在开发环境中或使用工具软件查看）；不同的C语言编译系统提供的库函数的数量和功能会有一些不同
  - 用户自定义函数：由用户自己编写，为文本形式（可用记事本等文本编辑软件查看）

- 使用函数的规则
  - 在ANSI C中，与变量类似，函数需要在使用（称为函数调用）前进行定义或声明
  - main()函数是特殊函数，无需事先定义或声明，可以调用其它函数，但不能被其它函数调用
  - 其它任何函数在定义或声明后，都可以被一个或多个函数调用任意多次

◎ 例：计算圆面积的程序

用户自定义函数

主函数

```
#include<stdio.h>
#include<math.h>
#define PI 3.14159
double area(double x) {
    double y=0;
    y=PI*x;
    return(y);
}
int main() {
    double r,a;
    printf("Input: r=?\n");
    scanf("%lf",&r);
    r=pow(r,2.0);  //调用math.h中声明的函数
    a=area(r);     //调用用户自定义函数
    printf("The area is %f\n", a);
    return(0);
}
```

◎ 函数的组成成分

- 函数名：
  - 函数的名字，应尽量准确地反映函数功能（有命名规则）
  - 同一程序/文件中不能重名
  - 一个工程文件中，又且只能有一个main()函数
- 返回值类型：也称为函数类型
  - 规定了被调用的函数执行完成后返回值的数据类型
  - 不需要返回值时，应定义为void（空类型）
  - 缺省则为int类型

```
#include<stdio.h>
#include<math.h>
#define PI 3.14159
double area(double x) {
    double y=0;
    y=PI*x;
    return(y);
}
int main() {
    double r,a;
    printf("Input: r=?\n");
    scanf("%lf",&r);
    r=pow(r,2.0);  //调用math.h中声明的函数
    a=area(r);     //调用用户自定义函数
    printf("The area is %f\n", a);
    return(0);
}
```

◎ 函数的组成成分

- 形式参数：简称形参
  - 形参表规定了函数调用时传递的数据对象的个数、顺序及数据类型
  - 系统临时分配存储单元，以接收并存放调用时传递进来的数据（地位相当于函数的内部变量）
- 函数返回值：
  - return语句表示被调用函数完成功能并返回，可以返回一个值(按函数名前指定的类型)或不返回值(无论函数名前是否有返回类型)
  - exit()语句表示终止程序运行。main()函数中return和exit语句等价

```
#include<stdio.h>
#include<math.h>
#define PI 3.14159
double area(double x) {
    double y=0;
    y=PI*x;
    return(y);
}
int main() {
    double r,a;
    printf("Input: r=?\n");
    scanf("%lf",&r);
    r=pow(r,2.0);  //调用math.h中声明的函数
    a=area(r);      //调用用户自定义函数
    printf("The area is %f\n", a);
    return(0);
}
```

◎ 函数的组成成分

- 局部变量：
  - 其作用域在函数内部（其它函数不能使用）；
  - 形参虽然在函数首部说明，但作用域同内部变量；
  - 形参可以认为是被初始化了的内部变量，因此二者不能重名
- 执行语句：
  - 完成特定的功能；其中不能嵌套定义函数，但可以调用其它函数，甚至调用自己（递归调用）
- 局部变量声明和执行语句合称函数体

```
#include<stdio.h>
#include<math.h>
#define PI 3.14159
double area(double x) {
    double y=0;
    y=PI*x;
    return(y);
}
int main() {
    double r,a;
    printf("Input: r=?\n");
    scanf("%lf",&r);
    r=pow(r,2.0);  //调用math.h中声明的函数
    a=area(r);      //调用用户自定义函数
    printf("The area is %f\n", a);
    return(0);
}
```

◎ 函数的组成成分

- 特例：空函数

  返回值类型说明 函数名() { }

  - 如：void null () { }
  - 作用：可正常调用，但函数本身什么都不做，通常用来占位，开发大型程序时常见

- 关于main()函数：
  - 由于main()函数不会被其它函数调用，因此其返回值类型可以随便定义，return语句也可有可无（实际编译器有不同要求）
  - 规范的写法是：

```
int main() {...; return(0);}

void main() {...}
```

函数的定义与声明

- 函数名必须先定义或声明，然后才能使用

- 函数的定义是告诉编译器，名字所要完成的具体任务（做什么，怎么做）
- 函数的声明是告诉编译器，如何解释和使用名字（做什么）

- 函数的定义是函数的完整代码，包含了函数所需要的的所有成分
  - 函数定义也具有声明的作用。若被调函数定义在主调函数之前，则可以不对被调函数进行声明，因为编译系统已经知道被调函数的所有特征
  - 否则应先给出函数原型声明，然后才可以调用

◎ 函数的定义与声明

- 函数原型声明是对已定义函数或准备定义函数的函数名、函数类型、形参表等信息进行说明，不包含函数体

- 函数原型声明的两种形式（效果相同）：

  函数类型 函数名 (参数类型1，参数类型2，...);

  如：int max (int, int);

  函数类型 函数名 (类型1 形参1，类型2 形参2，...);

  如：int max (int x, int y);

◎ 函数的定义与声明

- 注意区分函数原型声明与函数定义
  - 形式上
    - 原型声明不包括函数体，且以“;”结尾
    - 函数定义包括函数体
  - 作用上
    - 原型声明仅是通知编译系统被调函数的特征，用以验证用户调用函数合法性
    - 函数定义则是整个函数的完整代码

◎ 例：计算圆面积的程序（函数定义置后）

```
#include<stdio.h>
#include<math.h>
#define PI 3.14159
int main() {
    double area(double x);   //函数声明
    double r,a;
    printf("Input: r=?\n");
    scanf("%lf",&r);
    r=pow(r,2.0);
    a=area(r);      //调用用户自定义函数
    printf("The area is %f\n", a);
    return(0);
}
double area(double x) {
    double y=0;
    y=PI*x;
    return(y);
}
```

```
#include<stdio.h>
#include<math.h>
#define PI 3.14159
double area(double x);   //外部声明函数
int main() {
    double r,a;
    printf("Input: r=?\n");
    scanf("%lf",&r);
    r=pow(r,2.0);
    a=area(r);      //调用用户自定义函数
    printf("The area is %f\n", a);
    return(0);
}
double area(double x) {
    double y=0;
    y=PI*x;
    return(y);
}
```

函数的定义与声明

- 函数声明可以在主调函数的声明部分，也可以在函数外部。通常集中放在文件的前面，或在头文件(.h)中
  - 建议将函数原型声明放在头文件中，并在定义函数的文件中包含该头文件，以便编译器及时发现不匹配的情况
- 在外部声明的函数名，可被之后所有函数调用，无需在函数内再次声明
- 库函数的声明包含在头文件(*.h)里，包含了头文件就无须另写

◎ 函数调用

- 函数调用的一般形式

  函数名(实参表)

  - 例如：r=pow(r, 2.0); a=area(r);
  - 实参，即实际传递给函数的参数（或表达式）
  - 实参表中实参的个数、出现的顺序和实参的类型一般应与函数定义中形参的设置相同
  - 如果函数定义中没有形参，则函数调用时也没有实参，但函数名后的括号不能省
  - 当实参表中有多个实参时，各实参之间要以逗号分开

◎ 函数调用

- 函数调用时，存在流程控制转移和数据传递
- 调用者称为主调函数，被调用者称为被调函数

- 当被调函数存在形参时，主调函数向其传递“实际参数”（简称实参），并将流程控制转移到被调函数，被调函数的形参接收实参后，执行并完成预定的任务

- 被调函数执行完语句后，把流程控制返回主调函数调用它的地方，并通过函数返回值向主调函数返回信息

函数调用

- 例，计算并输出三个电阻的串联和并联值，分别由函数series()和parallel()实现

```
#include<stdio.h>
float series(float a1,float a2,float a3) {
    return(a1+a2+a3);
}
float parallel(float b1,float b2,float b3) {
    float rp,rr;
    rr=1.0/b1+1.0/b2+1.0/b3;
    rp=1.0/rr;
    return(rp);
}
int main() {
    float r1,r2,r3,rs,rp;
    scanf("%f%f%f",&r1,&r2,&r3);
    rs=series(r1,r2,r3);
    printf("The series values is %f\n",rs);
    rp=parallel(r1,r2,r3);
    printf("The parallel values is %f\n",rp);
}
```

◎ 函数调用

- 按函数调用在程序中出现的形式和位置来分，可以有以下3种函数调用方式：

1. 函数调用语句

- 把函数调用单独作为一个语句

  如 printf (“Hello world!”);

  strcpy(str1,str2);

- 不要求函数带回值（虽然这两个函数实际上是有返回值的），只要求函数完成一定的操作

2. 函数表达式

- 函数调用也是表达式，可以出现在任何表达式中

  如 c=max(a,b); rs=series(r1,r2,r3);

- 此时函数必须返回一个确定的值，以参与运算

◎ 函数调用

3. 函数参数

- 函数调用作为另一函数调用时的实参

如 printf("%f",series(r1,r2,r3));

m＝max(a,max(b,c));

- 本质都是函数返回值参与后续运算
  - 函数的返回值才是它的“本职工作”
  - 函数附带的效果，例如显示等，是它的“副作用”

◎ 参数传递

- 函数定义中的形式参数
  - 形参用于定义函数拟接收数据的类型和顺序
  - 函数也可以没有形参，表示函数不需要接收数据
  - 形参只是一种说明，不能初始化（C++支持缺省参数，可以初始化）

- 函数调用时的实际参数
  - 实参可以是常量、变量或表达式，但必须有确定的值
  - 实参的个数、类型应与形参一致，个数不一致会出错，类型不一致会进行隐式转换（随编译器不同，可能发生意想不到的结果）

◎ 参数传递

- C语言参数的传递遵循“值拷贝”原则，并分成两种情况
  - 当实参是常量、变量或表达式时，传递的是数据对象的内容（值），简称传值方式
- 例，

```
#include<stdio.h>
void swap(int x,int y)
{    int t;
     t=x; x=y; y=t;
     printf(“in swap():x=%d y=%d\n”,x,y);
}
int main()
{    int a=3,b=5;
     printf(“Before call swap():a=%d b=%d\n”,a,b);
     swap(a,b);
     printf(“After call swap():a=%d b=%d\n”,a,b);
}
```

运行结果是：

```
Before... :a=3 b=5
in        ...:x=5 y=3
After  ...  :a=3 b=5
```

◎ 参数传递

- C语言参数的传递遵循“值拷贝”原则，并分成两种情况
  - 当实参是常量、变量或表达式时，传递的是数据对象的内容（值），简称传值方式
  - 当实参是数据对象的首地址值时（如数组名），简称传址方式或传地址方式
    - 传递的是地址的值
    - 被调函数通过地址访问（读/写）数据存储空间——具有数据双向传递的功效
- 例，

```
#include<stdio.h>
void add(int b[], int n) {
    int i;
    for(i=0; i<n; i++) b[i]++;
}
int main() {
    int a[5]={1,2,3,4,5},i;
    add(a,5);
    for(i=0; i<5; i++) printf("a[%d]=%d\n",i+1,a[i]);
}
```

运行结果是：
a[5]={2,3,4,5,6}

参数传递

- 当有多个实参时，实参的计算顺序未严格规定
  - 实参规定是从右向左入栈的
  - 实参的计算未见严格定义，但与入栈顺序一致就最容易实现的

- 要警惕赋值相关运算的副作用发生时间
  - 例如：printf(“%d,%d,%d\n”, i++, i++, i++); 在VC6中调用的是..., i, i, i， 在Dev C++中调用的是...i+2, i+1, i

- 应避免使用类似未严格定义的写法

◎ 参数传递

- return语句
  - return语句的作用：结束当前函数的执行，并将返回值传给主调函数。例：

```
int test(int n)
{   if(n>10) return 1;
    return(0);
}
```

注意return语句两种不同的格式，等效

  - 当return的数值类型与函数类型不一致时，会转换成函数类型
  - 函数需要返回值时，若缺少return语句，或return语句未带返回值，则返回一个不确定值（多数编译器会给出警告）
  - C语言的函数不需要返回值时，可以使用void来将函数定义为空类型，此时，函数体中不出现return语句；或使用不带返回值的return语句：return;

数组作为函数参数

- 一维数组元素作为函数参数
  - 数组元素是单值变量，因此作实参使用时和一般变量没什么区别

- 一维数组名作为函数参数
  - 函数的形参如果是数组，调用时实参要用数组名，此时是传地址调用，形参和实参指向同一块存储空间，相当于对这块空间分别命名
  - 事实上，数组作为函数参数时，无论函数调用前后，都不会为形参数组分配另外的存储空间，形参数组只接收实参数组的首地址（即指针）
  - 实参数组名是个常数，不可更改；形参数组名是个指针，其值可以更改
  - 一维数组作形参时，一般不定义数组的大小，即使定义，也没有实际意义。传递的仅是首地址，并不关心有多少元素。数组元素个数一般另用参数传递
  - 实参也可以不用数组名，而是某段内存的起始地址。至于内容，要由程序员掌控

◎ 数组作为函数参数

- 例，数组分段交换，即将一个长度为n的数组a分为两段，前k个元素一段，后n-k个元素一段，要求编写函数实现两段交换，例如：

  a={1,2,3,4,5,6,7,8,9}, k=3交换后变成

  a={4,5,6,7,8,9,1,2,3}

- 最直接的思路是通过一个临时数组暂存和处理

数组作为函数参数

```
#include <stdio.h>
void array_swap(int a[], int n, int k) {
    int t[30],i;
    for (i=0; i<k; i++) t[i]=a[i];
    for (i=k; i<n; i++) a[i-k]=a[i];
    for (i=0; i<k; i++) a[n-k+i]=t[k];
}
void main() {
    int a[]={1,2,3,4,5,6,7,8,9};
    int i, n=9, k=3;
    for (i=0; i<n; i++) printf("%3d",a[i]);
    printf("\n");
    array_swap(a,n,k);
    for (i=0; i<n; i++) printf("%3d",a[i]);
    printf("\n");
}
```

数组作为函数参数

- 二维数组元素作为函数参数
  - 数组元素是单值变量，因此作实参使用时和一般变量没什么区别

- 二维数组名作为函数参数
  - 行优先排列，有序地占用一片连续的内存区域
  - 形参定义成二维数组时，必须指定第二维的大小，编译器才知道如何由数组下标计算元素存储位置
  - 第一维行的大小可以根据需要选择传递，以告知数组大小
  - 因为是地址传递，原则上并不要求形参数组与实参数组的结构一样，被调用函数按照形参指定的格式理解数据。但不一致时很难理解，极少使用

◎ 数组作为函数参数

- 例，矩阵乘法

```
#include<stdio.h>
void matpro (int a[][3], int b[][2], int c[][2], int m, int n, int p) {
    int i,j,k,s;
    for(i=0; i<m; i++)
        for(j=0; j<p; j++)
        {   for(k=s=0; k<n; k++) s+=a[i][k]*b[k][j];
            c[i][j]=s;
        }
}
void main() {
    static int a[2][3]=..., b[3][2]=..., c[2][2];
    ...;
    matpro(a,b,c,2,3,2);
    ...
}
```

数组作为函数参数

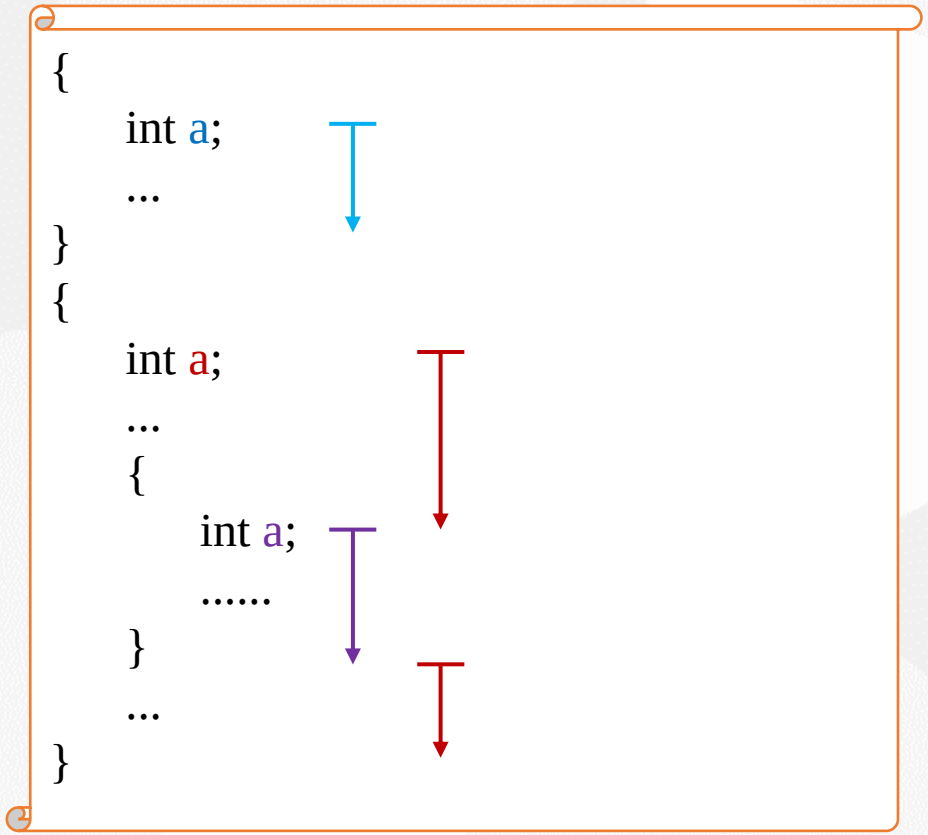
- 多维数组作为函数参数
  - 与二维情况类似
  - 多维数组名作为函数参数时，形参定义需要指定第二维及之后所有维度的大小，以便寻址

局部变量与全局变量

- 变量声明的位置有三种
  - 在函数的开头，称为局部变量
  - 在语句（特别是循环语句或复合语句）中，称为局部变量
  - 在函数的外面，称为全局变量（注意与外部变量不同）
- 作用域
  - 即该变量名字可以被使用的范围
  - 局部变量限制在本程序块内
    - 在函数开头声明的，作用域直到本函数结束
    - 在语句中声明的，作用域直到本语句结束
  - 全局变量限制在本源代码文件内

局部变量与全局变量

- 同一个程序块内，不允许出现重名变量
- 嵌套的程序块出现重名局部变量时，使用本层次或最接近本层次的外层程序块的声明
- 例，



◎ 局部变量与全局变量

- 全局变量的使用应适度
  - 是全局可访问的公共数据
  - 节省函数参数的传递
  - 破坏了函数/类的封装
  - 无论数据是否有用，始终占据存储单元
  - 多线程程序可能导致冲突和死锁

◎ 变量的存储类别

- 自动的 auto
  - 例，auto int a;
  - 局部变量缺省都是自动的
  - 在函数被调用时自动分配存储空间，在函数退出时自动释放
- 静态的 static
  - 例，static int a;
  - 全局变量都是静态的。局部变量可以声明为静态的
  - 在程序载入时分配且独占存储空间，在程序退出时释放
  - 对全局变量声明static，其含义是“内部的”，限制该变量只能被本文件使用
- 寄存器的 register
  - 例，register int a;
  - 变量优先存储在寄存器中
- 外部的 extern
  - 例，extern int a;
  - 声明变量a在其它文件中，链接时进行拼装

变量的存储类别

- 静态局部变量的例子

```
#include <stdio.h>
int main() {
    int fac(int n);
    int i;
    for (i=1; i<=5; i++)
        printf("%d!=%d\n", i, fac(i);
    return 0;
}
int fac(int n) {
    static int f=1;
    f=f*n;
    return();
}
```

程序运行结果：

```
1!=1
2!=2
3!=6
4!=24
5!=120
```

◎ 例，排序算法

对n个数进行排序（以从小到大为例）

- 交换排序法
  - 第i轮将第i个数与后面所有数依次比较，不满足小于等于关系时交换（或仅与最小的数交换）。每轮排好第i小的数
- 选择排序法
  - 第i轮将第i个数与后面所有数依次比较，若第j个数不满足小于等于关系时，将第j个数继续与后面的所有数比较，直至所有数比较完成，得到最小的数，与i交换。每轮排好第i小的数
- 冒泡排序法
  - 每轮对相邻的两个数进行比较，不满足小于等于关系时交换。第i轮排好第i大的数
- 以上算法复杂度均为$O(n^2)$
- 插入排序法
  - 前m个数已排好序，将第m+1个数插入到合适位置。

◎ 例，排序算法

对n个数进行排序（以从小到大为例）

- 快速排序法
  - 分治法
  - 以第1个数(设为a)为基准，将小于等于a的数挪到数组前半段，将大于等于a的数挪到数组后半段
    - 更好的方法是取第一个元素、中间元素、最后一个元素的中间值
  - 对前半段和后半段两个数组分别递归执行快速排序
  - 算法复杂度为O(n log n)

例，二分法查找算法

在一个已从小到大排序的数组中，查找特定key

- 分治法
  - 双指针，low指向数组开始，high指向数组末尾
  - 计算mid指向low和high的中间
  - 若mid指向的数等于key，则找到
  - 否则根据mid指向的数大于key或小于key，修改low或high，继续迭代查找

```
#include <stdio.h>
void main() {
    static int a[]={5,13,19,21,37,56,64,75,80,88,92};
    int n, key, low, mid, high;
    n=sizeof(a)/sizeof(a[0]);
    scanf(“%d”, &key);
    low=0; high=n-1;
    while (low<high) {
        mid=(low+high)/2;
        if (key==a[mid]) {
            printf(“a[%d]=%d”\n, mid, key);
            break;
        }
        else if (key>a[mid]) low=mid+1;
        else high=mid-1;
    }
    printf(“not found!\n”);
}
```

◎ 例，爬楼梯

有n(0<n<=1000)级台阶，每次可以上一级或两级，问有几种上法？

- 动态规划法
  - 最后一步可以上一级或两级 f(n)=f(n-1)+f(n-2)

```
#include <stdio.h>
int steps(int n) {
    if (n==1) return(1);
    else if (n==0) return(1);
    else return(steps(n-1)+steps(n-2));
}
void main() {
    int n;
    scanf(“%d”, &n);
    printf(“%d ways.\n”, steps(n));
}
```

```
#include <stdio.h>
int main() {
    int n, i, f0=1, f1=1;
    scanf(“%d”, &n);
    for(i=2; i<=n; i++) {
        f1=f0+f1;
        f0=f1-f0;
    }
    printf(“%d ways.\n”, f2);
    return 0;
}
```

程序员之歌

99 little bugs in the code,

99 bugs in the code,

fix one bug, compile it again,

101 little bugs in the code.

101 little bugs in the code….

(Repeat until BUGS = 0)

作业

A13. 随机产生一个整型数组（元素不超过10000），一边产生一边用插入排序法进行非递减排序。

- 产生数组、排序、输出数组，用三个函数来实现
- 插入排序：假设前t个数据已排好序，将第t+1个数据插入到前面适当位置，使得前t+1个数据也是有序的

A14. 用递归法求解N皇后问题（N<=8)。

A15. 编写程序，输入一个数字字符串s，将它转换为科学计数法输出。

- 数字串s仅包含数字和小数点，小数点位于最后时，小数点可有可无；小数点前只有0时，0可有可无
- 科学计数法应保持原数字的有效精度，例如 10.0 = 1.00E+0；科学计数法的指数部分用E表示，不缺省；E后用一位表示指数的符号位，不缺省

作业（一）

A16. 随机产生N个城市间巴士班次表，形式如下图所示。（$2 \le N \le 20$)

出发城市	到达城市	出发时间	到达时间
A	B	0700	0900
A	B	0730	1000
B	C	1200	1800

- 每个城市有多个班次（4班到10班）到达另一个城市，路程时间可以不相同，但相差不超过2小时
- 所有巴士运行时间不会跨过2400

B6. 在A16题产生的班次表上，

a) 给定从S市出发的时间，求到达T市的最短时间（假定只中转一次）。注意你的旅行可以跨越2400

b) 若出发时间不固定，求从S市到T市的最短时间及路径

B7. 用非递归法求解N皇后问题。

作业（二）

B8. 约翰抓牛：从前有一个农夫约翰，他养了很多牛。有一天一只牛走丢了，农夫要尽快的抓住它！现在将问题简单化为数字问题：假设农夫和母牛都站在一条数轴上，农夫开始的位置为N，母牛的位置为 K。假设约翰当前位置为 X，他有三种行动方式，他可以向前走一格到 X+1，也可以向后走一格走到 X-1，还可以一下子跳到 2*X。每行动一次需要一秒钟时间。我们的问题是，假设母牛不会动，农夫最少需要多少秒才能抓到母牛？（$0 \leq N,K \leq 100000$）

C2. 关于N位二进制大数运算的程序，

- 将加、减运算封装为独立的函数
- 添加乘、除运算函数
- 努力让你的算法效率尽量高
- 保存好你的程序，后面我们还会对它进行改造

C3. 关于N位二进制大数运算的程序，

- 添加取模运算，模数也是一个二进制大数
- 针对模运算下的加、减、乘、除，进行优化改造
- 保存好你的程序，后面我们还会对它进行改造