

Drake Equation (drake)

Professor Caleb Fowler

June 11, 2019

Problem.

The Drake equation is named after Dr. Frank Drake. He was the first radio astronomer to look for intelligent life in the universe. He developed this equation to answer the question, "How many potential alien civilizations are there in the universe? ". You are going to write a program to calculate the Drake equation. You will declare and initialize the variables in the formula and display the output. Use the formula in figure 1 for your work. The N is the number of potential alien civilizations.

$$N = R \cdot p \cdot n \cdot f \cdot i \cdot c \cdot L$$

Drake Equation Factor Values		Estimated Values
Rate of star creation	R	7 [†]
Percentage of stars with planets	p	40%
Average number of planets that can potentially support life for each star with planets	n	(no consensus)
Percentage of those that go on to develop life	f	13%
Percentage of those that go on to intelligent develop life	i	(no consensus)
Percentage of those willing and able to communicate	c	(no consensus)
Expected lifetime of civilizations	L	(no consensus)
[†] Estimate of NASA and the European Space Agency		

The parentheses contains the short file name for this homework assignment. Use it for your source file name. This is the name I look for in my scripts.

The problem section contains a basic description of the problem and any additional contextual information.

Figure 1: The Drake equation with commonly accepted values. No consensus means you get to choose a value. You can use any value you wish.

Requirements.

- Come up with your own variable names. Do NOT use the single letter names from the formula - they are poor variable names.
- Program activities are split into logical 'chunks' or paragraphs.
- Hard-code the data into the program. You do not need to worry about prompting the client for additional input. Do not use cin (there will time for that later ...).
- Use white-space and comments to make your code more readable.

These are the requirements for the assignment. This means they are general and apply to the entire assignment, rather than one specific part. Not every assignment will have a requirements section.

Specification Bundles.

These are additional requirements for the assignment. You code the specifications from the various groups to control the maximum potential grade you can get for this assignment. The more work you do, the better grade you can get. Specifications are bundled into groups: "A", "B", "C", "D". You must meet the specifications of the lowest group before I will count the specifications for the highest group. For example, you must meet the "D" specifications before I will count the "C" specifications. If you miss one element of a specification bundle, that is the grade you will get for the assignment - regardless of how much extra work you do. **Note: sometimes you will get comments stacking on top of each other - that's OK even though you would never do that in the real world.**

You MUST follow the commenting convention below for the specifications. If they aren't commented, they DON'T count. To confirm I can find them, use the grep trick discussed below.

"D Grade" Specification Bundle.

If you do not complete any specifications or if you forget to add specification comments to your code, this is the grade I will give you. If you do the work - ALWAYS add the comment!

"C" Specification Bundle.

- ☐ // Specification C1 – Program Output
Program displays output of the Drake equation on the console.
- ☐ // Specification C2 - Variable Declaration
Create variables to hold your data, calculation, and output. Put this comment above them.
- ☐ // Specification C3 - Program Greeting
Include this section of your code as well (see the Style Guide below for more details).
- ☐ // Specification C4 - Source File Header
Program has a valid source file header (see Style Guide below).
- ☐ // Specification C5 - Variable Initialization
Initialize your variables to appropriate values as well as initialize them. If you declared and initialized in the same step, but the specification comment C5 on the line below the specification comment C2. Specification comments should always have a line to themselves.

You can just copy and paste the comment into your code. Whenever you see a specification comment, the grep trick works. Check off the boxes when you complete writing the code for that specification - helps keep track of your work.

"B" Specification Bundle.

- ☐ // Specification B1 - Calculation
Put this comment above the code which actually performs the calculation. Do not put the calculation in your cout statement(s).
- ☐ // Specification B2 - Double Output

Display on the console your calculation results doubled. Don't forget to label it so we know what we are looking at.

- // Specification B3 - Double Output Headings
Put a heading above the output you generated for Specification B2. The heading should look something like:

```
{ blank line }
OUTPUT DOUBLED
=====
{ Your output goes here. }
```
- // Specification B4 - Half Output Display on the console your calculation results reduced by half. B2 and B4 are called sensitivity analysis.
- // Specification B5 - Half Output Headings
Put a heading above the output you generated for Specification B4. The heading should look something like:

```
{ blank line }
OUTPUT HALVED
=====
{ Your output goes here. }
```

"A" Specification Bundle.

- // Specification A1 – Constant Variables
Use the proper syntax as well as the `const` keyword to create appropriate constant variables.
- // Specification A2 – Display variables
Display all the variable names and their values below the results of the Drake equation on the console.
- // Specification A3 - Variable Name Heading
On the lines above the output for Specification A2, display a title which looks like this:

```
{ blank line }
VARIABLE NAMES AND VALUES
=====
{ Your output goes here. }
```
- // Specification A4 - Program Greeting Heading
Put a heading on your program greeting which looks something like this:

```
{ blank line }
PROGRAM GREETING
=====
{ Your output goes here. }
```
- // Specification A5 - Drake Equation Heading
Add a heading to your equation's output which looks like this:

```
{ blank line }
```

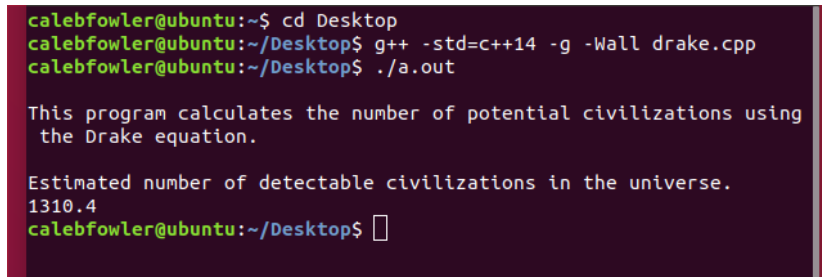
```

CHANCE OF INTELLIGENT LIFE
=====
{ Your output goes here.  }

```

Sample Run.

Use the same data to check your answers with mine. Figure 2 shows a sample "C" quality run with sample data. Figure 3 shows an "A" quality run with sample data. Don't worry if the numbers don't match exactly - many times they won't. However, the numbers should be very close.



```

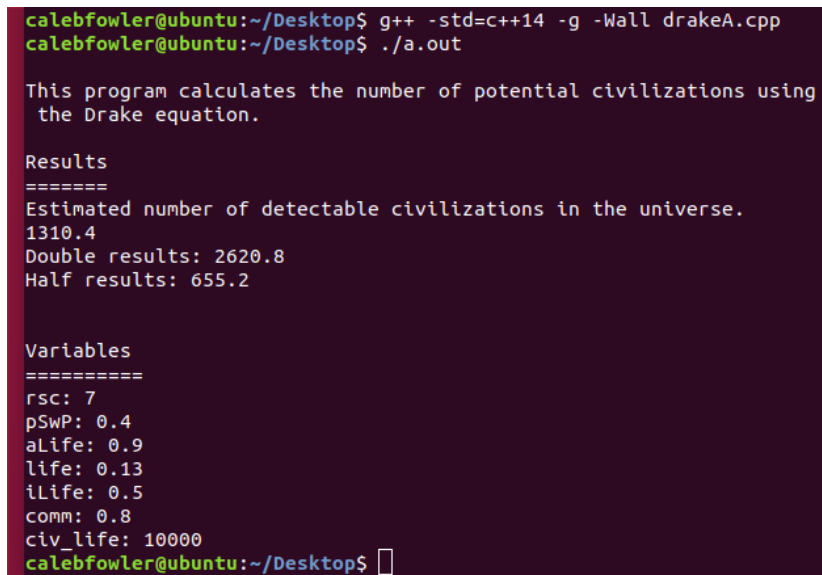
calebfowler@ubuntu:~$ cd Desktop
calebfowler@ubuntu:~/Desktop$ g++ -std=c++14 -g -Wall drake.cpp
calebfowler@ubuntu:~/Desktop$ ./a.out

This program calculates the number of potential civilizations using
the Drake equation.

Estimated number of detectable civilizations in the universe.
1310.4
calebfowler@ubuntu:~/Desktop$ 

```

Figure 2: "C" quality program run.



```

calebfowler@ubuntu:~/Desktop$ g++ -std=c++14 -g -Wall drakeA.cpp
calebfowler@ubuntu:~/Desktop$ ./a.out

This program calculates the number of potential civilizations using
the Drake equation.

Results
=====
Estimated number of detectable civilizations in the universe.
1310.4
Double results: 2620.8
Half results: 655.2

Variables
=====
rsc: 7
pSwP: 0.4
aLife: 0.9
life: 0.13
iLife: 0.5
comm: 0.8
civ_life: 10000
calebfowler@ubuntu:~/Desktop$ 

```

Figure 3: "A" quality program run.

Finally, figure 4 shows the grep trick applied to the "A" quality homework. You can see how the specifications all stand out. It makes it easy to see if you covered all of them.

Homework Checklist.

Check the following before you turn in your work:

```

calebfowler@ubuntu:~/Desktop$ grep -i 'specification' drakeA.cpp
// Specification C4 - Source File Header
{
    // Specification C2 - Variable Declaration
    // Specification C3 - Program Greeting
    // Specification B1 - Calculation
    // Specification C1 - Program Output
    // Specification A1 - Output Headers
    // Specification B2 - Double Output
    // Specification B3 - Half Output
    // Specification A2 - Display Variables
calebfowler@ubuntu:~/Desktop$

```

Figure 4: The grep trick in action.

- ☐ You coded your homework.
- ☐ Does it meet all the requirements?
- ☐ Test your code.
 - ☐ Does it compile?
 - ☐ Does it have any compiler warnings?
 - ☐ Does it run?
 - ☐ Does it produce correct output?
 - ☐ Did you use the grep trick to make sure I can see your work?
- ☐ Upload to Canvas.
- ☐ What's the plagiarism checker score?

Due Date.

This assignment is due by 11:59 PM on **Sunday** on the date on the calendar in Canvas. Example, if this assignment appears on the Canvas calendar during week 2, the assignment will be due that Sunday at 11:59 PM. All the assignments are open the first day of class and you can begin working on them immediately. I encourage you to start sooner rather than later with your homework, these always seem to take longer than you think.

Late Work.

If you miss the due date and time specified above, your work is late. Canvas records the date and time your homework upload COMPLETES. Late work is still acceptable, but it suffers a -15% penalty. You may turn late work in up until MONDAY 11:59 PM AFTER THE ASSIGNMENT WAS DUE. That is, you have 1 day to turn your work in - after that the Canvas drop box closes. Once Canvas closes I will not accept an assignment. Do not email your homework files to me; I will not accept them. Keep in mind the time Canvas uses to record

your submission - build 5 - 10 minutes into your estimates to upload the file!

Pro-Tip: Get a bare bones copy of your code running and turn it in¹. Then go ahead and modify it, fix it and whatnot. Upload it with the same name when you finish. That way, if something unexpected happens, you have some working code turned in. Risk management, class, risk management.

¹ If you really want to go pro, get some sort of version control system running (like Git).

How to Turn in your Homework.

Turn homework in by uploading to the appropriate Canvas Dropbox folder. Save your homework as a .cpp file. Don't zip or otherwise compress your files. Create a file with the following naming format: Short_file_name.cpp. Do NOT split your file up into multiple files. I know that is a standard industry practice, but it just gets in the way for this class.

I ONLY accept homework through the Canvas Dropbox. Do not add it to the submission comments or email it to me - I will not accept it. If you are having trouble submitting the assignment, email me immediately. Make sure you upload it a few minutes before the assignment closes in Canvas. If you go over by just one second - you are late.

Style Guide

~~All programs you write MUST have the following code and/or comments.~~ Again, I look for these elements with my scripts, you want me to find them.

Comments.

Use white space and comments to make your code more readable. I run a program called cloc (count lines of code) which actually looks for this stuff.

End of line comments are only permitted with variable declarations. Full line comments are used everywhere else.

Specification Comments.

Specifications are bundled into groups: "A", "B", "C", "D". You must meet the specifications of the lowest group before I will count the specifications for the highest group. For example, you must meet the "D" specifications before I will count the "C" specifications. If you miss one element of a specification bundle, that is the grade you will get for the assignment - regardless of how much extra work you do.

Use whole line comments for Specifications. Put the comment on the line above the start of the code implementing the Specification. If the same Specification code appears in more than 1 place, only comment the first place that Specification code appears. Number your Specifications according to the specification bundle and the specific specification you are using, also provide a very short description. DO NOT BUNCH ALL YOUR SPECIFICATIONS AT THE TOP OF THE SOURCE FILE. Example specification comment:

```
// Specification A2 - Display variables
Your code to do this starts here;
```

It's very important to get the specifications down correctly. **If your specification code isn't commented, it doesn't count.** I use the grep trick to find your specification code. Proper documentation is part of the solution, just like actually coding the solution is.

Compiler Warnings.

Compiler warnings are a potential problem. They are not tolerated in the production environment. In CISP 360 you can have them. I will deduct a small number of points. CISP 400 - I will deduct lots of points if compiler warnings appear. Make sure you compile with -Wall option. This is how you spot them.

C++ Libraries.

We are coding in C++, not C. Therefore, you must use the C++ libraries. The only time you can use the C libraries is if they haven't been ported to C++ (very, very rare).

Non-Standard Language Extensions.

Some compilers support unapproved extensions to the C++ syntax. These extensions are **unacceptable**. Unsupported extensions are compiler specific and non-portable. Do not use them in your programs.

Program Greeting.

Display a program greeting as soon as the program runs. This is a simple description of what the program does. Example:

```
// Program Greeting
cout << "Simple program description text here." << endl;
```

Source File Header.

Start your source file with a program header. This includes the program name, your name, date and this class. I use the grep trick for .cpp (see below) to look for this. I focus on that homework name and display the next 3 lines. Example:

```
// drake.cpp
// Pat Jones, CISP 413
// 12/34/56
```

Specifications and Specification Bundles.

You document specifications like this: // Specification C1 - Some stuff

You do not need to code them in order. You will probably want to because the specifications get harder as you move up in bundles (not THAT much harder). You also don't need to worry about the specification comments appearing in order in your code, either.

However, all of a specification bundle must be coded to reach that bundle grade (ie all C bundle to get a C). Partially completed bundles DO NOT COUNT. Say you code all specifications for a B bundle and only 1 for an A bundle (out of 5 for example). The highest grade you would get would be a B because that's the last bundle you've completed.

You can stop at any bundle you want, you just can't get a higher grade (ex, you code all specifications for bundle B - the best you can get for this homework is a B). This is designed to mirror the work word, the more features your code has, usually, the happier your clients are. This also gives you some control over your grade.

This style guide has more information on the specifics of these comments.

Variables.

Constant variables - anytime you have a value which is not supposed to change, that's a constant. We make it read only with the const keyword and signify it with the ALL CAPS style: const PI = 3.14; We prefer using constants because they make the code easier to read. There are a few situations where we do not usually use them, such as starting a loop at zero. However, if we have that loop end at, say, 33, then it's a magic number. What's 33? Who knows? If we use const SIZE = 33; we know what 33 is.

When we have numeric literals appearing in the program we call these magic numbers. We don't know what they are, but if we change them, the program breaks. hence, magic. Magic numbers are generally frowned upon.

Grep Trick.

Always run your code immediately before you turn it in. I can't tell you how many times students make 'one small change' and turn in broken code. It kills me whenever I see this. Don't kill me.

You can check to see if I will find your specification and feature comments by executing the following command from the command line. If you see your comments on the terminal, then I will see them. If not, I will NOT see them and you will NOT get credit for them. The following will check to see if you have commented your specifications:

```
grep -i 'specification' homework.cpp
```

This will generate the following output. Notice the specifications are numbered to match the specification number in the assignment. This is what I would expect to see for a 'C' Drake assignment. Note the cd Desktop changes the file location to the desktop - which is where the source file is located.

```
calebfowler@ubuntu:~$ cd Desktop
calebfowler@ubuntu:~/Desktop$ grep -i 'specification' cDrake.cpp
// Specification C2 - Declare Variables
// Specification C3 - Separate calculation
// Specification C1 - Program Output
calebfowler@ubuntu:~/Desktop$
```

This is what I would expect to see for an 'A' level Drake assignment.

```
calebfowler@ubuntu:~/Desktop$ grep -i 'specification' aDrake.cpp
{ // Specification C2 - Declare Variables
  // Specification C3 - Separate calculation
  // Specification B1 - Calculation
  // Specification C1 - Program Output
  // Specification B 2 - double and half
  // Specification A1 - Output Headers
  // Specification A2 - Display variables
calebfowler@ubuntu:~/Desktop$
```

We can also look at the line(s) after the grep statement. I do this to pay attention to code segments.

```
grep -i -C 1 'specification' aDrake.cpp
```

```

calebfowler@ubuntu:~/Desktop$ grep -i -C 1 'specification' aDrake.cpp
int main()
{ // Specification C2 - Declare Variables
  int r_starcreation = 7; // rate of star creation
  --

  // Specification C3 - Separate calculation
  float drake = 0; // initialize to 0
  // Specification B1 - Calculation
  drake = r_starcreation * perc_starswithplanets * ave_numberofplanetslife *
  perc_devlife * perc_devintlife * perc_comm * exp_lifetime;

  // Specification C1 - Program Output
  cout << "The estimated number of potential alien civilizations in the univer
se is ";
  --

  // Specification B 2 - double and half
  cout << "Half this value: " << drake * .5 << endl;
  --

  // Specification A1 - Output Headers
  cout << endl;
  --

  // Specification A2 - Display variables
  cout << "Variables:" << endl;
calebfowler@ubuntu:~/Desktop$

```

We can also use this to look for other sections of your code. The `grep` command searches for anything withing the single quotes "", and the `-i` option makes it case insensitive. This is how I will look for your program greeting:

```

calebfowler@ubuntu:~/Desktop$ grep -i -C 1 'greeting' aDrake.cpp
// Program Greeting
cout << "This program calculates and displays the number of potential";
calebfowler@ubuntu:~/Desktop$

```

The `grep` trick is extremely powerful. Use it often, especially right before you turn in your code. This is the best way I can think of for you to be sure you met all the requirements of the assignment.

Client System.

Your code must compile and run on the client's system. That will be Ubuntu Desktop Linux, version 18.04. Remember, `sourcefile.cpp` is YOUR program's name. I will type the following command to compile your code:

```
g++ -std=c++14 -g -Wall sourcefile.cpp
```

If you do not follow this standard it is likely I will detect errors you miss - and grade accordingly. If you choose to develop on another system there is a high likelihood your program will **fail to compile**. You have been warned.

Using the Work of Others.

This is an individual assignment, you may use the Internet and your text to research it, but I expect you to work alone. You **may** discuss code and the assignment. Copying code from someone else and turning it in as your own is plagiarism. I also consider isomorphic

homework to be plagiarism. You are ultimately responsible for your homework, regardless of who may have helped you on it.

Canvas has a built in plagiarism detector. You should strive to generate a green color box. If you submit it and the score is too high, delete it, change your code and resubmit. You are still subject to the due date, however. This does not apply if I have already graded your homework.

Often, you will not be able to change the code to lower the score. In this case, include as a comment with your homework, what you did and why you thought it was ineffective in lowering your score. This shows me something very important - you are paying attention to what you are doing and you are mindful of your plagiarism score.

ProTip: Get a bare bones copy of your code running and turn it in. Then go ahead and modify it with bonuses and whatnot. Upload it with the same name so it replaces your previous homework. This way, if something comes up or you can't finish your homework for some reason, you still have something turned in. A "C" is better than a zero. Risk management class, risk management.