```cpp
#include <iostream> // cin cout 的头文件
#include <iomanip>  // setprecision 的头文件
#include <cstdio>   // printf scanf 的头文件
using namespace std;

// 函数声明
void ascii();
void printf();
void addition();
void precision();
void circle();
void e_notation();
void address();
void pass_by_value(int x, int y);
void pass_by_pointer(int* x, int* y);

int main()
{
  // 如果想使用哪一个函数，就把这个函数的注释去掉，其他函数的注释加上
  //  ascii();
  // e_notation();
  // printf();
  precision();
  // addition();
  // circle();
  // address();
  /* // 指针的程序想要测试可以对27-34行一起取消注释
    int a = 5;
    int b = 100;
    pass_by_value(a, b);
    cout << "main: " << a << " " << b << endl; // 5 100
    pass_by_pointer(&a, &b);
    cout << "main: " << a << " " << b << endl; // 6 101
  */
  return 0;
};

void e_notation()
{
  /*
    科学计数法 3E5 = 3 * 10^5
    0.3E-3 = 0.3 * 10^-3
    注意 E前面的数字可以是小数 后面的指数必须是整数

    十六进制数字 0x12f3 前面用0x表示 后面的数字是16进制的数字
  */
  int x = 3E5;
```

```
  int a = 3e5;
  int y = 0x3E5; // 一旦用0x表示，e/E就不是科学计数法了
  int z = 0x3e5;
  cout << x << " " << a << " " << y << " " << z << endl;

  double f = 0.3E-3;
  cout << f << endl;
}

void ascii()
{
  /*
    整数
      short
      int
      long long int
      unsigned int
    小数（浮点）
      float
      double
    字符
      char
    他们之间可以互相转化，但是注意转化是要留意数据的大小是否会溢出(大小超出合理范围)
  */
  int x = 5;
  short y = x;
  cout << y << endl;

  x = 65;
  char c = x; // 'A' char 的转化遵循ascii表格

  x = -100;
  unsigned int u = x; // unsigned int 不能存储负数 所以虽然不报错 但是答案不会正确
  cout << u << endl;

  x = 2147483647;
  short p = x; // short 的范围是 -32768 ~ 32767 所以这里也会溢出
  cout << p << endl;

  /*
    强制类型转化
    (数据类型)(表达式)
  */
  // 0.2 + 0.3 = 0.5 但是 类型转换时是保留整数 所以两个都是0
  x = (int) (0.2 + 0.3) + (int) (0.4 + 0.5);
  cout << x << endl; // 0
  x = 0.2 + 0.3 + 0.4 + 0.5;
  cout << x << endl; // 1
```

```
};

void precision()
{
  /*
    double 15位有效数字
    float 7位有效数字
  */
  float f = 0.1284427;
  double x = 12.23452356455749;
  /*
    c语言保留精度 %m.nlf
    m是总长度 (包括整数,小数点和小数部分)
    n是小数点后的位数
    lf 是double类型特有的输入输出符号标志 告诉程序这里按照double类型进行解析
    float - f
    double - lf
    int - d
    char - c
    string - s
    short - h
    long long int - ld
  */
  printf("%.12lf\n", x);
  // 注意如果这里如果保留30位小数 并不会成功 从第16位开始就不准确了，因为double类型只能保留15位有效数字
  printf("%.30lf\n", x);
  // c++ 默认输出保留6位有效数字
  cout << x << endl;
  // c++ 保留精度 所有有效数字
  cout << setprecision(10) << x << endl; // setprecision 可以设置输出的精度
  // c++ 保留精度 小数部分
  cout << fixed << setprecision(10) << x << endl; // fixed 表示保留小数部分10位
}

void addition()
{
  int x = 10;
  cout << x << endl;
  int y = 20;
  int z = y - (x++); // x++ 先取值 在进行++这个运算
  // x = 11
  int p = y - (++x);          // ++x 先进行++运算 在取值
  cout << "z: " << z << endl; // z = 10
  cout << "p: " << p << endl; // p = 8
}

const double PI = 3.14159;
void circle()
```

```
{
  // 课堂练习
  double radius, diameter, circumference, area;
  double 半径, 直径, 周长, 面积;
  scanf("%lf", &radius);
  diameter = 2 * radius;
  circumference = 2 * PI * radius;
  area = PI * radius * radius;
  printf("%.2lf", diameter);
  printf("直径: %.2lf 周长: %.2lf 面积: %.2lf", diameter, circumference, area);
}
/*
  cout c++ 里的输出 cin c++ 里的输入
  printf c 里的输出 scanf c 里的输入
*/
void printf()
{
  int a,b,c;
  /* & 获取a该变量的地址
     * 通过地址获取变量的值
  */
  // 注意scanf确定了输入的格式
  // 如果引号里是%d-%d-%d 那么输入的时候必须是 1-2-3
  scanf("%d-%d-%d", &a, &b, &c);
  printf("这些变量的值分别是%d %d %d", a, b, c);

  float d, e, f;
  // 如果引号里是%d %d %d 那么输入的时候必须是 1 2 3
  scanf("%f %f %f", &d, &e, &f);
  printf("这些变量的值分别是%.2f %.2f %.2f", d, e, f);

  // 如果引号里是%d%d%d 注意这里数字也需要用空格或回车或tab隔开 1 2 3 不然123 会被当成一个数字
  scanf("%d%*d%d", &a, &c); // %*d表示跳过一个整数
  printf("%d %d", a, c);
  // %4d表示读取4位整数 这时候可以连着写
  //12345678910 会被当成两个数字 1234 5678 多余的数字会被丢弃
  scanf("%4d%4d", &a, &b);

  printf("%d %d", a, b);
}

void address()
{
  /*
     & 获取a该变量的地址
     * 通过地址获取变量的值
     如果 p = 0x7ffeeb1b3a3c (地址)
     *p = 5 这个地址存放的值
```

```
    如果 x = 5
    &x = 0x7ffeeb1b3a3c
  */

  int a = 10;
  int b = 100;
  int *p = &a; // int* 专门用来存放地址的数据类型

  cout << "a的地址是" << &a << endl;
  cout << "p的值是" << p << endl;
  cout << "p指向的值是" << *p << endl;

  cout << "b的地址是" << &b << endl;
  cout << "b的值是" << b << endl;
}

// 通过值传递无法改变原来变量的值
void pass_by_value(int x, int y){
  x = x + 1;
  y = y + 1;
}
// 这里通过地址可以改变原来变量的值
void pass_by_pointer(int *x, int *y){
  *x = *x + 1;
  *y = *y + 1;
}
```