习题一

在C++中，当一个浮点数被转换成整型数时，数值的小数部分会被直接舍去。因此，将4.99999转换为整型数时，结果是4。在许多场合，将浮点数转换为最接近的整型数将会更有用。现有一个浮点型变量x，你可以通过将变量加上0.5并舍去小数部分来得到其最接近的整数。因为数值四舍五入取整时会朝着数轴上0的方向，所以负数取整时需要为其数值减去0.5，而不是加上0.5。

编写一个`roundToNearestInt(x)`函数，将浮点数取整为与其最接近的整数，并编写一个适当的main函数来验证它

```cpp
#include <cassert>
#include <cstdlib>
#include <vector>
#include <cmath>

int roundToNearestInt(double x) {
    int sign = std::signbit(x) ? -1 : 1;
    x = std::abs(x);
    int res = x + 0.5;
    return sign * res;
}

int main() {
    std::vector<double> test_arr = {1.2, 4, 1.6, 1.4};
    std::vector<int>    vaild_arr = {1, 4, 2, 1};
    for (int i = 0; i < test_arr.size(); i++) {
        assert(vaild_arr[i] == roundToNearestInt(test_arr[i]));
    }
    return 0;
}
```

习题二

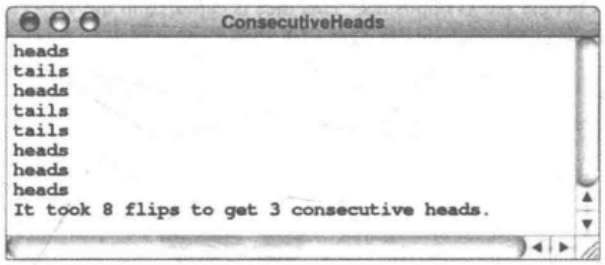
一个比1大的整数如果除了自身和1之外没有其他因子，则被称为素数（prime）。例如17就是一个素数，因为除了1和17之外没有其他数可以整除它。91不是素数，因为它还可以被7和13整除。

编写一个判定函数`isPrime(n)`，如果整数n是素数，则返回true，反之则返回false。为了测试你的算法，编写一个main函数来列出1到100之间的素数。

```
bool isPrime(int n) {
    if (n ⩽ 2) {
        return n == 2;
    }
    if (n % 2 == 0) {
        return false;
    }
    for (int i = 3; i ⩽ sqrt(n); i += 2) {
        if (n % i == 0) {
            return false;
        }
    }
    return true;
}
```

习题三

15. 正面…
正面…
正面…
在一定的概率下，如果没有任何其他情况发生，一个弱者可能会重新审视他的信仰。
——汤姆·斯托帕德，《罗森克兰茨和吉尔登斯特恩已死》*Rxencrantz and Guildenstern Are Dead*，1967
编写一个模拟重复抛硬币过程的程序，一直抛直到连续三次正面朝上。此时，你的程序必须显示出共抛了多少次硬币。下面是一个可能的程序运行例子：




```
void consecutiveHeads() {
    int count = 0;
    int tc = 0;
    double p = .5;
    while (tc < 3) {
        if (randomChance(p)) {
            cout << "heads" << endl;
            tc++;
        }
        else {
            cout << "tails" << endl;
        }
        count++;
    }
    cout << "It took " << count << " filps to get "
        << tc << " consecutive heads." << endl;
}
```