Evan Hedges

CIS 3190: Software for Legacy Systems

Professor Michael Wirth

This assignment has enlightened me to why archaic ways of programming and why they should be updated. For this assignment I decided to do the Ackermann Function. The provided implementation was very difficult to understand and following the properly flow of the assignment was difficult. I created flow diagrams and other methods to understand how this implementation of the algorithm worked. My overall approach to the assignment was to ensure that at all times I had a code that would return the correct information. If at any point the code did not work, I would debug my changes or revert to a previously working version. I began by getting user input so that I could easily run tests rather than using static values and require compiling. I proceeded to implement the stack by taking the example from the notes and modifying it to match my requirements for the assignment. Then I proceeded to convert the Pascal program into Ada. I began by creating three stacks, due to the original program pushing three values into a single stack. I then removed the GOTO statements, which allowed me to delete the stack used for labels. I made an additional copy of the Ackerman functions inside to replace the case statement of the original program. I then realized that by pushing the initial value of m and n onto the stack, all I would need is a single instance of Ackermann and could delete the function that was outside of the loop. At this point I discovered that the n stack was not needed, which left me with a single m stack. The resulting program was one that was more compact, easier to understand and follow. Overall, I found the most difficult part of the assignment was understanding the flow of the program. The GOTO statements were not easy to understand and made following the program difficult. Furthermore, some labels appeared to do nothing, such as label 4. This made deciding on what parts of the code could be stripped away difficult because it may change the behavior of the code. Eventually I understood that label 4 was used to create a loop, but this was initially difficult to understand. Overall, I found the assignment highlighted why GOTO statements are not recommended for coding. Modern languages have implemented most constructs that GOTOs were used for (such as loops) and that even with relatively simple functions it can obscure proper functioning.

I found the variety features of Ada very useful. I found the error handling particularly helpful. Due to my familiarity with C, I would often forget to put then after if statements. I appreciated the fact that the compiler tell me that I was missing a then statement. Furthermore, when I forget a semicolon, the compiler would provide the line number and the location of the missing semicolon. This is compared to C, which may throw numerous errors for forgetting a single semicolon. I also find procedures to be useful. The ability to edit multiple values inside a function can be very useful. I also found the record to be useful in creating the stack. It appears to be similar to structs from C. This allows for Ada to implement a variety of abstract data structures such as linked lists or trees. Furthermore, this can be allocated on the stack rather

than the heap which prevents a developer from making memory management mistakes. However, this can be also be considered somewhat of a draw back as I will discuss shortly.

The differences in the language was probably the most difficult part of Ada. Due to my familiarity with C and C based languages, I often found myself using a lone equal sign to set values rather than colon and equal. Furthermore, I would often use a double equal sign for comparisons, but Ada uses a single. This is not the fault of the language, but rather personal experience. One thing I do not like about Ada is that you can only have one package per file. I spent awhile trying to condense the stack and the Ackermann program into a single program because I thought that you restricted to a single file. I can see this preventing errors but also adding a large amount of files to a program. Another difference with Ada is the restricted access to certain variables. I dislike that you cannot change the value of an in only variable. This requires the creation of additional variables to store the values. Although this is not very important to this assignment, it may be useful on larger Ada programs to improve readability and clarity.

For this particular assignment, Ada may not have been much better than C. The ability to use an array to create a stack is very useful and an important concept to learn, however, the ability to have a dynamic array maybe more useful. Ada does have the ability to have a dynamically allocated array, although the size does need to be specified. The problem is that Ackermann Function can grow quite large and knowing the size may not be easy. This could be solved by using a linked list in Ada and converting this to a stack. However, at this point it would be easier to implement in C rather than Ada due to my familiarity with the C language. Furthermore, the since majority of memory is allocated to the heap, not the stack, using pointers would be better for dynamically allocated data structures.

I did not add many embellishments. The only embellishment I implemented was a loop to make marking easier.