Extensions

Introduction

I concentrated on two extensions:

- A WhiteSpace translator and combining tool.
- A hashing based password encoder.

WhiteSpace

This was my own personal take on the parody language *Whitespace*[1]*.* In WhiteSpace any typed words are counted as comments, and spaces, tabs, newlines and other whitespace symbols make up the wording of the program. My extended version of the parser now gives you some WhiteSpace orientated capabilities that I will demonstrate with an example.

Example:

If we have the file *pseudo_torus.txt* and we want to translate this into a WhiteSpace program we type at the terminal:

```
./turtle pseudo_torus.txt TRANSLATE
```

This will then output a file *reg_to_ws_translation.txt.* On opening that file it should look as if it is empty. However if you select the contents it should actually contain an assortment of spaces and tabs, separated into different lines. If we then run

```
./turtle reg_to_ws_translation.txt SECRET
```

Then this ‘secret’ blank message is decoded to draw the pseudo_torus picture. It also outputs a file *ws_to_reg_translation.txt* that can be used to see what the original WhiteSpace code was.

However, the point of the WhiteSpace language is that it can be combined with other languages to create source files that when compiled in one language can do one thing, whilst if they are compiled in another then they do another. By typing:

```
./turtle circle_square.txt reg_to_ws_translation.txt COMBINE
```

[1] http://en.wikipedia.org/wiki/Whitespace_%28programming_language%29

We can create one of these source files! Note that the order of arguments within this function is important. The regular LOGO code must go first, with the WhiteSpace code second in order for them to be combined effectively.

The output of this is a file called *Combined_Files.txt.* Now running

```
./turtle Combined_Files.txt
```

Gives the *circle_square.txt* output, whereas running

```
./turtle Combined_Files.txt SECRET
```

Gives the *pseudo_torus.txt* output!

I have enclosed a variety of shapes to try in my submission in the Working/Fun_Shapes_to_Play_With directory.

Hashing Based Password Encryption

WARNING: DO NOT FORGET YOUR PASSWORD!

The default password for the program is set to ‘1’. By entering: ./turtle and no extra arguments you can change it to any number between 1 and 1000 that you like. However, please try not to forget it as you might have to fiddle around with my code to bypass the password section!

The password system takes its inspiration from hashing and uses two files, *password_file_1.txt* and *password_file_2.txt* (which are purposefully cryptically named to prevent people figuring out how they work!). Your password then forms a parameter for a hashing key for a hashing table filled with values from *password_file_2.txt.* Using this hashing key, values from *password_file_1.txt* are mapped to the table and checked against the values that are already populating it. If they all match then the password is correct and you are let into the program.

To explain more clearly I will enclose a simplified example where the number of integers in the above files is not 1000, but is just 10.

File 1	0	0	0	1	0	0	0	0	0	0
File 2	0	0	0	0	0	0	0	1	0	0

Above we have mapped the two files to two separate vectors, and denoted which file each of the vectors has come from. The password for this encryption would then be 4. By moving the elements of File 1 four places up the two vectors match and the password is accepted.

Note that this has a wrap around property. The number that was originally at place 9 is now at place 0.

File 1	0	0	0	0	0	0	0	1	0	0
File 2	0	0	0	0	0	0	0	1	0	0