MODEL

Purpose: Model handles all of the data that is given from the user. It will also save input/changes and then tell the VIEW file when to update new information for the user. It has two subclasses that assist in its work. Memory and CPU

Description of Arguments:

- CPU cpu :the CPU class object
- VIEW view: the instance of the view
- MEMORY memory :the memory class object

Precondition: Depending on position in the current project, the MODEL will either have data stored in it's variables or it will not. It waits to get information from the CONTROLLER and is ready to send information to the VIEW page as well

Postcondition: The data brought in from the Controller will be loaded into the appropriate data structures and then it will ready itself for the next instruction from the controller.

CPU

Purpose: The CPU will handle the switch case, this is the main structure that handles the instructions from the user. it interacts with the memory, pulling data and working it when needed.

Description of Arguments:

- map<> memory is the raw memory
- Int registers[3] stores the accumulator, IC, and Opcode
- String IR stores the operand
- Bool halt trips when the CPU is done executing

Precondition: None

Postcondition: Once runCPU executed, the arguments will be populated to be read in MemoryDump

Memory

Purpose: As a subclass of the Model, the memory class is the framework for the memory within which the data will be stored. Most of the data will be pulled from the memory to be manipulated and stored back into memory after manipulation.

Description of Arguments:

- map<size_t, string> memory: The object which the memory lines are stored

Precondition: It's a blank memory object, awaiting input from the CPU and Model.

Postcondition: New memory is stored, filled with instructions and stored data from the user's input program.

VIEW

Purpose: The VIEW handles everything that will be displayed to the user. It is responsible for printing menu pages, displaying error messages, and CMD text formatting at a resolution of 120x30 characters. It will receive instructions on what to display from the MODEL and the CONTROLLER. It has 2 FACADEs, MenuStrings and a subclass MemoryDump.

Description of Arguments:

- String pageFormat[4]: data structure that hosts 4 important strings. The menu number the program is currently in, what page Execution mode is in, what page Edit mode is in, How many lines until the next page needs to be printed.
- String currPage: is what is currently be displayed
- Const string TOKENS[5]: list of tokens for regex replace
- Const char MESSAGES[14]: list of messages to print
- Stringstream currPage: is where the screen is saved

Precondition: VIEW will be created in the CONTROLLER and the instance will be passed to the MODEL. It will wait until the CONTROLLER or MODEL calls it to display something

Postcondition: Console is updated and VIEW will go back to being idle

CONTROLLER

Purpose: The CONTROLLER is responsible for collecting user input and decides what next step the simulator needs to take. User input is stored in memory and sent to MODEL. When the user is done with editing mode, the CONTROLLER tells the MODEL to start execution. The CONTROLLER tells MODEL what menu pages it needs to send to View, and is also in charge of loading and saving programs from within the simulator during runtime.

Description of Arguments:

- MODEL model :an instance of the MODEL
- VIEW view :an instance of the VIEW
- Const string saveFile: name of the save file

Precondition: Waits for user input
Postcondition: The program ends

MenuStrings

Purpose: Acts as a facade wrapper for loading all the menu strings files we will be using in the program.

Description of Arguments:

- Vector<string> pages stores the loaded menu pages
- String pageNames[] holds the names of all the txt files

Precondition: Menus aren't loaded

Postcondition: Menus loaded

MemoryDump

Purpose: Formats the memory from the Model into a printable form and passes it to the view.

Description of Arguments:

- CPU- the cpu object passed to it from the MODEL. At this point, CPU is acting as a data structure.

Precondition: CPU must have executed runCPU()

Postcondition: Prints the CPU dump on the console