```
/*
 * Programming Challenge 2
 */
#include <cassert>
#include <cmath>
#include <iostream>
using namespace std;

bool isFactor (int numerator, int denominator);
void makeChange (int initialValue, int& quarters, int& dimes, int& nickels, int& pennies);
double launchHumanCannonball (double initialVelocity, double launchAngle);

/* for unit testing -- do not alter */
template <typename X, typename A>
void btassert(A assertion);
void unittest ();

int main (int argc, char* argv[])
{
    unittest();

    return 0;
}

/*
 * Determine if one integer is a factor (evenly divisible into)
 * another integer.
 * @param numerator an int representing the numerator (the value being tested against)
 *        in the equation
 * @param denominator an int representing the denominator (the potential factor)
 *        in the equation
 * @return true if the numerator is evenly divisible by the denominator (making the denominator
 *         a factor, else return false
 */
bool isFactor (int numerator, int denominator)
{
    if( !(numerator % denominator) )    {

        return true;

    } else {

        return false;

    }
}

/*
 * Given an initial integer value (representing change to be given, such as in a financial
 * transaction), break the value down into the number of quarters, dimes, nickels, and pennies
```

```
 * that would be given as change.
 * @param initialValue an int representing the amount of change to be broken down
 * @param quarters the number of quarters that come out of the initial value
 * @param dimes the number of dimes that come out of the initial value, after quarters have
 *        been taken out
 * @param nickels the number of nickels that come out of the initial value, after quarters
 *        and dimes have been taken out
 * @param pennies the number of pennies that come out of the initial value, after quarters,
 *        dimes, and nickels have been taken out
 */
void makeChange (int initialValue, int& quarters, int& dimes, int& nickels, int& pennies)
{
    int remainder;

    quarters    = initialValue / 25;
    remainder   = initialValue % 25;

    dimes       = remainder / 10;
    remainder   = remainder % 10;

    nickels     = remainder / 5;
    remainder   = remainder % 5;

    pennies     = remainder;
}

/*
 * Computes the horizontal distance traveled by a human cannonball given an initial
 * velocity and launch angle. Simplified -- does not account for many factors that
 * affect projectile motion.
 * @param initialVelocity a double representing the initial velocity of the human
 *        cannonball in meters/second
 * @param launchAngle a double representing the launch angle of the human cannonball
 *        in degrees
 * @return a double representing the horizontal distance the human cannonball will
 *         travel
 */
double launchHumanCannonball (double initialVelocity, double launchAngle)
{
    double radangle     = launchAngle * (M_PI/180);
    double xveloc       = initialVelocity * cos( radangle );
    double yveloc       = initialVelocity * sin( radangle ) * -1;
    double flighttime   = yveloc * 2 / -9.8;

    return xveloc * flighttime;
}

/*
 * Unit testing functions. Do not alter.
 */
```

```
void unittest ()
{
    cout << "\nSTARTING UNIT TEST\n\n";

    try {
        btassert<bool>(isFactor(100, 25) == true);
        cout << "Passed TEST 1: isFactor 100/25\n";
    } catch (bool b) {
        cout << "# FAILED TEST 1 isFactor 100/25 #\n";
    }

    try {
        btassert<bool>(isFactor(100, 26) == false);
        cout << "Passed TEST 2: isFactor 100/26\n";
    } catch (bool b) {
        cout << "# FAILED TEST 2 isFactor 100/26 #\n";
    }

    try {
        btassert<bool>(isFactor(100, 26) == false);
        cout << "Passed TEST 3: isFactor 100/100\n";
    } catch (bool b) {
        cout << "# FAILED TEST 3 isFactor 100/100 #\n";
    }

    int q = 0, n = 0, d = 0, p = 0;

    try {
        makeChange(0, q, d, n, p);
        btassert<bool> ((q == 0) && (d == 0) && (n == 0) && (p == 0));
        cout << "Passed TEST 4: makeChange $0.00\n";
    } catch (bool b) {
        cout << "# FAILED TEST 4 makeChange $0.00 #\n";
    }

    try {
        makeChange(41, q, d, n, p);
        btassert<bool> ((q == 1) && (d == 1) && (n == 1) && (p == 1));
        cout << "Passed TEST 5: makeChange $0.41\n";
    } catch (bool b) {
        cout << "# FAILED TEST 5 makeChange $0.41 #\n";
    }

    try {
        makeChange(99, q, d, n, p);
        btassert<bool> ((q == 3) && (d == 2) && (n == 0) && (p == 4));
        cout << "Passed TEST 6: makeChange $0.99\n";
    } catch (bool b) {
        cout << "# FAILED TEST 6 makeChange $0.99 #\n";
    }
```

```
    double value = launchHumanCannonball(25.0, 45.0);
    double scale = 0.01;  // round to nearest one-hundreth

    try {
        value = (int)(value / scale) * scale;
        btassert<bool>((value <= 63.9) && (value >= 63.5));
        cout << "Passed TEST 7: launchHumanCannonball(25, 45)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 7 launchHumanCannonball(25, 45) #\n";
    }

    try {
        value = launchHumanCannonball(40.0, 60.0);
        value = (int)(value / scale) * scale;
        btassert<bool>((value <= 141.5) && (value >= 141.1));
        cout << "Passed TEST 8: launchHumanCannonball(40, 60)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 8 launchHumanCannonball(40, 60) #\n";
    }

    try {
        value = launchHumanCannonball(10.0, 30.0);
        value = (int)(value / scale) * scale;
        btassert<bool>((value <= 9.0) && (value >= 8.6));
        cout << "Passed TEST 9: launchHumanCannonball(10, 30)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 9 launchHumanCannonball(10, 30) #\n";
    }

    try {
        value = launchHumanCannonball(10.0, 90.0);
        value = (int)(value / scale) * scale;
        btassert<bool>((value <= 0.2) && (value >= -0.2));
        cout << "Passed TEST 10: launchHumanCannonball(10, 90)\n";
    } catch (bool b) {
        cout << "# FAILED TEST 10 launchHumanCannonball(10, 90) #\n";
    }

    cout << "\nUNIT TEST COMPLETE\n\n";
}

template <typename X, typename A>
void btassert (A assertion)
{
    if (!assertion)
        throw X();
}
```