# C# Unit Test project

You’ve been hired by Crate Builder Inc (CBI) to finish a C# application that another programmer began but did not finish. Being negligent, the former employee also did not add unit testing!

Here is how it should run.

A screenshot of a cell phone

Description automatically generated

But here’s what it does so far.

A screenshot of a cell phone

Description automatically generated

See the next page for the explanation and the task list of things to do.

Then download and unzip one of the projects and begin. The source code is included at the end of this file so you can go to it and reference it while reading the document, and also included in the projects.

# Explanation:

A crate has a height, width, and depth. The user will be asked to enter these values. The program will then calculate the volume of the crate. The charge for the crate, and the price, and therefore the profit, are then calculated based on volume.

The ***volume*** of a crate = ***height \* width \* depth***. This should be implemented as the ***getVolume()*** method.

The ***cost*** to build the crate = **get*Volume() \* COST\_PER\_CUBIC\_FOOT***, which is a constant. This will be the ***getCost()*** method.

The ***charge*** to build the crate = ***getVolume() \* CHARGE\_PER\_CUBIC\_FOOT***, which is a constant. This will be the ***getCharge()*** method.

The ***profit*** is calculated = ***getCharge() – getCost()***. This will be the **getProfit()** method.

The programmer was worried about floating point rounding problems, so he made all the variables “decimal” rather than “double.” This won’t change your program any, and you could change them all to double if you wanted to.

# Tasks

*Examine the source code, and make it work as shown on the first page.*

Finish the input\_data method first, making it look like the sample shown on the first page.

Correct the getVolume function. Run the program to make sure it works.

Add a unit test for the getVolume function. See the next page for details.

Run the test to make the unit test works.

Correctly implement the getCost, getCharge, and getProfit methods.

After each method you implement, add a unit test for it

.

Finish the display\_output method. Make it look like the sample shown on the first page.

…

Once you have done this, you are pretty much finished. However, as a programmer you love to keep improving things, so there are two more things to try. (a) Go to the Main() method and tackle task 3, which requires adding a parameterized constructor. (b) Once that is done, modify the getVolume() test to use the new constructor.

// sample unit tests – for Windows

//

// To get you started, once you’re done implementing getVolume(), create your test

// framework, and make your first tests look like this. Copy-and-paste is fine.

[TestMethod()]

public void CrateTest()

{

// let's just make a crate object and

// make sure that doesn't crash.

Crate c = new Crate();

}

[TestMethod()]

public void input\_dataTest()

{

// Assert.Fail();

// this requires user input so we

// should completely delete this test.

}

[TestMethod()]

public void getVolumeTest()

{

Crate c = new Crate();

c.Length = 0;

c.Width = 0;

c.Height = 0;

decimal result = c.getVolume();

decimal target = 0;

Assert.AreEqual(target, result, "getVolume zero test failed");

c.Length = 1;

c.Width = 2;

c.Height = 3;

result = c.getVolume();

target = 6;

Assert.AreEqual(target, result, "getVolume failed");

}

[TestMethod()]

public void display\_outputTest()

{

// create a crate, and call

// display\_output, for a 1x1x1

// crate. As long as it hasn't crashed,

// it passes the test.

Crate c = new Crate();

c.display\_output();

c.Height = 1;

c.Width = 1;

c.Length = 1;

c.display\_output();

}

// sample unit tests – for Mac (nnunit)

//

// To get you started, once you’re done implementing getVolume(), create your test

// framework, and make your first tests look like this. Copy-and-paste is fine.

[SetUp]

public void Setup()

{

}

[Test]

public void CrateTest()

{

// let's just make a crate object.

// if it doesn't crash doing this,

// we've passed.

Crate c = new Crate();

Assert.Pass();

}

[Test]

public void getVolumeTest()

{

Crate c = new Crate();

c.Length = 0;

c.Width = 0;

c.Height = 0;

decimal result = c.getVolume();

decimal target = 0;

Assert.AreEqual(target, result, "getVolume zero test failed");

c.Length = 1;

c.Width = 2;

c.Height = 3;

result = c.getVolume();

target = 6;

Assert.AreEqual(target, result, "getVolume failed");

}

[Test]

public void display\_outputTest()

{

// Let's create a crate, and call

// display\_output for a 1x1x1

// crate.

//

// As long as it hasn't crashed,

// it passes the test.

Crate c = new Crate();

c.display\_output();

c.Height = 1;

c.Width = 1;

c.Length = 1;

c.display\_output();

Assert.Pass();

}

}

Appendix: Here is the source code for the app. However, you will probably want to unzip one of the projects and use that instead.

using System;

namespace CrateCalculator2

{

public class Crate

{

const decimal COST\_PER\_CUBIC\_FOOT = 0.23M;

const decimal CHARGE\_PER\_CUBIC\_FOOT = 0.5M;

// A crate has a Length, Width, and Height

// you are not allowed to add additional properties

// here (don't add volume, cost, charge or profit).

public decimal Length { get; set; }

public decimal Width { get; set; }

public decimal Height { get; set; }

public Crate() { } // Don't change me. If you're adding a new

// constructor, leave this default constructor

// alone.

public void input\_data()

{

Console.WriteLine("---------------------");

Console.WriteLine("CBI Crate builder Inc");

Console.WriteLine("---------------------");

this.Width = 0;

this.Length = 0;

this.Height = 0;

Console.WriteLine("Please enter all dimensions in feet.");

Console.Write("Enter width:");

this.Width = decimal.Parse(Console.ReadLine());

Console.WriteLine("??? Finish me!");

}

public decimal getVolume()

{

return this.Height \* this.Width \* this.Length;

}

//public decimal getCharge()

//{

// return this.getVolume() \* Crate.CHARGE\_PER\_CUBIC\_FOOT;

//}

//public decimal getCost()

//{

// return this.getVolume() \* Crate.COST\_PER\_CUBIC\_FOOT;

//}

//public decimal getProfit()

//{

// return this.getCharge() - this.getCost();

//}

public void display\_output()

{

decimal volume = 0;

decimal cost = 0, charge = 0, profit = 0;

volume = getVolume();

// cost = getCost();

// charge = getCharge();

// profit = getProfit();

Console.WriteLine("---------------------");

Console.WriteLine("CBI Crate builder Inc");

Console.WriteLine("---------------------");

Console.WriteLine("Crate purchase");

Console.Write("Crate dimensions in feet:");

Console.Write($"width = {Width}");

Console.Write($", length = {Length}");

Console.WriteLine($", height = {Height}");

Console.WriteLine($"Crate volume: {volume:0.00} cubic ft");

Console.WriteLine("??? Finish me!");

Console.WriteLine("---------------------");

}

};

class Program

{

static void Main(string[] args)

{

// part 1

//

// Fix the Crate class so that the input, calculations,

// and output work in the way shown in the assignment

// document.

Crate c1 = new Crate();

c1.input\_data();

c1.display\_output();

// part 2

//

// Uncomment the code below and

// make sure this code works.

//

// If part 1 is done, it should fine

// without further change.

//

// Crate c = new Crate();

// c.Height = 1;

// c.Width = 2;

// c.Length = 3;

//

// decimal v1 = c.getVolume();

// decimal p1 = c.getProfit();

// Console.WriteLine($"Crate volume = {v1} profit = {p1}");

// part 3

//

// Uncomment this code and change the Crate

// class to make it work. You'll have

// to add a parameterized constructor to

// the Crate class.

//

// Crate c2 = new Crate(1,2,3);

//

// decimal v2 = c.getVolume();

// decimal p2 = c.getProfit();

// Console.WriteLine($"Crate volume = {v2} profit = {p2}");

Console.WriteLine("(Done - press enter)");

Console.ReadLine();

}

}

}