Kyle Durfee

VBA Final Project

**Executive Summary**

I built a workbook to handle my personal finances. This workbook allows the user to input account names, keep a general journal based on modified accounting principles, and build a budget. The macros post the different accounts to their own separate T-charts on individual worksheets, build an income statement by account by month, build a balance sheet, and set up a budget. Separate macros enable the user to make and save different budget scenarios based on individual input, or using exponential smoothing.

The system also does some basic data validation to make sure dates run chronologically, debits and credits add up, the accounts used are the accounts listed, etc. The macros also form the spreadsheets.

This tool makes it very easy to tell where your money is going, how much you’re worth, and identify places for improved spending habits. I made a less effective version before, but created this one to be much more flexible, dynamic, robust, and quick.

**Implementation documentation**

My project has 2 major parts:

* Building the ledgers and statements from the journal inputs
* Handling the budget.

***Building the Ledgers and Statements***

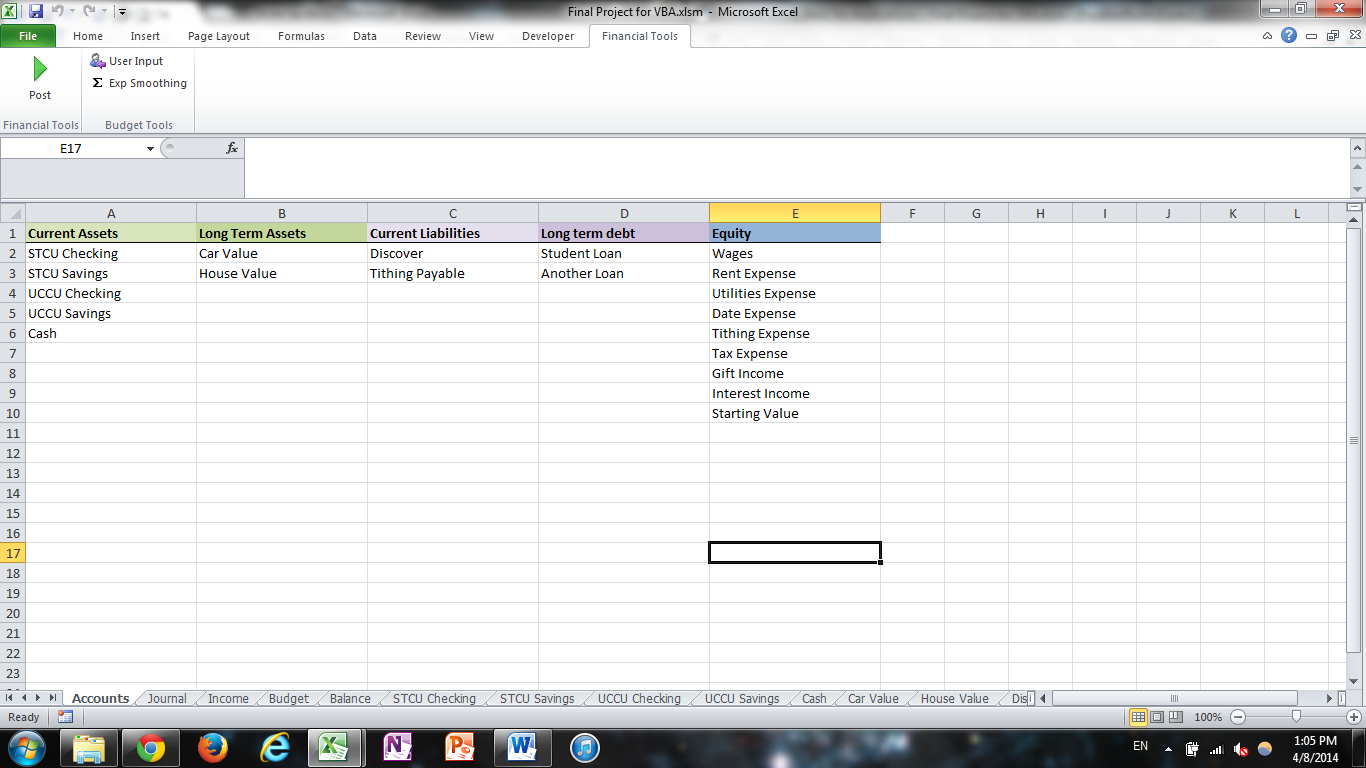
I have several subprocedures that run in succession. These procedures:

1. Put the different kinds of user-defined accounts into appropriate arrays
2. Make sure the data put into the general journal has been input correctly
3. Makes a worksheet for each different account and names the worksheets for the accounts
4. Posts the transactions into each account-specific worksheet
5. Makes a running total of the transactions to that point
6. Builds the structure and formatting for an income statement
7. Populates the income statement with the accounts and transactions for each account in each month
8. Builds the structure and formatting for a balance sheet
9. Populates the balance sheet with the appropriate accounts and values
10. Colors the income statement and general journal
11. Builds the structure and formatting for a budget
12. Populates the actual figures for the budget
13. Appropriately colors the budget sheet

*General Details on This Section*

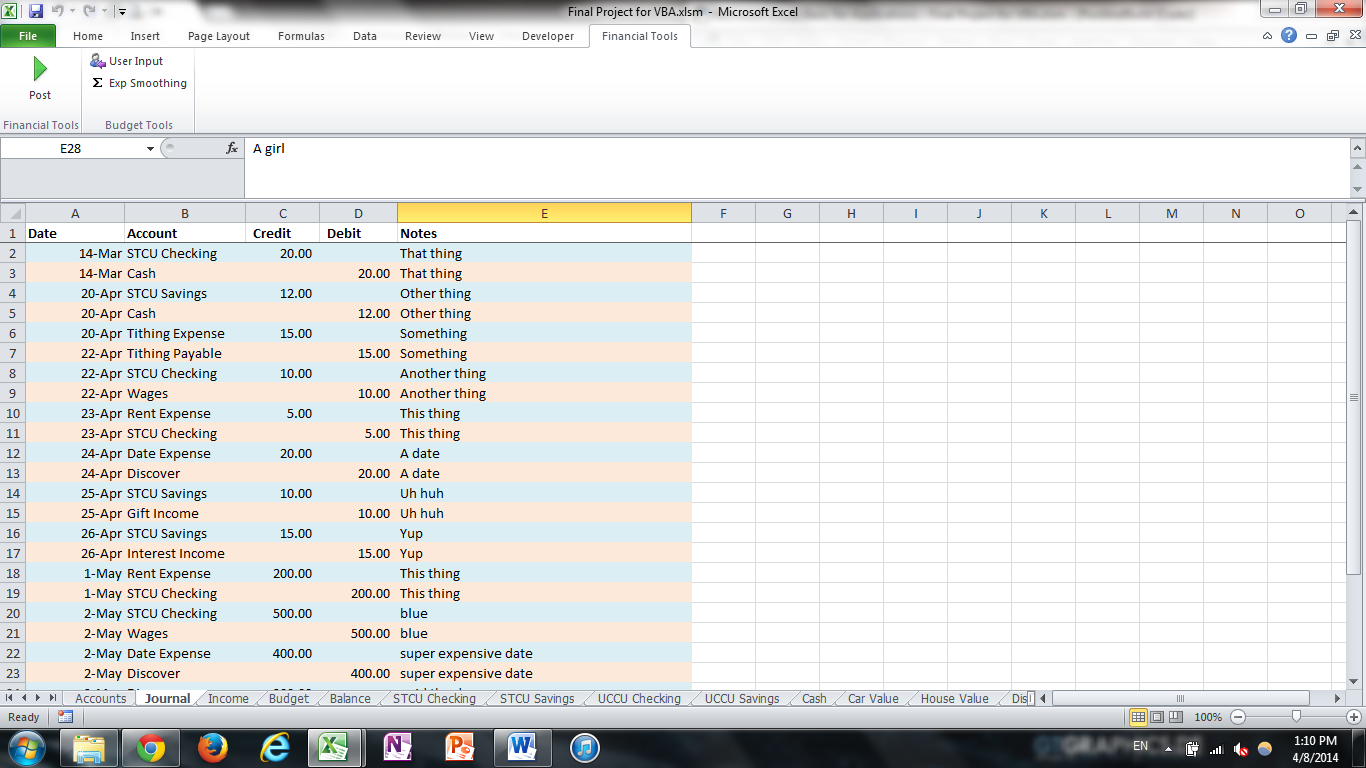
* Module level arrays are used to house the names of the user defined accounts and are declared and populated dynamically each time the macro is run
* The user is able to input any number of accounts
* The macros is controlled by a modified ribbon
* It is assumed that the user understands some basic accounting principles

*Details on specific steps*

**1**. The screenshot below shows where the user is able to input the accounts that he or she wants to use. The macro reads from this page and builds a separate array for each kind of account (current assets, long term asset, etc.) and uses those arrays to also define a larger array for the 3 basic elements of the accounting equation (Assets, Liabilities, Equity). This enables the macro to easily build the income statement, budget, and balance sheet appropriately.

You can also see the modified ribbon on top. The “Post” button is what the user presses to run this section of the project. The other two smaller buttons are for budget activities.

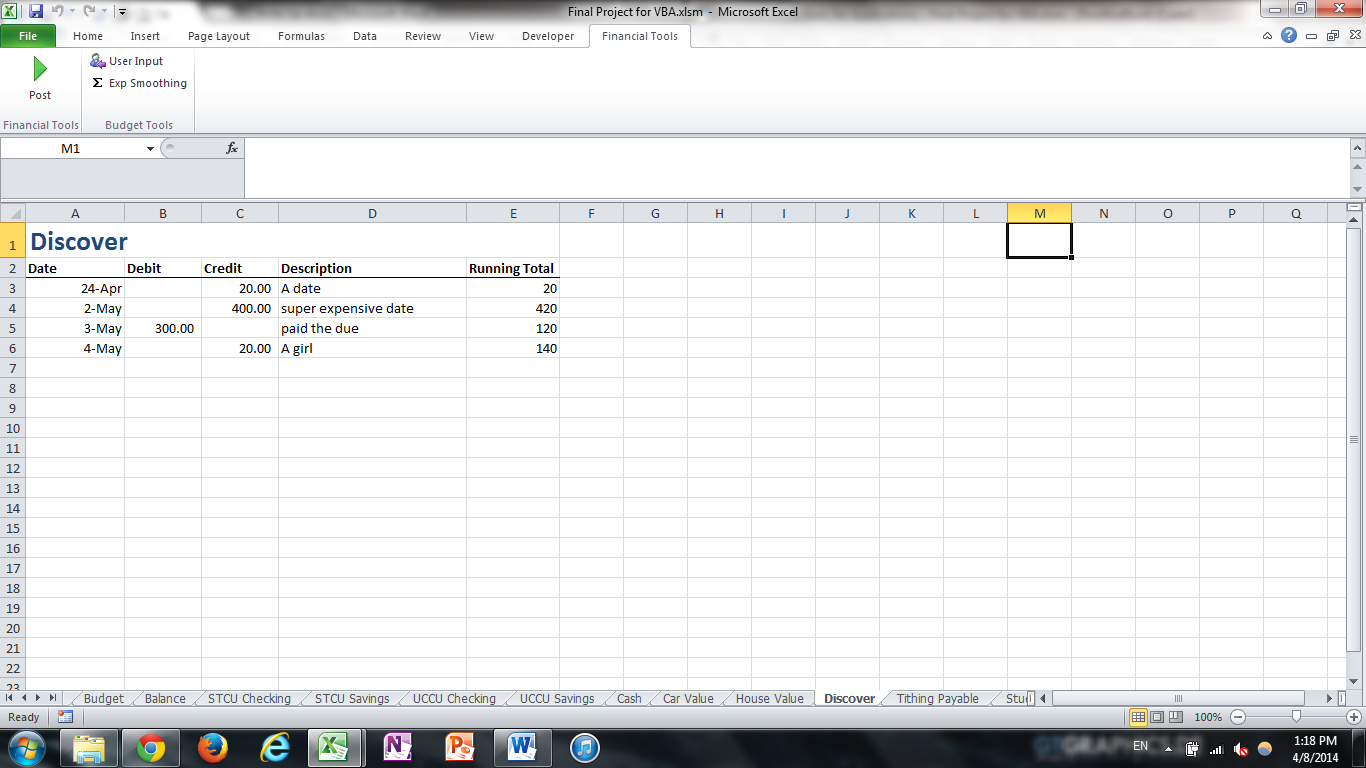
**2**. The screenshot below shows the page of the general journal where the user inputs transactions. It is anticipated that the user puts the data in chronologically and that debits will equal credits. The macro first makes sure that each item in the “Account” column refers to an account actually listed in the “Accounts” worksheet. If something doesn’t match, a message box appears telling the user what cell the validation failed at, and ends the macro (as a result, it will only find the first instance of a problem and won’t find multiple problems in the same go, but it really isn’t hard to keep things straight on the user’s end). It then checks to make sure all of the transactions are listed chronologically, and if they aren’t, a message box appears telling the user which cell is out of order. After it does these things, it will add up the credits and debits columns and make sure they’re equal. If they’re not, a message box appears notifying the user, and the program ends.



**3-5**. The macro then deletes all of the worksheets that aren’t part of the “core” worksheets. For each of the core worksheets, I modified their code names in the properties window of VBA so that even if the user gets crazy and modifies the worksheet names the program won’t break. The “core” worksheets are labeled “Accounts”, “Journal”, “Income”, “Budget”, “Balance”, and a hidden worksheet labeled “BudgetHolding”.

The non-core worksheets are the different t-charts for each of the individual accounts. The macro deletes all of the non-core worksheets, and then makes a new worksheet for each account and puts the basic formatting into each one (see the screenshot below as an example). The macro will post the date, the transaction amount, whether it was a debit or a credit, and the description of the transaction. In step 5 it will also make a running total column. The macro also makes the sheet name the name of the account, and puts the account name in a merged, header cell.

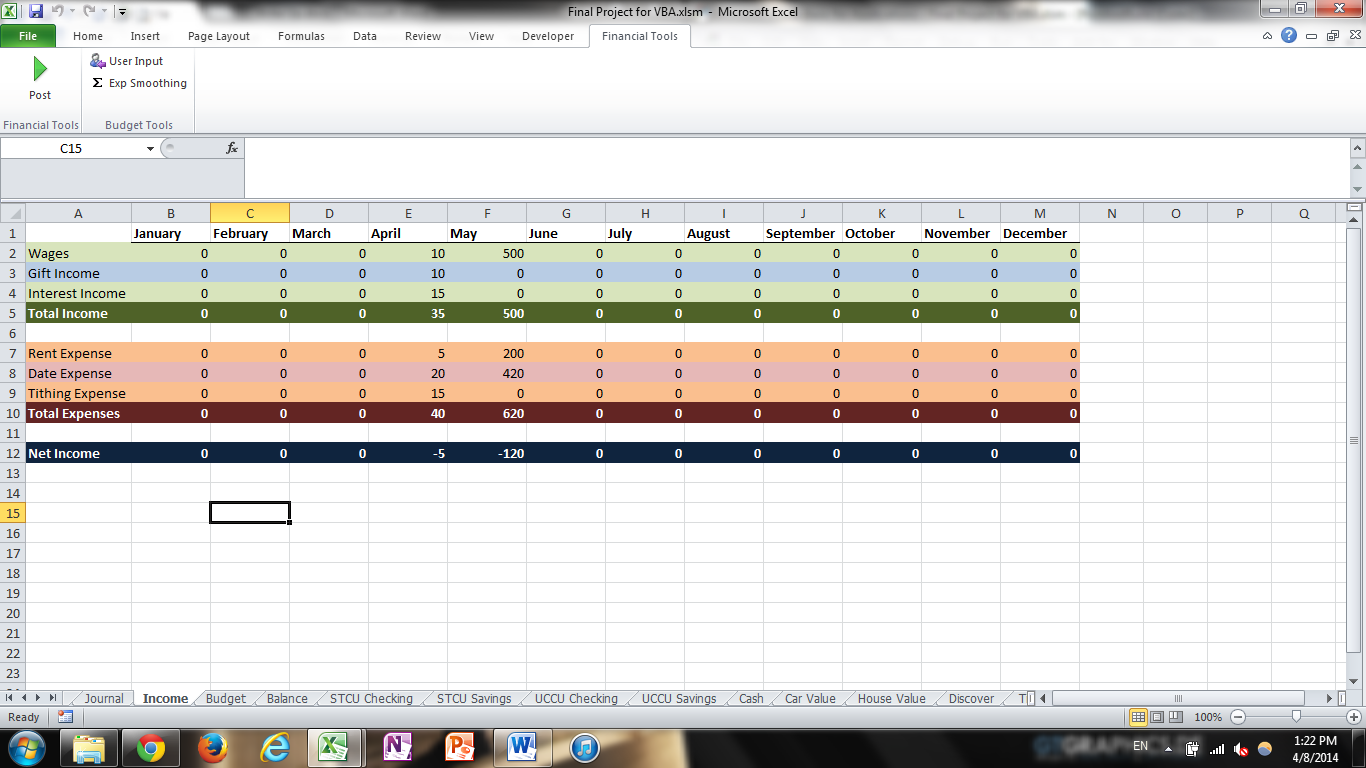
When worksheets are being deleted, the “DisplayAlerts” property is set to “False” to avoid having lots of windows interrupt the program. Deleting and creating worksheets takes the largest amount of run-time for the program, but the program still finishes the job pretty quickly.



**6-7**. This section builds the income statement as shown below. It first clears out everything on the sheet to start over. It then puts the month headings up top with the necessary formatting. The macro then goes through the equity accounts to find which ones increase on the credit side. Each account that increases on the credit side is a revenue account and gets placed in the first section (green and blue on the screenshot). A “sum” row is then placed under all of the revenue accounts that the macro finds.

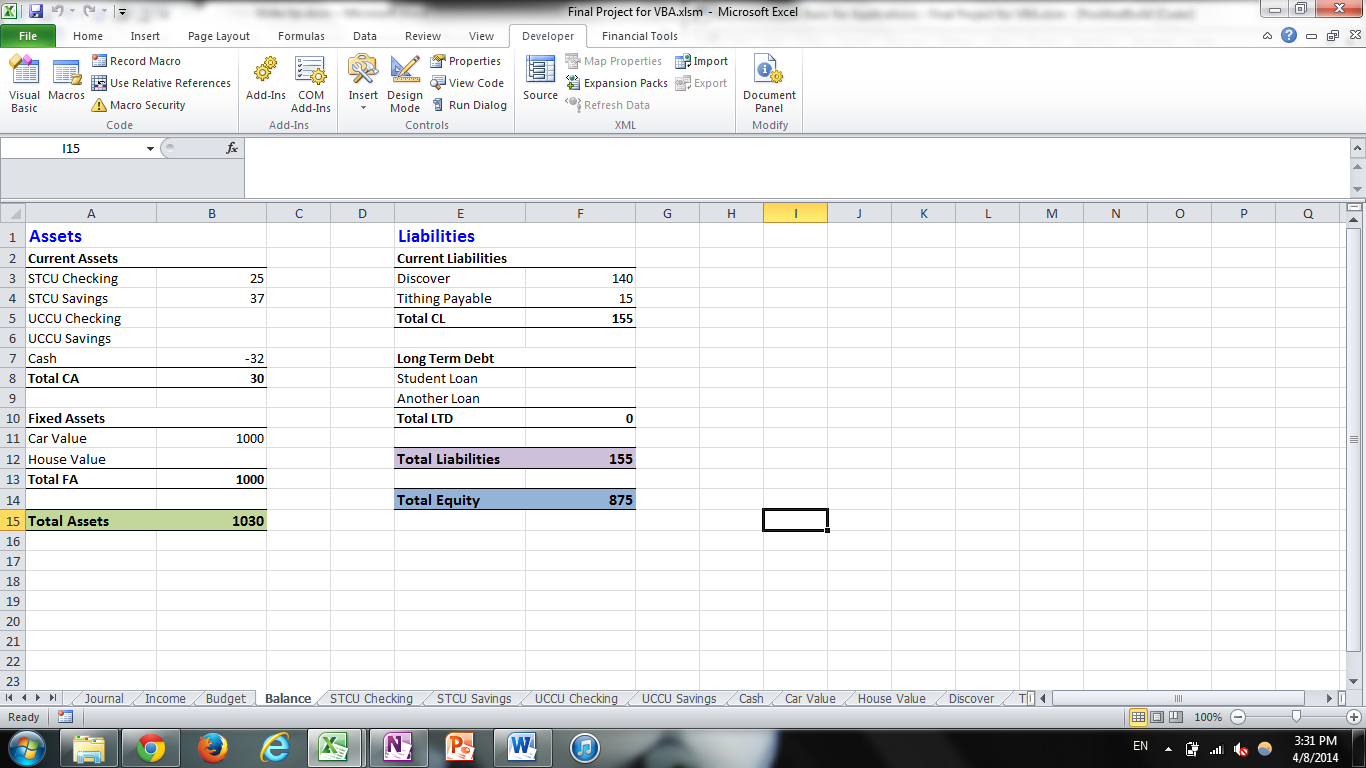
The macro then skips a line and puts in the expense accounts, makes a sum row, and then makes a “Net Income” row. The macro then formats the column widths to their appropriate sizes, and makes the sum rows bold.

The macro then sorts through all of the sheets for the income statement for each month, sums the transactions for each account for each month, and puts the appropriate value in the appropriate cell in the income statement. This required looking at the month value of each of the dates on each of the account sheets, and adding up all of the transactions that fit the date into a variable, then outputting the value of the variable into the appropriate location on the income statement. It does this without activating any of the sheets so it runs pretty quickly.



**8-9**. These steps build the balance sheet. The macro clears out the sheet, then sets up the top assets and liabilities headings. It then runs through the arrays to put in the account names and format everything. After it finishes the first section of accounts, it puts in totals rows, skips a row, and does the next section of accounts (long term accounts for assets and liabilities). Each main category also gets its own total, and the macro puts in the coloring. The macro also handles column widths to adjust for whatever size the account names are.

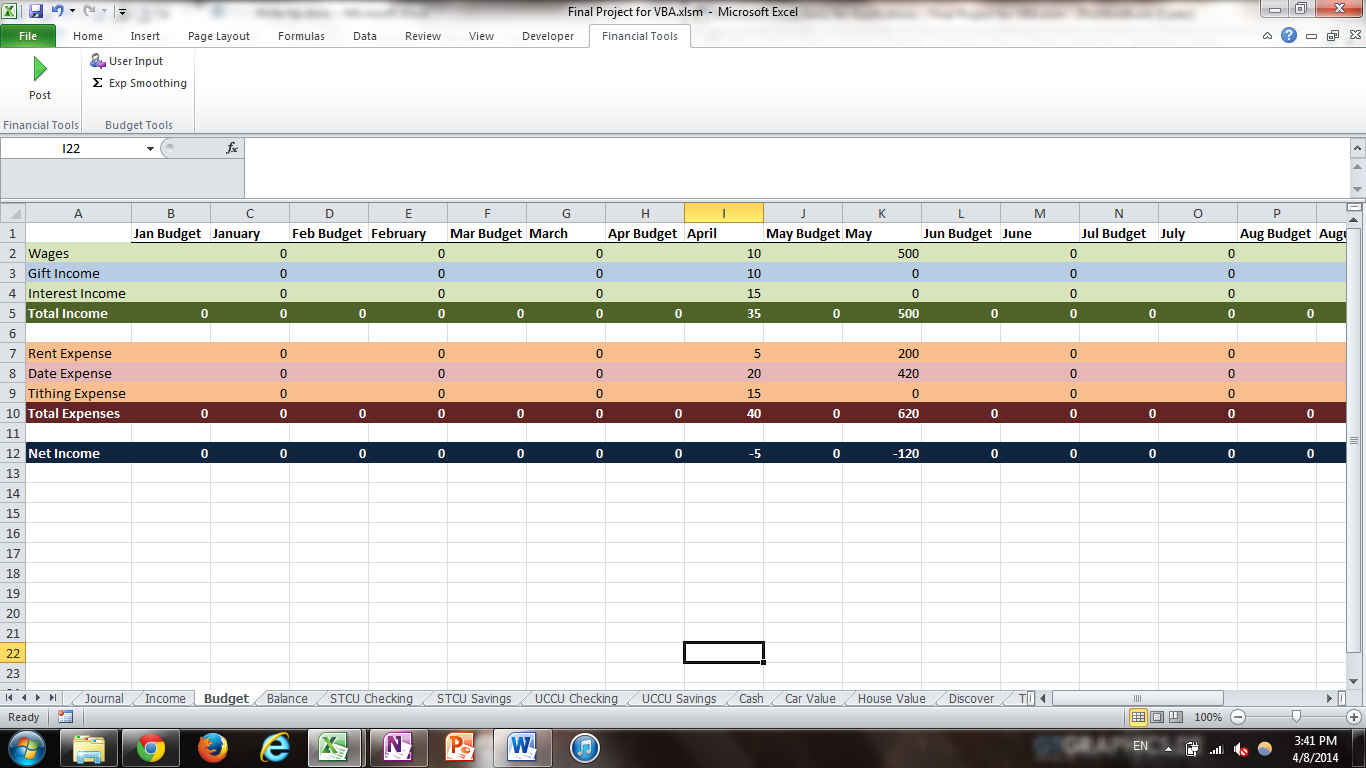
After the structure is set up, the macro goes through the account sheets to look at the running totals and apply that value to the category in the balance sheet. It also puts in the values to sum each category and subcategory.



**10**. This step puts the coloring in for the income statement and the general journal. For the income statement, it does different colors depending on whether an item is an income or an expense, alternating the colors on each line. I used the RGB codes to call for the colors after looking up the codes of the colors I wanted. It also looks for the sum lines and colors them differently as well.

In the journal, it simply adds alternating colors to each row that has values input from the user. This just helps readability a little bit.

**11-13**. These steps build the structure to prepare for a budget. Basically, it follows many of the same steps as the income statement above, but includes columns each month for a budget (see screenshot below). The details on how the user interacts with the budget and uses the other buttons will be outlined below.



***Building the Ledgers and Statements***

This section is simpler than the previous one, and is outside the scope of the original project, but it added an extra bit of fun to the workbook. There are two major subprocedures for the different buttons the user can click on the ribbon.

The first button is a “User Input” button. Clicking this button will switch the budget figures to those input by the user. It allows the user to make and keep a customized budget. The second button is an option to create a budget based on exponential smoothing, where the budget uses the following equation to put in budget values for each month:

Alpha\*(last month’s actual-last month’s budget)+last month’s budget

The “Alpha” is a user defined value. When the exponential smoothing option is selected, an input box appears asking for the user to choose an alpha, with the default value being 0.2.

*Details on the User Input Macro*

When this button is clicked, the macro looks at a hidden sheet called “BudgetHolding”. This hidden sheet contains the budget values previously input by the user for each account. By putting the values here, it enables the user to add new accounts, as well as use the exponential smoothing option, without losing previously entered budget amounts. After the user input button is clicked, the macro will write the budgeted values to the budget sheet, and the user is free to make changes, etc.

*Details on the Exponential Smoothing Macro*

This macro is slightly more complicated than the user input macro. This macro will first take the values listed that the user had for his or her user inputs. The macro will take all of these values and copy them to the “BudgetHolding” hidden worksheet, clearly labeled for easy access when the user wants to flip back to his or her custom budget.

The macro then displays the input box asking for the user to select an Alpha. 0.2 is a common number, so it is placed as the default to let the user just click “OK” most of the time. The macro then goes through and runs the equation listed above on each of the cells.

Exponential smoothing helps the user forecast what future amounts will likely be. When a budget is off from the actual, it will adjust the following month’s budget by the difference times the alpha. With an alpha of 0.2, it will add or subtract 20% of the amount the budget was off. This helps soften the effects of non-repeated events and is generally the most accurate way of forecasting. It’s not really a “budget” per se, but can help guide a person’s budget making process.

**Learning and Conceptual Difficulties Encountered**

What you learned through the project. Elements you wanted to include but couldn’t make work.

The bullet point summary of what I learned includes:

* How to use arrays more effectively
* How to modify a ribbon with multiple buttons and groups
* How to use option buttons on the Excel sheet (not included in the final project, but was my solution before learning how to modify the ribbon)
* How to better handle borders with formatting
* How to nest multiple loops within multiple loops
* The value of breaking apart long programs into multiple subprocedures
* How to use worksheet code names
* The value of including detailed comments

There weren’t any significant elements that I tried to make but couldn’t. I did originally try to make option buttons appear in the ribbon, but since that was beyond the original scope of the project I didn’t worry about it for very long and just made regular buttons work.

**Assistance**

I did not receive substantial outside assistance.