# CSC 101 Program 2 Part B, Test Plan

This is a test plan for Programming Assignment 2, Part B. As with Program 1, this Program 2B test plan consists of *test cases*, each one of which runs the `containers` program with different inputs. For example, the 26th test case runs the program with the sample inputs shown in the Program 2, Part B writeup.

For Part B of this assignment, the program being tested must be named "`containers`". You make this program with the compiler command

```
gcc -ansi -pedantic -Wall -Werror containers.c -o containers
```

as described in the writeup.

There is an executable testing script named `run.csh`. This script implements the test plan defined in the table below. This is the script that will be run on your submitted program to compute your score.

It is strongly recommended that you run this script yourself, before submitting your program. To do so, you need a copy of the script itself, as well as a complete copy of both the `inputs` and `expected-output` testing directories. To obtain all of the testing files, run the following UNIX commands from the directory where you've stored your executable `containers` program:

```
cp -rp ~gfisher/classes/101/programs/1/testing .
cd testing
./run.csh
```

Note that you only need to run the "`cp -rp ...`" command one time, not every time you want to test your program. A two-part name is used for the testing files, corresponding to the two program inputs for each test case. For example, the file `input/r,4,8.5` has the inputs for the first test case. The same file name is used in all of the testing sub-directories:

- files in the `input` directory have the input values for the program
- files in the `expected-output` directory have the correct program results for those inputs
- files in the `output` directory have the actual program results produced by your program
- files in the `diffs` directory have any differences between the expected output and your actual output

Before your programs runs, both the `output` and `diffs` directories will be empty. After your program runs, the `output` directory will have one file for each test case run through your program.

If your `containers` program passes all of the test cases, the only terminal output you will see is the score, which will be "`100/100`" points. If one or more cases fail, the script will report the failure(s), and print the appropriate score at the end. The difference files in the `diffs` directory show the details of how your output differs from the expected output. The testing script uses the UNIX `diff` utility to compare expected and actual output files. An explanation of how to read UNIX `diff` files is available at `http://www.gnu.org/software/diffutils/manual/diffutils.html#Comparison`

The complete test plan has 25 test cases. Each test case is worth 4 points, for a total of 100 points possible.

The following table describes what each test case is for.

| **Test Case** | **Input: purchased,tendered** | **Correct Output** | **Description of Test Case** |
|---|---|---|---|
| 1 | r,4,8.5 | `expected-output/r,4,8.5` | First sample from the writeup. |
| 2 | d,3.25,9 | `expected-output/d,3.25,9` | Second sample from the writeup. |
| 3 | c,6.5,4.25 | `expected-output/c,6.5,4.25` | Second sample from the writeup. |
| 4 | r,10,20 | `expected-output/r,10,20` | Simple case for a rect. |
| 5 | d,10,20 | `expected-output/d,10,20` | Simple case for a dome. |
| 6 | c,10,20 | `expected-output/c,10,20` | Simple case for a cylinder. |
| 7 | r,4,8 | `expected-output/r,4,8` | Another simple case for a rect. |
| 8 | d,4,8 | `expected-output/d,4,8` | Another simple case for a dome. |
| 9 | c,4,8 | `expected-output/c,4,8` | Another simple case for a cylinder. |
| 10 | r,.08,.08 | `expected-output/r,.08,.08` | Minimum non-zero volume for rect. |
| 11 | d,.1,.1 | `expected-output/d,.1,.1` | Minimum non-zero volume for dome. |
| 12 | c,.09,.09 | `expected-output/c,.09,.09` | Minimum non-zero volue for a cylinder. |
| 13 | r,0,0 | `expected-output/r,0,0` | Non-positive inputs. |
| 14 | r,1,0 | `expected-output/r,1,0` | Non-positive inputs. |
| 15 | r,0,1 | `expected-output/r,0,1` | Non-positive inputs. |
| 16 | r,-1,-1 | `expected-output/r,-1,-1` | Non-positive inputs. |
| 17 | r,1,-1 | `expected-output/r,1,-1` | Non-positive inputs. |
| 18 | r,-1,1 | `expected-output/r,-1,1` | Non-positive inputs. |
| 19 | x,0,0 | `expected-output/x,0,0` | Unrecognized shape. |
| 20 | r,1,100 | `expected-output/r,1,100` | Really tall rect. |
| 21 | d,1,100 | `expected-output/d,1,100` | Really tall dome. |
| 22 | c,1,100 | `expected-output/c,1,100` | Really tall cylinder. |
| 23 | r,100,1 | `expected-output/r,100,1` | Really fat rect. |
| 24 | d,100,1 | `expected-output/d,100,1` | Really fat dome. |
| 25 | c,100,1 | `expected-output/c,100,1` | Really fat cylinder. |