

目录

一、数学问题.....	4
1.精度计算——大数阶乘.....	4
2.精度计算——乘法（大数乘小数）	5
3.精度计算——乘法（大数乘大数）	6
4.精度计算——加法.....	7
5.精度计算——减法.....	8
6.任意进制转换.....	9
7.最大公约数、最小公倍数.....	10
8.组合序列.....	11
9.快速傅立叶变换（FFT）	12
10.Ronberg 算法计算积分	14
11.行列式计算.....	16
12.求排列组合数.....	17
13.求某一天星期几.....	18
14.卡特兰 (Catalan) 数列 原理.....	18
15.杨辉三角.....	19
16.全排列.....	20
17.匈牙利算法---最大匹配问题.....	22
18.最佳匹配 KM 算法	23
二、字符串处理.....	26
1.字符串替换.....	26
2.字符串查找.....	27
3.字符串截取.....	27
4.LCS-最大公共子串长度	28
5.LCS-最大公共子串长度	29
6.数字转换为字符.....	30
三、计算几何.....	31
1.叉乘法求任意多边形面积.....	31

2.求三角形面积.....	32
3.两矢量间角度.....	32
4.两点距离（2D、3D）.....	33
5.射向法判断点是否在多边形内部.....	33
6.判断点是否在线段上.....	34
7.判断两线段是否相交.....	35
8.判断线段与直线是否相交.....	36
9.点到线段最短距离.....	36
10.求两直线的交点.....	37
11.判断一个封闭图形是凹集还是凸集.....	38
12.Graham 扫描法寻找凸包.....	39
13.求两条线段的交点.....	40
四、数论.....	41
1.x 的二进制长度.....	41
2.返回 x 的二进制表示中从低到高的第 i 位.....	41
3.模取幂运算.....	42
4.求解模线性方程.....	42
5.求解模线性方程组(中国余数定理).....	43
6.筛法素数产生器.....	44
7.判断一个数是否素数.....	45
8.求矩阵最大和.....	46
8.求一个数每一位相加之和.....	47
10.质因数分解.....	47
11.高斯消元法解线性方程组.....	48
五、图论.....	48
1.Prim 算法求最小生成树.....	48
2.Dijkstra 算法求单源最短路径.....	49
3.Bellman-ford 算法求单源最短路径.....	51
4.Floyd-Warshall 算法求每对节点间最短路径.....	51
5.解欧拉图.....	52
六、排序/查找.....	53

1.快速排序.....	53
2.希尔排序.....	54
3.选择法排序.....	55
4.二分查找.....	56
七、数据结构.....	56
1.顺序队列.....	56
2.顺序栈.....	59
3.链表.....	62
4.链栈.....	66
5.二叉树.....	69
八、高精度运算专题.....	71
1.专题函数说明.....	71
2.高精度数比较.....	72
3.高精度数加法.....	72
4.高精度数减法.....	73
5.高精度乘 10.....	73
6.高精度乘单精度.....	74
7.高精度乘高精度.....	75
8.高精度除单精度.....	75
9.高精度除高精度.....	76
九、标准模板库的使用.....	77
1.计算求和.....	77
2.求数组中的最大值.....	78
3. sort 和 qsort.....	79
九、其他.....	80
1.运行时间计算.....	80

一、数学问题

1.精度计算——大数阶乘

语法: `int result=factorial(int n);`

参数:

`n`: `n` 的阶乘

返回值: 阶乘结果的位数

注意:

本程序直接输出 `n!` 的结果, 需要返回结果请保留 `long a[]`

需要 `math.h`

源程序:

```
int factorial(int n)
{
    long a[10000];
    int i,j,l,c,m=0,w;
    a[0]=1;
    for(i=1;i<=n;i++)
    {
        c=0;
        for(j=0;j<=m;j++)
        {
            a[j]=a[j]*i+c;
            c=a[j]/10000;
            a[j]=a[j]% 10000;
        }
        if(c>0) {m++;a[m]=c;}
    }

    w=m*4+log10(a[m])+1;
    printf("\n%ld",a[m]);
    for(i=m-1;i>=0;i--) printf("%4.4ld",a[i]);
    return w;
}
```

2.精度计算——乘法（大数乘小数）

语法: `mult(char c[],char t[],int m);`

参数:

`c[]`: 被乘数, 用字符串表示, 位数不限

`t[]`: 结果, 用字符串表示

`m`: 乘数, 限定 10 以内

返回值: `null`

注意:

需要 `string.h`

源程序:

```
void mult(char c[],char t[],int m)
{
    int i,l,k,flag,add=0;
    char s[100];
    l=strlen(c);
    for (i=0;i<l;i++)
        s[l-i-1]=c[i]-'0';
    for (i=0;i<l;i++)
    {
        k=s[i]*m+add;
        if (k>=10) {s[i]=k%10;add=k/10;flag=1;} else
        {s[i]=k;flag=0;add=0;}
    }
    if (flag) {l=i+1;s[i]=add;} else l=i;
    for (i=0;i<l;i++)
        t[l-1-i]=s[i]+'0';
    t[l]='\0';
}
```

3.精度计算——乘法（大数乘大数）

语法: `mult(char a[],char b[],char s[]);`

参数:

`a[]`: 被乘数, 用字符串表示, 位数不限

`b[]`: 乘数, 用字符串表示, 位数不限

`t[]`: 结果, 用字符串表示

返回值: `null`

注意:

空间复杂度为 $O(n^2)$

需要 `string.h`

源程序:

```
void mult(char a[],char b[],char s[])
{
    int i,j,k=0,alen,blen,sum=0,res[65][65]={0},flag=0;
    char result[65];
    alen=strlen(a);blen=strlen(b);
    for (i=0;i<alen;i++)
        for (j=0;j<blen;j++) res[i][j]=(a[i]-'0')*(b[j]-'0');
    for (i=alen-1;i>=0;i--)
    {
        for (j=blen-1;j>=0;j--) sum=sum+res[i+blen-j-1][j];
        result[k]=sum%10;
        k=k+1;
        sum=sum/10;
    }
    for (i=blen-2;i>=0;i--)
    {
        for (j=0;j<=i;j++) sum=sum+res[i-j][j];
        result[k]=sum%10;
        k=k+1;
        sum=sum/10;
    }
    if (sum!=0) {result[k]=sum;k=k+1;}
    for (i=0;i<k;i++) result[i]+='0';
    for (i=k-1;i>=0;i--) s[i]=result[k-1-i];
    s[k]='\0';
    while(1)
    {
        if (strlen(s)!=strlen(a)&&s[0]=='0')
            strcpy(s,s+1);
        else
            break;
    }
}
```

4.精度计算——加法

语法: `add(char a[],char b[],char s[]);`

参数:

`a[]`: 被加数, 用字符串表示, 位数不限

`b[]`: 加数, 用字符串表示, 位数不限

`s[]`: 结果, 用字符串表示

返回值: `null`

注意:

空间复杂度为 $O(n^2)$

需要 `string.h`

源程序:

```
void add(char a[],char b[],char back[])
{
    int i,j,k,up,x,y,z,l;
    char *c;
    if (strlen(a)>strlen(b)) l=strlen(a)+2; else l=strlen(b)+2;
    c=(char *) malloc(l*sizeof(char));
    i=strlen(a)-1;
    j=strlen(b)-1;
    k=0;up=0;
    while(i>=0||j>=0)
    {
        if(i<0) x='0'; else x=a[i];
        if(j<0) y='0'; else y=b[j];
        z=x-'0'+y-'0';
        if(up) z+=1;
        if(z>9) {up=1;z%=10;} else up=0;
        c[k++]=z+'0';
        i--;j--;
    }
    if(up) c[k++]='1';
    i=0;
    c[k]='\0';
    for(k=1;k>=0;k--)
        back[i++]=c[k];
    back[i]='\0';
}
```

5.精度计算——减法

语法: `sub(char s1[],char s2[],char t[]);`

参数:

`s1[]`: 被减数, 用字符串表示, 位数不限

`s2[]`: 减数, 用字符串表示, 位数不限

`t[]`: 结果, 用字符串表示

返回值: `null`

注意:

默认 $s1 \geq s2$, 程序未处理负数情况

需要 `string.h`

源程序:

```
void sub(char s1[],char s2[],char t[])
{
    int i,l2,l1,k;
    l2=strlen(s2);l1=strlen(s1);
    t[l1]='\0';l1--;
    for (i=l2-1;i>=0;i--,l1--)
    {
        if (s1[l1]-s2[i]>=0)
            t[l1]=s1[l1]-s2[i]+'0';
        else
        {
            t[l1]=10+s1[l1]-s2[i]+'0';
            s1[l1-1]=s1[l1-1]-1;
        }
    }
    k=l1;
    while(s1[k]<0) {s1[k]+=10;s1[k-1]-=1;k--;}
    while(l1>=0) {t[l1]=s1[l1];l1--;}
loop:
    if (t[0]=='0')
    {
        l1=strlen(s1);
        for (i=0;i<l1-1;i++) t[i]=t[i+1];
        t[l1-1]='\0';
        goto loop;
    }
    if (strlen(t)==0) {t[0]='0';t[1]='\0';}
}
```


6.任意进制转换

语法: `conversion(char s[],char s2[],char t[]);`

参数:

`s[]`: 转换前的数字

`s2[]`: 转换后的数字

`d1`: 原进制数

`d2`: 需要转换到的进制数

返回值: `null`

注意:

高于 9 的位数用大写 'A'~'Z' 表示, 2~16 位进制通过验证

源程序:

```
void conversion(char s[],char s2[],long d1,long d2)
{
    long i,j,t,num;
    char c;
    num=0;
    for (i=0;s[i]!='\0';i++)
    {
        if (s[i]<='9'&&s[i]>='0') t=s[i]-'0'; else t=s[i]-'A'+10;
        num=num*d1+t;
    }
    i=0;
    while(1)
    {
        t=num%d2;
        if (t<=9) s2[i]=t+'0'; else s2[i]=t+'A'-10;
        num/=d2;
        if (num==0) break;
        i++;
    }
    for (j=0;j<i/2;j++)
        {c=s2[j];s2[j]=s2[i-j];s2[i-j]=c;}
    s2[i+1]='\0';
}
```

7.最大公约数、最小公倍数

语法: `result=hcf(int a,int b)`、`result=lcd(int a,int b)`

参数:

a: int a, 求最大公约数或最小公倍数

b: int b, 求最大公约数或最小公倍数

返回值: 返回最大公约数 (hcf) 或最小公倍数 (lcd)

注意:

lcd 需要连同 hcf 使用

源程序:

```
int hcf(int a,int b)
{
    int r=0;
    while(b!=0)
    {
        r=a%b;
        a=b;
        b=r;
    }
    return(a);
}

lcd(int u,int v,int h)
{
    return(u*v/h);
}
```

8.组合序列

语法: `m_of_n(int m, int n1, int m1, int* a, int head)`

参数:

`m`: 组合数 C 的上参数

`n1`: 组合数 C 的下参数

`m1`: 组合数 C 的上参数, 递归之用

`*a`: $1\sim n$ 的整数序列数组

`head`: 头指针

返回值: `null`

注意:

`*a` 需要自行产生

初始调用时, `m=m1`、`head=0`

调用例子: 求 $C(m,n)$ 序列: `m_of_n(m,n,m,a,0)`;

源程序:

```
void m_of_n(int m, int n1, int m1, int* a, int head)
{
    int i,t;
    if(m1<0 || m1>n1) return;
    if(m1==n1)
    {
        return;
    }
    m_of_n(m,n1-1,m1,a,head); // 递归调用
    t=a[head];a[head]=a[n1-1+head];a[n1-1+head]=t;
    m_of_n(m,n1-1,m1-1,a,head+1); // 再次递归调用
    t=a[head];a[head]=a[n1-1+head];a[n1-1+head]=t;
}
```

9.快速傅立叶变换 (FFT)

语法: `kkfft(double pr[],double pi[],int n,int k,double fr[],double fi[],int l,int il);`

参数:

`pr[n]`: 输入的实部

`pi[n]`: 数入的虚部

`n, k`: 满足 $n=2^k$

`fr[n]`: 输出的实部

`fi[n]`: 输出的虚部

`l`: 逻辑开关, 0 FFT, 1 ifFT

`il`: 逻辑开关, 0 输出按实部/虚部; 1 输出按模/幅角

返回值: `null`

注意:

需要 `math.h`

源程序:

```
void kkfft(pr,pi,n,k,fr,fi,l,il)
int n,k,l,il;
double pr[],pi[],fr[],fi[];
{
    int it,m,is,i,j,nv,l0;
    double p,q,s,vr,vi,poddr,poddi;
    for (it=0; it<=n-1; it++)
    {
        m=it; is=0;
        for (i=0; i<=k-1; i++)
            {j=m/2; is=2*is+(m-2*j); m=j;}
        fr[it]=pr[is]; fi[it]=pi[is];
    }
    pr[0]=1.0; pi[0]=0.0;
    p=6.283185306/(1.0*n);
    pr[1]=cos(p); pi[1]=-sin(p);
    if (l!=0) pi[1]=-pi[1];
    for (i=2; i<=n-1; i++)
    {
        p=pr[i-1]*pr[1];
        q=pi[i-1]*pi[1];
        s=(pr[i-1]+pi[i-1])*(pr[1]+pi[1]);
        pr[i]=p-q; pi[i]=s-p-q;
    }
    for (it=0; it<=n-2; it=it+2)
    {
        vr=fr[it]; vi=fi[it];
        fr[it]=vr+fr[it+1]; fi[it]=vi+fi[it+1];
        fr[it+1]=vr-fr[it+1]; fi[it+1]=vi-fi[it+1];
    }
}
```

```

    }
    m=n/2; nv=2;
    for (l0=k-2; l0>=0; l0--)
    {
        m=m/2; nv=2*nv;
        for (it=0; it<=(m-1)*nv; it=it+nv)
            for (j=0; j<=(nv/2)-1; j++)
            {
                p=pr[m*j]*fr[it+j+nv/2];
                q=pi[m*j]*fi[it+j+nv/2];
                s=pr[m*j]+pi[m*j];
                s=s*(fr[it+j+nv/2]+fi[it+j+nv/2]);
                poddr=p-q; poddi=s-p-q;
                fr[it+j+nv/2]=fr[it+j]-poddr;
                fi[it+j+nv/2]=fi[it+j]-poddi;
                fr[it+j]=fr[it+j]+poddr;
                fi[it+j]=fi[it+j]+poddi;
            }
    }
    if (l!=0)
        for (i=0; i<=n-1; i++)
        {
            fr[i]=fr[i]/(1.0*n);
            fi[i]=fi[i]/(1.0*n);
        }
    if (il!=0)
        for (i=0; i<=n-1; i++)
        {
            pr[i]=sqrt(fr[i]*fr[i]+fi[i]*fi[i]);
            if (fabs(fr[i])<0.000001*fabs(fi[i]))
            {
                if ((fi[i]*fr[i])>0) pi[i]=90.0;
                else pi[i]=-90.0;
            }
            else
                pi[i]=atan(fi[i]/fr[i])*360.0/6.283185306;
        }
    return;
}

```

10.Ronberg 算法计算积分

语法: `result=integral(double a,double b);`

参数:

a: 积分上限

b: 积分下限

function f: 积分函数

返回值: f 在 (a,b) 之间的积分值

注意:

function f(x)需要自行修改, 程序中用的是 `sina(x)/x`

需要 `math.h`

默认精度要求是 `1e-5`

源程序:

```
double f(double x)
{
    return sin(x)/x; //在这里插入被积函数
}
```

```
double integral(double a,double b)
{
```

```
    double h=b-a;
```

```
    double t1=(1+f(b))*h/2.0;
```

```
    int k=1;
```

```
    double r1,r2,s1,s2,c1,c2,t2;
```

```
loop:
```

```
    double s=0.0;
```

```
    double x=a+h/2.0;
```

```
    while(x<b)
```

```
    {
```

```
        s+=f(x);
```

```
        x+=h;
```

```
    }
```

```
    t2=(t1+h*s)/2.0;
```

```
    s2=t2+(t2-t1)/3.0;
```

```
    if(k==1)
```

```
    {
```

```
        k++;h/=2.0;t1=t2;s1=s2;
```

```
        goto loop;
```

```
    }
```

```
    c2=s2+(s2-s1)/15.0;
```

```
    if(k==2){
```

```
        c1=c2;k++;h/=2.0;
```

```
        t1=t2;s1=s2;
```

```
        goto loop;
```

```
    }
```

```
r2=c2+(c2-c1)/63.0;
if(k==3){
    r1=r2; c1=c2;k++;
    h/=2.0;
    t1=t2;s1=s2;
    goto loop;
}
while(fabs(1-r1/r2)>1e-5){
    r1=r2;c1=c2;k++;
    h/=2.0;
    t1=t2;s1=s2;
    goto loop;
}
return r2;
}
```

11.行列式计算

语法: `result=js(int s[][],int n)`

参数:

`s[][]`: 行列式存储数组

`n`: 行列式维数, 递归用

返回值: 行列式值

注意:

函数中常数 `N` 为行列式维度, 需自行定义

源程序:

```
int js(s,n)
int s[][N],n;
{
    int z,j,k,r,total=0;
    int b[N][N];/*b[N][N]用于存放, 在矩阵 s[N][N]中元素 s[0]的余子式*/
    if(n>2)
    {
        for(z=0;z<n;z++)
        {
            for(j=0;j<n-1;j++)
                for(k=0;k<n-1;k++)
                    if(k>=z) b[j][k]=s[j+1][k+1]; else
b[j][k]=s[j+1][k];
            if(z%2==0) r=s[0][z]*js(b,n-1); /*递归调用*/
            else r=(-1)*s[0][z]*js(b,n-1);
            total=total+r;
        }
    }
    else if(n==2)
        total=s[0][0]*s[1][1]-s[0][1]*s[1][0];
    return total;
}
```


12.求排列组合数

语法: $\text{result} = P(\text{long } n, \text{long } m); / \text{result} = \text{long } C(\text{long } n, \text{long } m);$

参数:

m: 排列组合的上系数

n: 排列组合的下系数

返回值: 排列组合数

注意:

符合数学规则: $m \leq n$

源程序:

```
long P(long n, long m)
{
    long p=1;
    while(m!=0)
        {p*=n;n--;m--;}
    return p;
}
long C(long n, long m)
{
    long i, c=1;
    i=m;
    while(i!=0)
        {c*=n;n--;i--;}
    while(m!=0)
        {c/=m;m--;}
    return c;
}
```

13.求某一天星期几

语法: `result=weekday(int N,int M,int d)`

参数:

`N,M,d`: 年月日, 例如: 2003,11,4

返回值: 0: 星期天, 1 星期一.....

注意:

需要 `math.h`

适用于 1582 年 10 月 15 日之后, 因为罗马教皇格里高利十三世在这一天启用新历法.

源程序:

```
int weekday(int N,int M,int d)
{
int m,n,c,y,w;
m=(M-2)%12;
if (M>=3) n=N;else n=N-1;
c=n/100;
y=n%100;
w=(int)(d+floor(13*m/5)+y+floor(y/4)+floor(c/4)-2*c)%7;
while(w<0) w+=7;
return w;
}
```

14.卡特兰 (Catalan) 数列 原理

令 $h(1)=1$, catalan 数满足递归式:

$h(n)=h(1)*h(n-1)+h(2)*h(n-2)+\dots+h(n-1)h(1)$ (其中 $n\geq 2$)

该递推关系的解为: $h(n)=c(2n-2,n-1)/n$ ($n=1,2,3,\dots$)

1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845, 35357670, 129644790, 477638700, 1767263190, 6564120420, 24466267020, 91482563640, 343059613650, 1289904147324, 4861946401452, ...

1.括号化问题。

矩阵链乘: $P=a_1\times a_2\times a_3\times\dots\times a_n$, 依据乘法结合律, 不改变其顺序, 只用括号表示成对的乘积, 试问有几种括号化的方案? ($h(n)$ 种)

2.出栈次序问题。

一个栈(无穷大)的进栈序列为 1,2,3,...,n,有多少个不同的出栈序列?

类似: 有 $2n$ 个人排成一行进入剧场。入场费 5 元。其中只有 n 个人有一张 5 元钞票, 另外 n 人只有 10 元钞票, 剧院无其它钞票, 问有多少中方法使得只要有 10 元的人买票, 售票处就有 5 元的钞票找零? (将持 5 元者到达视作将 5 元入栈, 持 10 元者到达视作使栈中某 5 元出栈)

3.将多边形划分为三角形问题。

将一个凸多边形区域分成三角形区域的方法数?

类似: 一位大城市的律师在她住所以北 n 个街区 and 以东 n 个街区处工作。每天她走 $2n$ 个街区去上班。如果他从不穿越 (但可以碰到) 从家到办公室的对角线, 那么有多少条可能的道路?

类似: 在圆上选择 $2n$ 个点, 将这些点成对连接起来使得所得到的 n 条线段不相交的方法数?

15.杨辉三角

语法: void gen()

预置:

Const int Max;

注意: (Max 一般不能超过 22,否则要用高精度计算)

int inta[Max][Max];

结果:inta 为杨辉三角序列.inta[1][1]=1;inta[2][1]=1;inta[2][2]=1...

void gen()

```
{
    int i ,j;
    for( i =1 ;i <= Max ;i++)
    {
        inta[i][1] = 1 ;
        inta[i][i] = 1 ;
    }
    inta[2][2] = 1 ;
    for( i = 3 ; i<=Max ;i++)
    {
        for(j =2 ;j<i ;j++ )
        {
            inta[i][j] = inta[i-1][j-1] + inta[i-1][j] ;
        }
    }
}
```

前十行:

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
```

16.全排列

语法: void qp(int Array[],int begin,int end);

注意:当参与全排列的数字稍大的时候将会有很大的计算量

```
#include<iostream>
using namespace std;
const int MaxNum=20;
static int a[MaxNum];
void qp(int Array[],int begin,int end);
int main()
{
    int i;
    for(i=0;i<MaxNum;i++)
        a[i]=i+1;
    //初始化数组为:1,2,3..
    qp(a,0,10);
    return 0;
}
void qp(int Array[],int begin,int end)
{
    int i;
    if(begin==end)
    {
        for(i=0;i<end;i++)
            cout<<Array[i]<<"\t";
        cout<<endl;
    }
    else for(i=begin;i<end;i++)
    {
        swap(a[begin],a[i]);
        qp(a,begin+1,end);
        swap(a[begin],a[i]);
    }
}
```

//stl 里面的全排列生成函数 next_permutation

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

```
int main()
{
    int i;
    20 / 80
```

```
int A[] = {0,1,2,3};
while(next_permutation(A, A+4)==true)    //prev_permutation(A, A+4);

{
    for(i=0;i<4;i++)
        cout<<A[i]<<"\t";
    cout<<endl;
}
}
```

17.匈牙利算法----最大匹配问题.

```

#include<stdio.h>
#include<string.h>
bool g[201][201];           //为边之间的关系,如 g[0][1]==true 表示为左边 0 点可与右边 1 点相连.
int n,m,ans;                //n 左边的点,m 右边的点,ans 为最大匹配的边数.
bool b[201];                //b[i]表示该点是否有左边的点连上
int link[201];
bool init()
{
    int _x,_y;
    memset(g,0,sizeof(g));           //对二维数组也可以这样初始化!
    memset(link,0,sizeof(link));
    ans=0;
    if(scanf("%d%d",&n,&m)==EOF)return false;
    for(int i=1;i<=n;i++)           //n 左边的点,m 右边的点
    {
        scanf("%d",&_x);
        for(int j=0;j<_x;j++)
        {
            scanf("%d",&_y);
            g[ i ][_y]=true;
        }
    }
    return true;
}
bool find(int a)
{
    for(int i=1;i<=m;i++)
    {
        if(g[a][ i ]==true&&!b[ i ])
        {
            b[ i ]=true;
            if(link[ i ]==0||find(link[ i ]))
            {
                link[ i ]=a;
                return true;
            }
        }
    }
    return false;
}
int main()
{
    while(init())

```

```

{
    for(int i=1;i<=n;i++)
    {
        memset(b,0,sizeof(b)); //
        if(find(i))
            ans++;
    }
    printf("%d\n",ans);
}
}

```

18.最佳匹配 KM 算法

```

#include <cstdio>
#include <queue>
#include <algorithm>
using namespace std;
const int N = 301;
const int INF = 10000;

class Graph
{
private:
    bool xckd[N], yckd[N];
    int n, edge[N][N], xmate[N], ymate[N];
    int lx[N], ly[N], slack[N], prev[N];
    queue<int> Q;
    bool bfs();
    void agument(int);
public:
    bool make();
    int KMMatch();
};

bool Graph::make()
{
    int i, j;
    while(scanf("%d", &n) != EOF)
    {
        for(i = 0; i < n; i++)
        {
            for(j = 0; j < n; j++)
            {
                scanf("%d", &edge[i][j]);
            }
        }
    }
}

```

```

        return true;
    }
    return false;
}

bool Graph::bfs() {
    while(!Q.empty()) {
        int p = Q.front(), u = p>>1; Q.pop();
        if(p&1) {
            if(ymate[u] == -1) { agument(u); return true; }
            else { xckd[ymate[u]] = true; Q.push(ymate[u]<<1); }
        } else {
            for(int i = 0; i < n; i++)
                if(yckd[i]) continue;
            else if(lx[u]+ly[i] != edge[u][i]) {
                int ex = lx[u]+ly[i]-edge[u][i];
                if(slack[i] > ex) { slack[i] = ex; prev[i] = u; }
            } else {
                yckd[i] = true; prev[i] = u;
                Q.push((i<<1)|1);
            }
        }
    }
    return false;
}

void Graph::agument(int u) {
    while(u != -1) {
        int pv = xmate[prev[u]];
        ymate[u] = prev[u]; xmate[prev[u]] = u;
        u = pv;
    }
}

int Graph::KMMatch()
{
    int i, j;
    memset(ly, 0, sizeof(ly));
    for(i = 0; i < n; i++) {
        lx[i] = -INF;
        for(j = 0; j < n; j++)
            lx[i] = lx[i] > edge[i][j] ? lx[i] : edge[i][j];
    }
    memset(xmate, -1, sizeof(xmate)); memset(ymate, -1, sizeof(ymate));
    bool agu = true;
    for(int mn = 0; mn < n; mn++) {
        if(agu) {
            memset(xckd, false, sizeof(xckd));

```

```
    memset(yckd, false, sizeof(yckd));
    for(i = 0; i < n; i++) slack[i] = INF;
    while(!Q.empty()) Q.pop();
    xckd[mn] = true; Q.push(mn<<1);
}
if(bfs()) { agu = true; continue; }
int ex = INF; mn--; agu = false;
for(i = 0; i < n; i++)
    if(!yckd[i]) ex = ex < slack[i] ? ex : slack[i];
for(i = 0; i < n; i++) {
    if(xckd[i]) lx[i] -= ex;
    if(yckd[i]) ly[i] += ex;
    slack[i] -= ex;
}
for(int i = 0; i < n; i++)
    if(!yckd[i] && slack[i] == 0) { yckd[i] = true; Q.push((i<<1)|1); }
}
int cost = 0;
for(i = 0; i < n; i++) cost += edge[i][xmate[i]];
return cost;
}
int main()
{
    Graph g;
    while(g.make())
    {
        printf("%d\n", g.KMMatch());
    }
    return 0;
}
```

二、字符串处理

1.字符串替换

语法: `replace(char str[],char key[],char swap[]);`

参数:

`str[]`: 在此源字符串进行替换操作

`key[]`: 被替换的字符串, 不能为空串

`swap[]`: 替换的字符串, 可以为空串, 为空串表示在源字符中删除 `key[]`

返回值: `null`

注意:

默认 `str[]` 长度小于 1000, 如否, 重新设定 `tmp` 大小

需要 `string.h`

源程序:

```
void replace(char str[],char key[],char swap[])
{
    int l1,l2,l3,i,j,flag;
    char tmp[1000];
    l1=strlen(str);
    l2=strlen(key);
    l3=strlen(swap);
    for (i=0;i<=l1-l2;i++)
    {
        flag=1;
        for (j=0;j<l2;j++)
            if (str[i+j]!=key[j]) {flag=0;break;}
        if (flag)
        {
            strcpy(tmp,str);
            strcpy(&tmp[i],swap);
            strcpy(&tmp[i+l3],&str[i+l2]);
            strcpy(str,tmp);
            i+=l3-1;
            l1=strlen(str);
        }
    }
}
```

2.字符串查找

语法: `result=strfind(char str[],char key[]);`

参数:

`str[]`: 在此源字符串进行查找操作

`key[]`: 被查找的字符串, 不能为空串

返回值: 如果查找成功, 返回 `key` 在 `str` 中第一次出现的位置, 否则返回-1

注意:

需要 `string.h`

源程序:

```
int strfind(char str[],char key[])
{
    int l1,l2,i,j,flag;
    l1=strlen(str);
    l2=strlen(key);
    for (i=0;i<=l1-l2;i++)
    {
        flag=1;
        for (j=0;j<l2;j++)
            if (str[i+j]!=key[j]) {flag=0;break;}
        if (flag) return i;
    }
    return -1;
}
```

3.字符串截取

语法: `mid(char str[],int start,int len,char strback[])`

参数:

`str[]`: 操作的目标字符串

`start`: 从第 `start` 个字符串开始, 截取长度为 `len` 的字符

`len`: 从第 `start` 个字符串开始, 截取长度为 `len` 的字符

`strback[]`: 截取的到的字符

返回值: 0: 超出字符串长度, 截取失败; 1: 截取成功

注意:

需要 `string.h`

源程序:

```
int mid(char str[],int start,int len,char strback[])
{
    int l,i,k=0;
    l=strlen(str);
    if (start+len>l) return 0;
    for (i=start;i<start+len;i++)
        strback[k++]=str[i];
    strback[k]='\0';
}
```

```
    return 1;  
}
```

4.LCS-最大公共子串长度

语法: `result=lcs_len(char *a, char *b);`

参数:

`a,b[]`: 根据 `a,b` 生成最大公共子串

返回值: 最大公共子串的长度

注意:

需要 `string.h`

`M`、`N` 是 `a,b` 数组的最大可能长度

如果不需要生成公共子串, `c[M][N]` 不可设置为全局变量

源程序:

```
#define M 20  
#define N 20  
int c[M][N];  
int lcs_len(char *a, char *b)  
{  
    int m=strlen(a),n=strlen(b),i,j;  
    for(i=0;i<=m;i++) c[i][0]=0;  
    for(j=0;j<=n;j++) c[0][j]=0;  
    for(i=1;i<=m;i++)  
        for(j=1;j<=n;j++)  
        {  
            if(a[i-1]==b[j-1])  
                c[i][j]=c[i-1][j-1]+1;  
            else if(c[i-1][j]>c[i][j-1])  
                c[i][j]=c[i-1][j];  
            else  
                c[i][j]=c[i][j-1];  
        }  
    return c[m][n];  
}
```

5.LCS-最大公共子串长度

语法: `result=build_lcs(char s[], char *a, int blen, int clen);`

参数:

*a: 生成公共子串的字符串 a,b 中的 a

s[]: 接受返回结果的字符串数组

blen: 生成公共子串的字符串 a,b 中的 b 的长度

clen: 最大公共子串的长度, 通过 `lcs_len` 函数求得

返回值: 最大公共子串的长度

注意:

需要 `string.h`

需要 `lcs_len` 函数求 clen 并且生成 `c[M][N]`

可通过 `result=build_lcs` 返回指针或者通过 `build_lcs(s,a,blen,clen)`, 用 s 接受结果

源程序:

```
char *build_lcs(char s[], char *a, int blen, int clen)
{
    int k=clen,alen=strlen(a),i,j;
    s[k]='\0';
    i=alen,j=blen;
    while(k>0)
    {
        if(c[i][j]==c[i-1][j])
            i--;
        else if(c[i][j]==c[i][j-1])
            j--;
        else
        {
            s[--k]=a[i-1];
            i--;j--;
        }
    }
    return s;
}
```

6.数字转换为字符

语法: `cstr(int k,char o[]);`

参数:

k: 转换的数字

o[]: 存储转换结果的字符串

返回值: null

注意:

需要 `math.h`

源程序:

```
void cstr(int k,char o[])
{
    int len,i,t;
    len=log10(k)+1;
    for (i=len;i>0;i--)
    {
        t=k%10;
        k-=t;k/=10;
        o[i-1]='0'+t;
    }
    o[len]='\0';
}
```

三、计算几何

1.叉乘法求任意多边形面积

语法: `result=polygonarea(Point *polygon,int N);`

参数:

`*polygon`: 多边形顶点数组

`N`: 多边形顶点数目

返回值: 多边形面积

注意:

支持任意多边形, 凹、凸皆可

多边形顶点输入时按顺时针顺序排列

源程序:

```
typedef struct {
    double x,y;
} Point;
double polygonarea(Point *polygon,int N)
{
    int i,j;
    double area = 0;
    for (i=0;i<N;i++) {
        j = (i + 1) % N;
        area += polygon[i].x * polygon[j].y;
        area -= polygon[i].y * polygon[j].x;
    }
    area /= 2;
    return(area < 0 ? -area : area);
}
```

2.求三角形面积

语法: `result=area3(float x1,float y1,float x2,float y2,float x3,float y3);`

参数:

`x1~3`: 三角形 3 个顶点 x 坐标

`y1~3`: 三角形 3 个顶点 y 坐标

返回值: 三角形面积

注意:

需要 `math.h`

源程序:

```
float area3(float x1,float y1,float x2,float y2,float x3,float y3)
{
    float a,b,c,p,s;
    a=sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
    b=sqrt((x1-x3)*(x1-x3)+(y1-y3)*(y1-y3));
    c=sqrt((x3-x2)*(x3-x2)+(y3-y2)*(y3-y2));
    p=(a+b+c)/2;
    s=sqrt(p*(p-a)*(p-b)*(p-c));
    return s;
}
```

3.两矢量间角度

语法: `result=angle(double x1, double y1, double x2, double y2);`

参数:

`x/y1~2`: 两矢量的坐标

返回值: 两的角度矢量

注意:

返回角度为弧度制, 并且以逆时针方向为正方向

需要 `math.h`

源程序:

```
#define PI 3.1415926

double angle(double x1, double y1, double x2, double y2)
{
    double dtheta,theta1,theta2;
    theta1 = atan2(y1,x1);
    theta2 = atan2(y2,x2);
    dtheta = theta2 - theta1;
    while (dtheta > PI)
        dtheta -= PI*2;
    while (dtheta < -PI)
        dtheta += PI*2;
    return(dtheta);
}
```


4. 两点距离（2D、3D）

语法: `result=distance_2d(float x1,float x2,float y1,float y2);`

参数:

`x/y/z1~2`: 各点的 `x`、`y`、`z` 坐标

返回值: 两点之间的距离

注意:

需要 `math.h`

源程序:

```
float distance_2d(float x1,float x2,float y1,float y2)
{
    return(sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2)));
}
```

```
float distance_3d(float x1,float x2,float y1,float y2,float z1,float z2)
{
    return(sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2)+(z1-z2)*(z1-z2)));
}
```

5. 射向法判断点是否在多边形内部

语法: `result=insidepolygon(Point *polygon,int N,Point p);`

参数:

`*polygon`: 多边形顶点数组

`N`: 多边形顶点个数

`p`: 被判断点

返回值: 0: 点在多边形内部; 1: 点在多边形外部

注意:

若 `p` 点在多边形顶点或者边上, 返回值不确定, 需另行判断

需要 `math.h`

源程序:

```
#define MIN(x,y) (x < y ? x : y)
#define MAX(x,y) (x > y ? x : y)
typedef struct {
    double x,y;
} Point;
int insidepolygon(Point *polygon,int N,Point p)
{
    int counter = 0;
```

```

int i;
double xinters;
Point p1,p2;
p1 = polygon[0];
for (i=1;i<=N;i++) {
    p2 = polygon[i % N];
    if (p.y > MIN(p1.y,p2.y)) {
        if (p.y <= MAX(p1.y,p2.y)) {
            if (p.x <= MAX(p1.x,p2.x)) {
                if (p1.y != p2.y) {
                    xinters = (p.y-p1.y)*(p2.x-p1.x)/(p2.y-p1.y)+p1.x;
                    if (p1.x == p2.x || p.x <= xinters)
                        counter++;
                }
            }
        }
    }
    p1 = p2;
}
if (counter % 2 == 0)
    return(OUTSIDE);
else
    return(INSIDE);
}

```

6.判断点是否在线段上

语法: result=Pointonline(Point p1,Point p2,Point p);

参数:

p1、p2: 线段的两个端点

p: 被判断点

返回值: 0: 点在不在线段上; 1: 点在线段上

注意:

若 p 线段端点上返回 1

需要 math.h

源程序:

```

#define MIN(x,y) (x < y ? x : y)
#define MAX(x,y) (x > y ? x : y)
typedef struct {
    double x,y;
} Point;
int FC(double x1,double x2)
{

```

```

    if (x1-x2<0.000002&& x1-x2>-0.000002) return 1; else return 0;
}

```

```

int Pointonline(Point p1,Point p2,Point p)
{
    double x1,y1,x2,y2;
    x1=p.x-p1.x;
    x2=p2.x-p1.x;
    y1=p.y-p1.y;
    y2=p2.y-p1.y;
    if (FC(x1*y2-x2*y1,0)==0) return 0;
    if ((MIN(p1.x,p2.x)<=p.x&& p.x<=MAX(p1.x,p2.x))&&
        (MIN(p1.y,p2.y)<=p.y&& p.y<=MAX(p1.y,p2.y)))
        return 1; else return 0;
}

```

7.判断两线段是否相交

语法: result=lineintersect(Point p1,Point p2,Point p3,Point p4);

参数:

p1~4: 两条线段的四个端点

返回值: 0: 两线段不相交; 1: 两线段相交; 2 两线段首尾相接

注意:

p1!=p2;p3!=p4;

源程序:

```
#define MIN(x,y) (x < y ? x : y)
```

```
#define MAX(x,y) (x > y ? x : y)
```

```
typedef struct {
```

```
    double x,y;
```

```
} Point;
```

```
int lineintersect(Point p1,Point p2,Point p3,Point p4)
```

```
{
```

```
    Point tp1,tp2,tp3;
```

```
    if
```

```

((p1.x==p3.x&& p1.y==p3.y)|| (p1.x==p4.x&& p1.y==p4.y)|| (p2.x==p3.x&& p2.y==p3.y)|| (p2.x==p4.x&& p2.y==p4.y))
        return 2;

```

```
//快速排斥试验
```

```
    if
```

```

((MIN(p1.x,p2.x)<=p3.x&& p3.x<=MAX(p1.x,p2.x)&& MIN(p1.y,p2.y)<=p3.y&& p3.y<=MAX(p1.y,p2.y))||

```

```

(MIN(p1.x,p2.x)<=p4.x&& p4.x<=MAX(p1.x,p2.x)&& MIN(p1.y,p2.y)<=p4.y&& p4.y<=MAX(p1.y,p2.y)))

```

```

        ;else return 0;
//跨立试验
    tp1.x=p1.x-p3.x;
    tp1.y=p1.y-p3.y;
    tp2.x=p4.x-p3.x;
    tp2.y=p4.y-p3.y;
    tp3.x=p2.x-p3.x;
    tp3.y=p2.y-p3.y;
    if ((tp1.x*tp2.y-tp1.y*tp2.x)*(tp2.x*tp3.y-tp2.y*tp3.x)>=0) return 1;
else return 0;
}

```

8.判断线段与直线是否相交

语法: result=lineintersect(Point p1,Point p2,Point p3,Point p4);

参数:

p1、p2: 线段的两个端点

p3、p4: 直线上的两个点

返回值: 0: 线段直线不相交; 1: 线段和直线相交

注意:

如线段在直线上, 返回 1

源程序:

```

typedef struct {
    double x,y;
} Point;
int lineintersect(Point p1,Point p2,Point p3,Point p4)
{
    Point tp1,tp2,tp3;
    tp1.x=p1.x-p3.x;
    tp1.y=p1.y-p3.y;
    tp2.x=p4.x-p3.x;
    tp2.y=p4.y-p3.y;
    tp3.x=p2.x-p3.x;
    tp3.y=p2.y-p3.y;
    if ((tp1.x*tp2.y-tp1.y*tp2.x)*(tp2.x*tp3.y-tp2.y*tp3.x)>=0) return 1;
else return 0;
}

```

9.点到线段最短距离

语法: result=mindistance(Point p1,Point p2,Point q);

参数:

p1、p2: 线段的两个端点

q: 判断点

返回值: 点 q 到线段 p1p2 的距离

注意:

需要 math.h

源程序:

```
#define MIN(x,y) (x < y ? x : y)
#define MAX(x,y) (x > y ? x : y)
typedef struct {
    double x,y;
} Point;
double mindistance(Point p1,Point p2,Point q)
{
    int flag=1;
    double k;
    Point s;
    if (p1.x==p2.x) {s.x=p1.x;s.y=q.y;flag=0;}
    if (p1.y==p2.y) {s.x=q.x;s.y=p1.y;flag=0;}
    if (flag)
    {
        k=(p2.y-p1.y)/(p2.x-p1.x);
        s.x=(k*k*p1.x+k*(q.y-p1.y)+q.x)/(k*k+1);
        s.y=k*(s.x-p1.x)+p1.y;
    }
    if (MIN(p1.x,p2.x)<=s.x&& s.x<=MAX(p1.x,p2.x))
        return sqrt((q.x-s.x)*(q.x-s.x)+(q.y-s.y)*(q.y-s.y));
    else
        return
        MIN(sqrt((q.x-p1.x)*(q.x-p1.x)+(q.y-p1.y)*(q.y-p1.y)),sqrt((q.x-p2.x)*(q.x-p2.x)+(q.y-p2.y)*(q.y-p2.y)));
}
```

10.求两直线的交点

语法: result=mindistance(Point p1,Point p2,Point q);

参数:

p1~p4: 直线上不相同的两点

*p: 通过指针返回结果

返回值: 1: 两直线相交; 2: 两直线平行

注意:

如需要判断两线段交点, 检验 k 和对应 k1 (注释中) 的值是否在 0~1 之间, 用在 0~1 之间的那个求交点

源程序:

```
typedef struct {
```

```

double x,y;
} Point;
int linecorss(Point p1,Point p2,Point p3,Point p4,Point *p)
{
    double k;
    if ((p4.y-p3.y)*(p2.x-p1.x)-(p4.x-p3.x)*(p2.y-p1.y)==0) return 0;
    if ((p4.x-p3.x)*(p1.y-p3.y)-(p4.y-p3.y)*(p1.x-p3.x)==0&&
        (p2.x-p1.x)*(p1.y-p3.y)-(p2.y-p1.y)*(p1.x-p3.x)==0) return 0;

    k=((p4.x-p3.x)*(p1.y-p3.y)-(p4.y-p3.y)*(p1.x-p3.x))/((p4.y-p3.y)*(p2.x-p1.x)-(p4.x-p3.x)*(p2.y-p1.y));
    //k1=((p2.x-p1.x)*(p1.y-p3.y)-(p2.y-p1.y)*(p1.x-p3.x))/((p4.y-p3.y)*(p2.x-p1.x)-(p4.x-p3.x)*(p2.y-p1.y));
    (*p).x=p1.x+k*(p2.x-p1.x);
    (*p).y=p1.y+k*(p2.y-p1.y);
    return 1;
}

```

11.判断一个封闭图形是凹集还是凸集

语法: result=convex(Point *p,int n);

参数:

*p: 封闭曲线顶点数组

n: 封闭曲线顶点个数

返回值: 1: 凸集; -1: 凹集; 0: 曲线不符合要求无法计算

注意:

默认曲线为简单曲线: 无交叉、无圈

源程序:

```

typedef struct {
    double x,y;
} Point;
int convex(Point *p,int n)
{
    int i,j,k;
    int flag = 0;
    double z;
    if (n < 3)
        return(0);
    for (i=0;i<n;i++) {
        j = (i + 1) % n;
        k = (i + 2) % n;
        z = (p[j].x - p[i].x) * (p[k].y - p[j].y);
        z -= (p[j].y - p[i].y) * (p[k].x - p[j].x);
        if (z < 0)
            flag |= 1;
    }
}

```

```

        else if (z > 0)
            flag |= 2;
        if (flag == 3)
            return -1; //CONCAVE
    }
    if (flag != 0)
        return 1; //CONVEX
    else
        return 0;
}

```

12.Graham 扫描法寻找凸包

语法: `Graham_scan(Point PointSet[],Point ch[],int n,int &len);`

参数:

`PointSet[]`: 输入的点集

`ch[]`: 输出的凸包上的点集, 按照逆时针方向排列

`n`: `PointSet` 中的点的数目

`len`: 输出的凸包上的点的个数

返回值: `null`

源程序:

```

struct Point{
    float x,y;
};
float multiply(Point p1,Point p2,Point p0)
{
    return((p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y));
}
float distance(Point p1,Point p2)
{
    return(sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y)));
}
void Graham_scan(Point PointSet[],Point ch[],int n,int &len)
{
    int i,j,k=0,top=2;
    Point tmp;

    for(i=1;i<n;i++)
        if
((PointSet[i].y<PointSet[k].y)||((PointSet[i].y==PointSet[k].y)&&(PointSet[i].x<PointSet[k].x)))
        k=i;
    tmp=PointSet[0];
    PointSet[0]=PointSet[k];

```

```

    PointSet[k]=tmp;
    for (i=1;i<n-1;i++)
    {
        k=i;
        for (j=i+1;j<n;j++)
            if ( (multiply(PointSet[j],PointSet[k],PointSet[0])>0) ||
                ((multiply(PointSet[j],PointSet[k],PointSet[0])==0)

&&(distance(PointSet[0],PointSet[j])<distance(PointSet[0],PointSet[k])))
        )
            k=j;
        tmp=PointSet[i];
        PointSet[i]=PointSet[k];
        PointSet[k]=tmp;
    }
    ch[0]=PointSet[0];
    ch[1]=PointSet[1];
    ch[2]=PointSet[2];
    for (i=3;i<n;i++)
    {
        while (multiply(PointSet[i],ch[top],ch[top-1])>=0) top--;
        ch[++top]=PointSet[i];
    }
    len=top+1;
}

```

13.求两条线段的交点

语法: Result=IntersectPoint (Point p1,Point p2,Point p3,Point p4,Point &p);

参数:

P1~P4: 两条线断 4 个端点

P: 线段交点

返回值: 如果两条线段平行无交点, 返回 0, 否则返回 1

源程序:

```

struct Point{
    float x,y;
};

int IntersectPoint (Point p1,Point p2,Point p3,Point p4,Point &p)
{
    float a,b,c,d,e,f;
    a=p2.y-p1.y;
    b=p1.x-p2.x;

```



```

c=p1.y*(p2.x-p1.x)+p1.x*(p2.y-p1.y);
d=p4.y-p3.y;
e=p3.x-p4.x;
f=p3.y*(p4.x-p3.x)+p1.x*(p4.y-p3.y);
if (a*e==b*d)
    return 0;
else
{
    p.x=(e*c-b*f)/(b*d-a*e);
    p.y=(d*c-a*f)/(a*e-b*d);
    return 1;
}
}

```

四、数论

1.x 的二进制长度

语法: result=BitLength(int x);

参数:

x: 测长的 x

返回值: x 的二进制长度

源程序:

```

int BitLength(int x)
{
    int d = 0;
    while (x > 0) {
        x >>= 1;
        d++;
    }
    return d;
}

```

2.返回 x 的二进制表示中从低到高的第 i 位

语法: result=BitAt(int x, int i);

参数:

x: 十进制 x

i: 要求二进制的第 i 位

返回值：返回 x 的二进制表示中从低到高的第 i 位

注意：

最低位为第一位

源程序：

```
int BitAt(int x, int i)
{
    return ( x & (1 << (i-1)) );
}
```

3.模取幂运算

语法：result=Modular_Exponent(int a,int b,int n);

参数：

a、b、n： $a^b \bmod n$ 的对应参数

返回值： $a^b \bmod n$ 的值

注意：

需要 BitLength 和 BitAt

源程序：

```
int Modular_Exponent(int a,int b,int n)
{
    int i, y=1;
    for (i = BitLength(b); i > 0; i--)
    {
        y = (y*y)%n;
        if (BitAt(b,i) > 0)
            y = (y*a)%n;
    }
    return y;
}
```

4.求解模线性方程

语法：result=modular_equation(int a,int b,int n);

参数：

a、b、n： $ax \equiv b \pmod{n}$ 的对应参数

返回值：方程的解

源程序：

```
int ext_euclid(int a,int b,int &x,int &y) //求 gcd(a,b)=ax+by
{
    int t,d;
    if (b==0) {x=1;y=0;return a;}
}
```

```

    d=ext_euclid(b,a %b,x,y);
    t=x;
    x=y;
    y=t-a/b*y;
    return d;
}

void modular_equation(int a,int b,int n)
{
    int e,i,d;
    int x,y;
    d=ext_euclid(a,n,x,y);
    if (b%d>0)
        printf("No answer!\n");
    else
    {
        e=(x*(b/d))%n;
        for (i=0;i<d;i++)
            printf("The %dth answer is : %ld\n",i+1,(e+i*(n/d))%n);
    }
}

```

5.求解模线性方程组(中国余数定理)

语法: result=Modular_Exponent(int a,int b,int n);

参数:

B[]、W[]: $a=B[] \pmod{W[]}$ 的对应参数

返回值: a 的值

注意:

其中 W[],B[] 已知, $W[i]>0$ 且 $W[i]$ 与 $W[j]$ 互质, 求 a

源程序:

```

int ext_euclid(int a,int b,int &x,int &y) //求 gcd(a,b)=ax+by
{
    int t,d;
    if (b==0) {x=1;y=0;return a;}
    d=ext_euclid(b,a %b,x,y);
    t=x;
    x=y;
    y=t-a/b*y;
    return d;
}

```

```

int China(int B[],int W[],int k)
{
    int i;
    int d,x,y,a=0,m,n=1;
    for (i=0;i<k;i++)
        n*=W[i];
    for (i=0;i<k;i++)
    {
        m=n/W[i];
        d=ext_euclid(W[i],m,x,y);
        a=(a+y*m*B[i])%n;
    }
    if (a>0) return a;
    else return(a+n);
}

```

6.筛法素数产生器

语法: result=prime(int a[],int n);

参数:

a[]: 用于返回素数的数组

n: 产生 n 以内的素数, 按升序放入 a[]中

返回值: n 以内素数的个数

注意:

其中 W[],B[]已知, W[i]>0 且 W[i]与 W[j]互质, 求 a

源程序:

```

int prime(int a[],int n)
{
    int i,j,k,x,num,*b;
    n++;
    n/=2;
    b=new int[(n+1)*2];
    a[0]=2;a[1]=3;num=2;
    for(i=1;i<=2*n;i++)
        b[i]=0;
    for(i=3;i<=n;i+=3)
        for(j=0;j<2;j++)
        {
            x=2*(i+j)-1;
            while(b[x]==0)
            {
                a[num++]=x;
                for(k=x;k<=2*n;k+=x)

```

```

        b[k]=1;
    }
}
return num;
}

```

7.判断一个数是否素数

语法: result=comp(int n);

参数:

n: 判断 n 是否素数

返回值: 素数返回 1, 否则返回 0

源程序:

//传统方法

```

int comp(int n){
    if(n<2)return 0;
    if(n%2==0)return 0;
    for(int i=3;i<=sqrt(n);i+=2) if(n%i==0) return 0;
    return 1;
}

```

//大数素数判定, $(1/4)^K$ 出错概率

```

int powermod(int a,int b,int n)
{
    //get (a^b)%n
    i64 d=1, t=a;
    while(b>0){
        if(t==1)return d;
        if(b%2==1)d=(t*d)%n;
        b/=2; t=(t*t)%n;
    }
    return d;
}

int isprime(int n,int k){
    int a;
    while(k--){
        a=rand();
        a%=n-3; a+=2;
        if(powermod(k+2,n-1,n)!=1)return 0;
    }
    return 1;
}

```

8.求距阵最大和

语法: `result=maxsum2(int n);`

参数:

a: 距阵

n,m: 距阵行列数

返回值: 一维, 二维距阵最大和

源程序:

```
int a[101][101];
int maxsum(int a[],int n)//一维最大串
{
    int sum=-10000000,b=0;
    int i;
    for(i=0;i<n;i++)
    {
        if (b>0)
            b+=a[i];
        else
            b=a[i];
        if(b>sum)
            sum=b;
    }
    return sum;
}
int maxsum2(int m,int n)//二维最大串
{
    int sum = -10000000;
    int i,j,k,max;
    int* b = new int[n+1];
    for (i=0;i<m;i++)
    {
        for(k=0;k<n;k++)
            b[k]=a[i][k];
        max = maxsum(b,n);//第 i 列的一维最大串
        if(max>sum)
            sum=max;
        for(j=i+1;j<m;j++)
        {
            for (k=0;k<=n;k++)b[k]+=a[j][k];
            max = maxsum(b,n);//类似 maxsum, 通过每列相加求二维最大串
            if(max>sum)sum=max;
        }
    }
    delete []b;
    return sum;
}
```

}

8.求一个数每一位相加之和

语法: `result=digadd(int n)`

参数:

n: 待求数字

返回值: 各数字之和

源程序:

```
int digadd(int n)
{
    int i=0,k=0;
    while(i=n%10,n/=10) k+=i;
    return k+i;
}
```

10.质因数分解

语法: `result=int reduce(int prime[],int pn,int n,int rest[])`

参数:

Prime[]: 素数表, 至少需要达到 `sqrt(n)`

pn: 素数表的元素个数

N: 待分解的数

Rest: 分解结果, 按照升序排列

返回值: 分解因子个数

源程序:

```
int reduce(int prime[],int pn,int n,int rest[])
{
    int i,k=0;
    for(i=0;i<pn;i++)
    {
        if (n==1) break;
        if (prime[i]*prime[i]>n) {rest[k++]=n;break;}
        while(n%prime[i]==0)
        {
            n/=prime[i];
            rest[k++]=prime[i];
        }
    }
    return k;
}
```

11.高斯消元法解线性方程组

语法: gauss(int n,double ** a)

参数:

N: 变量个数

Rest: 变量系数行列式

源程序:

```
void gauss(int n,double **a)
{
    int i, j, k;
    double client, temp = 0.0;

    for(k = 0; k < n - 1; k++)
        for(i = k + 1; i < n; i++)
        {
            client = a[i][k]/a[k][k];
            for(j = k + 1; j < n; j++)
                a[i][j] = a[i][j] - client * a[k][j];
            a[i][n] = a[i][n] - client * a[k][n];
        }
    a[n - 1][n] = a[n - 1][n]/a[n - 1][n - 1];
    for(i = n - 2; i >= 0; i--)
    {
        for (j = i + 1; j < n; j++)
            temp += a[i][j] * a[j][n];
        a[i][n] = (a[i][n] - temp) / a[i][i];
    }
}

//打印
//for(i = 0; i < n; i++)
// printf("X%d = %lf\n", i + 1, a[i][n]);
```

五、图论

1.Prim 算法求最小生成树

语法: prim(Graph G,int vcount,int father[]);

参数:

G: 图, 用邻接矩阵表示

vcount: 表示图的顶点个数

father[]: 用来记录每个节点的父节点

返回值: null

注意:

常数 max_vertexes 为图最大节点数

常数 infinity 为无穷大

源程序:

```
#define infinity 1000000
#define max_vertexes 5

typedef int Graph[max_vertexes][max_vertexes];

void prim(Graph G,int vcount,int father[])
{
    int i,j,k;
    int lowcost[max_vertexes],closeset[max_vertexes],used[max_vertexes];
    for (i=0;i<vcount;i++)
    {
        lowcost[i]=G[0][i];
        closeset[i]=0;
        used[i]=0;
        father[i]=-1;
    }
    used[0]=1;
    for (i=1;i<vcount;i++)
    {
        j=0;
        while (used[j]) j++;
        for (k=0;k<vcount;k++)
            if ((!used[k])&&(lowcost[k]<lowcost[j])) j=k;
        father[j]=closeset[j];
        used[j]=1;
        for (k=0;k<vcount;k++)
            if (!used[k]&&(G[j][k]<lowcost[k]))
                { lowcost[k]=G[j][k];
                  closeset[k]=j; }
    }
}
```

2.Dijkstra 算法求单源最短路径

语法: `result=Dijkstra(Graph G,int n,int s,int t, int path[]);`

参数:

G: 图, 用邻接矩阵表示

n: 图的顶点个数

s: 开始节点

t: 目标节点

path[]: 用于返回由开始节点到目标节点的路径

返回值: 最短路径长度

注意:

输入的图的权必须非负

顶点标号从 0 开始

用如下方法打印路径:

```
i=t;
while (i!=s)
{
    printf("%d<--",i+1);
    i=path[i];
}
printf("%d\n",s+1);
```

源程序:

```
int Dijkstra(Graph G,int n,int s,int t, int path[])
{
    int i,j,w,minc,d[max_vertexes],mark[max_vertexes];
    for (i=0;i<n;i++) mark[i]=0;
    for (i=0;i<n;i++)
    { d[i]=G[s][i];
      path[i]=s; }
    mark[s]=1;path[s]=0;d[s]=0;
    for (i=1;i<n;i++)
    {
        minc=infinity;
        w=0;
        for (j=0;j<n;j++)
            if ((mark[j]==0)&&(minc>=d[j])) { minc=d[j];w=j; }
        mark[w]=1;
        for (j=0;j<n;j++)
            if ((mark[j]==0)&&(G[w][j]!=infinity)&&(d[j]>d[w]+G[w][j]))
                { d[j]=d[w]+G[w][j];
                  path[j]=w; }
    }
    return d[t];
}
```

3. Bellman-ford 算法求单源最短路径

语法: `result=Bellman_ford(Graph G,int n,int s,int t,int path[],int success);`

参数:

G: 图, 用邻接矩阵表示

n: 图的顶点个数

s: 开始节点

t: 目标节点

path[]: 用于返回由开始节点到目标节点的路径

success: 函数是否执行成功

返回值: 最短路径长度

注意:

输入的图的权可以为负, 如果存在一个从源点可达的权为负的回路则 `success=0`

顶点标号从 0 开始

用如下方法打印路径:

```
i=t;
while (i!=s)
{
    printf("%d<--",i+1);
    i=path[i];
}
printf("%d\n",s+1);
```

源程序:

```
int Bellman_ford(Graph G,int n,int s,int t,int path[],int success)
{
    int i,j,k,d[max_vertexes];
    for (i=0;i<n;i++) {d[i]=infinity;path[i]=0;}
    d[s]=0;
    for (k=1;k<n;k++)
        for (i=0;i<n;i++)
            for (j=0;j<n;j++)
                if (d[j]>d[i]+G[i][j]) {d[j]=d[i]+G[i][j];path[j]=i;}
    success=0;
    for (i=0;i<n;i++)
        for (j=0;j<n;j++)
            if (d[j]>d[i]+G[i][j]) return 0;
    success=1;
    return d[t];
}
```

4. Floyd-Warshall 算法求每对节点间最短路径

语法: `Floyd_Warshall(Graph G,int n,Graph D,Graph P);`

参数:

G: 图, 用邻接矩阵表示

n: 图的顶点个数

D: $D[i,j]$ 表示从 i 到 j 的最短距离

P: $P[i,j]$ 表示从 i 到 j 的最短路径上 j 的父节点

自定义:MaxN;

返回值: null

注意:此算法允许图中带有负权的弧,但不允许有路径长度为负值的回路.

源程序:

```
void Floyd(int G[][MaxN],int n,int D[][MaxN],int P[][MaxN])
{
    int i,j,k;
    for (i=0;i<n;i++)
        for (j=0;j<n;j++)
            { D[i][j]=G[i][j];
              P[i][j]=i; }
    for (i=0;i<n;i++) { D[i][i]=0;P[i][i]=0; }
    for (k=0;k<n;k++)
        for (i=0;i<n;i++)
            for (j=0;j<n;j++)
                if (D[i][j]>D[i][k]+D[k][j])
                    { D[i][j]=D[i][k]+D[k][j];
                      P[i][j]=P[k][j]; }
}

void Floyd(int n,int D[][MaxN])          //G 不用再输出的情况下.
{
    int i,j,k;
    for (i=0;i<n;i++) { D[i][i]=0; }
    for (k=0;k<n;k++)
        for (i=0;i<n;i++)
            for (j=0;j<n;j++)
                if (D[i][j]>D[i][k]+D[k][j])
                    { D[i][j]=D[i][k]+D[k][j];
                      }
}
```

5.解欧拉图

语法: int Euler(int graph[8][8], int v, int *path)

参数:

graph: 图, 用邻接矩阵表示

v: 图的顶点个数

path: $D[i,j]$ 表示从 i 到 j 的最短距离

注意:

此函数会删除图中的边

返回值：若找到欧拉回路则返回路径长度，否则返回-1

源程序：

```
int Euler(int graph[8][8], int v, int *path)
{
    int start,a,b,count=0, deg,i;
    int s[1000], sp=0;//栈，大小可根据需要改变

    start=0;
    while(true)
    {
        a=start;
        s[sp++]=a;
        for(b=-1,i=0;i<v;i++) if (graph[i][a]&& a!=i) {b=i;break;}
        for(;b!=start&&b!=-1;)
        {
            s[sp++]=b;
            graph[a][b]=graph[b][a]=0;
            a=b;
            for(b=-1,i=0;i<v;i++) if (graph[i][a]&& a!=i) {b=i;break;}
        }
        if (b==-1) return(-1);//若找不到 Euler 回路返回-1
        s[sp++]=b;
        graph[a][b]=graph[b][a]=0;

        while(sp>0)
        {
            b=s[--sp];
            for(i=0,deg=0;i<v;i++) if (graph[i][b]==1) {deg++; break;}
            if (deg>0) break;
            path[count++]=b;
        }
        if (sp==0) return(count);
        start=b;
    }
}
```

六、排序/查找

1.快速排序

语法: `quicksort(int l,int r,int b[]);`

参数:

`l`: 排序上界, 开始时 `l=0`

`r`: 排序下界, 开始时 `r=数组元素个数`

`b[]`: 被排序的元素

返回值: `null`

注意:

输出升序序列

源程序:

```
void quicksort(int l,int r,int b[])
{
    int i,j,x;
    if(l>=r) return;
    i=l;
    j=r;
    x=b[i];
    while(i!=j)
    {
        while(b[j]>x&& j>i) j--;
        if(i<j)
        {
            b[i]=b[j];
            i++;
        }
        while(b[i]<x&& j>i) i++;
        if(i<j)
        {
            b[j]=b[i];
            j--;
        }
    }
    b[i]=x;
    quicksort(l,j-1,b);
    quicksort(i+1,r,b);
}
```

2.希尔排序

语法: `shellsort(int a[],int n);`

参数:

`n`: 数组元素个数

`a[]`: 待排序数组

返回值: `null`

注意：

输出升序序列

源程序：

```
void shellsort(int a[],int n)
{
    int i,j,g;
    int temp,k;
    g=n/2;
    while(g!=0)
    {
        for(i=g+1;i<=n;i++)
        {
            temp=a[i];
            j=i-g;
            while(j>0)
            {
                k=j+g;
                if(a[j]<=a[k])
                    j=0;
                else
                {
                    temp=a[j];a[j]=a[k];a[k]=temp;
                }
                j=j-g;
            }
        }
        g=g/2;
    }
}
```

3.选择法排序

语法：sort(int t[],int n);

参数：

t[]：待排序数组

n：数组 t[]元素的个数

返回值：null

注意：

输出升序序列

小规模排序用

源程序：

```
void sort(int t[],int n)
{
```

```

int i,j,k,temp;
for (i=0;i<n;i++)
{
    k=i;
    for (j=i;j<n;j++) if (t[j]<t[k]) k=j;
    temp=t[i];t[i]=t[k];t[k]=temp;
}
}

```

4.二分查找

语法: result=search_bin(int *t,int k);

参数:

t[]: 待查找数组

k: 查找关键字

返回值: 如果 k 在 t[] 中存在, 输出 i: t[i]=k, 否则输出 -1

注意:

要求查找数组是有序升序序列

源程序:

```

int search_bin(int *t,int k)
{
    int low=1,high=10,mid;
    while (low<=high)
    {
        mid=(low+high)/2;
        if (k==t[mid]) return mid;
        else if (k<t[mid]) high=mid-1;
        else low=mid+1;
    }
    return -1;
}

```

七、数据结构

1.顺序队列

源程序:

```

#define maxsize 100
typedef struct
{

```



```
    int data[maxsize];
    int front;
    int rear;
} sqqueue;
int sqinit(sqqueue *p) //队列初始化
{
    p->front=0;
    p->rear=0;
    return 1;
}
int enqueue(sqqueue *q, int e) //入队
{
    if((q->rear+1)%maxsize==q->front)
        return 0;
    else
        q->data[q->rear]=e;
    q->rear=(q->rear+1)%maxsize;
    return 1;
}
int dequeue(sqqueue *q) //出队
{
    int e;
    if (q->front==q->rear)
        return 0;
    e=q->data[q->front];
    q->front=(q->front+1)%maxsize;
    return e;
}
int empty(sqqueue *q) //判空
{
    int v;
    if (q->front==q->rear)
        v=1;
    else
        v=0;
    return v;
}
int gethead(sqqueue *q) //取得头元素
{
    int e;
    if (q->front==q->rear)
        e=-1;
    else
        e=q->data[q->front];
    return e;
}
```

```

}
void display(sqqueue *q) //显示所有元素
{
    int s;
    s=q->front;
    printf("the sequeue is display:\n");
    if (q->front==q->rear)
        printf("the sequeue is empty!");
    else
    {
        while(s<q->rear)
        {
            printf("->%d", q->data[s]);
            s=(s+1)%maxsize;
        }
        printf("\n");
    }
}
}
main(sqqueue *head) //函数使用样例
{
    int n,i,m,x,y,select,xq;
    printf("create a empty sequeue\n");
    sqinit(head);
    printf("please input the sequeue length:\n");
    scanf("%d",&n);
    for (i=0;i<n;i++)
    {
        printf("please input a sequeue value:\n");
        scanf("%d",&m);
        enqueue(head,m);
    }
    printf("head->rear:%d\n",head->rear);
    printf("head->front:%d\n",head->front);
    display(head);
    printf("select 1 ***** enqueue() \n");
    printf("select 2 ***** dequeue() \n");
    printf("select 3 ***** empty () \n");
    printf("select 4 ***** gethead() \n");
    printf("select 5 ***** display() \n");
    printf("please select (1--5):");
    scanf("%d",&select);
    switch(select)
    {
        case 1:
            {

```

```
        printf("please input a value :\n ");
        scanf("%d",&x);
        enqueue(head,x);
        display(head);
        break;
    }
    case 2:
    {
        dequeue(head);
        display(head);
        break;
    }
    case 3:
    {
        if(empty(head))
            printf("the sequeue is empty");
        else
            printf("the sequeue is full");
    }
    case 4:
    {
        y=gethead(head);
        printf("output head value:%d\n",y);
        break;
    }
    case 5:
    {
        display(head);
        break;
    }
    }
}
```

2.顺序栈

源程序:

```
#define m 100
typedef struct
{
    int stack[m];
    int top;
} stackstru;
```

```
init(stackstru *s) /*装入栈*/
{
    s->top=0;
    return 1;
}
int push(stackstru *s,int x) /*入栈操作*/
{
    if (s->top==m)
        printf("the stack is overflow!\n");
    else
    {
        s->top=s->top+1;
        s->stack[s->top]=x;
    }
}
void display(stackstru *s) /*显示栈所有数据*/
{
    if(s->top==0)
        printf("the stack is empty!\n");
    else
    {
        while(s->top!=0)
        {
            printf("%d->",s->stack[s->top]);
            s->top=s->top-1;
        }
    }
}
int pop(stackstru *s) /*出栈操作并返回被删除的那个记录*/
{
    int y;
    if(s->top==0)
        printf("the stack is empty!\n");
    else
    {
        y=s->stack[s->top];
        s->top=s->top-1;
        return y;
    }
}

int gettop(stackstru *s) /*得到栈顶数*/
{
    int e;
    if(s->top==0)
```

```
        return 0;
    else
        e=s->stack[s->top];
    return e;
}
main(stackstru *p) //函数使用演示
{
    int n,i,k,h,x1,x2,select;
    printf("create a empty stack!\n");
    init(p);
    printf("input a stack length:\n");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("input a stack value:\n");
        scanf("%d",&k);
        push(p,k);
    }
    printf("select 1:display()\n");
    printf("select 2:push()\n");
    printf("select 3:pop()\n");
    printf("select 4:gettop()\n");
    printf("input a your select(1-4):\n");
    scanf("%d",&select);
    switch(select)
    {
        case 1:
        {
            display(p);
            break;
        }
        case 2:
        {
            printf("input a push a value:\n");
            scanf("%d",&h);
            push(p,h);
            display(p);
            break;
        }
        case 3:
        {
            x1=pop(p);
            printf("x1->%d\n",x1);
            display(p);
            break;
        }
    }
}
```

```

    }
    case 4:
    {
        x2=gettop(p);
        printf("x2->%d",x2);
        break;
    }
}
}

```

3.链表

源程序:

```

# define null 0

typedef char ElemType; /* 字符型数据*/

typedef struct LNode
{
    ElemType data;
    struct LNode *next;
};

setnull(struct LNode **p);
int length (struct LNode **p);
ElemType get(struct LNode **p,int i);
void insert(struct LNode **p,ElemType x,int i);
int delete(struct LNode **p,int i);
void display(struct LNode **p);
main()
{
    struct LNode *head,*q; /*定义静态变量*/
    int select,x1,x2,x3,x4;
    int i,n;
    int m,g;
    char e,y;

    head=setnull(&head); /*建议链表并设置为空表*/
    printf("请输入数据长度: ");
    scanf("%d",&n);
    for(i=1;i<n;i++);
    {
        printf("将数据插入到单链表中: ");
    }
}

```

```

scanf("%d",&y);
insert(&head,y,i);} /*插入数据到链表*/
display(&head); /*显示链表所有数据*/

printf("select 1 求长度 length()\n");
printf("select 2 取结点 get()\n");
printf("select 3 求值查找 locate()\n");
printf("select 4 删除结点 delete()\n");
printf("input your select: ");
scanf("%d",&select);
switch(select)
{
    case 1:
        {
            x1=length(&head);
            printf("输出单链表的长度%d ",x1);
            display(&head);
        }break;
    case 2:
        {
            printf("请输入要取得结点: ");
            scanf("%d",&m);
            x2=get(&head,m);
            printf(x2);
            display(&head);
        }break;
    case 3:
        {
            printf("请输入要查找的数据: ");
            scanf("%d",&e);
            x3=locate(&head,e);
            printf(x3);
            display(&head);
        }break;
    case 4:
        {
            printf("请输入要删除的结点: ");
            scanf("%d",&g);
            x4=delete(&head,g);
            printf(x4);
            display(&head);
        }break;
    }
}
}

```

```
setnull(struct LNode **p)
{
    *p=null;
}
int length (struct LNode **p)
{
    int n=0;
    struct LNode *q=*p;
    while (q!=null)
    {
        n++;
        q=q->next;
    }
    return(n);
}
ElemType get(struct LNode **p,int i)
{
    int j=1;
    struct LNode *q=*p;
    while (j<i&&q!=null)
    {
        q=q->next;
        j++;
    }
    if(q!=null)
        return(q->data);
    else
        printf("位置参数不正确!\n");
}
int locate(struct LNode **p,ElemType x)
{
    int n=0;
    struct LNode *q=*p;
    while (q!=null&&q->data!=x)
    {
        q=q->next;
        n++;
    }
    if(q==null)
        return(-1);
    else
        return(n+1);
}
void insert(struct LNode **p,ElemType x,int i)
```



```
{
int j=1;
struct LNode *s,*q;
s=(struct LNode *)malloc(sizeof(struct LNode));
s->data=x;
q=*p;
if(i==1)
{
s->next=q;
p=s;
}
else
{
while(j<i-1&&q->next!=null)
{
q=q->next;
j++;
}
if(j==i-1)
{
s->next=q->next;
q->next=s;
}
else
printf("位置参数不正确!\n");
}
}

int delete(struct LNode **p,int i)
{
int j=1;
struct LNode *q=*p,*t;
if(i==1)
{
t=q;
*p=q->next;
}
else
{
while(j<i-1&&q->next!=null)
{
q=q->next;
j++;
}
if(q->next!=null&&j==i-1)
{
```

```

        t=q->next;
        q->next=t->next;
    }
    else
        printf("位置参数不正确!\n");
    }
    if(t==null)
        free(t);
}

void display(struct LNode **p)
{
    struct LNode *q;
    q=*p;
    printf("单链表显示: ");
    if(q==null)
        printf("链表为空!");
    else if (q->next==null)
        printf("%c\n",q->data);
    else
    {
        while(q->next!=null)
        {
            printf("%c->",q->data);
            q=q->next;
        }
        printf("%c",q->data);
    }
    printf("\n");
}

```

4.链栈

源程序:

```

#define null 0

typedef struct stacknode
{
    int data;
    struct stacknode *next;
} stacklink;

typedef struct
{
    stacklink *top;

```

```
int stacksize;
}stackk;

initlink(stackk *s)
{
    s->top=(stacklink *)malloc(sizeof(stacklink));
    s->top->data=0;
    s->top->next=null;
}

int poplink(stackk *s)
{
    stackk *p;int v;
    if(s->top->next==null) printf("the stackis empty\n");
    else
    {
        v=s->top->next->data;
        p=s->top->next;
        s->top=s->top->next;
    }
    free(p);
    return v;
}

int pushlink(stackk *s,int x)
{
    stackk *p;
    p=(stacklink *)malloc(sizeof(stacklink));
    p->data=x;
    p->next=s->top->next;
    s->top->next=p;
}

int gettop(stackk *s)
{
    int e;
    if(s==null) printf("the stack is empty!\n");
    e=s->top->next->data;
    return e;
}

display(stackk *s)
{
    stackk *p;
    p=s->top->next;
    printf("display the stacklink:\n");
```

```
if (s->top=null) printf("the stacklink is empty!\n");
else
{
    while(p)
    {
        printf("->%d",p->data);
        p=p->next;
    }
}
```

```
main(stacklink *p)
{
    int n,k,i,select,h,x1,x2;
    printf("create a empty stacklink!\n");
    initlink(p);
    printf("input a stacklink length:\n");
    scanf("%d",&n);
    for (i=1;i<=n;i++)
        {printf("input a stacklink value:\n");
        scanf("%d",&k);
        pushlink(p,k);
        }
    printf("select 1:display()\n");
    printf("select 2:pushlink()\n");
    printf("select 3:poplink()\n");
    printf("select 4:gettop()\n");
    printf("input a your select(1-4):\n");
    scanf("%d",&select);
    switch(select)
    { case 1:
        { display(p);break;}
      case 2:
        { printf("input a push a value :\n");
          scanf("%d",&h);
          pushlink(p,h);
          display(p);
          break;}
      case 3:
        { x1=poplink(p);printf("x1->%d\n",x1);
          display(p);
          break;}
      case 4:
        { x2=gettop(p);printf("x2->%d",x2);
          break;}
    }
```

```

    }
}

```

5. 二叉树

源程序:

```

typedef struct bitnode
{
    char data;
    struct bitnode *lchild, *rchild;
}bitnode, *bitree;
void createbitree(t,n)
bitnode ** t;
int *n;
{
    char x;
    bitnode *q;
    *n=*n+1;
    printf("\n Input %d DATA:",*n);
    x=getchar();
    if(x!='\n') getchar();
    if (x=='\n')
        return;
    q=(bitnode*)malloc(sizeof(bitnode));
    q->data=x;
    q->lchild=NULL;
    q->rchild=NULL;
    *t=q;
    printf(" This Address is: %o, Data is: %c,\n Left Pointer is: %o,
Right Pointer is: %o",q,q->data,q->lchild,q->rchild);
    createbitree(&q->lchild,n);
    createbitree(&q->rchild,n);
    return;
}

void visit(e)
bitnode *e;
{
    printf(" Address: %o, Data: %c, Left Pointer: %o, Right Pointer:
%o\n",e,e->data,e->lchild,e->rchild);
}

void preordertraverse(t)

```

```
bitnode *t;
{
    if(t)
    {
        visit(t);
        preordertraverse(t->lchild);
        preordertraverse(t->rchild);
        return ;
    }
    else
        return ;
}

void countleaf(t,c)
bitnode *t;
int *c;
{
    if(t!=NULL)
    {
        if (t->lchild==NULL && t->rchild==NULL)
        { *c=*c+1;
        }
        countleaf(t->lchild,c);
        countleaf(t->rchild,c);
    }
    return;
}

int treehigh(t)
bitnode *t;
{
    int lh,rh,h;
    if(t==NULL)
        h=0;
    else
    {
        lh=treehigh(t->lchild);
        rh=treehigh(t->rchild);
        h=(lh>rh ? lh:rh)+1;
    }
    return h;
}

main()
{
    bitnode *t; int count=0;
    int n=0;
```

```

printf("\n Please input TREE Data:\n");
createbitree(&t,&n);
printf("\n This is TREE struct: \n");
preordertraverse(t);
countleaf(t,&count);
printf("\n This TREE has %d leaves ",count);
printf(" , High of The TREE is: %d\n",treehigh(t));
}

```

八、高精度运算专题

1.专题函数说明

说明:

本栏为本专题所有程序的公用部分，调用本专题任何一个程序必须加上本栏的代码
input/print 为高精度数输入输出，调用格式为 input(hp HightPoint,"123456")/print(hp
HighPoint)

本栏为纯 C++代码

源程序:

```

#include <iostream>
using namespace std;
#define maxsize 100
struct hp
{
    int len;
    int s[maxsize+1];
};

void input(hp &a,string str)
{
    int i;
    while(str[0]=='0' && str.size()!=1)
        str.erase(0,1);
    a.len=(int)str.size();
    for(i=1;i<=a.len;++i)
        a.s[i]=str[a.len-i]-48;
    for (i=a.len+1;i<=maxsize;++i)
        a.s[i]=0;
}

void print(const hp &y)
{

```

```
int i;
for(i=y.len;i>=1;i--)
    cout<<y.s[i];
cout<<endl;
}
```

2.高精度数比较

语法: `int result=compare(const hp &a,const hp &b);`

参数:

a,b: 进行比较的高精度数字

返回值: 比较结果, a>b 返回正数, a=b 返回 0, a<b 返回负数

源程序:

```
int compare(const hp &a,const hp &b)
{
int len;
if(a.len>b.len)
    len=a.len;
else
    len=b.len;
while(len>0 && a.s[len]==b.s[len]) len--;
if(len==0)
    return 0;
else
    return a.s[len]-b.s[len];
}
```

3.高精度数加法

语法: `plus(const hp &a,const hp &b, hp &c);`

参数:

a,b: 进行加法的高精度数字

返回值: 返回相加结果到 c 中

源程序:

```
void plus(const hp &a,const hp &b, hp &c)
{
int i,len;
for(i=1;i<=maxsize;i++) c.s[i]=0;
if(a.len>b.len) len=a.len;
else len=b.len;
for(i=1;i<=len;i++)
```



```

{
    c.s[i]+=a.s[i]+b.s[i];
    if(c.s[i]>=10)
    {
        c.s[i]-=10;
        c.s[i+1]++;
    }
}
if(c.s[len+1]>0) len++;
c.len=len;
}

```

4.高精度数减法

语法: `subtract(const hp &a,const hp &b,hp &c);`

参数:

a,b: 进行减法的高精度数字, a 是被减数, b 是减数, 不支持负数

返回值: 返回结果到 c 中

源程序:

```

void subtract(const hp &a,const hp &b,hp &c)
{
    int i,len;
    for(i=1;i<=maxsize;i++) c.s[i]=0;
    if(a.len>b.len) len=a.len;
    else len=b.len;
    for(i=1;i<=len;i++)
    {
        c.s[i]+=a.s[i]-b.s[i];
        if(c.s[i]<0)
        {
            c.s[i]+=10;
            c.s[i+1]--;
        }
    }
    while(len>1&& c.s[len]==0) len--;
    c.len=len;
}

```

5.高精度乘 10

语法: `multiply10(hp &a);`

参数:

a: 进行乘法的高精度数字

返回值: 返回结果到 a 中

源程序:

```
void multiply10(hp &a)
{
    int i;
    for(i=a.len;i>=1;i--)
        a.s[i+1]=a.s[i];
    a.s[1]=0;
    a.len++;
    while(a.len>1&& a.s[a.len]==0) a.len--;
}
```

6.高精度乘单精度

语法: multiply(const hp &a,int b, hp &c);

参数:

a: 进行乘法的高精度数字

b: 进行乘法的单精度数字

返回值: 返回结果到 c 中

源程序:

```
void multiply(const hp &a,int b, hp &c)
{
    int i,len;
    for(i=1;i<=maxsize;i++) c.s[i]=0;
    len=a.len;
    for(i=1;i<=len;i++)
    {
        c.s[i]+=a.s[i]*b;
        c.s[i+1]+=c.s[i]/10;
        c.s[i]%=10;
    }
    len++;
    while(c.s[len]>=10)
    {
        c.s[len+1]+=c.s[len]/10;
        c.s[len]%=10;
        len++;
    }
    while(len>1&&c.s[len]==0) len--;
    c.len=len;
}
```

7.高精度乘高精度

语法: multiplyh(const hp &a,const hp &b,hp &c);

参数:

a,b: 进行乘法的高精度数字

返回值: 返回结果到 c 中

源程序:

```
void multiplyh(const hp &a,const hp &b,hp &c)
{
    int i,j,len;
    for(i=1;i<=maxsize;i++) c.s[i]=0;
    for(i=1;i<=a.len;i++)
        for(j=1;j<=b.len;j++)
        {
            c.s[i+j-1]+=a.s[i]*b.s[j];
            c.s[i+j]+=c.s[i+j-1]/10;
            c.s[i+j-1]%=10;
        }
    len=a.len+b.len+1;
    while(len>1&& c.s[len]==0) len--;
    c.len=len;
}
```

8.高精度除单精度

语法: divide(const hp &a,int b,hp &c,int &d);

参数:

a: 进行除法的高精度数字

返回值: 返回商到 c 中, 余数到 d 中

源程序:

```
void divide(const hp &a,int b,hp &c,int &d)
{
    int i,len;
    for(i=1;i<=maxsize;i++) c.s[i]=0;
    len=a.len;
    d=0;
    for(i=len;i>=1;i--)
    {
        d=d*10+a.s[i];
        c.s[i]=d/b;
    }
}
```

```

    d%=b;
}
while(len>1&& c.s[len]==0) len--;
c.len=len;
}

```

9.高精度除高精度

语法: `divideh(const hp &a,const hp &b, hp &c, hp &d);`

参数:

a, b: 进行除法的高精度数字

返回值: 返回商到 c 中, 余数到 d 中

注意:

需要 `compare`、`multiply10`、`subtract`

源程序:

```

void divideh(const hp &a,const hp &b, hp &c, hp &d)
{
    hp e;
    int i,len;
    for(i=1;i<=maxsize;i++)
    {
        c.s[i]=0;
        d.s[i]=0;
    }
    len=a.len;
    d.len=1;
    for(i=len;i>=1;i--)
    {
        multiply10(d);
        d.s[1]=a.s[i];
        while(compare(d,b)>=0)
        {
            subtract(d,b,e);
            d=e;
            c.s[i]++;
        }
    }
    while(len>1&& c.s[len]==0) len--;
    c.len=len;
}

```

九、标准模板库的使用

1. 计算求和

```
TYPE accumulate( iterator start, iterator end, TYPE val );
```

The accumulate() function computes the sum of *val* and all of the elements in the range [*start,end*).

```
#include <iostream>
#include <vector>
////////////////////////////////////用法 1
#include <numeric>
using namespace std;
int main ()
{
    int arr[] = { 1,2,3,4,5 };
    vector<int> v(arr,arr+5);
```

```
// 1+2+3+4+5 and + 0
```

```
    int sum = accumulate(v.begin(), v.end(), 0, );
int sum
    cout << "sum = " << sum << endl;

    return 0;
}
//OUTPUT:
// sum = 15
```

```
////////////////////////////////////用法 2
```

```
#include <iostream>
#include <vector>
#include <numeric>
using namespace std;
int mul(int x,int y)
{
    return x*y;
}
int main ()
{
    int arr[] = { 1,2,3,4,5 };
    vector<int> v(arr,arr+5);
```

```

// 1+2+3+4+5 and + 0
int sum = accumulate(v.begin(), v.end(), 1, mul);
int sum
cout << "sum = " << sum << endl;

return 0;
}
//OUTPUT:
// sum = 120

////////////////////////////////////用法 3
#include <iostream>
#include <algorithm>
#include <numeric>
using namespace std;

int main ()
{
    int x[] = {1,2,3,4,5};

    // 1 * 2 * 3 * 4 * 5 and * 1
    int sum = accumulate(x, x+5, 1, multiplies<int>());
    cout << "sum = " << sum << endl;

    // 1 * 2 * 3 * 4 * 5 and * 2
    sum = accumulate(x, x+5, 2, multiplies<int>());
    cout << "sum = " << sum << endl;

    // 2 * 3 * 4 and * 10
    sum = accumulate(x+2, x+4, 10, multiplies<int>());
    cout << "sum = " << sum << endl;

    return 0;
}
OUTPUT:
// sum = 120
// sum = 240
// sum = 120

```

2.求数组中的最大值

max_element

Syntax:

78 / 80

```
#include <algorithm>
iterator max_element( iterator start, iterator end );
iterator max_element( iterator start, iterator end, BinPred p );
```

The `max_element()` function returns an **iterator** to the largest element in the range $[start, end)$.

If the binary predicate p is given, then it will be used instead of the `<` operator to determine the largest element.

Example code:

For example, the following code uses the `max_element()` function to determine the largest integer in an array and the largest character in a vector of characters:

```
int array[] = { 3, 1, 4, 1, 5, 9 };
unsigned int array_size = 6;
cout << "Max element in array is " << *max_element( array, array+array_size) << endl;
Max element in array is 9
```

3. sort 和 qsort

1.sort

用法:

```
#include <algorithm>
void sort( iterator start, iterator end );
void sort( iterator start, iterator end, StrictWeakOrdering cmp );
```

例子:

```
#include<iostream>
#include<algorithm>
using namespace std;
int cmp(int a,int b)
{
    return a<b;
}
int main()
{
    int a[10]={1,5,68,32,15,48,78,45,35,45};
    for(int i=0;i<10;i++)
        cout<<a[i]<<"\t";
    sort(a,a+10,cmp);
    cout<<endl;
    for(int i=0;i<10;i++)
        cout<<a[i]<<"\t";
    return 0;
```

```
}
```

2.qsort

```
#include<cstdlib>
```

一:对整型数据进行排序

```
int num[100];
```

Sample:

```
int cmp ( const void *a , const void *b )
```

```
{
```

```
return *(int *)a - *(int *)b;
```

```
}
```

```
qsort(num,100,sizeof(num[0]),cmp);
```

二:对结构体进行排序

```
struct In
```

```
{
```

```
double data;
```

```
int other;
```

```
}s[100];
```

//按照 data 的值从小到大将结构体排序,关于结构体内的排序关键数据 data 的类型可以很多种

```
int cmp( const void *a ,const void *b)
```

```
{
```

```
struct In *c=(In *)a;
```

```
struct In *d=(In *)b;
```

```
return c->data > d->data ? 1 : -1;
```

```
}
```

```
qsort(s,100,sizeof(s[0]),cmp);
```

九、其他

1.运行时间计算.

```
#include<time.h>
```

```
clock_t start,finish;
```

```
start=clock();
```

```
finish=clock();
```

```
cout<< (double)(finish - start) / CLOCKS_PER_SEC;//运行时间:
```