Cat, Cot, Cap Temperature Calculator (ccctc)

Professor Caleb Fowler

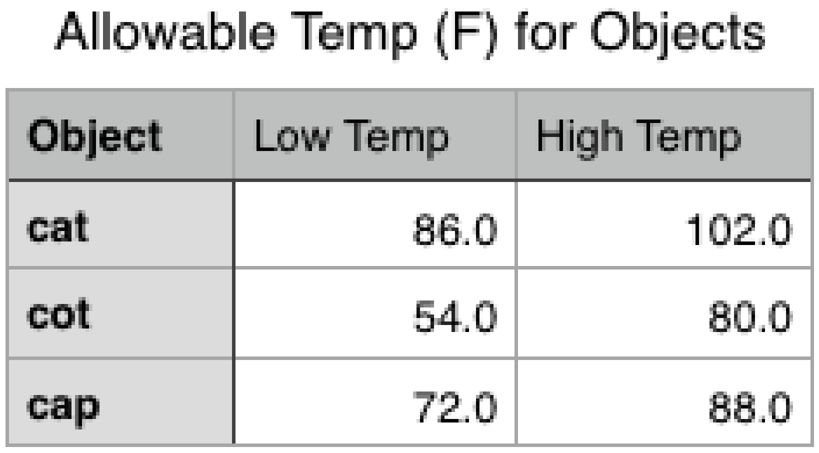
June 7, 2020

Problem.

Your friend has asked you to help them with a project. He needs to convert a Fahrenheit temperature to either Celsius, Kelvin or Rankine temperature scales. However, your friend is a bit - odd. He needs this to apply to only three objects: a cat, a cot and a cap. Each object has a valid temperature range for his calculations.

Requirements.

- Prompt for the object to calculate the temperature for. Note: This is NOT a menu! The client is expected to enter the proper words.
- Ask for the temperature of the object in Fahrenheit. Check to see if it's in the proper range.
- Create a menu of valid temperature scales to select from. Selecting the scale option will immediately display the converted temperature. Include an option to calculate the temperature for ALL the options.
- Re-prompt if incorrect data was entered. Keep re-prompting until the client get's it right.
- Don't forget to incorporate these elements from the style guide:
  - Comments
  - Source file header.
  - Program Greeting - this is NOT the same as the menu!
  - Constant variable style if applicable.
  - No global variables or global variable look a-likes.



Allowable Temp (F) for Objects

Object	Low Temp	High Temp
cat	86.0	102.0
cot	54.0	80.0
cap	72.0	88.0

Valid temperature ranges for the 3 items.

Specification Bundles.

Code elements from the specification bundles to control the maximum potential grade you can get for this assignment. The more work you do, the better grade you can get. This is the starting point for your grade.

"C Grade" Specification Bundle.

- ☐ // Specification C1 – Only Valid Words
  We only want to accept valid words cat, cap, cot no matter how they are spelled (ie Cat, CAT, caT all Ok too).
- ☐ // Specification C2 – Temp Scale Menu
  This menu shows us what temperatures we can convert to. This can be a numeric menu.
- ☐ // Specification C3 – Valid Scale Menu Check
  Only allow valid menu options to be entered. Give the client feedback if it's wrong and re-prompt.

"B Grade" Specification Bundle.

- ☐ // Specification B1 – Floats for Temps
  Allow the client to enter a decimal temperature.
- ☐ // Specification B2 – Valid Temps Only
  Make sure the user can input the temperatures only within a valid range. If out of range, tell them (too high, too low) and re-prompt. Temperatures must be valid floats with in the proper range. We can accept an error of less than 0.2 degrees. Example: a valid temperature for a cat will be 102.2, but 102.3 will not.
- ☐ // Specification B3 - Auto Variable Type
  Use the `auto` variable type at least once in the program.

"A Grade" Specification Bundle.

- ☐ // Specification A1 – Sophisticated Word Check
  Your validation check can only test against 3 lowercase words: cat, cot, cop. Yet, somehow, you must still address the issue of our client entering any combination of upper and lower case letters. See discussion below for more information.
- ☐ // Specification A2 – Another Thing
  Add another thing (dog as well as cat, etc etc). Modify the program to process it correctly.

- ☐ // Specification A3 – Another Temperature Scale
  Add another temperature scale. You will be amazed at how many other scales there are out there.

Homework Checklist.

Check the following before you turn in your work:

- ☐ 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?

Sample Run.

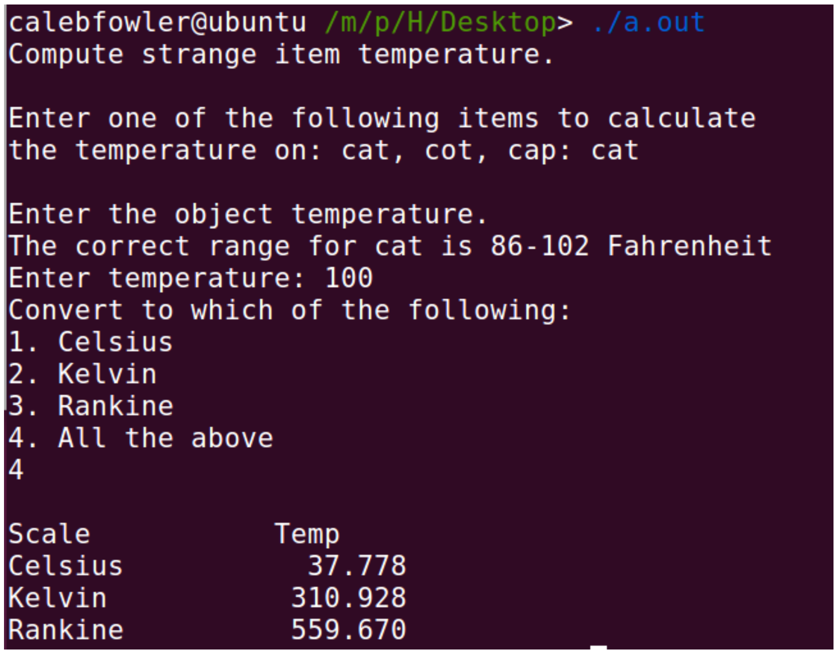
Use the same data to check your answers with mine. Figure 1 shows a sample run with sample data.

Discussion.

You will need to research the formula's to convert Fahrenheit temperatures to Celsius, Kelvin and Rankine scales. Wikipedia is one easy to use source.

It's a huge pain to check all the ways we could spell the input words. Student's commonly enumerate every single possible spelling in one gigantic compound if statement. That's the brute force approach. It works but it's difficult to maintain. There is a more sophisticated approach. One way to simplify it is to convert all input strings to lower case, then test against the lower case word only. This is called a canonical form of input and it will make your life easier.

Another thing you can do to make your life easier is to look for patterns in the input. In this case, all the valid input words begin with 'c'. So one check we could make is look at the first letter of the input string. If it's not a 'c' then we know the word can't possibly be correct. Google extracting characters from strings if you aren't sure how to do this.



```
calebfowler@ubuntu /m/p/H/Desktop> ./a.out
Compute strange item temperature.

Enter one of the following items to calculate
the temperature on: cat, cot, cap: cat

Enter the object temperature.
The correct range for cat is 86-102 Fahrenheit
Enter temperature: 100
Convert to which of the following:
1. Celsius
2. Kelvin
3. Rankine
4. All the above
4

Scale           Temp
Celsius           37.778
Kelvin           310.928
Rankine          559.670
```

Figure 1: Sample program run.

Due Date

This assignment is due by 11:59 PM on Sunday on the date on the calendar in Canvas. 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 1 letter grade 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.

Pro-Tip: Get a bare bones copy of your code running and turn it in. 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]

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

How to Turn in your Homework

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.

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. Do NOT split your file up into multiple files. I know that is a standard industry practice, but it just get's in the way for this class.

Create a file with the following naming format: W12345678.cpp (your w number). This allows me to sort the class in alphabetical order - don't stand out here! If you are having trouble submitting the assignment, email me immediately. Don't change your filename if you make multiple submissions - Canvas will keep track of them and download the latest one by default.

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". 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 "B" specifications before I will count the "A" 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.

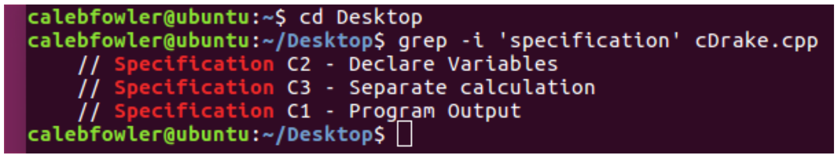
Always run your code immediately before your turn it in. I can't tell you how many times students make 'one small change' and turn in broken code. It kills me whenever I see this. Don't kill me.

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
```

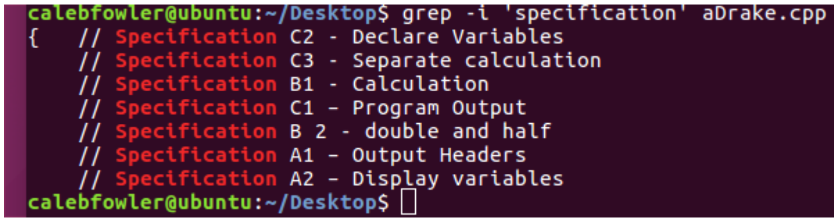
This will generate the following output. Notice the specifications are numbered to match the specification number in the assignment.

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$ 
```

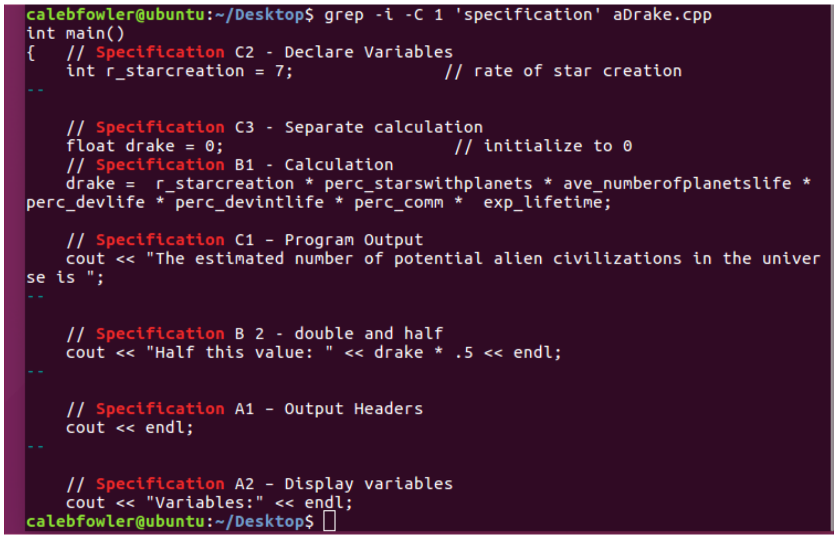
This is what I would expect to see for an ′A′ level Drake assignment.



```
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$ 
```

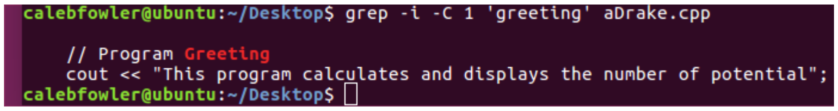
We can also look at the line(s) after the grep statement. I do this to pay attention to code segments.

```
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.

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.

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.