How to Write a Program that writes a Program

August 26, 2020

Dr. Thomas B. Kinsman, Ph.D.

Someone asked for help thinking about the next homework assignment. If you have never written a program that writes a program before, I can understand the confusion.

The gist of this is that you have to begin with the end in mind. You have to think about what the results are going to do, and work backwards.

Imagine for a moment that you just had to write a program, which always outputs the same program, every single time:

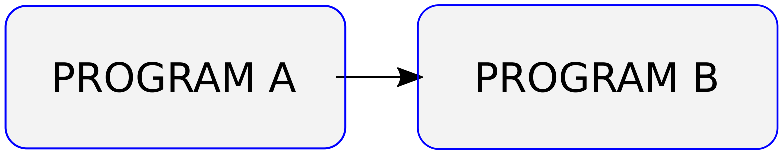

**Figure 1 -- When PROGRAM A runs, it creates PROGRAM B.
In this simple scenario, Program B is the same every time.**

This is no different then writing out data, or writing to a file. The only difference is that the output file, "ProgramB.py" has computer instructions in it, in Python (or whatever language you choose). In the C language, I would use filePointer = fopen("filename.py", "w").

I don't know anything about python. I'm a C/C++ and Matlab expert. However, I would guess that the mentor program file might look something like:

```
file_object = open( "MyFileName.py", "wt" )

# stuff happens.
file_object.write('# Line 1\n')
file_object.write('print("Created on August 26th, 2020 at 12:04 PM.\\n")\n')
file_object.write('# Line 3\n')
file_object.write('# Line 3\n')
file_object.write('# Line 4\n')

file_object.close()
```

Try it. Try something simple. See what errors you get, fix the errors. Then grow the program.

Actually, that program would need to add a routine in it to figure out the current time and date. Then it would print that time and date out to the file.

So, for this homework, the Mentor program needs to print some things in the file.

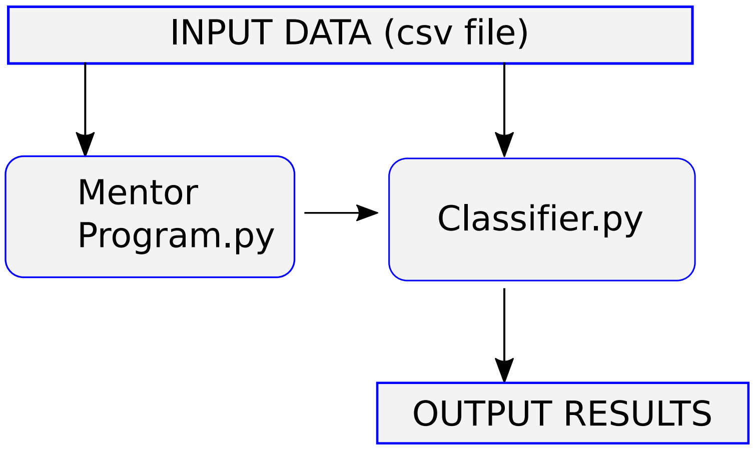

Figure 2 – Control Flow of homework 00, sort of.

To start with:

1. Imagine that you know what is constant about the program “Classifier.py”
2. Then get that fixed version working so that when you create the MentorProgram.py, it always creates the correct program in “classifier.py”.
3. Then add in the things that need to change.
   Change Mentor.py so that it inserts either the correct strings into the output, such as the date.

4. For our class, if a column of data contains nothing but spaces, it is an empty column and should be deleted.
5. The same with rows.

The final resulting program, “classifier.py”, or “decision_tree_program.py” (or whatever the specs say to call it) will have several parts to it:

A. It will have a section that opens the input file, and reads in the data to be classified.
   This part of your program might be called the “prologue”.

   This part of the code is constant. It is always the same.
   It is a fixed set of commands that are always identical.

B. A loop is used to read in each record of the input file and then classify it.

   a. Classification could be done with a call to a routine that you are going to have later in the program (the decision tree classifier subroutine or object method). If this part of the program is always the same, it could be part of the prologue too. If so, then the classifier is written at the end of the resulting program, after everything else.

   b. A simpler way to do classify the record is to have an if-then-else statement in the middle of the loop. This tests the attributes of each record, as determined by your decision tree creation algorithm, and sets the classification to return.

C. The program will then print out the results of the classification of each record.
   Printing might be to a file, or it might be to the standard output.

D. It will close the input file, and the last part of the program will close any open braces and loops.
   Notice that the previous part, and this part, do the same thing, regardless of how your decision tree works.
   So you can call these last two parts the “epilogue”.

I assign this homework here, so that you can learn how to figure these things out now. Otherwise when you get to the homework where you write a decision tree, you will not know how to do meta-programming, and might be completely lost. Last semester, everyone did fine.

So, what does your homework program do?

1. It will read in the training data.

2. It will analyze the training data, and form an internal structure (of your choice) that contains the decisions for the decision tree.

3. It will open up the output file, that will be your resulting decision tree program:
   ```
   output_file_handle =  file.open... ("HW_Kinsman_Resulting_Decision_tree.py"), or
   ```
   whatever the specifications say to call this program.

4. To this file it will print the prologue.
   ```
   emit_prologue( output_file_handle );
   ```
   This function just prints a chunk of program to a file. You have to know your syntax, and it takes a few iterations to get it right. The function is a large block of print statements.

5. Then you emit the body of the main loop, which does the classification.
   ```
   emit_body( output_file_handle, other_possible_parameters );
   ```
   This either prints the if-then-else decisions you need to make, or prints out calls to your function (which will be appended later on.)

6. Then you close the program, by printing the end of the main loop.
    ```
    emit_epilogue( output_file_handle );
    ```

7. If necessary, you add a function to the end of the file that defines the decision tree subroutine.
    ```
    emit_decision_tree( output_file_handle, other_possible_parameters );
    ```

    This will require several iterations to debug. The great thing about it is that once you have it debugged, you can write many other programs, based on the same idea, by having it make different decisions when building the form and shape of the decision tree.