TODO List

Professor Caleb Fowler

Problem.

Create a command line TODO list program. Prompt your client. The first character/symbol they enter will be the command. Follow that with a space. Finally, enter the todo list item. Example: "+ Study for Final" (don't enter the quotes). This causes `Study for Final` to be entered into the TODO list with today's date. You will need to have the following fields at a minimum: TODO itself, date added, and TODO Identification number, but you can add as many other fields as you wish.

Requirements.

- No C libraries (no .h).
- No global variables or analogues of global variables.
- No standard template library anything.
- Create a alphabetic menu to handle program options.
- Only allow valid letters or symbols as input for this menu. Upper and lower case if appropriate. Re-prompt in event of error. Can you do this without displaying an actual menu?
- Style guide elements apply: comments, layout, Program Greeting, Source File Header, and variables etc. etc. In your program greeting function, be sure to display the current date. Get this from the system hardware.
- Create at least 1 object.
- All objects must have a component testing method which runs at least 2 diagnostic tests on that object. Include this in main() right after the program greeting.

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. Start with bundle "C" and work your way up to "A".

"C" Specification Bundle.

1. // Specification C1 - Overload «
   Overload the stream insertion operator to output a TODO.
2. //Specification C2 - Overload »
   Overload thestream extraction operator to input a TODO.
3. // Specification C3 - Test TODO's
   Generate at least 5 TODO's in your component testing method.

4. //Specification C4 - TODO array
   Put your TODO's in a dynamic array of TODO's

"B" Specification Bundle.

1. // Specification B1 - + Symbol
   Allow the user to enter tasks with a "+" symbol.
2. // Specification B2 - ? Symbol
   Allow the user to display all tasks with a ? symbol.
3. // Specification B3 - - symbol
   Allow the user to remove a task with a "-" symbol (use an ID number to remove the TODO). This doesn't necessary mean you need to delete it immediately.
4. Specification B4 - Persistence
   Have your TODO list survive program termination by dumping the TODO's to disk when program ends. Load the data when the program first runs - if the file exists.

"A" Specification Bundle.

1. // Specification A1 - Overload Copy Constructor
   Overload the default copy constructor to handle your TODO's. Do this even if you don't have any pointers in your TODO object/struct. This is a great method to put in your component testing method.
2. // Specification A2 - Overload Assignment Operator
   To handle TODO assignment.
3. // Specification A3 - System Date.
   Pull the date for your TODO record directly from the system date method.
4. // Specification A4 - Overload Constructor
   Allow empty input for an add from the menu. However, empty adds trigger the regular constructor which creates a test record filled with obviously dummy data. When the client actually enters data during an add, trigger an overloaded constructed filling it with the data supplied by the client.

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?

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 in [1] . 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 get's in the way for this class.

I ONLY accept homework through the Canvas Dropbox. Do not add it to the comments or email 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 conform to the following style specifications.

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.

Comment Rate.

I use a program called cloc to calculate the total number of blank lines, total number of comments and the total number of lines in your program. When you divide the number of comments by the number of lines you get the comment rate. This number gives me a rough approximation of how well commented your code is. When I see a rate of 10% (for example) I have a good idea of what I will find when I look at the program - pretty much no comments at all. Ditto, when I see a rate of 45% (for example). This code will have comments for many tricky or complex sections as well as functions.

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).

Functions.

Functions are used to segment your code into easier to work with chunks. You want your functions to do only one thing - one activity. Functions are coded BELOW main(), not above it. Use arguments and parameters to pass information to your functions - global variables are discouraged with prejudice . Whenever possible, try to have a brief comment below the function signature describing what the function does.

```
void foo()
// a meaningless function to show a function comment in
the style guide.
```

Function Prototypes.

Use function prototypes and comment them. This is a constraint in this class even if not expressly stated in the homework. I use the grep trick for Function Prototype to look for this. You only need to comment it once, at the top of your source file - above main(). Example:

```
// Function Prototype
void ProgramGreeting();
```

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 description of what the program does. Example:

```
// Function Greeting
cout « "Simple program description text here." « endl;
```

You can make this much more elaborate - usually used as cover so clients don't realize we are loading huge data files. If your assignment calls for a menu, DO NOT put the menu in here - that goes in it's own section.

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:

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

Space Rate.

I use a program called cloc to calculate the total number of blank lines, total number of comments and the total number of lines in your program. When you divide the number of blank lines by the total number of lines you get the space rate. This number gives me a rough approximation of how well laid out your code is. When I see a rate of 10% (for example) I have a good idea of what I will find when I look at the program - wall of words. Ditto, when I see a rate of 45% (for example). This code will have good spacing between major blocks of code and functions. Note: when this number get's too high (like 70% of so) the blank space becomes a distraction.

Variables.

Document constants with ALLCAPS variables and the const keyword. Magic numbers are generally frowned upon.

~~*Grep Trick.*~~

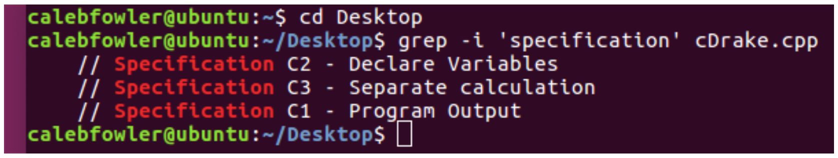
Always run your code immediately before your turn it in. I can't tell you how many times students make 'one small change' and turn in broken code. It kills me whenever I see this. Don't kill me.

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:

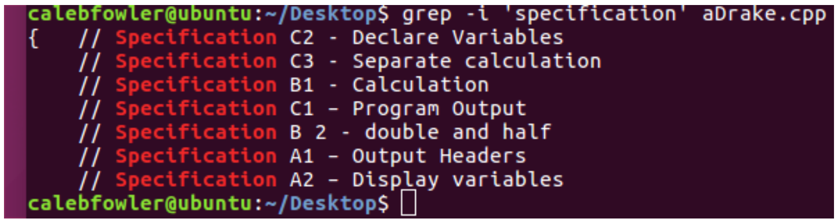
```
grep -i 'specification' homework.cpp
```

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$ 
```

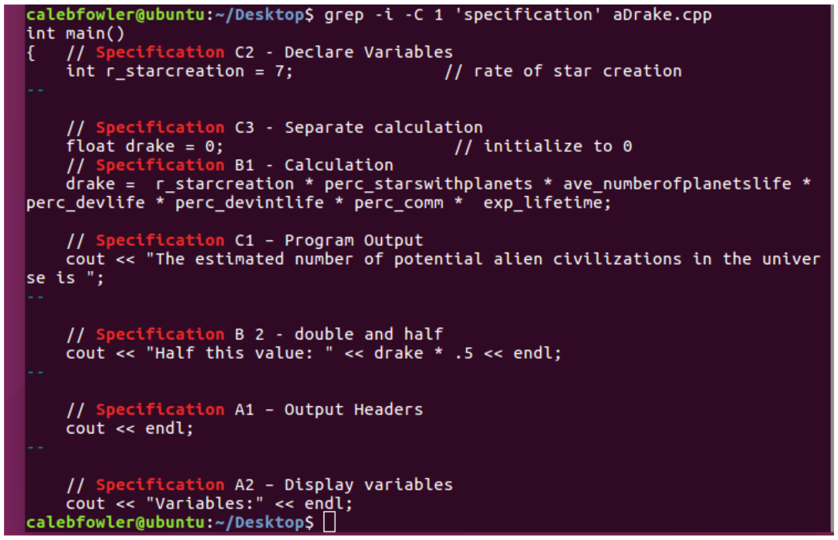
This is what I would expect to see for an 'A' level Drake assignment.



```
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$ 
```

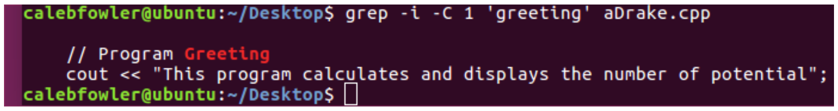
We can also look at the line(s) after the grep statement. I do this to pay attention to code segments.

```
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

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.

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.