The goal of this project was to create an assembler using the C++ language. My solution started out with the starting kit that handled getting the data from the .asm file. The only files I added were the assembler source file and its corresponding header file. My main file is called myAssembler and it uses the main file in the starting kit as a base. myAssembler is responsible for getting the data from the .asm and creating the output file when the conversion from assembly language to machine code is complete. When myAssebler is getting the data from the .asm file it places the information into a temporary Instruction struct to store the data from the .asm file. Afterwards the assembler handles the storing the Instruction struct into a vector and after all the Instruction structs are placed into the vector, assembler converts all the instructions into machine language. Assembler places the result into a vector, then myAssembler takes that vector and prints the results of that file to an output file.

Assembler contains various information aiding in the creation machine code. I had assembler store several map data structures to help with the conversion. rtypeInsts map contains the mnemonic of all the R-type format instructions and its corresponding function value, and I did not store the opcode data because for R-type instructions the opcode is the same for all of them, the value zero. The itypeInsts map has the I-type mnemonic and its corresponding opcode, and the function data is not stored because in I-type instructions the function information is not represented. The numbers map has a list of all the registers and their corresponding values represented as a 5-bit binary. The toConvert vector is the vector that assembler will assemble into machine code. The machineCode vector is the result of the assembly. The hex map contains the 4-bit binary values of number 1 to 15 and its corresponding value in hexadecimal. Finally, assembler contains a boolean variable called needSecondPass that represents if a beq or bne instruction exists in the instruction set.

All of assembler’s functions are used to create the necessary machine code. Most of the functions are private because myAssembler has no need for them. It is also within these functions that assembler checks for any errors. If any error is found within the instructions the program throws an exception, produces an error message, and exits the program. When myAssembler calls the createMachineCode function it starts the process of assembling all of the instructions. Assembler first goes through each instruction in toConvert and determines if each instruction is either an I-type or R-type instruction using the assignInstructionTypes function. Then it converts each instruction into binary using a specialized function depending on what type of instruction it is. Within these functions if a beq or bne instruction is found then the function secondPass will complete the appropriate binary for these instructions. Next, it will place the binary into the machineCode vector. Finally, it takes the binary values in the machineCode vector and converts it into hexadecimal format.