栈和队列

7-1 银行业务队列简单模拟

设某银行有A、B两个业务窗口，且处理业务的速度不一样，其中A窗口处理速度是B窗口的2倍 —— 即当A窗口每处理完2个顾客时，B窗口处理完1个顾客。给定到达银行的顾客序列，请按业务完成的顺序输出顾客序列。假定不考虑顾客先后到达的时间间隔，并且当不同窗口同时处理完2个顾客时，A窗口顾客优先输出。

输入格式:

输入为一行正整数，其中第1个数字N(≤1000)为顾客总数，后面跟着N位顾客的编号。编号为奇数的顾客需要到A窗口办理业务，为偶数的顾客则去B窗口。数字间以空格分隔。

输出格式:

按业务处理完成的顺序输出顾客的编号。数字间以空格分隔，但最后一个编号后不能有多余的空格。

输入样例:

```
8 2 1 3 9 4 11 13 15
```

输出样例:

```
1 3 2 9 11 4 13 15
```

代码：

```
#include<iostream>
#include<queue>
#include<cstdio>
#include<algorithm>
using namespace std;
int main()
{
    queue<int> q1,q2;
    int n;
    cin>>n;
    while(n--)
    {
        int m;
        cin>>m;
        if(m%2) q1.push(m);//进入奇数队列
        else q2.push(m);//进入偶数队列
    }
    while(!q1.empty())//弹出奇数队列
    {
        int cnt=2,i=0;
        while(cnt--&&!q1.empty()) {
            if(i++) cout<<" ";
```

```
            cout<<q1.front();
            q1.pop();
        }
        if(!q2.empty()){
            cout<<" "<<q2.front()<<" ";
            q2.pop();
        }
    }
    int i=0;
    while(!q2.empty())//弹出多余的偶数队列
    {
        if(i++) cout<<" ";
        cout<<q2.front();
        q2.pop();
    }
    return 0;
}
```

7-2 表达式转换

算术表达式有前缀表示法、中缀表示法和后缀表示法等形式。日常使用的算术表达式是采用中缀表示法，即二元运算符位于两个运算数中间。请设计程序将中缀表达式转换为后缀表达式。

输入格式:

输入在一行中给出不含空格的中缀表达式，可包含`+`、`-`、`*`、`/`以及左右括号`()`，表达式不超过20个字符。

输出格式:

在一行中输出转换后的后缀表达式，要求不同对象（运算数、运算符号）之间以空格分隔，但结尾不得有多余空格。

输入样例:

```
2+3*(7-4)+8/4
```

输出样例:

```
2 3 7 4 - * + 8 4 / +
```

代码:

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

typedef struct pheap{
    char data[50];
    int top;
}heap;
```

```
int isEmpty(heap *H)//是否空
{
    if(H->top==-1)return 1;
    return 0;
}
void add(heap *H,char x)//入栈
{
    H->data[++H->top]=x;
}
char pop(heap *H)//出栈
{
    return H->data[H->top--];
}
int Big(char a,char b){//注意括号也要比较优先级 ，此函数若第一个参数优先级大，返回1
    if((a=='*'&&(b=='+'||b=='-'))||(a=='/'&&(b=='+'||b=='-'))||b=='('
    return 1;
    else return 0;
}
int isnumber(char a)
{
    if(a>=48&&a<=57)return 1;
    return 0;

}
int get(int start,int *next,char *token,char *str)
{
    int i,j=0;
    //无符号的时候
    if(isnumber(str[start]))
    {
        for(i=start;isnumber(str[i])||str[i]=='.';i++)
        {
                token[j++]=str[i];
        }

        token[j]='\0';
        *next=i;
        return 1;
    }
    //有符号,两种情况;最开头的时候，或者前一位是括号
    else if((str[start]=='+'||str[start]=='-') && (start==0||str[start-1]=='('))
    {
        if(str[start]=='-')
        {
            token[0]='-';
            j=1;
        }
        else
        j=0;
        for(i=start+1;isnumber(str[i])||str[i]=='.';i++)
        {
            token[j++]=str[i];
        }
        token[j]='\0';
        *next=i;
        return 1;
```

```
    }
    else
    {
        token[0]=str[start];
        token[1]='\0';
        *next=start+1;
        return 0;
    }
}
int main()
{
    int i,j=0,next,start=0;
    char str[100],token[100];
    heap *S1,*S2;
    S1=(heap*)malloc(sizeof(heap));S2=(heap*)malloc(sizeof(heap));//S1储存运算符,S2
储存运算数
    S1->top=S2->top=-1;
    S1->data[S1->top]=S2->data[S2->top]='\0';
    gets(str);

    for(i=0;i<strlen(str);i=next)
    {
        int isnum=get(i,&next,token,str);
        //是数字的话入栈S2
        if(isnum)
        {
            for(j=0;token[j]!='\0';j++)
            {
                add(S2,token[j]);
            }
            add(S2,' ');
         }
         //是运算符
         else if(str[i]!='('&&str[i]!=')')
         {
            while(1){
                if(isEmpty(S1)||S1->data[S1->top]=='(')
            {
                add(S1,str[i]);
                break;
            }
            else if(Big(str[i],S1->data[S1->top]))
            {
                add(S1,str[i]);
                break;
            }
            else
            {
                add(S2,pop(S1));
                add(S2,' ');
            }
             }

          }
          //是左右括号
          else
```

```
            {
              if(str[i]=='(')
              {
                  add(S1,str[i]);
                }
                else if(str[i]==')')
                {
                  while(S1->data[S1->top]!='(')
                  {
                      add(S2,pop(S1));
                      add(S2,' ');
                    }
                   pop(S1);//把左括号废弃掉
                }
             }
    }
    while(!isEmpty(S1))
    {
        add(S2,pop(S1));
        add(S2,' ');
    }
    pop(S2);
    for(i=0;i<=S2->top;i++)
    {
        printf("%c",S2->data[i]);
    }
    return 0;
}
```

7-3 堆栈模拟队列

设已知有两个堆栈S1和S2，请用这两个堆栈模拟出一个队列Q。

所谓用堆栈模拟队列，实际上就是通过调用堆栈的下列操作函数:

- `int IsFull(Stack S)`：判断堆栈`S`是否已满，返回1或0；
- `int IsEmpty (Stack S )`：判断堆栈`S`是否为空，返回1或0；
- `void Push(Stack S, ElementType item )`：将元素`item`压入堆栈`S`；
- `ElementType Pop(Stack S )`：删除并返回`S`的栈顶元素。

实现队列的操作，即入队`void AddQ(ElementType item)`和出队`ElementType DeleteQ()`。

输入格式:

输入首先给出两个正整数`N1`和`N2`，表示堆栈`S1`和`S2`的最大容量。随后给出一系列的队列操作：`A item`表示将`item`入列（这里假设`item`为整型数字）；`D`表示出队操作；`T`表示输入结束。

输出格式:

对输入中的每个`D`操作，输出相应出队的数字，或者错误信息`ERROR:Empty`。如果入队操作无法执行，也需要输出`ERROR:Full`。每个输出占1行。

输入样例:

```
3 2
A 1 A 2 A 3 A 4 A 5 D A 6 D A 7 D A 8 D D D D T
```

输出样例:

```
ERROR:Full
1
ERROR:Full
2
3
4
7
8
ERROR:Empty
```

代码:

```
#include<stdio.h>
int main()
{
    int N1,N2;
    scanf("%d %d",&N1,&N2);
    int max,min;
    if(N1>N2)
    {
        max = N1;
        min = N2;
    }
    else
    {
        max = N2;
        min = N1;
    }//最大容量大的那个作为输出栈，小的为输入栈
    int s1[min];
    int s2[max];
    int top1 = 0;
    int top2 = 0;
    char c;
    int item;
    scanf("%c",&c);
    while(c != 'T')//T表示结束
    {
        if(c == 'A')//A入队
        {
            scanf("%d",&item);
            if(top1 < min)
            {
                s1[top1++] = item;//当s1没满，先将item压入s1输入栈中
            }
            else if(top2 == 0)//当s2为空栈时,注意这里是else if
            {
```

```
                while(top1 != 0)
                {
                    s2[top2++] = s1[--top1];//将s1栈中元素转移到s2中，直至s1又为空
                }
                s1[top1++] = item;//继续将元素压入s1
            }
            else//s1已满，s2中存有元素，输出ERROR:FULL
            {
                printf("ERROR:Full\n");
            }
        }
        else if(c == 'D')//D表示出队
        {
            if(top2 != 0)
            {
                printf("%d\n",s2[--top2]);//s2不是空的，从s2中直接输出元素
            }
            else if(top2 == 0 && top1 != 0)
            {
                while(top1 != 0)
                {
                    s2[top2++] = s1[--top1];
                }
                printf("%d\n",s2[--top2]);
            }//S2是空的，S1不是空的，将S1中的元素倒入S2中，再输出栈S2.top()
            else
            {
                printf("ERROR:Empty\n");
            }//S2是空的，S1也是空的，输出ERROR:Empty。
        }
        scanf("%c",&c);
    }
    return 0;
}
```

7-4 输出全排列

请编写程序输出前n个正整数的全排列（n<10），并通过9个测试用例（即n从1到9）观察n逐步增大时程序的运行时间。

输入格式:

输入给出正整数n（<10）。

输出格式:

输出1到n的全排列。每种排列占一行，数字间无空格。排列的输出顺序为字典序，即序列$a1,a2,\cdots,a**n$排在序列$b1,b2,\cdots,b**n$之前，如果存在k使得$a1=b1,\cdots,a**k=b**k$ 并且 $a**k+1<b**k+1$。

输入样例：

```
3
```

输出样例：

```
123
132
213
231
312
321
```

代码：

```
#include<stdio.h>
#define MAX 9
void LeftSwap(int a[],int Left,int i)
{
    int Tag,j;
    Tag = a[i];
    for(j=i;j>Left;j--)
    {
        a[j] = a[j-1];
    }
    a[Left]=Tag;
}
void RihtSwap(int a[],int Left,int i)
{
    int Tag,j;
    Tag=a[Left];
    for(j=Left;j<i;j++)
        a[j]=a[j+1];
    a[i]=Tag;
}
void Array(int a[],int Left,int Right)
{
    int i;
    if(Left==Right)
    {
        for(i=Left;i<=Right;i++)
            printf("%d",a[i]);
    printf("\n");
    }
    else
    {
        for(i=Left;i<=Right;i++)
        {
            LeftSwap(a,Left,i);
            Array(a,Left+1,Right);
            RihtSwap(a,Left,i);
        }
    }
```

```
}
int main()
{
    int n,a[MAX];
    scanf("%d",&n);
    int i;
    for(i=0;i<n;i++)
    {
        a[i]=i+1;
    }
    Array(a,0,n-1);
    return 0;
}
```

7-5 出栈序列的合法性

给定一个最大容量为 *m* 的堆栈，将 *n* 个数字按 1, 2, 3, ..., *n* 的顺序入栈，允许按任何顺序出栈，则哪些数字序列是不可能得到的？例如给定 *m*=5、*n*=7，则我们有可能得到{ 1, 2, 3, 4, 5, 6, 7 }，但不可能得到{ 3, 2, 1, 7, 5, 6, 4 }。

输入格式：

输入第一行给出 3 个不超过 1000 的正整数：*m*（堆栈最大容量）、*n*（入栈元素个数）、*k*（待检查的出栈序列个数）。最后 *k* 行，每行给出 *n* 个数字的出栈序列。所有同行数字以空格间隔。

输出格式：

对每一行出栈序列，如果其的确是有可能得到的合法序列，就在一行中输出`YES`，否则输出`NO`。

输入样例：

```
5 7 5
1 2 3 4 5 6 7
3 2 1 7 5 6 4
7 6 5 4 3 2 1
5 6 4 3 7 2 1
1 7 6 5 4 3 2
```

输出样例：

```
YES
NO
NO
YES
NO
```

代码：

```
#include<stdio.h>
int main(void)
{
    int stack[10000], b[10000];
```

```
    int m,n,k;
    int top;
    int index1,index2;
    scanf("%d %d %d",&m,&n,&k);
    while(k--)
    {
        int t=1;
        int i;
        index1=1,index2=1;
        top=0;
        for(i=1;i<=n;i++)
        scanf("%d",&b[i]);
        while(1)
        {
            //进一个出一个
            if(index1==b[index2])
            {
                index1++;
                index2++;
            }
            //全部进去，然后全部出来
            else if(top!=0&&stack[top-1]==b[index2])
            {
                top--;
                index2++;
            }
            //以上两种情况的综合
            else
            {
                if(index1>n)break;
                stack[top]=index1;//开始进栈
                top++;
                index1++;
                if(top>=m)//超过最大容量时
                {
                    t=0;
                    break;
                }
            }
        }
        //判断其是否合法
        if(t==0||top!=0)
        printf("NO\n");
        else
        printf("YES\n");
    }
    return 0;
}
```

7-6 括号匹配

检查一段C语言代码的小括号`( )`、 中括号`[ ]`和大括号`{ }`是否匹配。

输入格式:

在一行中输入一段C语言代码，长度不超过1000个字符（行末以换行符结束）。

输出格式:

第一行输出左括号的数量和右括号的数量，中间以一个空格间隔。

若括号是匹配的，在第二行打印`YES`，否则打印`NO`。

输入样例1:

```
for(int i=0; i<v; i++){ visited[i] = 0; for(int j=0; j<v; j++) scanf("%d",&(g->Adj[i][j])); }
```

输出样例1:

```
8 8
YES
```

输入样例2:

```
for(int i=0; i<v; i++) a(i]=0;
```

输出样例2:

```
2 2
NO
```

代码:

```
#include<iostream>
#include<string>
using namespace std;

//栈结构体，以SqStack命名
typedef struct {
    char *base;
    char *top;
    int stacksize;
}SqStack;

bool matchChar(char c,char c0);//匹配c和c0是否匹配，如c='[',c0=']'匹配，
                               //注意c='[',若c0='}' 或')' 时就不匹配了
void iniStack(SqStack&S,int size);//初始化栈
void popStack(SqStack&S);//出栈
void pushStack(SqStack&S,char celem);//celem入栈
char getStackTop(SqStack&S);//查看栈顶元素，栈内部结构和元素不改变


int main()
```

```
{
    string cstr;
    int stacksize;//本次匹配所需栈的空间，就是输入字符串的长度
    int l=0;//记录左括号
    int r=0;//记录右括号
    SqStack stk;

    getline(cin,cstr);//输入字符串
    stacksize=cstr.length();//本次匹配所需栈的空间，就是输入字符串的长度
    iniStack(stk,stacksize);//初始化栈

    for(int i=0;i<cstr.length();i++)
    {
            if(cstr[i]=='('||cstr[i]=='{'||cstr[i]=='[')//左括号
            {
                //若遍历到的字符为左括号，绝不可能在一个从左到右（[0]到[cstr.length()]）遍历的字符串中
                //找到匹配，只可能是右括号与栈中存在的左括号匹配，因此左括号不需匹配，直接入栈
                l++;//左括号数加一
                pushStack(stk,cstr[i]);//入栈
            }
            else if(cstr[i]=='}'||cstr[i]==']'||cstr[i]==')')//右括号
            {
                r++;//右括号数加一
                if(matchChar(cstr[i],getStackTop(stk)))//将此时的str[i]与此时栈顶元素匹配，匹配成功返回true，反之返回false
                {
                    popStack(stk);//匹配成功即将栈顶元素出栈
                }
                else
                {
                    pushStack(stk,cstr[i]);//匹配失败即将str[i]入栈
                }
            }
    }
    cout<<l<<' '<<r<<endl;
    if(stk.base==stk.top)//判断栈空，若栈空，则全部匹配成功，YES
        cout<<"YES"<<endl;
    else
        cout<<"NO"<<endl;
    return 0;
}

bool matchChar(char c,char c0) {
    bool is_match=false;
    if(c=='}')
        {
        if(c0=='{')
            is_match=true;
        }
    else if(c==']')
         {
        if(c0=='[')
            is_match=true;
        }
    else if(c==')')
```

```
        {
        if(c0=='(')
            is_match=true;
        }

    return is_match;
}

void iniStack(SqStack&S,int size) {
    S.base=new char[size];
    S.top=S.base;
    S.stacksize=size;
}

void popStack(SqStack&S) {
    if(S.top==S.base)
        cout<<"StackNull,so poperror."<<endl;
    S.top--;
}

void pushStack(SqStack&S,char celem) {
    if(S.top-S.base==S.stacksize)//判断栈满
        cout<<"StackFull,so pusherror."<<endl;
    *S.top=celem;
    S.top++;
}

char getStackTop(SqStack&S) {
    char celem;
    if(S.top==S.base)//判断栈空
        celem='#';
    else
    {
    S.top--;//先将top指针退回指向栈顶元素
    celem=*(S.top);//取栈顶元素
    S.top++;    //再将top指针恢复
    }
    //栈的top指针总是指向有元素存储位置的下一个存储位置（较高存储位）
    return celem;
}
```

7-7 后缀式求值

我们人类习惯于书写“中缀式”，如 `3 + 5 * 2` ，其值为 `13` 。(p.s. 为什么人类习惯中缀式呢？是因为中缀式比后缀式好用么？)

而计算机更加习惯“后缀式”（也叫“逆波兰式”，Reverse Polish Notation）。上述中缀式对应的后缀式是： `3 5 2 * +`

现在，请对输入的后缀式进行求值。

输入格式:

在一行中输入一个后缀式，`运算数`和`运算符`之间用空格分隔，运算数长度不超过`6`位，运算符仅有`+ - * /`四种。

输出格式:

在一行中输出后缀式的值，保留一位小数。

输入样例:

```
3 5.4 2.2 * +
```

输出样例:

```
14.9
```

代码:

```
#include <stdio.h>

int main()
{
    char str[10];
    double st[100];
    int i=0;
    while(~scanf("%s", str))
    {
        if(str[1] == '\0' && (str[0] == '+'||str[0] == '-'||str[0] == '*'||str[0]
== '/'))
        {
            double num1, num2, res;
            num1 = st[--i];//前面一定有两个数了-1
            num2 = st[--i];//前面一定有两个数了-2
            switch(str[0])
            {
            case '+':
                res = num2 + num1;
                st[i++]=res;
                break;
            case '-':
                res = num2 - num1;
                st[i++]=res;
                break;
            case '*':
                res = num2 * num1;
                st[i++]=res;
                break;
            case '/':
                res = num2 / num1;
                st[i++]=res;
                break;
            }
```

```
        }
        else
        {
            double number;
            sscanf(str, "%lf", &number);
            //sscanf()会将参数str 的字符串根据参数format（格式化字符串）来转换并格式化数据， 转换后的结果存于对应的变量中。
            st[i++]=number;
        }
    }
    printf("%.1lf", st[0]);
    return 0;
}
```

7-8 进制转换

输入十进制整数N和待转换的进制x（2、8、16），分别代表十进制N转换成二进制、八进制和十六进制，输出对应的结果。十六进制中A~F用大写字母表示。

输入格式:

输入两个整数N（十进制整数N）和x（x进制），中间用空格隔开。

输出格式:

输出对应的结果。

输入样例1:

在这里给出一组输入。例如：

```
123 2
```

输出样例1:

在这里给出相应的输出。例如：

```
1111011
```

输入样例2:

在这里给出一组输入。例如：

```
123 16
```

输出样例2:

在这里给出相应的输出。例如：

```
7B
```

代码：

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// void print_octal(int n) {
//     if (n == 0) return;
//     print_octal(n / 8);
//     printf("%d", n % 8);
// }

int main() {
    int n, x;
    scanf("%d %d", &n, &x);
    int b, i = 1, j = 0;

    if (x == 2) {
        while (n != 0) {
            b = n % 2;
            n /= 2;
            j += i * b;
            i *= 10;
        }
        printf("%d\n", j);
    }
    else if (x == 8) {
        // print_octal(n);
        // printf("\n");
        char s[10000];
        sprintf(s, "%o", n);
        printf("%s\n", s);
    }
    else if (x == 16) {
        char s[10000];
        sprintf(s, "%X", n);
        printf("%s\n", s);
    }

    return 0;
}
```

```c
// void print_octal(int n) {
//     if (n == 0) return;
//     print_octal(n / 8);
//     printf("%d", n % 8);
// }  的解释：
```

假设我们要打印整数 210（十进制）的八进制表示。

调用 print_octal(210)。
210 除以 8 等于 26 余 2，递归调用 print_octal(26) 并打印余数 2。
26 除以 8 等于 3 余 2，递归调用 print_octal(3) 并打印余数 2。
3 除以 8 等于 0 余 3，因为 n 等于 0，递归结束，打印余数 3。
现在，按照递归调用的顺序反向打印余数：322。

所以，整数 210（十进制）的八进制表示是 322。

7-9 行编辑器

一个简单的行编辑程序的功能是：接受用户从终端输入的程序或数据，并存入用户的数据区。
由于用户在终端上进行输入时，不能保证不出差错，因此，若在编辑程序中，“每接受一个字符即存入用户数据区”的做法显然不是最恰当的。较好的做法是，设立一个输入缓冲区，用以接受用户输入的一行字符，然后逐行存入用户数据区。允许用户输入出差错，并在发现有误时可以及时更正。例如，当用户发现刚刚键入的一个字符是错的时，可补进一个退格符"#"，以表示前一个字符无效；
如果发现当前键入的行内差错较多或难以补救，则可以键入一个退行符"@"，以表示当前行中的字符均无效。
如果已经在行首继续输入'#'符号无效。

输入格式:

输入一个多行的字符序列。但行字符总数（包含退格符和退行符）不大于250。

输出格式:

按照上述说明得到的输出。

输入样例1:

在这里给出一组输入。例如：

```
whli##ilr#e(s#*s)
```

输出样例1:

在这里给出相应的输出。例如：

```
while(*s)
```

输入样例2:

在这里给出一组输入。例如：

```
outcha@putchar(*s=#++);
```

输出样例2:

在这里给出相应的输出。例如：

```
putchar(*s++);
```

代码：

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
typedef struct Stack{
    char ca[201];
    int top;
    int base;
}*stack,STACK;
void initialize(stack);
void push(stack,char);
void pop(stack);
void clear(stack);
void show(stack);
int main()
{   char str[201];
    stack sta=(stack)malloc(sizeof(STACK));
    initialize(sta);
    char c;
    while(gets(str)!=NULL){//注意这里不能使用EOF判断，否则会答案错误，dotcpp上很多都是这样
        for(int i=0;i<strlen(str);i++){
            switch(str[i]){
                case '#':
                    pop(sta);
                    break;
                case '@':
                    clear(sta);
                    break;
                default:
                    push(sta,str[i]);
                    break;
            }
        }
        show(sta);
        clear(sta);
        }
}
void initialize(stack sta){
    sta->top=-1;
    sta->base=0;
}
void push(stack sta,char c){
    sta->ca[++sta->top]=c;
}
void pop(stack sta){
    if(sta->top!=-1)
    sta->top--;
    else
    return;
}
void show(stack sta){
    while(sta->base<=sta->top){
        printf("%c",sta->ca[sta->base++]);
    }
```

```
    printf("\n");
}
void clear(stack sta){
    sta->top=-1;
    sta->base=0;
}
```

7-10 选数

已知n个整数x1,x2,x3...xi，以及1个整数k(k<n)。从 n 个整数中任选 k个整数相加，可分别得到一系列的和。例如当 n=4，k=3，4个整数分别为3,7,12,19 时，可得全部的组合与它们的和为：
3+7+12=22，
3+7+19=29，
7+12+19=38，
3+12+19=34，

现在，要求你计算出和为素数共有多少种。

例如上例，只有一种的和为素数：3+7+19=29

输入格式:

第一行两个空格隔开的整数 n,k（1≤n≤20，k<n）
第二行n个整数，两数之间空格隔开（1≤xi≤1000000）

输出格式:

输出一个整数，表示种类数。

输入样例:

在这里给出一组输入。例如：

```
4 3
3 7 12 19
```

输出样例:

在这里给出相应的输出。例如：

```
1
```

代码：

```
#include<stdio.h>

#include<math.h>
using namespace std;
typedef long long ll;
const int maxn=50;
int a[maxn];
int n,k,ans;
bool prime(int x){
```

```
    for(int i=2;i<=sqrt(x);i++){
        if(x%i==0)return false;
    }
    return true;
}
//t 代表已经选择的数字的个数  sum代表已经选择的数字之和
//d是已经选择的数字中 最后一个数在数组中的位置
void dfs(int t,int sum,int d){
    if(t==k){
        if(prime(sum))ans++;//达到k个 检验是否是素数 计数
    }
    else{
        for(int i=d;i<n;i++){//没到k个  继续从后边找下一个数
            dfs(t+1,sum+a[i],i+1);
        }
    }
}
int main(){
    scanf("%d%d",&n,&k);
    for(int i=0;i<n;i++) scanf("%d",&a[i]);
    dfs(0,0,0);
    printf("%d\n",ans);
    return 0;
}
```

7-11 猴子选大王

由M只猴子围成一圈，从1到M进行编号，打算从中选出一个大王，经过协商，决定选出大王的规则：从第一个开始循环报数，数到K的猴子出圈，下一个猴子从1开始报数，如此循环下去，最后剩下的一只猴子选为猴王。

输入格式:

输入一行中给两个正整数m,k。

输出格式:

输出当选猴王的编号。

输入样例:

在这里给出一组输入。例如：

```
3 2
```

输出样例:

在这里给出相应的输出。例如：

```
3
```

代码：

```
#include<iostream>
using namespace std;
int main()
{
    int n,k;
    cin>>n>>k;
    int i,ans=0;
    for(i=2;i<=n;i++)
        ans=(ans+k)%i;
    cout<<ans+1;
    return 0;
}
```