本来是想写个《C语言经典题目系列》，本系列包括一些经典算法题目，但由于时间问题，现在只是收集了不多题目且只做了一部分，就先发上来了。写此目的帮助一些学c语言的人入门及运用一些算法，由于水平有限错误在所难免及本来这些题目不是很难高手就不用看了，其中错误欢迎大家指正。

1、

【问题描述】梯有N阶，上楼可以一步上一阶，也可以一步上二阶。编写一个程序，计算共有多少中不同的走法

【思路】看到此题目容易想到用递归的方法来做，因为递归是一种描述和解决结构自相似问题的基本算法，而N阶楼梯问题和N-1阶、N-2阶的结构完全相同。

解决递归问题可以分为两个部分，第一部分是一些特殊（基础）情况，用直接法解，即始基；第二部分与原问题相似，可用类似的方法解决（即递归），但比原问题的规模要小。

定义int count(int n)函数求解N阶楼梯的走法，基于上述思想，可知：

N阶楼梯问题的始基是N==1、N==2两种情况；

上楼可以一步上一阶，也可以一步上二阶，当上一阶时问题规模变为N-1，当上二阶时问题规模变为N-2，所以总的情况为count(n-1)+count(n-2)。

【代码】

cCODE:

#include<stdio.h>

#include<stdlib.h>

int count(int n);

/\*count how many ways to climb up N steps stairs.\*/

int main (int argc, char \*argv[])

{

int n,ct;

printf("please input n:\n");

scanf("%d",&n);

ct=count(n);

printf("there are %d ways to climb up N steps stairs!\n",ct);

system("PAUSE");

return 0;

}

int count(int n)

{

if(1==n)

return 1;

else if(2==n)

return 2;

else return count(n-1)+count(n-2);

}

【程序输入输出】for example

please input n:

5

there are 8 ways to climb up N steps stairs!

2、

【问题描述】Armstrong数具有如下特征：一个n位数等于其个位数的n次方之和。如：

153＝13＋53＋33

1634＝14＋64+34+44

找出2、3、4、5位的所有Armstrong数。

【思路】看到此题我第一反应是用枚举法，给定m(10<=m<=99999)，首先判断m的位数n，然后判断它是否等于各位数的n次方之和。

定义函数int judgeDigit(int m)，用于判断给定参数m的位数；

定义函数int judgeEqual(int m,int n)，其中m为给定的数，n为m的位数，用于判断m是否等于各位数的n次方之和。

【代码】

cCODE:

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

int judgeDigit(int m);

/\*This function return the digit of parameter m\*/

void judgeEqual(int m,int n);

/\*parameter m is a integer,parameter n is the digit of m,this function is used to judge m whether is a Armstrong integer and output it\*/

int main (int argc, char \*\*argv)

{

int i,len;

printf("All 2 digit to 5 digit Armstrong integers are following:\n");

for(i=10;i<=99999;i++)

{

len=judgeDigit(i);

judgeEqual(i,len);

}

printf("\n");

system("PAUSE");

return 0;

}

int judgeDigit(int m)

{/\*This function return the digit of parameter m\*/

int len=0;

do

{

++len;

m=m/10;

}while(m);

return len;

}

void judgeEqual(int m,int n)

{/\*parameter m is a integer,parameter n is the digit of m,this function is used to judge m whether is a Armstrong integer and output it\*/

int j,temp=m,sum=0;

for(j=1;j<=n;j++)

{

sum+=(int)(pow(temp%10,n));

temp=temp/10;

}

if(m==sum)/\*if m is equal to sum,that is to say m is a Armstrong integer\*/

printf("%d\t",m);

}

【程序输入输出】

no input；

output：

All 2 digit to 5 digit Armstrong integers are following:

153 370 371 407 1634 8208 9474 54748 92727 93084

注：用gcc调试就得不到153这个结果，但windows下用vc6.0就可以得到。不解中，这是编译器问题还是程序问题？

3、

【问题描述】将1,2,3,4,5,6,7,8,9共9个数分成三组,组成3个三位数,且使这3个三位数构成1:2:3的比例,例如:3个三位数192,384,576满足以上条件.192:384:576=1:2:3。试求出所有满足条件的3个三位数。

【思路】1〜9组成的最小三位数是123，最大的是987，由于要满足1：2：3的关系，最小的那个数应该不到于987/3＝329。这样的话第一个数的变化范围是123〜329，将这里面的数分别乘2、乘3，然后判断这三个数是否符合要求，即这三个数是否由1〜9组成，而且各个数字不能相同。

即对每个数n(123<=n<=329)用枚举法。

定义函数int judge(int n)，用于判断整数n的各位数字是否相同，如果有想同的就返回0；否则返回1；

对每个数n(123<=n<=329)，2\*n，3\*n调用judge()函数用于判断这三个数是否由1~9组成且各个数字不相同；

判断n，2\*n，3\*n这三个数中的各位数是否相同，所以对数n\*1000\*1000+2\*n\*1000+3\*n调用judge()判断。

所以(judge(n)==0||judge(2\*n)==0||judge(3\*n)==0||judge(n\*1000+2\*n\*100+3\*n)==0)为真的话，即其中给定的n不符合要求。其实只要(judge(n\*1000+2\*n\*100+3\*n)==0)这一个条件即可，因为它包含了前面两种情况。

　　caution：其实要判断这三个数是否由1~9组成且各个数组不相同，即这三个数的各位数是否覆盖了1~9，只要判断各位数字的积是否等于9！且各位数字的和是否等于45。

【代码】

cCODE:

#include<stdio.h>

#include<stdlib.h>

int judge(int n);

int main (int argc, char \*\*argv)

{

int l,m,n,p,q;

for(l=123;l<=329;l++)

{

m=2\*l,n=3\*l;

p=l\*1000+m,q=p\*1000+n;

if(judge(q)==0)

//判断l、m、n是否符合要求。如果不符合就跳出本次循环，进入下次循环

continue;

printf("%d,%d,%d\n",l,m,n);

}

system("PAUSE");

return 0;

}

int judge(int n)

{//用于判断整数n的各位数字是否相同，如果有想同的就返回0；否则返回1

int num[10],i,j,len=0,temp=n;

do

{

++len;

temp=temp/10;

}while(temp);//求出n的位数

for(i=1;i<=len;i++)

{//将n的各位数字存入num［］，并判断是否存在0及相同的数字，如果存在就返回0

if((num=n%10)==0)

return 0;

n=n/10;

for(j=1;j<i;j++)

if(num[j]==num)

return 0;

}

return 1;

}

cCODE：来自一位网友youshuang，即用判断各位数字的积是否等于9！且各位数字的和是否等于45。)

#include <stdio.h>

bool judge( int a, int b, int c )

{

char tmp\_buf[ 10 ];

sprintf( tmp\_buf, "%d%d%d", a, b, c );

int nTimeResult = 1;

int nSumResult = 0;

for ( int i = 0; i < 9; i++ )

{

nTimeResult \*= ( tmp\_buf[ i ] - '0' );

nSumResult += ( tmp\_buf[ i ] - '0' );

}

return ( ( nTimeResult == 362880 ) && ( nSumResult == 45 ) );

}

int main()

{

for ( int i = 123; i <= 329; i++ )

{

if ( judge( i, i \* 2, i \* 3 ) )

{

printf( "%d, %d, %d \n", i, i \* 2, i \* 3 );

}

}

return 0;

}

【程序输入输出】

no input；

output：

192,384,576

219,438,657

273,546,819

327,654,981

4、

【问题描述】和尚挑水

某寺庙里7个和尚:轮流挑水，为了和其他任务不能冲突，各人将有空天数列出如下表：

和尚1: 星期二,四;

和尚2: 星期一,六;

和尚3: 星期三,日;

和尚4: 星期五;

和尚5: 星期一,四,六;

和尚6: 星期二,五;

和尚7: 星期三,六,日;

请将所有合理的挑水时间安排表

【思路】用回朔法求解（递归方式实现，当然也可以用迭代方式）。用结构体存储和尚的信息（空闲时间、是否已经挑过水标记）回朔法即每进行一步，都试图在当前部分解的基础上扩大该部分解。扩大时，首先检查扩大后是否违反了约束条件，若不违反，则扩大之，然后继续在此基础上按照类似的方法进行，直至成为完整解；若违反，则放弃该步以及它所能生成的部分解，然后按照类似的方法尝试其他可能的扩大方式，直到尝试了所有的扩大方式。

【代码】

cCODE:

#include<stdio.h>

#include<stdlib.h>

void backtrack(int n);

/\*函数功能：回朔求解第n天至第7天的解（即第n〜7天分别安排和尚几）\*/

struct st

{

int spare[8];

/\*存储和尚的空闲时间，spare=0表示星期i没有空闲，spare=1表示星期i空闲,其中spare[0]不用\*/

int flag;

/\*用于标记和尚周内是否已经工作过，flag＝0表示没挑过水，flag＝1表示已经挑过水\*/

}monk[8];

int x[8],sum=0;/\*sum用于统计共有多少种方案\*/

int main (int argc, char \*\*argv)

{

int i,j;

for(i=1;i<=7;i++)

{/\*初始化和尚的空闲时间,初始化时和尚全部没挑过水即flag都为0\*/

printf("请输入和尚%d的空闲时间：",i);

for(j=1;j<=7;j++)

{

scanf("%d",&monk.spare[j]);

}

monk.flag=0;

}

backtrack(1);

printf("共有%d种方案\n",sum);

}

void backtrack(int n)

{/\*函数功能：回朔求解第n天至第7天的解（即第n〜7天分别安排和尚几）\*/

int j;

if(n>7)

{

sum++;

printf("方案%d:\n",sum);

for(j=1;j<=7;j++)

{

printf("星期%d和尚%d挑水\n",j,x[j]);

}

printf("\n");

}

else

{

for(j=1;j<=7;j++)

{

x[n]=j;

if(monk[j].flag==0&&monk[j].spare[n]==1)

{//判断和尚j是否已经挑过水及和尚星期n是否有空

monk[j].flag=1;

backtrack(n+1);

monk[j].flag=0;

}

}

}

}

【程序输入输出】

input:

请输入和尚1的空闲时间：0 1 0 1 0 0 0

请输入和尚2的空闲时间：1 0 0 0 0 1 0

请输入和尚3的空闲时间：0 0 1 0 0 0 1

请输入和尚4的空闲时间：0 0 0 0 1 0 0

请输入和尚5的空闲时间：1 0 0 1 0 1 0

请输入和尚6的空闲时间：0 1 0 0 1 0 0

请输入和尚7的空闲时间：0 0 1 0 0 1 1

output:

方案1:

星期1和尚2挑水

星期2和尚6挑水

星期3和尚3挑水

星期4和尚1挑水

星期5和尚4挑水

星期6和尚5挑水

星期7和尚7挑水

方案2:

星期1和尚2挑水

星期2和尚6挑水

星期3和尚7挑水

星期4和尚1挑水

星期5和尚4挑水

星期6和尚5挑水

星期7和尚3挑水

方案3:

星期1和尚5挑水

星期2和尚6挑水

星期3和尚3挑水

星期4和尚1挑水

星期5和尚4挑水

星期6和尚2挑水

星期7和尚7挑水

方案4:

星期1和尚5挑水

星期2和尚6挑水

星期3和尚7挑水

星期4和尚1挑水

星期5和尚4挑水

星期6和尚2挑水

星期7和尚3挑水

共有4种方案

5、

【问题描述】编写一个c程序，利用如下的格里高利公式求п的值，直到最后一项的值小于10-6为止。

【思路】由公式可以看出，每次n的值都会改变，这实际上就是迭代。

在程序设计中，迭代是经常使用的一种算法。使用迭代算法时要注意三个问题：

迭代的初始值，如本题中sum的初始值为1，n的初始值为1

迭代公式，这是迭代的关键，如果有几个迭代公式，要特别注意这些迭代的顺序。如i+=1和sum+=n的次序不能交换。

迭代终止条件。一般用一个表达式或者计数器来判断迭代式是否应该终止。

本题中迭代式中i+=1，i的初始值为1；sum+=n or sum-=n这通过一个标志变量flag来控制，flag==1时sum+=n，flag==0时sum-=n，sum的初始值为1。终止条件为。

【代码】

cCODE:

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

int main (int argc, char \*\*argv)

{

int flag=0,i=1;

double n=1,sum=1;

while(n>=pow(10,-6))

{

n=(double)1/(2\*i+1);

i+=1;

if(1==flag)

{

sum+=n;

flag=0;

}

else

{

sum-=n;

flag=1;

}

}

sum\*=4;

printf("%lf",sum);

system("PAUSE");

return 0;

}

【程序输入输出】

No input；

Output：

3.141595

6、

【问题描述】编写一个c程序，把下列数组延长到第50项：

1，2，5，10，21，42，85，170，341，682

【思路】由给定的数组元素可以看出偶数项是前一项的2倍，奇数项是前一项的2倍加1。

即，这是一中递推关系由前项推出后项，此题可以通过递推关系求解。

递推解题和迭代解题是很相似的，递推是通过其他变量来演化，而迭代则是通过自身不断演化。递推法的运用也有三个关键：

寻找递推关系。这是最重要的问题。递推关系有解析和非解析两种。解析递推关系是指能用一般数学公式描述的关系，也称递推公式。例如，本题的递推关系就是解析的。非解析递推关系是指不能用一般的数学公式描述的关系，这类关系的描述，也许本身就是一个过程。这类问题一般比较复杂，要结合其他的策略如分治法来解决。

递推关系必须有始基，即最小子解（针对初始规模的子解的值），没有始基，递推计算就不能开始。例如本题a1=1就是始基。

递推计算。即根据递推关系进行递推计算。递推计算可以由递归解析和非递归两种，递归计算是采用递归法，其形式是自顶向下，而非递归则是自底向上。本题是非递归的。

解此题还须注意一点：数列的项必须定义为double型，因为延长到第50项如果定义为int or float型，数列的项会被截断即超过int和float的表示范围。

【代码】

cCODE:

#include<stdio.h>

#include<stdlib.h>

int main (int argc, char \*\*argv)

{

double a1=1,a=0;

int i=1;

while(i<=50)

{

printf("%.0lf ",a1); //'.0’ is just for do not output the decimal place

i++;

if(i%2==1)

a=2\*a1+1;

else

a=2\*a1;

a1=a;

}

system("PAUSE");

return 0;

}

【程序输入输出】

No input；

Output：

1 2 5 10 21 42 85 .......（略）

7、

【问题描述】 用递归算法实现求一个数组中的最大元素。

【思路】解决递归问题可以分为两个部分，第一部分是一些特殊（基础）情况，用直接法解，即始基；第二部分与原问题相似，可用类似的方法解决（即递归），但比原问题的规模要小。

本题显然始基是a[0],关键是要找出递归关系，定义一个函数int max(int a[],int n)，其中整型a[]是一个数组，n是数组长度减1，即数组最大有效元素的下标，因为c语言中数组元素下标是从0开始的。

如果0==n，则a[0]就是最大的元素

如果n>0，则先求出a[0]到a[n-1]的最大元素，然后与a[n]比较，较大者即为最大元素。其中a[0]到a[n-1]又可以用这种方式求，此时需要将a[0],a[1]...a[n-1]看成一个由n-1个元素构成的一维数组。

【代码】

cCODE:

#include<stdio.h>

#include<stdlib.h>

int max(int a[],int n);

int main (int argc, char \*\*argv)

{

int a[]={1,2,3,4,5,6,7,6};

printf("The max element is %d!\n",max(a,7));

/\*caution:he length of a is 8,but the argument is 7\*/

system("PAUSE");

return 0;

}

int max(int a[],int n)

{

if(0==n)

return a[n];

else

return (a[n]>max(a,n-1)?a[n]:max(a,n-1));

}

【程序输入输出】

no input;

output:

The max element is 7!

8、

【问题描述】自然数的拆分：任何一个大于1的自然数N，总可以拆分成若干个自然数之和，并且有多种拆分方法。例如自然数5，可以有如下一些拆分方法：

5=1+1+1+1+1

5=1+1+1+2

5=1+2+2

5=1+4

5=2+3

【思路】自然数的拆分可以用回溯法。

知识点：回溯法解题时，对任一解的生产，一般采用逐步扩大解的方式。每进行一步，都试图在当前部分解的基础上扩大该部分解。扩大时，首先检查扩大后是否违反了约束条件，若不违反，则扩大之，然后继续在此基础上按照类似的方法进行，直至为完全解；若违反，则放弃该步以及它生成的部分解，然后按照类似的方法尝试其他可能的扩大方式，直到已经尝试了所有的扩大方式。

回溯法解题通常包含以下三个步骤：

针对所给问题，定义问题的解空间；如本题对5的拆分来说，1<=拆分的数<=5。

确定易于搜索的解空间结构；如本题对5的拆分来说，用x[]数组来存储解，每个数组元素的取值范围都是1<=拆分的数<=5，从1开始搜索直到5。

搜索解空间，并在搜索过程中用剪枝函数避免无效搜索。如本题对5的拆分来说，为了避免重复，x>=x[j](i>j)，如x[]={2,3}满足条件而x[]={3,2}就不满足条件不是可行解即无效。

回溯法通常有两种实现方式，一种是递归的方式，另一种是迭代的方式。在此就用递归方式，当然迭代的方式也可以。

【代码】cCODE:

#include<stdio.h>

#include<stdlib.h>

void splitN(int n,int m);//n是需要拆分的数，m是拆分的进度。

int x[1024]={0},total=0 ;//total用于计数拆分的方法数，x[]用于存储解

int main()

{

int n ;

printf("please input the natural number n:");

scanf("%d",&n);

splitN(n,1);

printf("There are %d ways to split natural number %d.\n",total,n);

system("PAUSE");

return 0 ;

}

void splitN(int n,int m)

{//n是需要拆分的数，m是拆分的进度。

int rest,i,j;

for(i=1;i<=n;i++)

{//从1开始尝试拆分。

if(i>=x[m-1])

{//拆分的数大于或等于前一个从而保证不重复

x[m]=i ;//将这个数计入结果中。

rest=n-i ;//剩下的数是n-i，如果已经没有剩下的了，并且进度(总的拆分个数)大于1，说明已经得到一个结果。

if(rest==0&&m>1)

{

total++;

printf("%d\t",total);

for(j=1;j<m;j++)

{

printf("%d+",x[j]);

}

printf("%d\n",x[m]);

}

else

{

splitN(rest,m+1);//否则将剩下的数进行进度为m+1拆分。

}

x[m]=0;//取消本次结果，进行下一次拆分。环境恢复，即回溯

}

}

}

【程序输入输出】

input:

please input the natural number n:5

output:

1 1+1+1+1+1

2 1+1+1+2

3 1+1+3

4 1+2+2

5 1+4

6 2+3

There are 6 ways to split natural number 5.