```cpp
/*
 * Programming Challenge 10
 */
#include <cassert>
#include <iomanip>
#include <iostream>
using namespace std;

/*
 * Class Converter.
 * A class that performs a variety of conversions.
 */
class Converter
{
    public:

        /*
         * Convert a temperature from metric to imperial units or from
         * imperial to metric units.
         * @param temp the temperature to be converted -- in degrees Fahrenheit
         *        or degrees Celsius
         * @param targetUnits a char representing the system to convert the
         *        temperature to -- 'M' to convert to metric, 'I' to convert
         *        to imperial
         * @return a float containing the converted temperature
         */
        float convertTemperature (float temp, char targetUnits);

        /*
         * Convert a distance from metric to imperial units or from
         * imperial to metric units.
         * @param distance the distance to be converted -- in feet or meters
         * @param targetUnits a char representing the system to convert the
         *        temperature to -- 'M' to convert to metric, 'I' to convert
         *        to imperial
         * @return a float containing the converted distance
         */
        float convertDistance (float distance, char targetUnits);

    private:

        /* Called by convertTemperature if target units is 'M' */
        float fahrenheitToCelsius (float temp);

        /* Called by convertTemperature if target units is 'I' */
        float celsiusToFahrenheit (float temp);

        /* Called by convertDistance if target units is 'M' */
        float feetToMeters (float distance);

```

```
        /* Called by convertDistance if target units is 'M' */
        float metersToFeet (float distance);
};

/* 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;
}

// CODE HERE -- FUNCTION DEFINITIONS

float Converter::convertTemperature (float temp, char targetUnits)
{
    if( targetUnits == 'M' )
        return fahrenheitToCelsius( temp );
    else if( targetUnits == 'I' )
        return celsiusToFahrenheit( temp );
}

float Converter::convertDistance (float distance, char targetUnits)
{
    if( targetUnits == 'M' )
        return feetToMeters( distance );
    else if( targetUnits == 'I' )
        return metersToFeet( distance );
}

float Converter::fahrenheitToCelsius (float temp)
{
    return (5.0/9.0 * (temp - 32));
}

float Converter::celsiusToFahrenheit (float temp)
{
    return ((9.0/5.0 * temp) + 32);
}

float Converter::feetToMeters (float distance)
{
    return (0.3048 * distance );
}

float Converter::metersToFeet (float distance)
```

```
{
    return ((1/0.3048) * distance);
}

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

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

    Converter c;

    try {
        btassert<bool>(static_cast<int>(c.convertTemperature(32, 'M')) == 0);
        cout << "Passed TEST 1: convertTemperature(32, 'M')\n";
    } catch (bool b) {
        cout << "# FAILED TEST 1 convertTemperature(32, 'M') #\n";
    }

    try {
        btassert<bool>(static_cast<int>(c.convertTemperature(212, 'M')) == 100);
        cout << "Passed TEST 2: convertTemperature(212, 'M')\n";
    } catch (bool b) {
        cout << "# FAILED TEST 2 convertTemperature(212, 'M') #\n";
    }

    try {
        btassert<bool>(static_cast<int>(c.convertTemperature(98.6, 'M')) == 37);
        cout << "Passed TEST 3: convertTemperature(98.6, 'M')\n";
    } catch (bool b) {
        cout << "# FAILED TEST 3 convertTemperature(98.6, 'M') #\n";
    }

    try {
        btassert<bool>(static_cast<int>(c.convertTemperature(0, 'I')) == 32);
        cout << "Passed TEST 4: convertTemperature(0, 'I')\n";
    } catch (bool b) {
        cout << "# FAILED TEST 4 convertTemperature(0, 'I') #\n";
    }

    try {
        btassert<bool>(static_cast<int>(c.convertTemperature(100, 'I')) == 212);
        cout << "Passed TEST 5: convertTemperature(100, 'I')\n";
    } catch (bool b) {
        cout << "# FAILED TEST 5 convertTemperature(100, 'I') #\n";
    }

    try {
```

```
        btassert<bool>(c.convertDistance(300, 'M') > 91.43f && c.convertDistance(300, 'M') < 91.45f);
        cout << "Passed TEST 6: convertDistance(300, 'M')\n";
    } catch (bool b) {
        cout << "# FAILED TEST 6 convertDistance(300, 'M') #\n";
    }

    try {
        btassert<bool>(c.convertDistance(5280, 'M') > 1609.3f && c.convertDistance(5280, 'M') < 1609.4f);
        cout << "Passed TEST 7: convertDistance(5280, 'M')\n";
    } catch (bool b) {
        cout << "# FAILED TEST 7 convertDistance(5280, 'M') #\n";
    }

    try {
        btassert<bool>(c.convertDistance(1, 'M') > .30f && c.convertDistance(1, 'M') < .31f);
        cout << "Passed TEST 8: convertDistance(1, 'M')\n";
    } catch (bool b) {
        cout << "# FAILED TEST 8 convertDistance(1, 'M') #\n";
    }

    try {
        btassert<bool>(c.convertDistance(1, 'I') > 3.28f && c.convertDistance(1, 'M') < 3.29f);
        cout << "Passed TEST 9: convertDistance(1, 'I')\n";
    } catch (bool b) {
        cout << "# FAILED TEST 9 convertDistance(1, 'I') #\n";
    }

    try {
        btassert<bool>(c.convertDistance(.5f, 'I') > 1.6f && c.convertDistance(.5f, 'M') < 1.7f);
        cout << "Passed TEST 10: convertDistance(.5f, 'I')\n";
    } catch (bool b) {
        cout << "# FAILED TEST 10 convertDistance(.5f, 'I') #\n";
    }

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

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