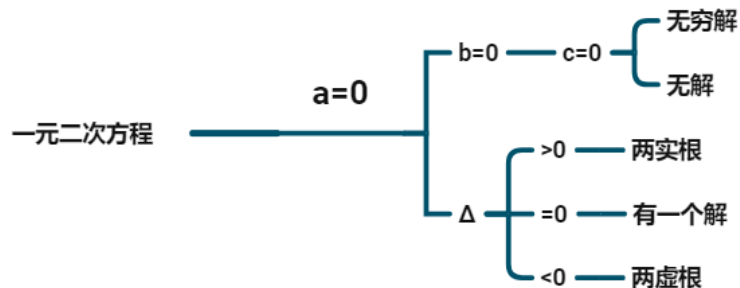


模块化程序

一元二次方程的根



判断能被3, 5, 7整除—switch妙用

```
1  a=b=c=0;
2  if(n%3==0) a=1;
3  if(n%5==0) b=2;
4  if(n%7==0) c=4;
5  switch(a+b+c){
6      case 0:printf("不能被整除"); break; / 最后加 default:
7      case 1:printf("能被3整除");break;
8      case 2:
9      case 4:
10     case 3:
11     case 6:
12     case 5:
13
14 }
```

求x年后 n个闰年

```
1  x=(x/4+1)*4; //求第一个可能的闰年 类似([x/4]+1)*4
2  do{
3      if((x%4==0&& x%100!=0) || (x%400==0)){ //闰年判别：能被4整除不能被100整除 || 能被400
        整除
4          printf x;
5          n--;
6      }
7      x+=4;
8  }while(n!=0);
```

素数

1不是素数

```

1 bool check_prime(int n){
2     for(i=n/2;i>=2;i--){ //'n/2'  '从2开始!!!'
3         if(n%i==0)
4             return false;
5     }
6     return true;
7 }

```

哥德巴赫猜想

```

1 bool goldbach(int n){
2     flag=false;
3     for(num=2;num<n/2;num++){ //从2到n/2
4         if(check_prime(num)&&check_prime(n-num)){ //check_prime是检查素数函数
5             printf(n=num+(n-num));
6             flag=true;
7         }
8     }
9     return flag;
10 }

```

公因数和公倍数

```

1 int gcd(int m,int n){
2     do{
3         t=m%n;
4         m=n;
5         n=t;
6     }while(t==0)
7     return m;
8 }
9 int gcd(int m,int n){
10    if(n==0) return m; //返回值即为gys
11    else return gcd(n,m%n);
12 }
13 gbs=m*n/gys;

```

$$9 \div 6 = 3$$

辗转相除法

$$6 \div 3 = 0$$

最终的除数

余数为零时

判断不可约分数

```
1 | if(gcd(i,j)==1)    //使用公因数函数
```

进制的转换

```
1 | #include <stdio.h>
2 | #include <string.h>
3 | char a[129];    //用来保存进制转化结果
4 | int i=0;
5 | /****将b1进制转化为10进制****/
6 | int change_ten(char num[],int b){
7 |     int n=0,i,len;
8 |     len=strlen(num);
9 |     for(i=0;i<len;i++){
10 |         if(num[i]<='9'&&num[i]>='0') n=n*b+num[i]-'0';//*b不是*10
11 |         else if(num[i]<='Z'&&num[i]>='A') n=n*b+(num[i]-
12 |         'A'+10);//+10!!!!
13 |         else n=n*b+(num[i]-'a'+10);
14 |     }
15 |     return n;
16 | }
17 | /****将10进制转化为b2进制(用递归倒序输出余数)****/
18 | void change_result(int num,int b){
19 |     int y=0;        //y=余数
20 |     if(num){
21 |         change_result(num/b,b);
22 |         y=num%b;
23 |         if(y>=0&&y<=9) a[i++]=y+48; //48=-i+(int)'1'
24 |         else a[i++]=y+55;    //55=-10+(int)'A'
25 |     }
26 | }
27 | /****将10进制转化为b2进制(用循环正序输出余数-逆转)****/
28 | void change_result(int num,int b){
29 |     int i=0,y;        //y=余数
30 |     while(num){
31 |         y=num%b;
32 |         if(y<10) a[i++]=y+48;
33 |         else a[i++]=y+55;
34 |         num=num/b;
35 |     }
36 |     a[i]='\0';
37 |     inverse();
38 | }
39 | int main()
40 | {
41 |     char num[129];
42 |     int num_10;
43 |     int b1,b2;
44 |     scanf("%d%d",&b1,&b2);
45 |     getchar();
46 |     gets(num);
47 |     num_10=change_ten(num,b1);
48 |     change_result(num_10,b2);
49 |     a[i]='\0';
```

```
49     printf("%s",a);
50     return 0;
51 }
```

斐波那契数列

```
1  v=1;
2  u=2;
3  printf(v,u)
4  while(结束条件){
5      w=u+v;
6      u=w;
7      v=u;
8      printf(w)
9  }
```

阶乘

```
1  int factorial(int n){
2      s=1;
3      for(i=1;i<=n;i++){
4          s=s*i;
5      }
6      return s;
7  }
```

跳出多重循环

**在控制条件中增加 &&flag

四舍五入

```
a=(int)((float)n+0.5)
```

浮点数精度

```
fabs(n)>=1e-5
```

输入相关

清除缓冲区

```
1  if(scanf("%d", &n) != 1) {
2      printf("请输入有效选项!\n");
3      while(getchar() != '\n');    //清除缓冲区
4      continue;
5  }
```

反复输入数据

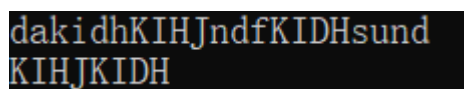
```
1 scanf()
2     while(结束条件){
3         ...//操作//
4         scanf()
5     }
```

滤掉前导字符

```
1 ch=getchar();
2 while(!(符合条件的)&&ch!='#'){
3     ch=getchar();
4 } //此时的ch符合条件
5 //+反复输入数据//
```

滤掉所有不符合条件字符

```
1 ch=getchar();
2 while(ch!='#'){ //注意!!!!这里不能写'\0'因为这不是字符串不会自动补'\0'结尾不是
    题目所给的结束符如'#'就是'\n'
3     while(!(符合条件)&&ch!='#'){ //注意!!!! 千万别漏ch!='#'不然可能会一直等待输入
4         ch=getchar();
5     } //滤掉前导字符; 此时的ch已符合条件
6     while(符合条件){
7         //操作// // (对单个字符操作/保存进数组)
8         ch=getchar();
9     }
10 }
```



此处操作是打印ch

读入单词

若想要将符合条件的字分段保存/当作单词(中间不符条件的即看作分隔符)

```
1 char string[100];
2 int read_word(){ //读入单词并返回字符串长度
3     int j=0;
4     ch=getchar();
5     while(!(符合条件)&&ch!='#'){
6         ch=getchar();
7     }
8     while(符合条件){
9         str[j++]=ch; //j++在此语句结束后才生效--正确
10        ch=getchar();
11    }
12    str[j]='\0'; //ch并非gets要自己补'\0'
13    return j;
14 }
```

```

15  int main(){
16      j=read_word();
17      while(j!=0){
18          //操作//
19          j=read_word;  //反复读入单词
20      }
21  }

```

返回[回文字](#)

gets函数

```

1  scanf("%d",&n);
2  getchar();      //前加getchar()滤掉回车!!!!
3  gets(array);

```

scanf&gets读字符串

- scanf**结束符**是空格或换行符
- gets**结束符**是换行符
- 无空格: `scanf("%s",str)`
- 有空格: `gets(str)`

同理先输入其他类型在输入字符时

```

1  scanf("%d %c",a,ch); //空格!!!!

```

多个字符串输入

```

1  char *str[5],str0[5][100];
2  int i;
3  for(i=0;i<5;i++)
4      str[i]=str0[i]; //填满躯壳
5  for(i=0;i<5;i++)
6      scanf("%s",str[i]); //也可以只用数组

```

- 多个字符串: scanf
- 多行字符串: gets

输入多行数据（有结束符）

包括多组测试数据。每组数据最多100个整数占用一行，以数字0结束(不计入100个整数里)。测试数据不超过20组，最后一行只包括-1，表示输入数据结束。

```

1  int main(void){
2      int i,j;
3      int x,a[101];
4      scanf("%d",&x);
5      while(x!=-1){      //用两层while循环外层做-1停止数据读取
6          i=0;
7          while(x!=0){    //内层做0停止行数据读取
8              a[i++]=x;
9              scanf("%d",&x);
10         }

```

```
11     /***操作***/
12     scanf("%d",&x); //注意还有一个读入是行首数据（外层while的）
13 }
14 return 0;
15 }
```

输出格式相关

puts函数

会自动在结束部分加'\n'

结尾去除多余字符

- 自设标记k

```
1 int k=0;
2 for(i=0;i<n;i++){
3     if(k++>0) printf(" %d",n);
4     else printf("%d",n);
5 }
```

输出图形

“每个数字占用2位英文字符宽度，宽度不足2位的在数字左侧补空格--右对齐”

- 空格：两个
数字：“%2d”（若是左对齐：“%-2d”）

时间输出

```
1 printf("%02d:%02d:%02d",a,b,c);
```

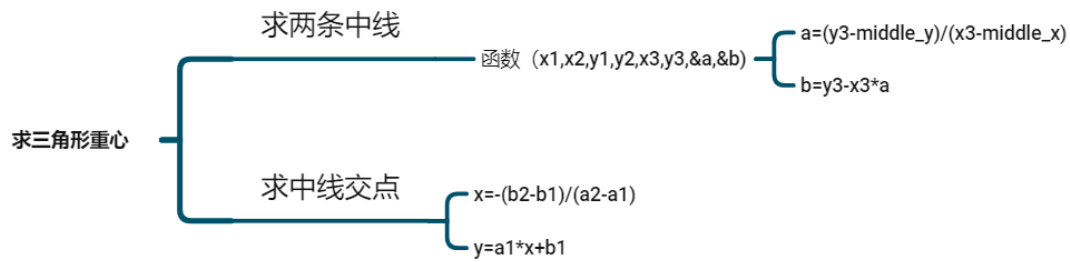
输出格式：小时、分钟、秒都是两位整数，数位不足用零补充

迭代法

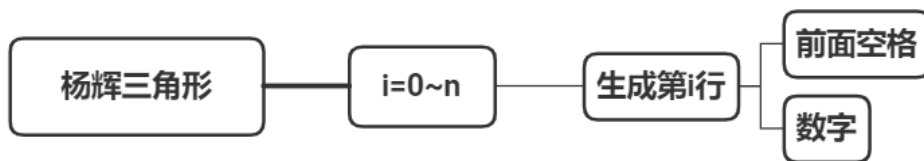
牛顿好牛

求三角形重心

- 坐标/3
- 中线交点



杨辉三角形



1					row0+1
1	1				row1+1
1	2	1			row2+1
1	3	3	1		row3+1
1	4	6	4	1	row4+1

从左至右

`a[i]=b[i]+b[i-1]`

```

1  for(i=0;i<n;i++){           //打印第i行
2      for(j=0;j<n-i-1;j++) printf("  "); //3个空格
3      for(j=0;j<=i;j++){
4          if(j==0 || j==i)
5              a[j]=1;
6          else
7              a[j]=b[j-1]+b[j];
8          printf("%-6d",a[j]); //左对齐x**
9      }
10     for(j=0;j<=i;j++) b[j]=a[j]; //只能打印该行完后才能将a拷贝到b
11     printf("\n");
12 }
```

从右至左

`a[i]=a[i]+a[i-1]`


```

1  for(i=0;i<n;i++){
2      for(j=0;j<n-i-1;j++) printf("  "); //3个空格
3      for(j=i;j>=0;j--){                //打印第i行
4          if(j==0||j==i)
5              a[j]=1;
6          else
7              a[j]=a[j-1]+a[j]; //由于a[j-1]=a[j-1]+a[j-2]用不到改变的a[j]
8          printf("%-6d",a[j]); //左对齐x**
9      }
10     printf("\n");
11 }

```

矩阵

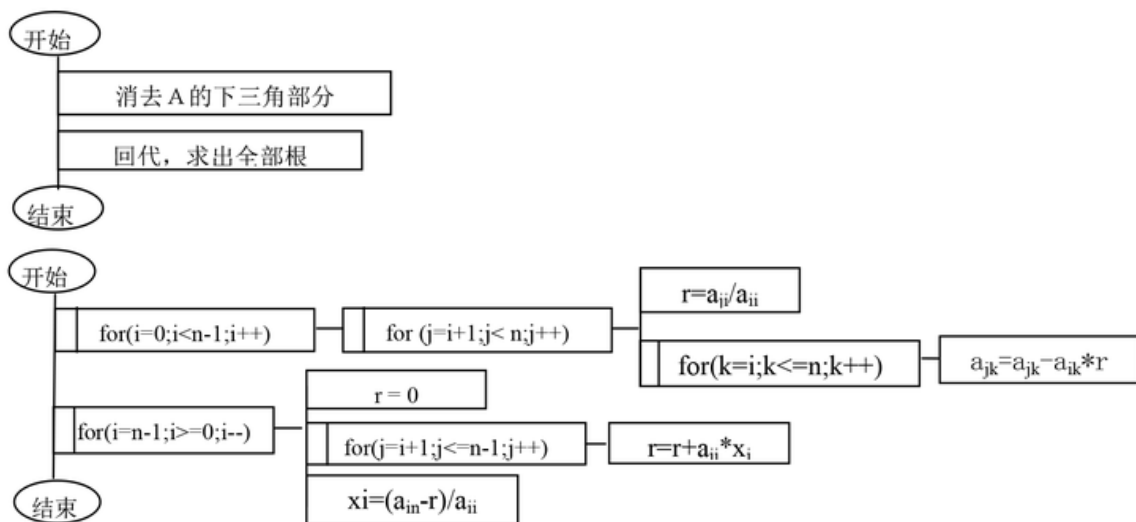
矩阵乘积

```

1  int a[m][p],b[p][n],c[m][n];
2  void matrixproduct(float a[m][p],float b[p][n],float c[m][n]){
3      for(i=0;i<m;i++){
4          for(j=0;j<n;j++){ //不用纠结i,j与m,n的关系
5              e=0;          //计算c每个元素e都要清零
6              for(k=0;k<p;k++){
7                  e=e+a[i][k]*b[k][j]; //左行乘右列
8              }
9              c[i][j]=e;
10     }

```

高斯消去法 $n \times (n+1)$



```

1  #include<stdio.h>
2  //define
3  float matrix[10][11];
4  float x[10];
5  //gauss
6  void gauss(int n){
7      int i,j,k;
8      float temp;

```

```

9  //transform-to-up-triangle//
10  for(j=0;j<n-1;j++){           //与pad图不同: i, j交换->i:行控制/j:列控制
11      for(i=j+1;i<n;i++){
12          temp=matrix[i][j]/matrix[j][j]; //注意是matrix[j][j]!!!!
13          for(k=j;k<n+1;k++)
14              matrix[i][k]=-temp*matrix[j][k]+matrix[i][k]; //第j列从
//j+1~n行依次消去元素a[i][j]变为0
15      }
16  }
17  //回代求根//
18  for(i=n-1;i>=0;i--){
19      temp=0;
20      for(j=i+1;j<=n-1;j++)
21          temp=temp+matrix[i][j]*x[j]; //求已解的解与系数相乘的和
22      x[i]=(matrix[i][n]-temp)/matrix[i][i];
23  }
24
25  }
26  //input//
27  int main (){
28      int i,j;
29      int n;
30      scanf("%d",&n);
31      for(i=0;i<n;i++)
32          for(j=0;j<n+1;j++)
33              scanf("%f",&matrix[i][j]);
34  //solve//
35      gaoss(n);
36  //output//
37      for(i=0;i<n;i++)
38          printf("%.2f ",x[i]);
39      return 0;
40  }

```

转置矩阵

```

1  void T(int matrix[][10],int n){
2      int i,j;
3      int temp;
4      for(i=0;i<n;i++){
5          for(j=i+1;j<n;j++){ //这里j从i+1开始!! 也就是表示遍历右上角元素
6              temp=matrix[i][j];
7              matrix[i][j]=matrix[j][i];
8              matrix[j][i]=temp;
9          }
10     }
11 }

```

n维矩阵的格式输出

```

1  for(i=0;i<n;i++){
2      for(j=0;j<n;j++){
3          if(j==n-1)
4              printf("%d\n",matrix[i][j]);
5          else
6              printf("%d ",matrix[i][j]);
7      }
8  }

```

矩阵 $M \times N$ 的旋转与对称

- 顺时针90°: `new_arr[i][j]=arr[M-1-j][i]` 行列互换 + 列 (j) 补差
- 顺时针180°: `new_arr[i][j]=arr[N-1-i][M-1-j]` 行列补差
- 顺时针270°: `new_arr[i][j]=arr[j][N-1-i]` 行列互换 + 行 (i) 补差
- 左右对称: `new_arr[i][j]=arr[N-1-i][j]` 行补差
- 上下对称: `new_arr[i][j]=arr[i][M-1-j]` 列补差

这里用 `M-1` or `N-1` 去补差取决于减数是 `j` or `i` 一一对应

以下为 $N \times N$ 方阵

- 主对角线对称: `new_arr[i][j]=arr[j][i]` 行列互换
- 副对角线对称: `new_arr[i][j]=arr[N-1-j][N-1-i]` 行列互换+行列补差

八皇后的本质解

运用回溯法与深度优先处理

```

1  #include <stdio.h>
2  #include<stdlib.h>
3  #include<math.h>
4  int m=1;
5  int a[9];
6  int count;
7  int s[92][9]={0};
8  #define N 8
9  /*****比较是否有旋转与对称的解*****/
10 /***关于主对角线对称(行列互换)*****/
11 int compare_symmetryMD(int i){
12     int j;
13     for(j=1;j<=N;j++){
14         if(a[s[i][j]]!=j) return 0;
15     }
16     return 1;
17 }
18 /***关于副对角线对称(行列互换+行列补差)*****/
19 int compare_symmetryVD(int i){
20     int j;
21     for(j=1;j<=N;j++){
22         if(a[N+1-s[i][j]]!=N+1-j) return 0;
23     }
24     return 1;
25 }
26 }
27 /***关于左右对称(行补差)*****/

```

```

28 int compare_symmetryLR(int i){
29     int j;
30     for(j=1;j<=N;j++){
31         if(a[N+1-j]!=s[i][j]) return 0;
32     }
33     return 1;
34 }
35 /**关于上下对称(列补差)***/
36 int compare_symmetryUD(int i){
37     int j;
38     for(j=1;j<=N;j++){
39         if(a[j]!=N+1-s[i][j]) return 0;
40     }
41     return 1;
42 }
43
44 /**顺时针旋转270(行列互换, 行补差)***/
45 int compare_270(int i){
46     int j;
47     for(j=1;j<=N;j++){
48         if(a[s[i][j]]!=N+1-j) return 0;
49     }
50     return 1;
51 }
52
53 /**顺时针旋转180(行列补差)***/
54 int compare_180(int i){
55     int j;
56     for(j=1;j<=N;j++){
57         if(a[N+1-j]!=N+1-s[i][j]) return 0;
58     }
59     return 1;
60 }
61
62 /**顺时针旋转90(行列互换, 列补差)***/
63 int compare_90(int i){
64     int j;
65     for(j=1;j<=N;j++){
66         if(a[N+1-s[i][j]]!=j) return 0;
67     }
68     return 1;
69 }
70 /**与所有s数组内(过滤过)比较是否有重复解***/
71 int equal_check(){
72     int i;
73     for(i=1;i<=count;i++){
74         if(compare_90(i)||compare_270(i)||compare_180(i)||compare_symmetryUD(i)||c
75 ompare_symmetryLR(i)|| compare_symmetryMD(i)||compare_symmetryVD(i)) return
76 0;
77     }
78     return 1;
79 }
80 /**过滤: 非重复解/第一个解保存进数组s***/
81 void filt(void){
82     int i;
83     if(count==0||equal_check()==1){
84         count++;
85         for(i=1;i<=8;i++)

```

```

83         s[count][i]=a[i];
84     }
85     else return;
86
87 }
88 /****不合规则/N行全符合规则就变换位置****/
89 void change(void){
90     while(a[m]==N) m--; //若列在最后一个，退行直到该行列不在最后一个(用来找位置)
91     a[m]++; //不在最后一个则列+1(注意!!! 不在while语句里)
92 }
93 /****符合规则且还没到N行时就延伸****/
94 void extend(void){
95     m++; //进行
96     a[m]=1; //初始化为1
97 }
98 /****检查r行前是否符合规则****/
99 int check(int r){
100     int i;
101     for(i=1;i<r;i++){
102         if(a[r]==a[i]) return 0; //在同行->不合规则
103         if(fabs(a[r]-a[i])==fabs(r-i)) return 0; //在斜对角线->符合规则
104     }
105     return 1;
106 }
107
108 int main(void){
109     int i,j;
110     a[0]=0; //a[0]无用
111     a[1]=1; //a[1]从1开始搜索
112     while(m>0){
113         if(check(m)){
114             if(m==N){ //即当N行全部合格时
115                 filt(); //过滤重复解
116                 change(); //换位进行下一个的查找
117             }else{
118                 extend(); //m<N时,延伸
119             }
120         }else{
121             change(); //不合规则换位
122         }
123     }
124     /****output****/
125     for(i=1;i<=count;i++){
126         printf("No%d:",i);
127         for(j=1;j<=8;j++)
128             printf("%d ",s[i][j]);
129         if(i!=count) printf("\n");
130     }
131     return 0;
132 }

```

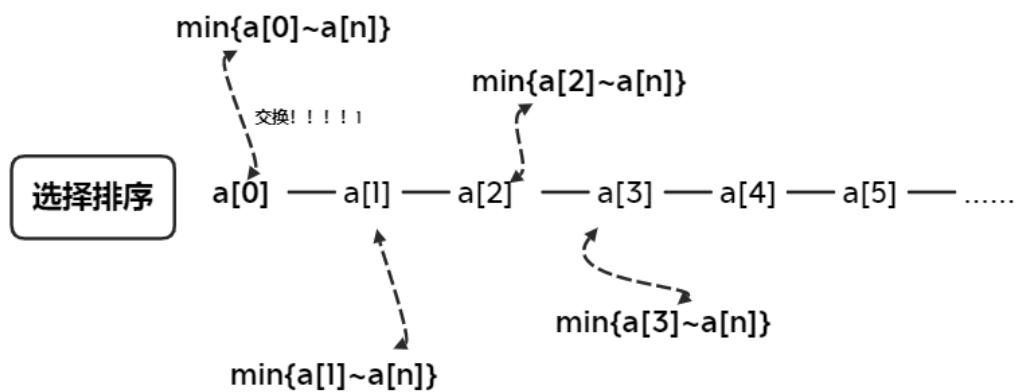
输出:

```
No1:1 5 8 6 3 7 2 4
No2:1 6 8 3 7 4 2 5
No3:2 4 6 8 3 1 7 5
No4:2 5 7 1 3 8 6 4
No5:2 5 7 4 1 8 6 3
No6:2 6 1 7 4 8 3 5
No7:2 6 8 3 1 4 7 5
No8:2 7 3 6 8 5 1 4
No9:2 7 5 8 1 4 6 3
No10:3 5 2 8 1 7 4 6
No11:3 5 8 4 1 7 2 6
No12:3 6 2 5 8 1 7 4
```

[跳回正整数分解](#)

分类

选择排序



```
1 void principal_sorting(int a[size]){
2     for(i=0;i<size;i++){
3         min=a[i];
4         for(j=i;j<size;j++){
5             if(min>a[j])
6                 min=a[j];
7         }
8         a[j]=a[i]; //注意这里是交换a[i]和min!!!不是单纯将min赋值给a[i]!!!!
9         a[i]=min;
10    }
11 }
```

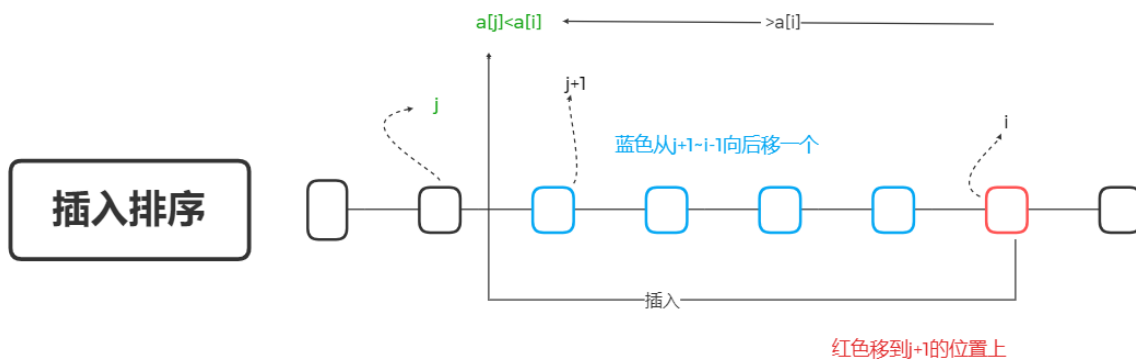
冒泡排序

```

1 void bubble_sort(int a[size]){
2     int flag=1;
3     while(flag){
4         flag=0;
5         for(i=0;i<size-1;i++) //size-1!!!!
6             if(a[i]>a[i+1]){
7                 temp=a[i];
8                 a[i]=a[i+1];
9                 a[i+1]=temp;
10                flag=1;
11            }
12    }
13 }

```

插入排序



```

1 void sequence_sort(int a[size]){
2     for(i=1;i<n;i++){ //i=1!!!!
3         j=i-1;
4         while(a[j]>a[i]&& j>=0) j--; //此时的j就是j
5         temp=a[i];
6         for(k=j+1;k<i;k++) a[k+1]=a[k]; //blue
7         a[j+1]=temp; //red
8     }
9 }

```

检索

顺序检索

```

1 int search(int a[],int n,int key){
2     for(i=0;i<n&& a[i]!=key;i++);
3     if(i==n)
4         return -1;
5     else
6         return i; //返回其下标
7 }

```

二分检索

适用于已排序的数据

```
1  int half_search(int a[],int n,int key){
2      int low,middle,high;
3      low=0;
4      high=n-1;
5      while(high>=low){    //>=!!!
6          middle=(high+low)/2;
7          if(key==a[middle])
8              return middle;
9          else{
10             if(key>a[middle])
11                 low=middle+1;    //+1!!!!
12             else
13                 high=middle-1;    //-1!!!
14         }
15     }
16     return -1;
17 }
```

回文

回文字

```
1  bool check_huiwenzi(char a[]){
2      length=strlen(a);
3      i=0;
4      j=length-1;
5      while(j>i){
6          if(a[i]!=a[j])
7              return false;
8          i++;
9          j--;
10     }
11     return true;
12 }
```

输入部分参考 [读入单词](#)

回文数


```

1  bool check_huiwenshu(int num){
2      int s=0;
3      do{
4          t=num%10;
5          temp=num/10;
6          s=s*10+t;
7      }while(temp!=0);
8      if(s==num)
9          return true;
10     else
11         return false;
12 }

```

自守数

若一个整数a满足条件a * a的尾数等于a则称a为自守数，例如25 * 25=625，76 * 76=5776，9376 * 9376=87909376

```

1  #include<stdio.h>
2  /****检查自守数****/
3  int check_number(int num){
4      long b;
5      b=num*num;
6      while(num!=0){
7          if(num%10!=b%10)
8              return 0;
9          num=num/10;
10         b=b/10;
11     }
12     return 1;
13 }

```

栈 stack

基本操作

声明

```

1  typedef element stack[size]; //栈
2  int top; //栈顶指针->指向栈顶第一个！！空单元！！

```

压入

```

1  bool push(type x){ //也可能是void型
2      if(top>=size)
3          return false;
4      else{
5          stack[top]=x;
6          top++;
7          return true;
8      }
9  }

```

弹出

```
1 type pop(){
2     if(top<=0) //<=0!!如果等于0即栈全空
3         return 错误信号;
4     else{
5         top--;
6         return stack[top]; //pop后此top指向栈顶第一个空单元
7     }
8 }
```

配对

括号的配对(以#为结束符)



```
1 #include<stdio.h>
2 #include<stdlib.h>
3 char stack[30];
4 int top=0;
5 void push(int ch){
6     stack[top]=ch;
7     top++;
8     return;
9 }
10 char pop(void){
11     top--;
12     return stack[top];
13 }
14 void check(char ch){ //易错!!!
15     char temp;
16     temp=pop();
17     if(temp=='#'){
18         printf("右括号多");
19         exit (0); //exit (0) 别忘记!!!
20     }
21     else
22         if(temp!=ch){
23             printf("括号交叉");
24             exit (0); //exit (0) 别忘记!!!
25         }
26     else
27         return ;
28 }
```

```

29 int main (void){
30     char ch;
31     push('#');
32     ch=getchar();
33     while(ch!='#'){
34         swith(ch){
35             case '(':
36             case '[':
37             case '{':push(ch);break;
38             case ')':check('(');break;
39             case ']':check('[');break;
40             case '}':check('{');break;
41         }
42         ch=getchar();
43     }
44     ch=pop;
45     if(ch=='#')
46         printf("匹配");
47     else
48         printf("左括号多");
49     return 0;
50 }

```

典型错误

```

1 void check(char ch){
2     if(pop()=='#') //第一次pop ()
3         printf("右括号多"); //没有exit
4     else
5         if(pop() !=ch) //第二次pop () ->相当于pop了两次!!!! 绝不可
6             printf("括号有交叉"); //没有exit
7         else
8             return 0;
9 }

```

队列 *queue*

基本操作

声明

```

1 typedef element queue[size]; //队列
2 int inpointer=0, outpointer=0, count=0; //送入指针(指向第一个空单元), 取出指针(第一个要
    取出的数据), 计数器

```

进队

```

1  bool putx(type x){
2      if(count>=size)
3          return false;
4      else{
5          queue[inpointer]=x;
6          inpointer=(inpointer+1)%size;
7          count=count+1;
8          return true;
9      }
10 }

```

出队

```

1  bool getx(type *x){          //也可设全局变量/返回值设为x/static
2      if(count<=0)
3          return false;
4      else{
5          x=queue[outpointer];
6          outpointer=(outpointer+1)%size;
7          count=count-1;
8          return true;
9      }
10 }

```

约瑟夫环(out的实现)

```

1  #include<stdio.h>
2  #include<stdlib.h>
3
4  int main (void){
5      int m,n;
6      int count,i=0,k=0;
7      int queue[100]={0};
8      scanf("%d%d",&n,&m);
9      count=n;
10     for(i=0;i<n;i++){
11         queue[i]=1;  //1即还活着
12     }
13     while(count!=0){
14         if(queue[i]!=0){  //当他没死时才能报数
15             k++;          //k用来报数
16             if(k==m){
17                 queue[i]=0;  //记为死亡
18                 printf("%d ",i+1);
19                 count--;      //记人数-控制循环条件
20                 k=0;          //下一个人从1开始报
21             }
22         }
23         i=(i+1)%n;          //队列的+1->因为是循环"%n! ! ! ! !" (不是%count)
24     }
25     return 0;
26 }

```

二维数组传参

声明定义 (形参类型)	调用	函数内操作
指针 (int *p,int m,int n)	(int*)数组名/数组名[0]	a[i*m+j] *(p+i*m+j)
二维数组 (int a[][n])	数组名	a[i][j] *(a[i]+j) *((a+i)+j) *((int*)a+i*n+j)
数组指针 (int (*a)[n])	数组名	a[i][j] *(a[i]+j) *((a+i)+j) *((int*)a+i*n+j)
指针数组 (int *a[])	数组名 (前有声明 int *a[])	*((int*)a+i*n+j)
二级指针 (int **a,int n)	(int**)数组名	*((int*)a+i*n+j)

实际上2 3等价

且当4 5实参为指针a时，操作时可用a[i]

```
1  #include <stdio.h>
2
3  void subfun(int n, char *subargs[])
4  {
5      int i;
6      for (i = 0; i < n; i++) {
7          printf("subargs[%d] = %s\n", i, subargs[i]);
8      }
9  }
10
11 void main()
12 {
13     char *args[3] = {"abc", "def", "ghi"};
14     subfun(3, args);
15 }
16
```

test用函数

循环右移k位

使用两个数组

```
1  void move_right(int a[],int size, int k){
2      int b[size],i;
3      for(i=0;i<n;i++)
4          b[i]=a[i];
5      k=k%size;                //注意这里必须有个‘周期’
6      for(i=0;i<size;i++)
7          a[i]=b[(i+3)%size];  //相当于队列加法
8  }
```

使用一个数组

```
1 void move_right(int a[],int size, int k){
2     int k0,i,temp;
3     k=k%size;
4     for(k0=0;k0<k;k0++){        //（循环右移一位）*k次
5         temp=a[size-1];
6         for(i=size-1;i>0;i--) a[i]=a[i-1];    //循环右移一位
7         a[i]=temp;
8     }
9 }
```

删除重复元素

```
1 int delete_repeat(int a[],int n){
2     int i,j,k;
3     for(i=0;i<n-1;i++){
4         for(j=i+1;j<n;j++){
5             if(a[i]==a[j]){
6                 for(k=j;k<n-1;k++){
7                     a[k]=a[k+1];
8                 }
9                 n--;                //直接n--相当与删除最后一个元素（不用再设一个变量h用
10                //n-h，反正在函数内）
11                j--;                //覆盖后回格
12            }
13        }
14    }
15    return n;
16 }
```

数域的构建

- (1) $1 \in M$;
- (2) 若 $x \in M$, 则 $2x+1 \in M$, $3x+1 \in M$;
- (3) 没有别的整数属于集合 M

编程序按递增顺序生成并输出集合 M 的前 n 项

穷举法

```
1 count=1;
2 a[0]=1;
3 num=1;
4 while(count<n){
5     num++;        //利用穷举法，自然递增
6     for(i=0;i<count;i++){
7         if(num==2*a[i]+1||num==3*a[i]+1){
8             a[count++]=num;
9             break;    //注意这里的break一旦发现该数符合条件，就跳出“循环”，造成重复保
10            //存
11        }
12    }
13 }
```

统计单词个数

“单词”是指连续不含空格的字符串，各单词之间用空格分隔，空格数可以是多个

```
1 int count_words(char str[]){
2     int i,k=0;
3     for(i=0;str[i+1]!='\0';i++){           //遍历字符串数组直到碰到'\0'
4         if(str[i]==' '&&str[i+1]!=' ') //若前一个有空格后一个非空格则计数
5             k++;
6     }
7     return k+1;           //k是空格数，+1即为单词数
8 }
```

访问(遍历)二维数组的元素

当作二维数组

```
1 int p,a[m][n];
2 for(p=0;p<m*n-1;p++){
3     a[p/n][p%n]
4 }
```

当作一维数组

```
1 int *p;
2 int a[m][n];
3 for(p=a[0];p<a[0]+m*n-1;p++){
4     *p
5 }
```

字符串

字符串的复制（赋值）

指针作为形参时，不可以将指针的地址改变，只可以对指针指向的值做改变

将s所指字符串复制到str1中

```
1 strcpy(str1,s); //对-指针指向的值做改变
2
3 v=-1;
4 do{
5     v++;
6     str1[v]=sp[v];
7 }while(sp[v]!='\0'); //对-循环赋值（或用指针或者双指针都可）
```

```
1 str1=s;           //错-指针的地址改变
2
3 *(str1)=*s;        //错-指针指向一个数组（如字符串）-会使空数组str1=s[0]（当如果指针只指向一个值是对的）
```

字符串排序

```
1 void sort_string(char *a[],int n){
2     char *temp; //声明char temp[20]也可，此时后面还可用strcpy
3     /****exchange****/
4     temp=a[i];
5     a[i]=a[k];
6     a[k]=temp;    //对于数组可以改变其地址，因为数组传递规则
7 }
```

求字符串长度

```
1 int str_length(char *str){
2     int i=0;
3     while(*(str+i)!='\0'){
4         i++;
5     }
6     return i;
7 }
```

字符串部分删除

使用自编函数char * str_delete(char *s, int v, int w)从字符串s的第v个字符开始删除w个字符，并将处理后的字符串首地址以函数返回值带回调用点

```
1 char* str_delete(char *s,int v,int w){
2     int i;
3     static char str[256]; //一定要加static!!! 否则则传了函数内声明值的地址（函数
4     执行结束后收回）造成错误
5     /****v前的字符直接复制****/
6     for(i=0;i<v-1;i++)
7         *(str+i)=*(s+i);
8     /****再将v+w后的字符拼接到str后****/
9     strcpy(str+i,s+v-w-1);//重要!!!! str+i即为str后的第一个地址，起拼接作用
10    return str;
11 }
```

复数加法乘法

```
1 /****复数类型结构体声明****/
2 typedef struct complex{
3     float real_part,imaginary_part;
4 }complex;
5 /****复数加法****/
6 complex complex_add(complex x,complex y){
7     complex new;
8     new.real_part=x.real_part+y.real_part;
9     new.imaginary_part=x.imaginary_part+y.imaginary_part;
10    return new;    //返回结构体变量
11 }
12 /****复数乘法****/
13 complex complex_mul(complex x,complex y){
14     complex new;
```

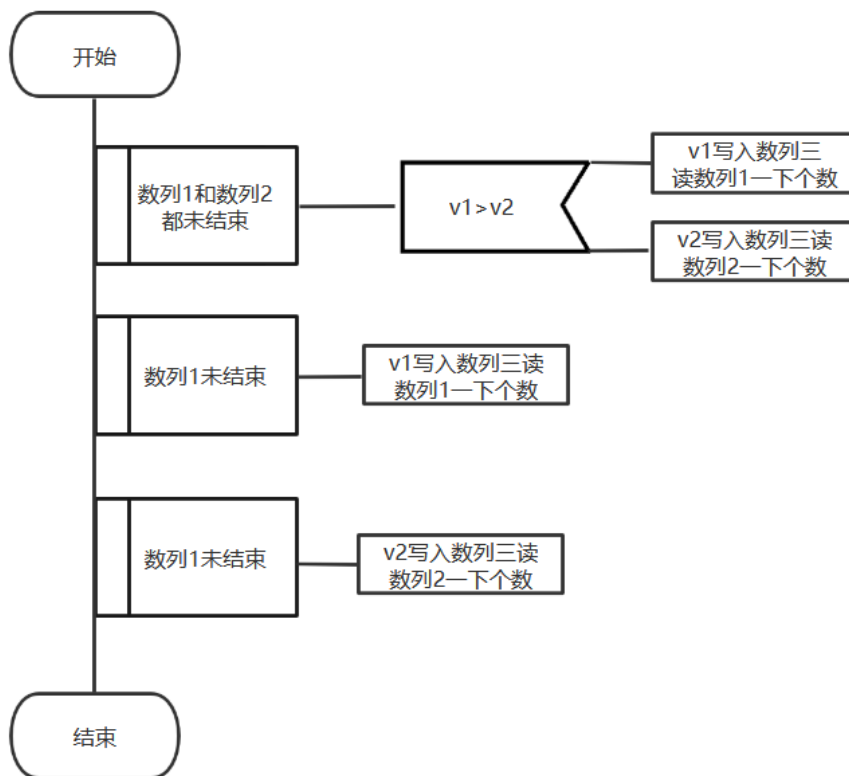


```

15     new.real_part=x.real_part*y.real_part-x.imaginary_part*y.imaginary_part;
16     new.imaginary_part=x.real_part*y.imaginary_part-
    x.imaginary_part*y.real_part;
17     return new;
18 }
19 /****input****/
20 void input(complex *x,complex *y){    //输入函数一般都传指针（不用加&）
21     scanf("%d%d%d%d",x->real_part,x->imaginary_part,y->real_part,y-
    >imaginary_part);
22 }
23 /****output****/
24 void output(complex x){
25     if(x.imaginary_part>0) printf("%d+%di",x.real_part,x.imaginary_part);
26     else
27         if(x.imaginary_part<0) printf("%d%i",x.real_part,x.imaginary_part);
28     else
29         printf("%d",x.real_part);
30 }

```

规则合并两个数列



>此处规则是递减排序

注意这里是while循环

计算调和级数（通分减小误差）

要求结果是分数形式 ($H_n = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}$)

```

1 #include<stdio.h>
2 /****分子分母相加****/
3 void add(int i,int *a,int *b){
4     *a=(*a)*i+(*b);

```

```

5      *b=(*b)*i;
6  }
7  /****求公因数****/
8  int gcd(int m,int n){
9      if(n==0) return m;
10     else return gcd(n,m%n);
11 }
12 /****约分****/
13 void reduce(int *a,int*b){
14     int gys;
15     gys=gcd(*a,*b);
16     *a=(*a)/gys;
17     *b=(*b)/gys;
18 }
19 int main (void){
20     int a=0,b=1,n;          //a=numerator b=denominator注意!!! 这里a相当于和
    单元 b相当于积单元->赋初值1
21     int i;
22     scanf("%d",&n);
23     for(i=1;i<=n;i++){
24         add(i,&a,&b);
25         reduce(&a,&b);
26     }
27     printf("      %d\n",a);
28     printf("Hn-----\n");
29     printf("      %d",b);
30 }

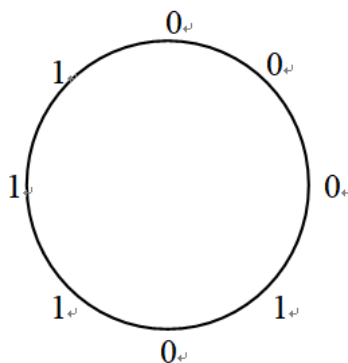
```

Debruijn问题

如图所示由 2^3 个二进制数字0和1组成一个环。使 2^3 个3位的二进制数正好在环中各出现一次。图中目前所示顺序是：0、1、2、5、3、7、6、4。设计生成这样环的程序，环由 2^n 个二进制数字组成，恰好包含 2^n 个互不相同的n位二进制数。

输入：n ($n \leq 4$)

输出：按照字典序输出符合的答案(当出现多组本质不同的解时，仅输出字典序中最小的那个序列)；每行数字间以一个西文空格间隔，行末有一个换行符。



```

1  #include <stdio.h>
2  #include<stdlib.h>
3  #include<math.h>
4  int bin[16]={0}; //用来保存2进制的穷举数
5  int n;
6  /*****二进制转换*****/
7  void binary(int num){

```

```

8     int i;
9     for(i=pow(2,n)-1;i>=0;i--){
10         bin[i]=num%2;
11         num=num/2;
12     }
13 }
14 int main(void){
15     int count,num,i,j,t;
16     int k,s=0;
17     int queue[16]={0}; //标记2^n个n位的二进制数正好在环中各出现一次
18     scanf("%d",&n);
19     t=pow(2,n); //代表有多少个需对应的数
20     count=t;
21     for(i=0;i<16;i++){
22         queue[i]=1; //1: 代表还未出现
23         num=0; //用来穷举数-环的二进制数对应的十进制数
24         while(num<pow(2,t)){ //最大数是2^n*(n-1)-全为1时
25             /*****num转换二进制分别保存进数组bin*****/
26             binary(num); //num->binary_array
27             /****判断环是否对应2^n个互不相同的数0~2^n-1****/
28             for(i=0;i<=t-1;i++){ //对bin数组做一个n元素的滑动窗口
29                 s=0;
30                 for(k=0;k<n;k++) s=s+bin[(i+k)%t]*pow(2,n-1-k); //将滑动窗口的n个
元素二进制->十进制
31                 for(j=0;j<t;j++){ //j代表需对应的数
32                     if(j==s&&queue[j]){ //是否对应一个数(0~2^n)且该数不能被对应过(未
out)
33                         queue[j]=0; //out的应用
34                         count--; //count表示还有n个数未被对应
35                         break;
36                     }
37                 }
38             }
39             /****如果包含2^n个互不相同的n位二进制数则打印****/
40             if(count==0){
41                 for(i=0;i<t;i++){
42                     if(i) printf(" %d",bin[i]);
43                     else printf("%d",bin[i]);
44                 }
45                 exit (0);
46             }
47             /****对下一个num的判断初始化!! 部分条件****/
48             for(i=0;i<16;i++) queue[i]=1;
49             count=t;
50             /****判断下一个数****/
51             num++;
52         }
53     }
54     return 0;
55 }
56

```

递归

深度优先算法用递归写起来比较方便。递归有**两个重要元素**：

- 递归出口
- 递归的表达式

递归对技巧性要求很高，大多数时候其关系式并不是很容易找到。而且对递归的设计与理解，很容易钻到具体细节的实现上。递归的优点就是可以让一些复杂问题简单化，把具体的细节交给计算机执行。而过分钻研细节，就非常容易陷入去理不清头绪。对于递归的学习应该是多看看经典的递归写法，遇到类似问题会模仿写就行了，不一定要自己创造一个递归关系式。

汉诺塔

六十四片->六十三片：把a针上的64片金片,移动到b针上,移动过程中可以利用c针。

1. 把 a 针上的 63 片金片，从 a 针移到 c 针上，移动过程中 b 针可以利用；
2. 把 a 针上的一片金片移到 b 针上；
3. 把 c 针上的 63 片金片，从 c 针移到 b 针上，移动过程中 a 针可以利用。

=>递推关系：把x针上的n片金片,移动到y针上,移动过程中可以利用z针。

1. 把 x 针上的 n-1 片金片，从 x 针移到 z 针上，移动过程中 y 针可以利用；
2. 把 x 针上的一片金片移到 y 针上；
3. 把 z 针上的 n-1 片金片，从 z 针移到 y 针上，移动过程中 x 针可以利用。

递归出口：金片n=0时

```

1  #include<stdio.h>
2  /*step2:移动一步*/
3  void move_one(char m,char n){
4      printf("%c->%c\n",m,n);
5  }
6  /*x 针上的n片金片,移动到y针上,移动过程中可以利用z针*/
7  void move(int n,char x,char y,char z){
8      if(n>0){
9          move(n-1,x,z,y); //1.
10         move_one(x,y); //2.
11         move(n-1,z,y,x); //3.
12     }
13     return;
14 }
15 int main (void){
16     int n;
17     scanf("%d",&n);
18     move(n,'a','b','c');
19     return 0;
20 }
```

齿轮

三齿轮啮合问题。设有三个齿轮互相衔接，求当三个齿轮的某两对齿互相衔接后到下一次这两对齿再互相衔接，每个齿轮最少各转多少圈

求三个数最小公倍数

等价

求其中两个的最小公倍数&第三个数的最小公倍数

```

1 lowestcm3(x,y,z){
2     lowestcm2(lowestcm2(x,y),z);
3 }

```

打印所有组合C(n,m)

从前n个自然数1,2,...,n中取r个数做组合，按递增排序打印所有组合。

n->n-1: 从1开始n个数取r个数进行组合

①从1开始从这n个数取一个数i(1~n-r+1)

②从i开始取r-1个数进行组合

递推关系：从s开始取j个数进行组合

①从s开始从这j个数取一个数i(s~n-j+1)

②从i+1开始取j-1个数进行组合

递归出口：当进行组合的个数(j)为一时

```

1  #include<stdio.h>
2  int r,n;
3  int a[10]={0}; //数组保存之前的i，当j=1(出口)时，打印
4  void combination(int s,int j){ //s=start即取数起点；j代表取j个数进行组合
5      int i,k;
6      for(i=s;i<=n-j+1;i++){ //从s开始取数i
7          a[r-j]=i; //r-j恰好是第i的位置(j=1时也要保存i)
8          if(j>1)
9              combination(i+1,j-1);
10         else{
11             for(k=0;k<r;k++) printf("%-2d",a[k]);
12             printf("\n");
13         }
14     }
15 }
16 int main(void){
17     scanf("%d%d",&n,&r);
18     combination(1,r); //j=r时，start=1，剩下的数为r
19     return 0;
20 }

```

[跳回正整数分解](#)

截木条

给定一个长度为n的木条，将其在大致2/5的位置截断，得到2个长度仍为整数的木条；如果新得到的木条的长度仍旧超过规定长度k，将继续按照上述方法处理得到的木条，直到所有木条的长度都不大于k。编写程序，用递归方法计算一个长度为n的木条，当规定长度为k时，其经过上述截断过程会得到多少根木条。

其中：n、k均为正整数，n>10，k>3，且假设木条截断所得短木条长度四舍五入为正整数，长木条长度为总长减去短木条长度。

```

1  #include <stdio.h>
2  int k;
3  int amount(int a[],int count){

```

```

4     int i;
5     /****遍历数组a若长度大于k就截****/
6     for(i=0;i<count;i++){
7         if(a[i]>k){
8             /****将新截的木棍长度保存进数组a****/
9             a[count]=(int)((float)a[i]*2/5+0.5); //a[count]就是“未开发”数组的第
            一个位置
10            a[i]=a[i]-a[count]; //将原来那个被截的木棍长度用新的木棍（长的）长度覆盖覆
            盖掉
11            i--; //由于此时a[i]是新的木棍长度所以要-1看看他
12            count++; //数组a“已开发”的长度+1
13        }
14    }
15    return count;
16 }
17
18 int main()
19 {
20     int n,num;
21     int a[100]={0}; //用a数组保存
22     scanf("%d",&n,&k);
23     a[0]=n;
24     num=amount(a,1);
25     printf("%d",num);
26     return 0;
27 }

```

求解最长字符串

```

1  #include<stdio.h>
2  #include<string.h>
3  int max;
4  char* StrMax(char *StrArr[ ],int n){ //注意第一个argument的传参
5      if(n==0) return StrArr[max]; //顺序很重要!!!要放在第一个!若和下面那条交换位
        置,就比较了StrArr[0-1]
6      if(strcmp(StrArr[max],StrArr[n-1])<0) max=n;
7      return StrMax(StrArr,n-1); //
8  }
9  int main (void){
10     char str[100][100];
11     int n,i;
12     char *str1[100];
13     char *str_max;
14     scanf("%d",&n);
15     max=n-1;
16     getchar();
17     for(i=0;i<n;i++){
18         gets(str[i]);
19         str1[i]=str[i];
20     }
21     str_max=StrMax(str1,n-1); //第二个参数用来计数
22     printf("%s",str_max);
23     return 0;
24
25 }

```

正整数分解

回溯法与深度优先处理

(仿八皇后方法)

```
1  #include<stdio.h>
2  int n;
3  int a[15]={0}; // 存放划分结果
4  int m= 0; // 数组指针
5  /****输出****/
6  void output(){
7      printf("%d=",n);
8      for(int i=0;i<=m;i++){
9          if(i) printf("+%d",a[i]);
10         else printf("%d",a[i]);
11     }
12     printf("\n");
13
14 }
15 /****检查sum与n的大小关系****/
16 int check(){
17     int sum=0; // 拆分项累加和
18     for(int i=0;i<=m;i++){
19         sum=sum+a[i];
20     }
21     if(sum>n) return 1;
22     if(sum<n) return -1;
23     return 0;
24 }
25 /****sum>n(不符合条件)时换位(回溯、加值)****/
26 void change(){
27     while(check()==1) m--; //回溯直到sum<=n(符合条件)
28     a[m]++; //试探下一个值
29 }
30 /****sum<n(不符合条件)时延伸****/
31 void extend(){
32     m++;
33     a[m]=a[m-1]; //从前一数的值开始,保证递增排序
34 } //注意!!! 延伸不需考虑a[m]++, 由于是“深度优先处理”: 优先扩张m, 不是a[m]!!!! (在回溯时会慢慢一步步扩张a[m])
35
36 int main (void){
37     scanf("%d",&n);
38     a[0]=1; //从1开始
39     while(m>=0){
40         if(check()==1) change(); //sum>n时换位(回溯、加值)
41         else if(check()==-1) extend(); //sum<n(不符合条件)时延伸
42         else{ //sum=n时
43             output(); //输出
44             change(); //换位(回溯、加值)
45         }
46     }
47     return 0;
48 }
49
```

输出结果:

```
4
4=1+1+1+1
4=1+1+2
4=1+3
4=2+2
4=4
```

(n=4)

广度优先处理

```
1  #include<stdio.h>
2  int n;
3  int a[15]={0}; // 存放划分结果
4  int m= 0; // 数组指针
5  /****输出****/
6  void output(){
7      printf("%d=",n);
8      for(int i=0;i<=m;i++){
9          if(i) printf("+%d",a[i]);
10         else printf("%d",a[i]);
11     }
12     printf("\n");
13
14 }
15 /****检查sum与n的大小关系****/
16 int check(){
17     int sum=0; // 拆分项累加和
18     for(int i=0;i<=m;i++){
19         sum=sum+a[i];
20     }
21     if(sum>n) return 1;
22     if(sum<n) return -1;
23     return 0;
24 }
25 /****sum>n(不符合条件)时归零、减值****/
26 void change(){
27     if(a[m]==1) m=0; //由于是“广度优先处理”，故若最右值为1(不能再减)，直接回到根列
28     a[m]--; //对最右/根数减值
29 }
30 /****sum<n(不符合条件)时延伸****/
31 void extend(){
32     m++;
33     a[m]=a[m-1]; //从前一数的值开始，保证递减排序
34 }
35
36 int main (void){
37     scanf("%d",&n);
38     a[0]=n;
39     while(a[0]>0){
40         if(check()==1) change(); //sum>n(不符合条件)时归零、减值
41         else if(check()==-1) extend(); //sum<n(不符合条件)时延伸
42         else{ //sum=n时
43             output(); //输出
44             change(); //换位(归零、减值)
45         }
46     }
47     return 0;
48 }
```


输出结果:

```
4
4=4
4=3+1
4=2+2
4=2+1+1
4=1+1+1+1
```

(n=4)

递归与回溯

(仿组合数法)

递推关系: 从s开始将数r分解, 已分解个数为count

①从s开始取一个数i(s~r)

②从s开始将数r-i分解, 已分解个数为count+1

递归出口: sum=N, 输出, 回溯; sum<N,回溯;

```
1  #include<stdio.h>
2  int N;
3  int a[15]; // 存放划分结果
4  /****输出****/
5  void output(int count){
6      printf("%d=", N);
7      for (int k=0; k<=count-1; k++) printf("%d+", a[k]);
8      printf("%d\n", a[count]);
9  }
10 /****检查sum与n的大小关系****/
11 int check(int count){
12     int sum=0; // 拆分项累加和
13     for(int i=0; i<=count; i++){
14         sum=sum+a[i];
15     }
16     if(sum>N) return 1;
17     if(sum<N) return -1;
18     return 0;
19 }
20 /****分解剩余数r(算法主体部分)****/
21 void division (int s,int r,int count) { //s=start r=rest count是记录已分解的个数
    用来计算sum&输出
22     int i;
23     for(i=s; i<=r; i++){
24         a[count]=i; //保存已分解数 注意!!! 千万别在这写count++下面传
count(still searching for the reason)
25         if(check(count)==-1) division(i,r-i,count+1); //sum<N-继续从i开始将剩
余数分解
26         else if(check(count)==0){output(count);return;} //sum==N-输出+回溯
27         else return; //sum>N-回溯
28     }
29 }
30 }
31
32 int main ()
33 {
34     scanf ("%d", &N);
35     division (1,N,0); //初始时, start=1, rest=N, count=0
```

```
36     return 0;
37 }
```

(以下别人的方法：没搞透彻)

```
1  #include<stdio.h>
2
3  int N;
4
5  int s[31]; // 存放划分结果
6  int m = 0; // 数组指针
7  int sum = 0; // 拆分项累加和
8
9  void division (int i);
10 void output();
11
12 int main ()
13 {
14     scanf ("%d", &N);
15     division (1);
16     return 0;
17 }
18
19 void division (int i) {
20     if (sum == N) {output(); return;}
21     if (sum > N) return; //回溯
22     for (int j=i; j<=N; j++) { //sum<N
23         s[m]=j;
24         m++; //下一层
25         sum+=j;
26         division(j);
27         sum-=j; //sum>=n时返回一步sum减去j，还原s至上一步
28         m--; //还原m至上一步
29     } // 算法主体
30 }
31
32 void output(){
33     printf("%d=", N);
34     for (int k=0; k<m-1; k++) printf("%d+", s[k]);
35     printf("%d\n", s[m-1]);
36 }
```

链表

基本操作

链表节点声明

```
1  typedef struct LinkNode{
2      int data;
3      struct LinkNode *next;
4  }node,*pnode;
```

初始化

```
1 void initial(pnode head){
2     *head=NULL;
3 }
```

销毁

```
1 void release(pnode head) { //释放单链表空间, head是单链表首结点指针
2     pnode p;
3     while(head!=NULL){
4         p=head;
5         head=head->next;
6         free(p);
7     }
8 }
```

打印

```
1 void print(pnode head){
2     pnode p=head;    //有哨兵项: p=head->next
3     while(p!=NULL){
4         printf("%d", p->data);
5         p=p->next;
6     }
7 }
```

创建单向链表

头插

单链表的头部插入都需要改变表头指针的指向，所以不分情况可以直接操作

```
1 /*外层循环调用headpush函数-创建链表*/
2 pnode head;
3 void headpush(int x){ //此时head必须为全局变量!!! 否则函数返回与形参内都需有head
4     pnode p;
5     p=(pnode)malloc(sizeof(node));
6     p->data=x;
7     p->next=head;    //联系
8     head=p;          //改变表头指针的指向
9 }
```

尾插 (以0结束)

```
1 pnode backpush(void){
2     pnode p, head, rear;
3     int x;
4     p=head=rear=NULL;
5     scanf("%d",&x);
6     while(x!=0){
7         p=(pnode)malloc(sizeof(node));
8         p->data=x;
9         p->next=NULL;    //处理尾节点
10        if(rear==NULL){ //判断为空链表
```

```

11         head=p;
12         rear=p;
13     }else{           //判断为非空链表
14         rear->next=p; //联系
15         rear=p;      //改变表尾指针的指向
16     }
17     scanf("%d",&x);
18 }
19 return head;
20 }

```

创建有n个节点的链表（尾插）

```

1  /****无哨兵项****/
2  pnode creatlink(int n){
3      pnode head,p,rear;
4      head=p=rear=NULL;
5      while(n>0){
6          p=(pnode)malloc(sizeof(node));
7          scanf("%d",&p->data);
8          p->next=NULL; //处理尾节点
9          if(rear==NULL){ //判断为空链表
10             head=p;
11             rear=p;
12         }else{           //判断为非空链表
13             rear->next=p; //联系
14             rear=p;      //改变表尾指针的指向
15         }
16         n--;
17     }
18     return head; //返回链表头
19 }
20 /****有哨兵项****/
21 pnode creatlink(int n){
22     pnode head,p,rear;
23     p=(pnode)malloc(sizeof(node)); //头节点单独处理
24     head=rear=p;
25     while(n>0){
26         p=(pnode)malloc(sizeof(node));
27         scanf("%d",&p->data);
28         rear->next=p;
29         rear=p;
30         n--;
31     }
32     rear->next=NULL; //尾节点单独处理
33     return head; //返回链表头
34 }

```

遍历

```

1  /****双指针****/
2  pnode p0,p;
3  p0=NULL; //注意:此时不能用 p0->next=p1;联系二者=>p0是空, 违法操作
4  p=head;
5  while(p!=NULL){
6      //操作

```

```

7     p0=p;
8     p=p->next;
9 }
10  /****单指针****/
11  pnode p;
12  p=head;
13  while(p!=NULL){
14      //操作
15      p=p->next;
16  }

```

查找位置

已知key元素

```

1  p0=NULL;
2  p=head;
3  while(p!=NULL&& p->data!=key){ //重点
4      p0=p;
5      p=p->next;
6  }

```

递增排序

```

1  while(p!=NULL&&n>p->data){
2      p0=p;
3      p=p->next;
4  }

```

中间节点

若中间结点有两个，则设定前一个为中间位置结点

相当总数/2后四舍五入

```

1  int k=1;
2  while(k<(int)((float)count/2+0.5)){
3      p=p->next;
4      k++;
5  }

```

已知位置

正数第k个：pos=k;

倒数第k个：pos=count-k-1

第一个指针从链表的头指针开始遍历，在第k-1步之前，第二个指针保持不动；在第k-1步开始，第二个指针也开始从链表的头指针开始遍历。由于两个指针的距离保持在k-1，当第一个（走在前面的）指针到达链表的尾结点时，第二个指针（走在后面的）指针正好是倒数第k个结点。

```

1  for (i=1;i<pos&&p!=NULL;i++) {
2      p0=p;
3      p=p->next; //pm->no.m
4  }

```

插入

```
1 p=(pnode)malloc(sizeof(node));
2 p->data=x;
3 if(r==head){           //插入在头节点之前
4     p->next=head;
5     head=p;
6 }else{
7     p->next=r;          //联系
8     r0->next=p;         //联系
9     p0=r;              //改变
10 }
```

删除

```
1 if(r==head){           //头删除
2     head=head->next;
3     free(r);
4     r=head;
5 }else{
6     r=r->next;
7     free(r0->next);
8     r0->next=r;
9 }
```

交换

```
1 /*****交换p->next, q->next(有中间变量g)*****/
2 if(p!=NULL&&q!=NULL&&m!=n){ //索引为m,n的结点位于链表中间
3     g=p->next;
4     p->next=q->next;
5     q->next=g;
6 }
7 /*****交换p0->next(p),q0->next(q)*****/
8 if(p==head&&q!=head){ //p0==NULL
9     q0->next=p;
10    head=q;
11 }
12 if(q==head&&p!=head){ //q0=NULL
13     p0->next=q;
14     head=p;
15 }
16 if(q!=head&&p!=head){ //q0=NULL
17     p0->next=q;
18     q0->next=p;
19 }
20 /*****重新配对p0,p,q0,q(若无后续操作可以不要)*****/
21 p=p0->next;
22 q=q0->next;
```

例题

实时保存排序

输入以0为结束符的0~1若干实数，按递增顺序实时保存

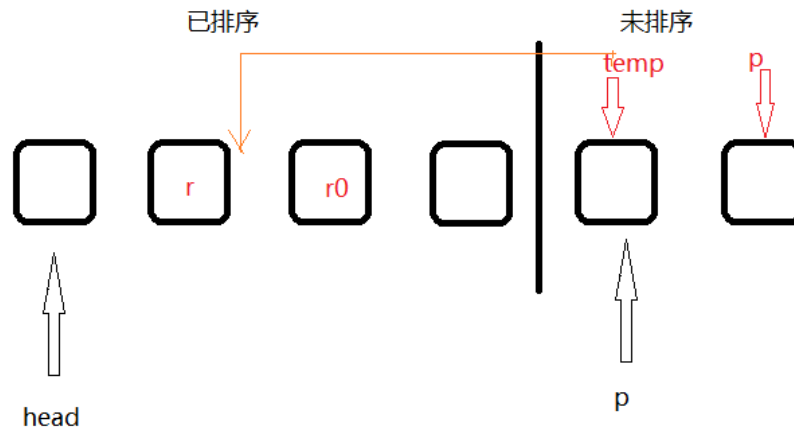
```
1  #include <stdio.h>
2  #include <math.h>
3  #include<stdlib.h>
4
5  typedef struct LinkNode{
6      float data;
7      struct LinkNode *next;
8  }node,*pnode;
9
10 pnode insert(void){
11     float n;
12     pnode head,rear,r,r0,p;  //r,r0标记插入位置;p为工作指针
13     /*****构造0*****/
14     head=(pnode)malloc(sizeof(node));
15     head->data=0;
16     /*****构造1*****/
17     rear=(pnode)malloc(sizeof(node));
18     rear->data=1;
19     /*****连接0、1*****/
20     head->next=rear;
21     rear->next=NULL;  //不要忘记!!!
22     /****以0结束创建单向链表****/
23     scanf("%f",&n);
24     while(fabs(n)>1e-5){
25         r=head;
26         r0=NULL;
27         /*****查找位置*****/
28         while(r!=NULL&&n>r->data){
29             r0=r;
30             r=r->next;
31         }
32         /*****插入*****/
33         p=(pnode)malloc(sizeof(node));
34         p->data=n;
35         r0->next=p;
36         p->next=r;
37         r0=p;
38         scanf("%f",&n);
39     }
40     return head;
41 }
42
43 void print(pnode head){
44     pnode p=head;  //有哨兵项:p=head->next
45     while(p!=NULL){
46         printf("%f ",p->data);
47         p=p->next;
48     }
49 }
50
51 void release(pnode head) {//释放单链表空间,head是单链表首结点指针
```

```

52     pnode p;
53     while(head!=NULL){
54         p=head;
55         head=head->next;
56         free(p);
57     }
58 }
59
60 int main(void) {
61     pnode head;
62     head=insert();
63     if(head!=NULL)print(head);
64     else printf("NULL");
65     release(head);
66     return 0;
67 }

```

插入排序



```

1  pnode insert_sort(pnode head){
2      pnode p,r,r0,temp; //p是无序链表头;head是有序链表头;r,r0标记p插入有序链表的位置;temp用来指向独立的p
3      p=head->next;      //p是无序链表头,从第二个数开始
4      head->next=NULL; //head是有序链表头,此时,头即是尾
5      while(p!=NULL){
6          /*****把p独立出来!!!!!!!!!!!!!!!!!!!!!! (不然后面疯狂牵扯他一塌糊涂)用temp指向它*****/
7              temp=p;
8              p=p->next;
9          /*****找有序链表中插入位置*****/
10             r=head;
11             r0=NULL;
12             while(r!=NULL&&(r->data<temp->data)){r0=r;r=r->next;}
13         /*****分情况插入*****/
14         if(r==head){
15             temp->next=head;
16             head=temp;
17         }else{
18             temp->next=r;
19             r0->next=temp;
20             r0=temp;

```



```

21         }
22     }
23     return head;
24 }

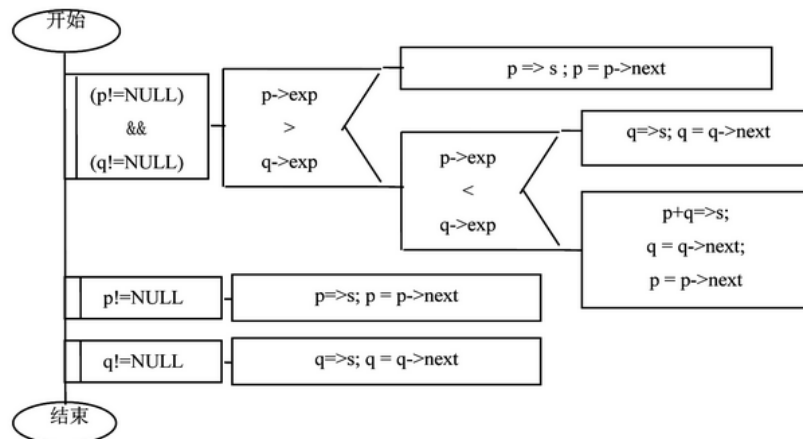
```

对于遍历用的p若要改变其连接顺序一定用temp标记后直接看下一个 (p=p->next;)，否则改变连接后，p=p->next就不是原来的下一个了

即遍历指针不能当作工作指针

多项式加法

$$p(x) = 6.5X^5 + 3.4X^2 + X + 0.5$$



```

1  struct LinkNode{
2      float x;
3      int exp;
4      struct LinkNode *next;
5  }node, pnode;
6  /*****构造新项放在r后面*****/
7  void add(int exp0, float x0, pnode r){
8      pnode p;
9      p=(pnode)malloc(sizeof(node));
10     p->exp=exp0;
11     p->x=x0;
12     r->next=p;
13     p->next=NULL;
14 }
15 /*****多项式加法*****/
16 pnode combine(pnode p, pnode q){
17     pnode head, r;    //r是工作指针
18     r=head=(pnode)malloc(sizeof(node)); //申请哨兵变量
19     while((p!=NULL)&&(q!=NULL)){ //两个多项式都没到尾
20         if(p->exp>q->exp){ //q的幂次小于p
21             add(p->exp, p->x, r);
22             p=p->next;
23         }else{
24             if(p->exp<q->exp){ //p的幂次小于q
25                 add(q->exp, q->x, r);
26                 q=q->next;
27             }else{ //p的幂次等于q
28                 add(q->exp, q->x+p->x, r); //注意exp是相同不要相加!!
29                 q=q->next;
30                 p=p->next;

```

```

31     }
32     }
33     r=r->next;
34 }
35 while(p!=NULL){           //p多项式都没到尾
36     add(p->exp,p->x,r);
37     p=p->next;
38     r=r->next;
39 }
40 while(q!=NULL){           //q多项式都没到尾
41     add(q->exp,q->x,r);
42     q=q->next;
43     r=r->next;
44 }
45 }

```

删除重复节点

```

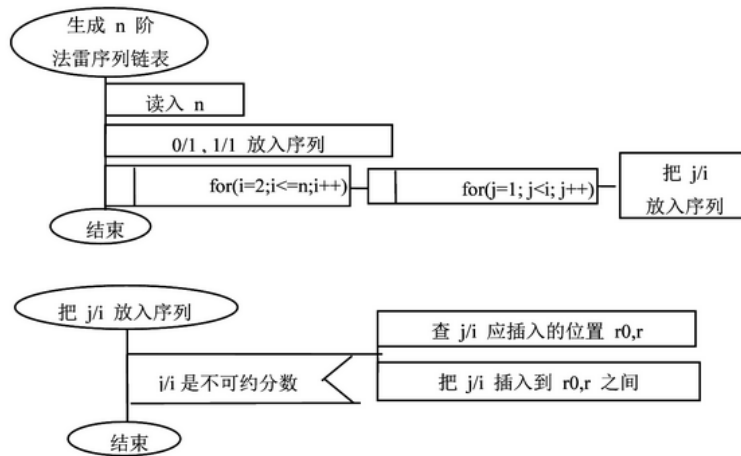
1 void delete_same(pnode head){
2     pnode p,r,r0;
3     p=head;
4     while(p!=NULL){       //不是p->next!=NULL!!!!!!
5         r0=p;
6         r=p->next;
7         while(r!=NULL){
8             if(r->data==p->data){
9                 r=r->next;
10                free(r0->next);
11                r0->next=r;
12            }else{
13                r0=r;
14                r=r->next; //注意!! 这里一定是else否则会漏掉一项
15            }
16        }
17        p=p->next;
18    }
19    return;
20 }

```

构造法雷序列

打印n 阶法雷序列。对任意给定的自然数n，把所有不可约分数按递增顺序排列起来，称该序列为n 阶法雷序列Fn。

$$\frac{j}{i} \quad (0 < i \leq n; 0 \leq j < i) \quad \frac{0}{1}, \frac{1}{8}, \frac{1}{7}, \frac{1}{6}, \frac{1}{5}, \frac{1}{4}, \frac{2}{7}, \frac{3}{8}, \frac{2}{5}, \frac{3}{7}, \frac{1}{2}, \frac{4}{7}, \frac{3}{5}, \frac{5}{8}, \frac{2}{3}, \frac{5}{7}, \frac{3}{4}, \frac{5}{6}, \frac{6}{7}, \frac{7}{8}, \frac{1}{1} \quad (\text{例: F8})$$



```

1 struct farlei_item {
2     int numerator, denominator; // 分子、分母
3     struct farlei_item* next; // 连接部分
4 }node, *pnode;
5 pnode farlei(int n){
6     int i, j;
7     pnode r, r0, p, head, rear;
8     head=rear=r0=r=p=NULL;
9     if(n<1) return NULL; //如果n<=0,则没有法雷序列
10    /*****构造0/1*****/
11    head=(pnode)malloc(sizeof(node));
12    head->numerator=0;
13    head->denominator=1;
14    /*****构造1/1*****/
15    rear=(pnode)malloc(sizeof(node));
16    rear->numerator=0;
17    rear->denominator=1;
18    /*****联系0/1, 1/1*****/
19    head->next=rear;
20    rear->next=p;
21    for(i=2; i<n; i++) //i=denominator
22        for(j=1; j<i; j++){ //j=numerator
23            if(gcd(i, j)==1) { //j/i不可约分
24                /*****找位置*****/
25                r=head->next;
26                r0=head;
27                while(j*(r->denominator)>i*(r->numerator)){
28                    r0=r;
29                    r=r->next;
30                }
31                /*****插入*****/
32                p=(pnode)malloc(sizeof(node));
33                p->numerator=j;
34                p->denominator=i;
35                p->next=r;
36                r0->next=p;
37                r0->next=r;
38            }
39        }
40    return head;
41 }
42
  
```

删除所有的x

```
1 cxxxxxxxxx22 1pnode DelAll(pnode head,int x){2    pnode r,r0;3
  r=r0=NULL;4    while(head&&head->data==x){ //单独处理头节点5        r=head;6
    head=head->next;7        free(r);8    }9    r=head->next; //此时head肯定不
等于x, 故可以从下一个开始遍历10    r0=head; //别忘将r0=head11
while(r!=NULL){12        if(r->data==x){13            r=r->next;14
free(r0->next);15            r0->next=r;16        }else{17
r0=r;18            r=r->next;19        }20    }21    return head;22}
```