**Project 1**

**< Loot for the Slaying>**

**CSC-11 (48982)**

**Name: Vo, Kevin**

**Date: 11/8/2015**

**Introduction**

Title: Loot for the Slaying

“Loot for the Slaying” is a role-playing game where the player must first succeed in the challenges available in order to earn weapons (where it provides more damage) and be successful in their final conflict. If the player fails to do so at the first couple of challenges, he/she will not be able to kill the boss in time. Luckily in the bonus challenge, out luck and process of elimination, the player may be able to still defeat to final boss by earning more turns to attack it despite doing less damage. The player will be alert of how much damage they can perform after obtain a loot. Furthermore, the reason behind doing this and why it is important, resides to the fact that it allows players to temporarily immerse themselves by using their own problem solving skills to improve the character within the game. From there, they will be able to see an instant reward from their own capabilities and that in many ways is fun to experience.

**Summary**

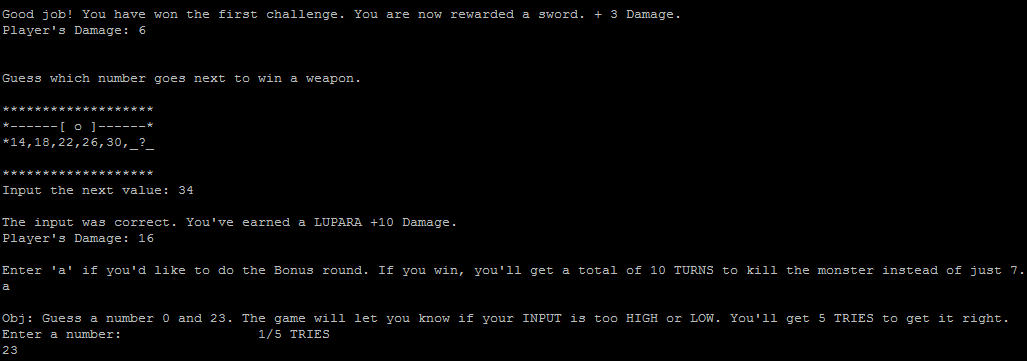
Project Size (C++ version): 194

Number of Variables: ~12 variables

The C version of the program contains nearly 200 lines and about 12 variables collectively when including all the header files along with main.cpp. The most challenging part of writing this programming is of course when translating from C to Assembly. Similar to the Fibonacci sequence along with the ISP package programs from the midterm, the hardest part is to figure out what my mistakes were when I get different results between the outputs from C and the outputs from assembly. This took up hours of my time when doing this first project as I kept on looking back and forth along with doing it on paper, only to find out it were simple mistakes like mistaken one registry for another, another part of the assembly program was affecting it, or the process was completely wrong. In addition, to scale the process out, it took me the weekend and Monday to think of what to do. I started my C code Monday afternoon and did some more it on Tuesday, but I didn’t like the idea and scrapped it and sketch another outline on paper. Finally, Wednesday evening I finished most of the C code and began doing it on Assembly while looking over the random number generator that was provided from class. Thursday and Friday was when I spend the day doing the assembly portion. Saturday was I fixed up the program and went to the C part to include the new things that I have added to the assembly along with writing this write-up. Moreover, whatever I have forgotten and whatever I felt I didn’t understand too well in assembly, thanks to the .pdf files provided on GitHub, I was able to find the site ThinkInGeek.com which has all of those tutorials and then some in separate chapters that acted as a guideline more or less and I find it to be more helpful than the book itself. In this project I was able to show that I have learned to do conditional branches, load and store information to/from memory, being able to perform inputs and outputs with printf and scanf, have a make and shell file, functions, shifts, being able to link multiple object files, loops, if-statements, and other conditions such as “ADDEQ” and “SUBEQ”.

**Description**

**Sample Input/Ouput**:



**Pseudo Code**

*Initialize*

*If the random number > 1 then minus 2 from random number*

*Else display random number*

*If input equals random number add three to player’s damage*

*Else player’s damage remains the same*

*Game ask player to figure out a pattern*

*If player enters the right answer adds ten to player’s damage*

*Else don’t add anything to player’s damager*

*Asks player to do a bonus mission*

*If player agrees to play then game gives player five turns to guess the number*

*If player guess the number correctly then game give ten rounds to fight boss*

*Else the game only gives seven rounds to fight boss*

*Else if player doesn’t want to do bonus then game jumps to boss*

*If player enter the right key then damage is done to boss*

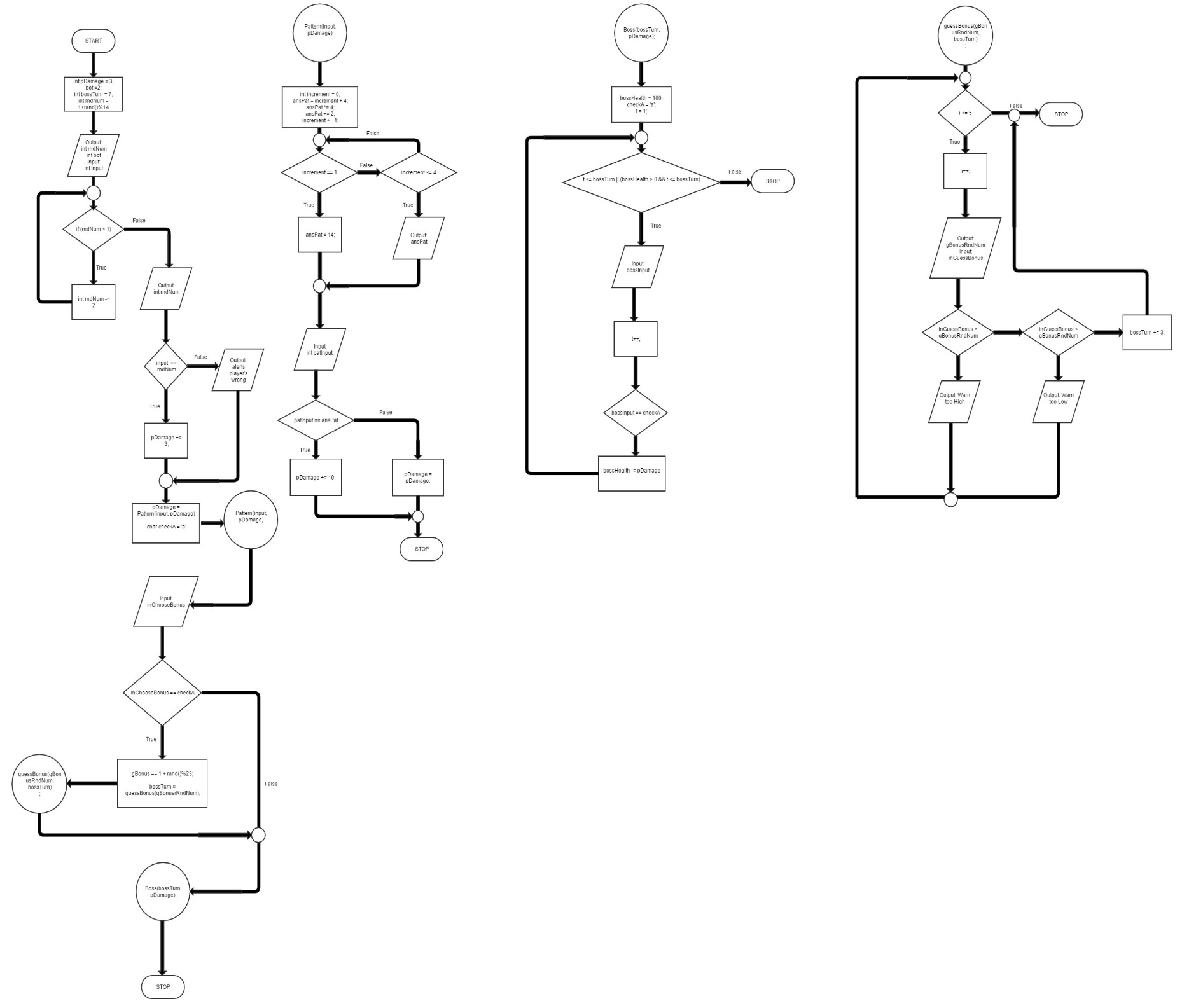
*Else the attack shall miss*

*If boss is dead before round ends player wins*

*Else player losses*

*Game exits*

**Flowchart**



**Major Variable**

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Variable | Description | Location |
| Integer | pDamage | Hold change value of player’s damage | main.cpp |
|  | bot | Stores value for challenge one | main.cpp |
|  | input | Takes player’s input | main.cpp |
|  | bossTurn | Holds number of turns for boss battle | main.cpp |
|  | ansPat | Stores the answer of a generated pattern | Pattern.h |
|  | increment | Keeps track of pattern’s terms | Pattern.h |
|  | patInput | Input for pattern | Pattern.h |
|  | bossHealth | Keeps track of boss’ health | Boss.h |
|  | t | Keeps track of turn | Boss.h |
|  | bossTurn | Maximum turn for boss battle | Boss.h |
| Char | checkA | Stores ‘a’ for comparison | Boss.h |
|  | bossInput | Char input for attacking boss | Boss.h |
| Integer | inGuessBonus | Input for the bonus challege | Guess.h |

**Language Constructs**

From the earliest chapters the MOV, ADD, SUB, BL can be found throughout the program. Not to mention, conditional branches can be found in any source file provide, one in particular is on line 78 and 79 (“BEQ correct” and “BNE incorrect”) in main.s. There is CMP as well and underneath some of those are ADDEQ and SUBEQ (line 118, main.s). Not to mention LDR and STR for when loading data to memory so that it can be later be loaded back to a registry for manipulation. There are also branches to printf and scanf are found in all source files. There are also two occasions where there are char comparisons (line 78, Boss.s). Also there are srand and rand branches for random number generation.

**References**

For this project I borrowed code from the mainDivModFuncV2.s and randTest.s that was present in class on Wednesday into order generation random numbers for Challenge 1 and the bonus challenge. I have also taken commands from Dr. Lehr’s repository for the make and shell files.

**Program (Part of it)**

.text

.global main

main:

LDR R1, address\_of\_return

STR LR, [R1]

LDR R0, address\_of\_Prompt1

bl printf

MOV R5, #3; //Setting starting damage

//Stores players health to memory

LDR R1, address\_of\_pDamage

STR R5, [R1]

bl ranNum//Calls function ranNum so generate random number

MOV R8, R1//For Displaying original value

calChal1:

SUBS R1, R1, #2//Answer

CMP R1, #1

BGT calChal1//simulates do while loop

Challenge1:

MOV R9, R1//Stores answer aftermath

MOV R2, #2 //Bottom

MOV R1, R8//R1 Top

LDR R0, address\_of\_chest1

bl printf

//Takes in answer for Challege 1

LDR R0, address\_of\_scan

LDR R1, address\_of\_storeit

bl scanf

//Loads it to R0 for comparison

LDR R0, address\_of\_storeit

LDR R0, [R0]

//Checks if answer is correct

CMP R9, R0

BEQ correct

BNE incorrect