作业6

自检

- Ex1. 一个神秘数的立方的后三位全为1。请编写一个程序，验证正整数n（小于1000，通过键盘输入）以内是否有神秘数（是，则显示“yes”；否，则显示“no”）。

```
int main()
{     int iUpperBound;
      printf("Input the top upper bound:");
      scanf("%d" , &iUpperBound);
      if(mysticalFind(iUpperBound))
            ……;
}
```

```
bool mysticalFind(int n)
{     for(int i = 1; i < n; ++i)
      {     if(i*i*i%1000 == 111)
                  return true;
      }
       return false;
}
```

- Ex2. 编写程序实现将一个大于1的正整数表示成所有素数因子的次方相乘的形式输出，次方用英文圆括号( )表示。要求按从小到大的顺序输出素数因子，例如，输入72，输出2(3)3(2)，输入181944，输出2(3)3(2)7(1)19(2)，输入21546465，输出3(1)5(1)1436431(1)。

- 思路：
  - 从2开始，判断是素数，不停地除，看n可以整除该素数的几次方；
  - 其实不需要判断素数，例如将n里所有的2除去之后就不会再出现4、8…了（思考筛法求素数的机制）。

```
int main()
{
	int n;

	cin >> n;
	fun(n);

	return 0;
}
```

```
bool prime(int x)
{
	int i = 2;
	while(i * i <= x)
	{
		if(x % i == 0)
			return false;
		++i;
	}
	return true;
}
```

```
void fun(int n)
{
    for(int i=2; i <= n; ++i)
    {
        if(n%i==0 && prime(i))    // if(n%i == 0)
        {
            int count = 0;
            while(n/i != 0 && n%i == 0)
            {
                n /= i;
                ++count;
            }
            cout << i <<"(" << count << ")";
        }
    }
}
```

如果 n 是最大的 int，
又恰好是素数，
i 会越界，成为负数，
于是...！！

```
void fun(int n)
{
    for(int i=2; i <= sqrt((float)n); ++i)
    {
        if(n%i==0 && prime(i))    // if(n%i == 0)
        {
            int count = 0;
            while(n/i != 0 && n%i == 0)
            {
                n /= i;
                ++count;
            }
            cout << i <<"(" << count << ")";
        }
    }
    if(n != 1)      cout << n <<"(1)";
}
```

如果 n 是
较大的 素数，
单独处理

- Ex3. 编写一个函数 double ItrNewton(double a, double b, double c, double d)，用牛顿迭代法求一元三次方程 ax3 + bx2 + cx + d = 0 在0附近的根，两次迭代结果变化小于10-6为止。在main函数中输入方程的四个系数，并输出该方程的根。不考虑分母为0的情况。（提示：牛顿迭代公式为Xn+1 = Xn - f(Xn)/f'(Xn)，其中f(x) = ax3 + bx2 + cx + d，f'(x) = 3ax2 + 2bx + c)

```
int main()
{
	double a, b, c, d;

	cin >> a >> b >> c >> d;
	cout << itrNewton(a, b, c, d) << endl;

	return 0;
}
```

牛顿迭代法求一元多次方程的根

x3-x=0

```
#include <cmath>
double itrNewton(double a, double b, double c, double d)
{
      double x1, x2;
      x2 = 0;
      do
      {
            x1 = x2;  // 不能放在循环体的最后
            double fn = a*x1*x1*x1 + b*x1*x1 + c*x1 + d;
            double fnp = 3*a*x1*x1 + 2*b*x1 + c;
            x2 = x1 - fn/fnp;
      }while( fabs(x2 - x1) >= 1e-6 );
      return x2;
}
```

```
if(!fnp) fnp = 1e-6;
```

- **Ex4. 设计函数，判断一个正整数是不是素数，并调用该函数验证哥德巴赫猜想：任一大于2的偶数，等于某两个素数之和。**

```
6
= 3 + 3
8
= 3 + 5
96
= 7 + 89
0
```

```
bool prime(int x)
{
      int i = 2;
      while(i * i <= x)
      {
            if(x % i == 0)
                  return false;
            ++i;
      }
      return true;
}
```

```
int n, i;
cin >> n;
while(n != 0)
{
      i = 2;
      while(!prime(i) || !prime(n - i)) // i不是素数或n-i不是素数
                  i += 1;
      cout << n << " = " << i << " + " << n - i << endl;
      cin >> n;
}
```

- 2. 编程实现分别按顺序和逆序输出26个英文字母，验证字符型数据可以进行算术、比较操作。

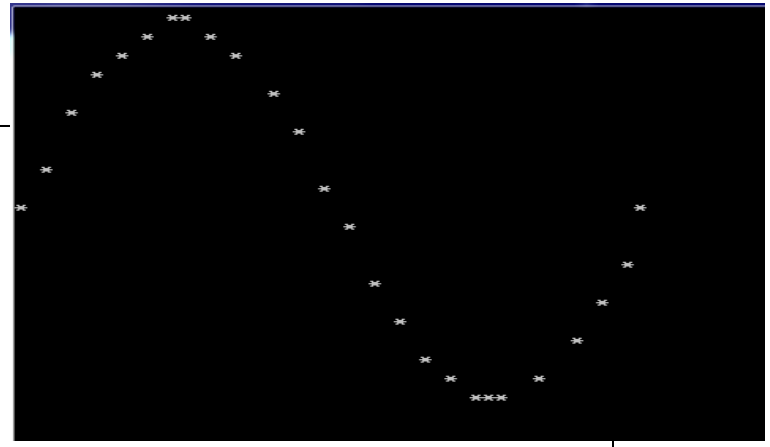
```
for(ch='a'; ch <= 'z'; ++ch)
     printf("%c, ", ch);
printf("\n");
for(ch='Z'; ch >= 'A'; --ch)
     printf("%c, ", ch);
printf("\n");
```

- 3. 设计一个函数，将一个数字字符（'0'~'9'），转换为对应的整数（0~9）。

```
int f(char ch)
{
     return ch - '0';
}
```

- 4. 编写程序，实现用*近似画[0, 360。]区间的正弦曲线y = sin(x)、[-3, 3]区间的抛物线y = x*x和半径r为10的圆x*x + y*y = r*r。（提示：为了让曲线显得好看一点，可以加调节因子T（≈2）拉伸横坐标。）

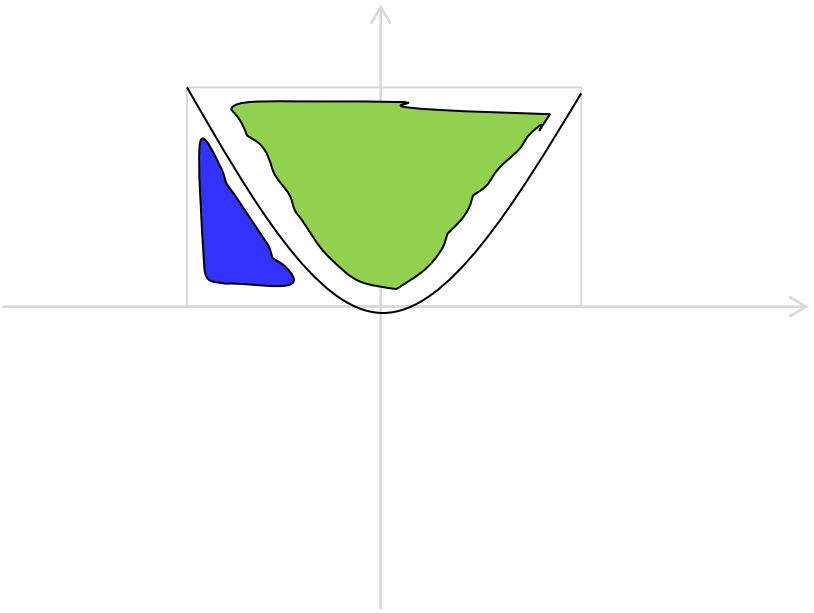
[0, 360]区间的正弦曲线。



```
#define PI 3.14

for( double y=1; y >= -1; y -= 0.1 )
{
     for( double x=0; x < 2*PI; x += 0.25 )
     {
          if( fabs(y-sin(x)) < 0.05 )
               cout << "*";
          else
               cout << "  ";
     }
     cout << endl;
}
```

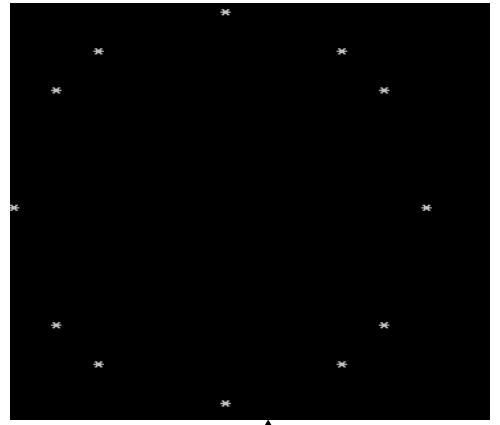
x可以分两个范围
[-3, -sqrt(y)]与[-sqrt(y), sqrt(y)]
加调节因子T之后，x的范围变为
[-3*T, -sqrt(y)*T]与[-sqrt(y)*T, sqrt(y)*T]

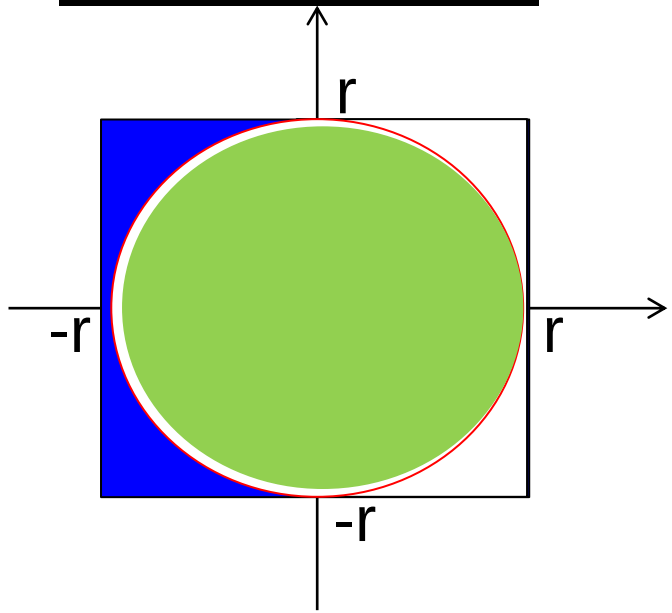
```
#include "cmath"
#define T 3

for(double y=9; y >= 0; --y)
{
        int i, x = T*sqrt(y);
        for(i=-3*T; i < -x; ++i)
                cout << " ";
        cout << "#";

        for(; i < x; ++i)
                cout << " ";
        cout << "#" << endl;
}
```



在控制台绘制空心圆：流程控制

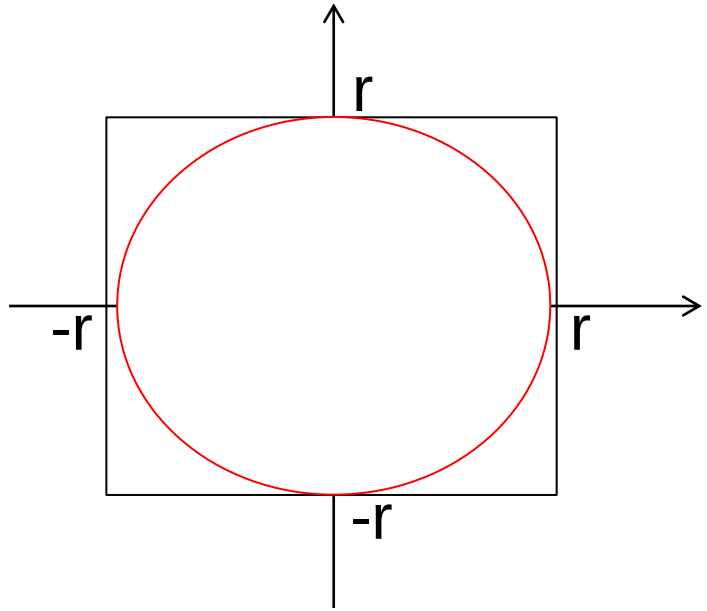
```
#include "cmath"
#define T 2.2

for(double y=r; y >= -r; --y)
{
      double i, x = T*sqrt(r*r - y*y);
      for(i = -r*T; i < -x; ++i)
            cout << " ";    //蓝色部分
      cout << "*";          //左半圆

      for(; i < x; ++i)
            cout << " ";    //绿色部分
      cout << "*" << endl;      //右半圆
}
```



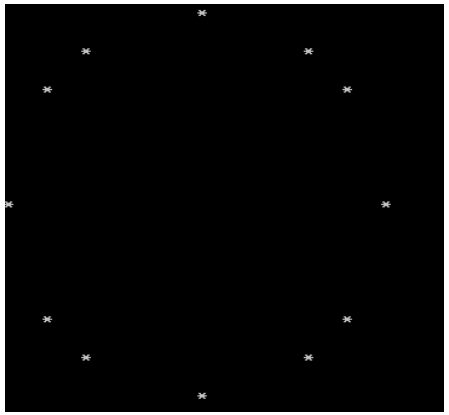


注意：
数据类型
变量的作用域

```
for(double x=-r*T; x <= r*T; ++x)
      if(x*x/T/T+y*y - r*r <= 0 && x*x/T/T+y*y - r*r > -10)
```




```
for(double y=r; y >= -r; --y)
{
     for(double x=-r; x <= r; ++x)
          if(x*x+y*y == r*r)
               cout << "*";    //红色部分的圆
          else
               cout << " ";    //白色部分
     cout << endl;
}
```



注意：
除法操作符，数据类型
关系操作的边界问题
逻辑运算符

自检

- **5. 请编写以下三个函数，并在main函数中：输入三个点的坐标；调用前两个函数输出前面两个点所决定的直线方程；调用第三个函数，判断第三个点是否在直线上。**

  double ComputeLineK(double p1x, double p1y, double p2x, double p2y); //计算斜率

  double ComputeLineB(double p1x, double p1y, double p2x, double p2y); //计算截距

  bool IsPointOnSegment(double px, double py, double p1x, double p1y, double p2x, double p2y)

  测试用例1:

  输入：1 1 0.5 0.5 0 0

  输出：The equation is: y=x；The point is on the line segment.

  测试用例2:

  输入：1 3 1 2 0 0

  输出：The equation is: x=1；The point is not on the line segment.

  测试用例3:

  输入：1 -2 2 -1 0 0

  输出：The equation is: y=x-3；The point is not on the line segment.

自检

- 测试用例1：
  - 输入：1 1 0.5 0.5 0 0
  - 输出：The equation is: y=x；The point is on the line segment.
- 测试用例2：
  - 输入：1 3 1 2 0 0
  - 输出：The equation is: x=1；The point is not on the line segment.
- 测试用例3：
  - 输入：1 -2 2 -1 0 0
  - 输出：The equation is: y=x-3；The point is not on the line segment.

自检

```
double ComputeLineK(double p1x, double p1y, double p2x, double p2y)
{
	return (p2y-p1y)/(p2x-p1x);
}

double ComputeLineB(double p1x, double p1y, double p2x, double p2y)
{
	return (p2y-p2x*ComputeLineK(p1x, p1y, p2x, p2y));
}
```

自检

```
bool IsPointOnSegment(double px, double py, double p1x, double
p1y, double p2x, double p2y)
{
    bool m = false;
    if(p1x == p2x)
    {   if(px == p1x)
            m = true;
    }
    else
    {   double k = ComputeLineK(p1x, p1y, p2x, p2y);
        double b = ComputeLineB(p1x, p1y, p2x, p2y);
        if(py == k*px + b)
            m = true;
    }
    return m;
}
```

main

自检

```
bool k_flag = true;

if(p1x == p2x)
    k_flag = false;  //斜率不存在

double k, b;
if(k_flag)
{
    k = ComputeLineK(p1x, p1y, p2x, p2y);  //得到斜率
    b = ComputeLineB(p1x, p1y, p2x, p2y);  //得到截距
}
```

自检

```
if(!k_flag)          //斜率不存在，平行于 y 轴
    cout << "The equation: " << 'x' << '=' << p1x << endl;
else if(k == 0)      //斜率为0，平行于 x 轴
    cout << "The equation: " << 'y' << '=' << b << endl;
else if(b > 0)
    if(k != 1)
        cout << "The equation:" << 'y' << '=' << k << 'x' << '+' << b << endl;
    else             //防止出现 y = 1x + 3 这种形式
        cout << "The equation: " << 'y' << '=' << 'x' << '+' << b << endl;
else if(b < 0)       //防止出现 y = x + -3 这种形式
    if(k != 1)
        cout << "The equation: " << 'y' << '=' << k << 'x' << b << endl;
    else
        cout << "The equation: " << 'y' << '=' << 'x' << b << endl;
else                 //防止出现 y = x + 0 这种形式
    if(k != 1)
        cout << "The equation: " << 'y' << '=' << k << 'x' << endl;
    else
        cout << "The equation: " << 'y' << '=' << 'x' << endl;
```

自检

```
double px, py;
cout << "请输入px, py: " << endl;
cin >> px >> py;

bool m = IsPointOnSegment(px, py, p1x, p1y, p2x, p2y);

if(m)
    cout << "在那条线上." << endl;
else
    cout << "不在那条线上." << endl;
```

自检

- 6.（选做）编写函数void Drawtri(double p1x, double p1y, double p2x, double p2y, double p3x, double 3y)，用*画出顶点坐标为(0,0)、(5,10)和(10,5)的近似三角形；进而计算三角形的面积。（提示：可利用两点之间的距离公式先求出边长a、b、c，再利用海伦公式s=sqrt(p(p-a)(p-b)(p-c))求面积，其中p=(a+b+c)/2。）

自检

```
double p1x, p1y, p2x, p2y, p3x, p3y;

cout << "请输入三个点的坐标：" << endl;
cin >> p1x >> p1y >> p2x >> p2y >> p3x >> p3y;

draw_line(p1x, p1y, p2x, p2y);
draw_line(p2x, p2y, p3x, p3y);
draw_line(p3x, p3y, p1x, p1y);
draw_tri(p1x, p1y, p2x, p2y, p3x, p3y);
```

划线

自检

```
void draw_line(double p1x, double p1y, double p2x, double p2y)
{
    bool k_flag = true;

    if(p1x == p2x)
        k_flag = false;  //斜率不存在

    double k, b;
    if(k_flag)
    {
        k = ComputeLineK(p1x, p1y, p2x, p2y);
        b = ComputeLineB(p1x, p1y, p2x, p2y);
    }
```

继续划线

自检

```
double py_max, py_min, px_max, px_min;
py_max = (p1y>p2y)? p1y : p2y;     //py_max是最大的纵坐标
py_min = (p1y<p2y)? p1y : p2y;     //py_min是最小的纵坐标
px_max = (p1x>p2x)? p1x : p2x;     //px_max是最大的横坐标
px_min = (p1x<p2x)? p1x : p2x;     //px_min是最小的横坐标
for( double y=py_max; y >= py_min; y -= 0.5 )
{
    for(double x=px_min; x < px_max; x += 0.5 )
    {
        if( (k_flag && (fabs(y-k*x-b)<0.05))
                || !k_flag && (fabs(x-p1x)<0.05) )
            cout<<"*";
        else
            cout<<"  ";
    }
    cout << endl;
}
```

画三角形

自检

```
void Drawtri(double p1x, double p1y, double p2x, double p2y,
double p3x, double p3y)
{
    bool k1_flag = true;
    if(p1x == p2x)
        k1_flag = false;  //斜率不存在
…//其他两条边

    double k1, b1;
    if(k1_flag)
    {
        k1 = ComputeLineK(p1x, p1y, p2x, p2y);
        b1 = ComputeLineB(p1x, p1y, p2x, p2y);
    }
…//其他两条边
```

继续画三角形

自检

```
double py_max, py_min, px_max, px_min;

py_max = (p1y>p2y)? (p1y>p3y ? p1y : p3y): (p2y>p3y ? p2y : p3y);
//py_max是最大的纵坐标
py_min = (p1y<p2y)? (p1y<p3y ? p1y : p3y): (p2y<p3y ? p2y : p3y);
//py_min是最小的纵坐标

px_max =  …//其他两条边
px_min =  …//其他两条边
```

继续画三角形

自检

```
for( double y=py_max; y >= py_min; y -= 0.5 )
{
    for(double x=px_min; x < px_max; x += 0.5 )
    {
        if( k1_flag && (fabs(y-k1*x-b1)<0.05)
              || k2_flag && (fabs(y-k2*x-b2)<0.05)
              || k3_flag && (fabs(y-k3*x-b3)<0.05)
              || !k1_flag && (fabs(x-p1x)<0.05)
              || !k2_flag && (fabs(x-p2x)<0.05)
              || !k3_flag && (fabs(x-p3x)<0.05) )
            cout<<"*";
        else
            cout<<"  ";
    }
    cout << endl;
}
```

继续加条件可以解决钝角三角形输出问题，比如：

```
if(p1x>p2x) {double x1=p1x; p1x=p2x;p2x=x1;}
if(p1y<p2y) {double y1=p1y; p1y=p2y;p2y=y1;}
if( (k1_flag && (fabs(y-k1*x-b1)<0.05
       && x>=p1x && x<=p2x
       && y<=p1y && y>=p2y)
    ||      …
```

计算面积

自检

```
double cmp_tri_area(double p1x, double p1y, double p2x, double
p2y, double p3x, double p3y)
{
    //length of P1P2 两点间距离公式
  double dA = sqrt( (p2x-p1x)*(p2x-p1x) + (p2y-p1y)*(p2y-p1y) );
    //length of P1P3
  double dB = sqrt( (p3x-p1x)*(p3x-p1x) + (p3y-p1y)*(p3y-p1y) );
    //length of P2P3
  double dC = sqrt( (p3x-p2x)*(p3x-p2x) + (p3y-p2y)*(p3y-p2y) );

  double dP = (dA+dB+dC)/2.0;

  return sqrt( dP*(dP-dA)*(dP-dB)*(dP-dC) );
}
```

自检

```
//把向量扩展到三维做叉乘来求面积的方法，可以少做一些平方根运算
double TriangleArea( double dP1x, double dP1y,
        double dP2x, double dP2y,
        double dP3x, double dP3y )
{
        //vector P1P2
        double dV1x = dP2x - dP1x;
        double dV1y = dP2y - dP1y;
        double dV1z = 0;
        //vector P1P3
        double dV2x = dP3x - dP1x;
        double dV2y = dP3y - dP1y;
        double dV2z = 0;

        //3d cross product
        double dNx = dV1y*dV2z - dV1z*dV2y;
        double dNy = dV1z*dV2x - dV1x*dV2z;
        double dNz = dV1x*dV2y - dV1y*dV2x;

        return 0.5 * sqrt( dNx*dNx + dNy*dNy + dNz*dNz );
}
```

- 7. 设计程序，生成10个0~1之间的随机小数。

```
float r;
srand(time(0));
for( int i = 0; i < 10; ++i )
{
      cout << (float)rand()/RAND_MAX << endl;
}
```

若要求其范围为 0.1~0.9

```
int j = 10*(float)rand()/RAND_MAX; //0~9
if(j != 0)
      r = j/10.0;
cout << r << endl;
```

Thanks !