Benjamin Frederickson

[bfrederickson@dmacc.edu](mailto:bfrederickson@dmacc.edu)

Data Structures Final Project Report

I. Formalized Proposal

My project is a writing program. This program allows you to use a variety of commands to allow the user to edit the words they have typed. This will allow the user to customize their writing to their own desires and allow them to record whatever they wish. Thus, solving the real-world problem. Realistically this program is a flawed version of better word writing programs due to some limitations of knowledge, time, and tools given.

A more specific description of this program is that it uses a linked list and two stacks to do everything in the program. The linked list holds strings of what the user has typed. The two stacks hold the last piece of information that was typed, deleted, or inserted. These two data structures work together to create a writing program.

There are several commands the user can use to edit what they’ve typed into the linked list. The first one being InsertWord which allows the user to insert a word at any index value of the linked list. The second one is DeleteWord it allows the user to delete a word at any index value of the linked list. The third being UndoWord which allows the user to undo their last action, whether it was deleting a word, inserting one, undoing a redo, or just undoing what the user last typed. The fourth is RedoWord it allows the user to redo their last action, like UndoWord. RedoWord can redo a deletion, an insertion, an undo, or just redo the last word typed. The fifth command is CommandsWord it enables the user to see the commands again at any point in the program. The final and sixth command is EndProgram which ends the program for the user. After ending the program, the user will be given a yes or no question: Would you like to save what you have written? If they answer yes, the strings written by the user will be sent to a file which is displayed to the user, if they answer no the program will just end. The program works exclusively through a GUI. Each time a user makes a mistake a ‘error message’ appears in the form of a window. The user can see what they did wrong and be able to fix their mistake. It also appears in the beginning of the program to welcome (with a splash screen) and inform the user about the program. The GUI is also the main body of the program. It allows users to input information and see the words in the linked list in real time. In all this a summary of my final project for data structures.

II. Time/Change Logs

Week One and Week Two: 10 hours and 13 minutes.

In week one I created the basic parts of the program in the first week (creating the linked list and the stack .java files and creating the basic main program). I went through my old programs and modeled the linked list and stack from them, making minor improvements on the code as I went and adding some Javadoc and comments. This was most of what I did for the first week, but I forgot to write where I was when I finished so that’s why week one and two are put together.

I continued from my work on the first week and added to it greatly. This is the portion where I set aside the most time to work on my program. I created all the command keywords that I needed, removed things I didn’t need from my stack and linked list. Created most of the logic in the main program (with a few bugs here and there). I error tested and fixed more minor issues and included comments while I went. I kept working on my program until that day I couldn’t think of any more things to bug test or change and I stopped there.

Week Three: 3 hours and 37 minutes

Week three was where I removed any remaining bugs I could think of. I talked in detail in my write up about these certain bugs. Some were as simple as making sure the user’s input was correct, or more complicated issues where the program would break if the user typed in a command before any words. Also, in this program I fixed any remaining issues with the undo and redo commands. For instance, one of the biggest problems is if the user used the undo command twice the program would have an ArrayIndexOutofBounds exception and end the program. It was a simple fix of just letting the user know they cannot use undo twice once the command has been called a second time. I had my family use the program at this stage as well to check for anything I had not encountered for. One issue that I wanted to fix remained: the user could type long sentences with whitespace in them and it would only count as one space in the program (deleting it would delete the whole string you typed). I decided to end things here.

Week Four: 6 hours and 20 minutes

In this week I started writing this final project report to allow myself to make sure I got all the goals in my sprints done. I also finalized some issues in my program and cleaned up the code as well. First, I added a GUI to my program, spreading it across from the welcome message, to each of the error messages, and even including a command that will pop up a window letting the user know of the commands. Second, I made sure everything was commented on and had Javadoc where appropriate. Third, I did some final bug testing and one of the errors I found was when calling my new command CommandsWord it would make you type something before you could continue writing words into the linked list. Another bug I encountered was the whitespace one I mentioned in the last week. I ended up changing how I read user input. So, when a user types something with whitespace it will enter in each word one at a time to the linked list. At first this wasn’t the solution I was thinking of, but I think it’s more effective than the other methods I had thought of previously. Other than those issues I could not find anymore bugs. Finally, I added the save feature at the end of the program and continued writing this report.

Week Five: 11 hours and 3 minutes

In this week I redid a big part of my project, I completely re-worked the GUI and parts of my main program to accommodate those changes. This process became my biggest blocker throughout this entire program, I detail this blocker more in the lessons learned section. The GUI elements I create was a splash screen, a welcome page, words typed page, and using the JOptionPane to take userinput. I used all these things to completely eliminate the console’s use out of my program. After adding the GUI elements, I needed to update some parts of the main program. I had to change the way I got and used the user input (set up JOptionPane’s where I was using a scanner). I needed to change how I handled strings (like I mentioned in the previous week, but I applied it to wherever I got userinput from a JOptionPane). I also added some precautions for when the user randomly hits the cancel statement (if statements to see if the userInput was equal to null). And the final thing I did to the program was go through and improve / change comments throughout. After all that work I began to re-write my user manual and parts of my report. This is where I ended this week’s work with a total of 30 hours and 13 minutes.

Week Six: 4 hours and 3 minutes  
 In this week I finished this report and went through my code a final time. I fixed some very minor issues and made sure each comment made sense for the program. I created the slide deck for the presentation and made sure to update my GitHub. I believe at this point I don’t have anything else to do for this project and have finalized everything. This puts the ending time I’ve spent on the project around 34 hours and 16 minutes.

III. Lessons Learned

At first, I believed the scope of my program was too big. When first figuring how to make the stacks, and linked list work I was overwhelmed. I thought there was too, much to do and I wasn’t going to make the deadline. Well quickly I found I was wrong about the scope, I was thinking too small. I quickly made work of the basics of both the linked list and stack and moved onto the main. At first, I did have difficulty thinking of solutions of how to make all the commands work correctly, but after a lot of debugging there was many different solutions I could take. Once those issues were finished the rest of the program came with ease, there was a few other bugs that caused me issues, but eventually I fixed those as well. I ended up adding more functionality to my program in the end by adding a GUI, save feature, and a commands feature.

Some blockers I encountered was how to handle each of the commands. How would I allow the user to insert information? How would I allow them to delete? Undo? Redo? There were a lot of questions I wasn’t sure how to answer in the beginning processes. I worked it through. For insertion I would allow them to specify where and a word to put there. I would do this by using the function I created in the linked list where it would take the index given and shift all the values in the array over to allow the new one to be put in the index specified. For the deletion I only needed an index value given by the user, so I could take that index and remove the value currently in it then shifting all the values back by one in the array making the value non-existent. For undo and redo it was a much more complicated process. I had to account for what command came before them, because if it was an insertion, undo would have to delete the value just inserted. If it was a deletion the undo would have to re-insert that value in the same index value. It was a similar process for the redo, except the opposite of the undo. Inserting again or removing an insertion. My solution was to keep a string of the last action done and update with every new action. So, if you inserted something, it would keep a string of InsertWord for the undo function. When the undo function is called it will do a deletion of the newly inserted word with the correct index from the insert function. This was the same for every command including redo. Redo follows a similar process as the undo, the main difference being that it doesn’t do anything when DeleteWord is called.

The other blockers I had were relatively simple, such as calling a command before typing any words, using the undo function twice, writing a letter instead of a number when prompted for a number, and a few other minor issues. Fixing the command issue was simple as checking if the linked list has any words in it, if it doesn’t then if the user types a command give them an error message and don’t do anything else. For the undo function twice, I just kept track if the last command was UndoWord and if it was, give the user an error message. Finally, for the letter instead of a number I created a do-while which has a try and catch statement in it. If the user’s input (an index value) is within a range and doesn’t cause an exception they will leave the do-while loop and continue the command. If it causes an exception, tell them the issue and ask for the input again.

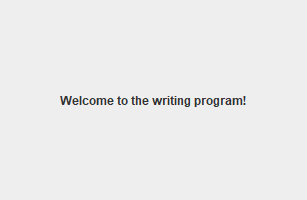
In the process of writing this report and working on my program I encountered my biggest blocker yet. When asking you about what I should do for a GUI you told me to have it so the console isn’t used at all. I began working at this by creating a GUI to allow input. At first I started by using a text field and buttons. You would type something in the text field and press okay to hand the program the input. This is where the problems began. At first after clicking the ok button the program went into a continuous loop entering the same user input repeatedly. This was because it never waited for the button, after it was clicked it was satisfied. I began to think of different solutions.. I could remove the main loop of my program and have buttons to call my functions. I began to re-write my program, but soon I learned that this approach didn’t work either. Next I tried to have different types of loops waiting until userinput was different again, but these didn’t work well either. I was quite frustrated at this point because I wasn’t sure what to do from here. Then I realized I could just use the JOptionPane for my input. This worked beautifully, and I implemented it in each of methods. When I was finished it worked quite well and looked much better than it did before.

One final blocker I encountered while trying to show you my program. For some reason the first word wasn’t updating in the typed info GUI. I went through the code and couldn’t figure out the issue at first. Then you suggested that it might be an error in the constructor, so I went back and checked how I created the window in the GUI. Turned out you were correct, I created it in a function instead of in the constructor. This was causing problems when trying to update the text. After I moved the label into the constructor and edited the main slightly, it worked great. I believe this is most of, if not all, of the blockers I encountered while writing my program. I learned a lot of lessons while coding in this program and learned some new tricks as well.

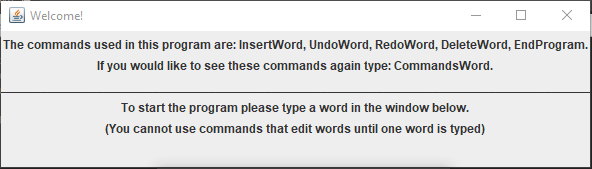
IV. CODE including comments:

<https://github.com/BenFred3/Data_Structures_Final>

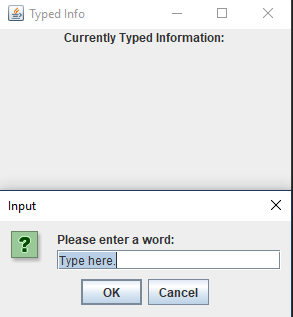
V. User Manual

Welcome to the user manual. In this manual you will learn how to use the writing program. It’s relatively simple to work through due to the fact the program prompts each action. To be able to start this program you will have to run the, “Final Project Tester” file in Eclipse. We’ll start with what you see when you start the program:

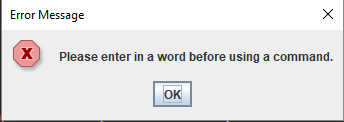
In the beginning of the program it shows you a splash screen that looks like this:



It welcomes you into the program then goes onto another screen that looks like this:

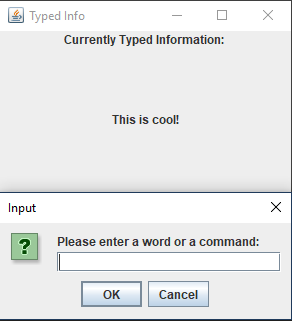


It tells you the commands and prompts you to begin typing. After that you can see a box where if you type it will be displayed and below it the box where you can begin typing:



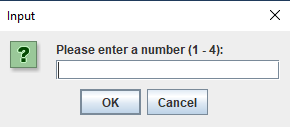
If you happen to ignore the request to type a word and enter a command (other than EndProgram or CommandsWord), you will encounter this error:

Simply just make sure to type a word (any word at all) to allow the use of commands. This program needs at least one word to allow you to do anything with commands.

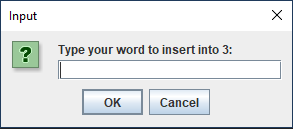
Once you begin typing some words you will be able to use any commands you would like. To begin we will type, “This is cool!” and press the ok button to see them on the typed info box.

Once you’ve added these words into the program, we will begin teaching you how to use each command starting with InsertWord.

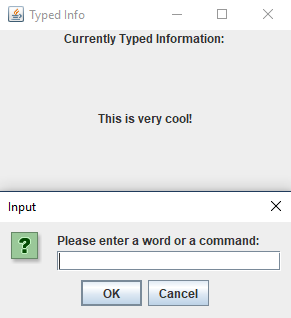
**InsertWord**: To start this command it is simple, if you type InsertWord into the program it will begin. Once you’ve done that it will prompt you to enter the number of where you want to place the word. The number values are simple. For the numbers from our example please reference this screenshot:

As you can see there is four choices, depending on how many words you type there will be a number value for each space. So, if you had 5 numbers, each word would be numbered from 1 to 5 with an extra number representing a 6th space. When you choose a number, your word will be placed in that number value. To show what that exactly means we will have to finish our example.

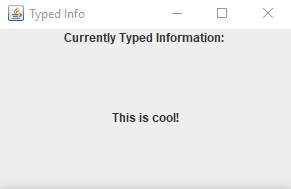
In our example after you type InsertWord you will be prompted with this box:



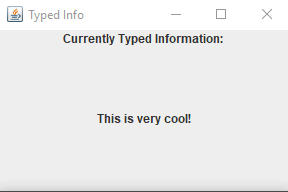
For our example we will choose 3. Next the program will ask you type in a word to insert into the value you’ve selected previously. It looks like this:



For our example we will type “very”. This is what it will look like after we have finished the InsertWord command:

**UndoWord**: To use this command all you must do is type UndoWord. Once you’ve done that the program will undo your last action. For example, after we just inserted a word, we could use the UndoWord command to remove it from our words typed. This screenshot from our example will show you just that:

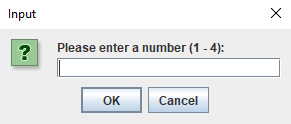
This command can also be used to undo a deletion, a word you’ve typed or undo the RedoWord command. Each of these is accomplished by just entering the UndoWord command. No other information or values are needed.

**RedoWord**: To use this command you just type RedoWord. This will then redo your last action in the program. For example, after we just undid the last action using the UndoWord command, you can redo it and make it insert the “very” again. This screenshot will allow you to see this in action:

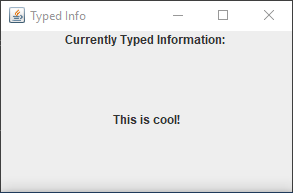
Like the UndoWord command, this command can redo a deletion, redo a word you just typed, or redo an insertion. You don’t need to do anything other than type in the RedoWord command, and then it will do just as the name says; redo.

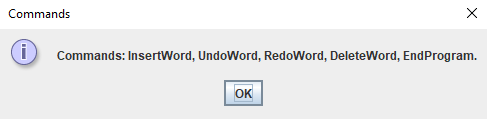
**DeleteWord**: The DeleteWord command is like the InsertWord command but has some distinct differences. For the DeleteWord command we will start a new example. In this example we will use the same words “This is very cool!” but with a misspelling in the word “very”. It will look like “vEry”, thus we need to delete it.

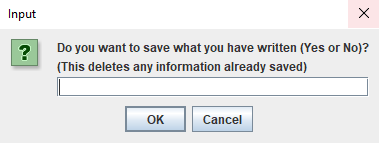
Like the command InsertWord, the program will prompt you for a number value to delete. So, type in the number corresponding to our example. Here is a screenshot to show what numbers correspond to our example:

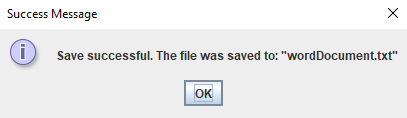


After typing the command, we are prompted for a number like this:

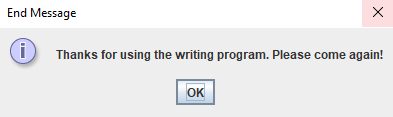
For our example we would choose the number 3. After choosing the number you are done using the command and the program will delete the word. This is the main difference between DeleteWord and InsertWord. Here is a screenshot showing the finished command:

**CommandsWord:** This command simply allows the user to see the commands listed again. They can do this command at any time and it will pop up with this window:

**EndProgram:** This command will end the program. After entering this command, you will get a pop-up window that says this:



If you enter in yes, the program will end, and you will get this message:



If the file was saved successfully and you will get the message in this screenshot (if you enter no you just get this message and the program ends):

**Errors during commands**: If you encounter any errors during the commands reference these three points:

(If you encounter an error not listed please contact the IT department ASAP)

1. You entered a word instead of a number when prompted for a number.

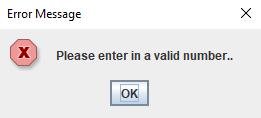
- Enter a number and this issue is solved.

2. You entered a number too high or low when prompted for a range.

- Each command will give you a range of numbers and if you don’t follow then you will encounter an error. Just choose a number in the range and this issue is solved.

3. You entered UndoWord twice.

- This isn’t allowed you need to choose a different command or type a word.



Here is a screenshot of one of these errors:

**Questions:** If you have any questions about the program please submit a ticket to IT department.

This is the end of the user manual, thank you for taking the time to read it!

VI. Conclusion

MERUSE -

Going through MERUSE’s principles of good programming I think that my code follows each principle. Starting with the M for modularity, my program incorporates several classes to create data structures for the main program. This fulfills the M’s statement of, “… writing classes and utilizing encapsulation”. Throughout the program I made sure to comment and create my code, so it would be able to be re-used throughout the program or future ones, which fulfills the other part of M’s statement (Statement: “This principle addresses writing pieces of code in Top-Down Design, so they are re-usable”).

Moving onto the E for efficiency, I made sure while creating my code not to incorporate methods that would needlessly use memory. There is one thing I did that could be considered not an efficient use of time (the splash screen), but I believe it adds a flair to my program, so I kept it. I fulfilled the E’s statement of, “This principle means efficient in memory and time usage. Algorithms should be efficient, you should not use more memory than necessary while allowing for scalability” throughout my program.

Next principle is R for robustness, through the development of my program I made sure to bug test extensively, account for when the user may be able to break the program (and handle it), and making sure the user could exit without throwing an error message. I have fulfilled the R’s statement of, “This principle means a program should not break, but since we cannot write bug-free code, the code handles errors when possible. When it cannot, the program should exit gracefully (ie, no blue screen of death)” by doing these things.

After that principle we move onto U for usability. For usability I made sure to make everything obvious to the end user even including a command to pull up a list of the other commands, labels on everything, and simple language. I also made sure to accomplish what I set out to create in a program. I allow the user to write whatever they want in my program with some commands. Making sure to include two data structures and a GUI. Thus, I have fulfilled the U’s statement of, “This principle addresses that a program should do what it was intended to do. Programmers may concentrate on bells and whistles and run into scope creep where they never accomplish the assigned task. Also, a program should be user-friendly and intuitive for the end-user (programmers often make code that is intuitive to themselves.)”.

Now moving onto the second to last principle S for should be readable. Throughout my program I made sure to keep this in mind by adding comments while creating my program and updating them making sure they made sense with any changes I made. I also made sure my variables names not only made sense to me, but to other people as well. Finally, I made sure my programs whitespace and indentation weren’t over the top (making sure it added to the program not making it harder to read). With these things I have fulfilled the S’s statement of, “This principle indicates the need for self-commenting code, appropriate use of indentation and whitespace, appropriate naming of variables and the inclusion of comments throughout”.

Finally getting to the last principle E for elegant. While working on my code I made sure to keep this principle in mind. Making sure to short solutions I had for problems in my code and working to make sure my program was easily readable. This fulfills the E’s statement of, “This principle addresses the idea of beautiful code. Programmers should be consistent in their style that leads to readable code. Additionally, shorter code and efficient algorithms lend to elegant code”. Thus, with this principle I believe I have fulfilled all MERUSE’s principles of good programming and have succeeded in using them while creating my program.

Summary -

This program is meant to solve the real-life problem of a writing program. Today many people use all sorts of different writing programs to allow them to document all sorts of things. I decided to create a simple one to fulfill that task. I decided to use a linked list to hold words that the user gave to the program. Then I wanted to add certain commands to add to my writing program, allowing the user to undo, redo, delete, and insert words as they please. To make the undo and redo work correctly, I decided on using a stack for both. After creating the linked list and the stack, I worked on the main body of my program. I created program that used Eclipse’s console to allow the user to type either words or commands they wanted to enter. After that was finished I started writing this report, working the beginning paragraphs, time table, and the user manual. After finishing the beginnings of my report, I went back to the program and double checked my work. I looked for bugs and encountered quite a few of them, but in the end I managed to fix all the ones I could find. Once that was finished I began to add a GUI to my program. It was no simple task and I struggled very much on it. I managed to get a working GUI in my program and was quite satisfied. Once the GUI was done I was just about finished with the project, only leaving redoing the user manual, updating the time table, finishing the conclusion of this report, and creating a slide deck. After finishing that I proceeded to turn in all my work and wrapped up the project.

Future Versions -

For future versions of this program, there is a lot of thing I would like to try and improve on. First thing I would like to do is improve my GUI. I would like to add buttons for the commands instead of having the user type them into the JOptionPane. It would add a lot more functionality. I tried to implement them but was unsuccessful in doing so. The second thing I would like to do is to remove the JOptionPane completely. I would like to add a TextBox, but I couldn’t get that working as well. The third thing I would like to add would be the ability to edit what you’ve typed more freely. Like how Microsoft word lets you edit words at any time by clicking in between words or highlighting them. I’m not sure how I would implement that with java, it might be as simple as having the text be set to the textbox and having the user hit enter to update the enter thing. These are just a few of the changes that could be done to the program to improve the GUI. One final change I would implement is making all the commands into functions outside of the main function. All of these changes would take quite a bit of time but would be very possible.