CIS*3190 A2 Design Report

Re-Engineering the Hangman Game from Fortran77 to ADA

Michael Tran

0524704

Synopsis

For this assignment we got to choose between two topics. The first topic was converting hangman code from FORTRAN to Ada and the second was creating a word jumble game from scratch in Ada. I chose the first topic where we were tasked with re-engineering a hangman game that was written in FORTRAN 77 to Ada. Hangman is a simple game where a player must guess what the hidden word is by suggesting letters. If the letter is in the word it uncovers where the letter is found in the word, otherwise if the letter is not in the word a drawing of a person being hung. The player gets a total of 10 wrong guesses to finish the puzzle, if not the player loses the game.

Before attempting to re-engineer the program, I first had to learn the basics of Ada which would allow me to convert FORTRAN 77 code to modern Ada. After learning much of the basics of Ada, I spent time looking at the FORTRAN 77 code of hangman to gain a better understanding of what is happening in the program. While examining the legacy code line by line I would locate where I would have to modify legacy features in the program. While locating all the legacy features in the program I grouped parts of the code where it would make sense to have a procedure/function to execute the code to modularize the program. This also would make the transition from FORTRAN 77 to Ada an easier process. After reading through the code, I essentially broke up the code into two groups, the first being initializing the arrays and game while the second was how the game was to be played.

The first group included the variable declarations, initializing arrays and the game. While going through the FORTRAN 77 code, I first had to convert the terrible identifiers to more suitable names before I could really start coding the game in Ada. Upon finishing renaming the identifiers, I began the process of converting the legacy code to Ada. With the new identifier names, I had to decide whether to keep the variable type as they are or change them to more suitable variable types. Afterwards, I had to create 3 user defined types to deal with the arrays. The first type was a 2D array of characters that was meant for drawing the hangman picture. The second was an array of unbounded strings holding up to 200 words. The last type was an array to hold characters that the player has guessed. The reason for using defined types was because it was necessary to have defined types to have arrays as parameters for procedure/functions. The next step was to decide how many procedures/function I needed to make this program more modularized. In the end, I created 2 procedures to execute all of this. The first procedure dealt with initializing arrays such as resetting the stored user guesses to all blank. The second procedure dealt with setting the game up for the player. The procedure would call the first procedure, reset variables crucial to the game and randomly choose a word from the dictionary for the player to guess. This took care of the first page of the legacy code given to us.

The second group dealt with game play which included checking the players guessed letter, drawing the picture, etc. I figured these could be made into procedures as well. Overall I created 4 procedures to carry out how the game would be played. The first procedure I dealt with was drawing the picture. Since the legacy code used nested go to statements, I figured I could use a nested if statements to execute drawing the picture. The procedure has two parameters which were the array to draw the picture and a counter which was used for tracking how many wrong guesses the player has made which corresponded with different parts of the person to be drawn. The second procedure was to print out the letters that the player has already used. The third procedure was checking if the letter the player has guessed was in the puzzle. In the legacy code, many go to statements as well as arithmetic IF statement were used for this part of the program. To get rid of all the go to and arithmetic IF statements, I decided to use the modern if statements to deal with this problem. Finally, the fourth procedure was for checking the players answer when they correctly guessed a letter within the word. This procedure would be called within the 3rd procedure if the player guessed a correct letter in the word and that it did not finish the puzzle. This took care of the how the game would be played.

The final thing to do before attempting the bonus was to implement all of this in the main program. I decided to use a while loop to run the game because I could create another procedure to check whether the player wanted to play again. If they decided they did not want to play again, it would break out of the loop and end the game. Otherwise, I could restart the game by using the procedures I created above to allow the player to guess a new word. This would allow the main program to be very small, simple to read and understand as well as modularized.

For the bonus portion of the program I created another procedure that would read in a file. When the game starts, the player will be prompted if they want to load their own dictionary. If they decide to go this route, they can use any text file they want; otherwise it defaults to the default dictionary. However, if the text file the user inputs does not exist, an exception is caught and defaults to the default dictionary. The default dictionary uses the words found in the legacy code of the game. After deciding whether to use your own text file, the player is asked what the maximum length the word can be, and how many of them would you like to guess. After finishing this I integrated this into the program and ran many test cases to make the program runs smoothly.

<u>Identify Legacy Features</u>

Legacy Code	Re-Engineered Code	Comments

Character P(12,12)
 	
Integer Q , M , I , J, etc
	hangmanPicture : vector(1..14,1..12);
	
	numWrongGuess, numGuesses, maxWords, : integer;
	
	etc
		• Occurs at line 7 - 11
		• All of the variable names were really bad
		• Modernized it to have much better variable names

Character P(12,12)	type vector is array (integer range <>, integer range <>) of character; hangmanPicture : vector(1..14,1..12);	• Occurs at line 7 • Legacy defined 2D array • Modernized it to Ada by creating an user defined type

Character A*20, B*20	wordToGuess, userGuess: unbounded_string;	• Occurs at line 8 • Legacy defined character array • Modernized it to Ada by using unbounded strings

Character*20 DICT(50)
 	
DATA DICT /'gum','sin', etc
	type wordList is array(1..50) of unbounded_string;
	
	wordBank : wordList;
		• Occurs at line 13 - 25
		• Legacy defined 2D string array
		• Modernized to Ada by using an user defined type of array of unbounded strings

DO 16 I = 1,12 DO 15 J = 1,12 P(I,J) = " " 15 CONTINUE 16 CONTINUE	for i in 1..14 loop hangmanPicture (i,1) := 'X'; end loop; etc	• Occurs at line 30 -54 • Legacy do loop using a label to initialize arrays • Modernized it to Ada by using for loops to initialize arrays
IF (C .LT. W)	if counter < maxWords then etc,	• Occurs at lines 62 • Legacy code using .LT. relational operator • Replaced with modernized "<" operator
IF (N(I) .EQ. ' ')	if storeUserGuess(i) = ' ' then etc,	• Occurs at line 80, 89, 98, 103, 120 • Legacy code using .EQ. relational operator • Replaced with modernized "=" operator
IF (N(I) .EQ. ' ') GO TO 200	if storeUserGuess(i) = ' ' then etc,	• Occurs at line 62 63 80 89 92 95 104 106 109 114 120 121 129 130 133-141 160 165 171 175 179 182 185 189 201 • Legacy code using go to statements • Used modern if statements blocks to deal with the go to statements

IF (U(Q) - 1) 120,100,120	if wordUsed(number) = true then …. elsif length(wordBank(number)) = 0 then …. Etc,	• Occurs at line 71 90 112 127 198 • Legacy code arithmetic IF statements • Replaced arithmetic IF statements by using nested if statements
GO TO (415,420,425,430,435,440,445,450,455,460), M	if counter = 1 then … elsif counter = 2 then … etc	• Occurs at line 132 143 • Legacy code nested go to statements • Replaced these with nested if statements

<u>Questions and Answers</u>

1. For a program of this size, I believe you could have easily re-written this program from scratch in a language such as C. It's a very simple game with simple logic so there isn't anything that is too difficult to understand. Furthermore there are many flow charts that are widely available online that could help code this game. Also it helps that I have much more experience with writing in the C language compared to Ada which is why it would have been much easier to re-write the program from scratch.

2. There were many problems that I faced during the re-engineering process. The first was trying to understand what the code was trying to accomplish. The code was so messy that at first glance I was not looking forward to breaking down what each section of the code is trying to achieve. In addition the multiple go to statements was much worse in this assignment compared to the first assignment. It took a bit of time trying to understand how a single go to statement with multiple values would execute. Also the arithmetic IF statements took me a bit of time to understand. However, the toughest challenge I encountered during the re-engineering process was dealing with strings. There is a learning curve involved with learning strings/characters with Ada which made it a bit difficult to write the program. Ada strings expect to have a range/constraint which must be satisfied otherwise problems occur while C programming you could declare an array of any size and just place the null terminator at the end and have no problems. This made it difficult to keep the compiler happy and having the program run correctly. These were most of the problems I faced during the re-engineering process.

3. My program was much longer than the original legacy code. Compared to the legacy code, my program is much more structured and easier to read. The legacy code relied on multiple go to statements and arithmetic IF statements. This allowed the code to be very short but very difficult to read. I removed all the go to statements and arithmetic IF statements completely by using correct loops and decision making statements which adds

length to the code but makes it much easier to read and follow what is happening in the program. Also, Ada made it much easier to have multiple procedures/functions to carry out small tasks which allowed to the program to be modularized, which allows better readability.

4. In my personal opinion, I believe this was a good approach of re-engineering the legacy code. When you have code to look at you can gain a better knowledge of what the program might be trying to accomplish, rather than starting from scratch. However, it depends on the size of the program. For this assignment, you could have easily started from scratch and just used a flow chart to code this program. On the other hand, trying to re-engineer legacy code that is 10,000+ lines would be a difficult task to accomplish. Having the legacy code and a flow chart allows for better understanding and makes it simpler to tackle the problem of re-engineering the legacy code.

5. The structures that made Ada a good choice was how modular you could make the program. With procedures carrying out simple commands to execute or having functions carrying out mathematical calculations you are able to tell at a glance what the procedure/function is supposed to accomplish. Also, with user defined types you are able to make the program more customized to your taste. I particularly liked the user defined types that made it simpler to define variables that have similar uses.

Being faced with learning a new language can sometimes be difficult. Having to learn the syntax of the language, how to declare variables, how to create functions/procedures and I/O can be difficult to grasp. For the most part, I found the Ada language easy to grasp except for strings. Having many different types of strings made it a bit difficult to fully understand how to use strings. It doesn’t help that there is very little information out there to help you better understand the different types of strings Ada have and how to use them correctly. Fortunately, past experiences with learning new languages with very little information I was able to manage and find what I needed to complete this assignment. Overall, the Ada language was fun to learn other than having to find information how to deal with strings.