ECS 40 **Program #3 (50 pts, 1.5 hours)** **Winter 2016**

Due: Wednesday, February 3rd, 11:59 PM. Executable Name: CPU.out

Filenames (case sensitive): authors.csv, decoder.cpp, decoder.h, instruction.cpp, instruction.h, main.cpp, reader.cpp, reader.h, registers.cpp, registers.h, and Makefile.

For this assignment you will be converting Program #1 to C++ code using classes, new, delete, and standard C++ input/output classes. All implementation code should be in the .cpp files. You are free (and encouraged) to use my source code for p1 as your starting point. You will find all my source code from p1 in ~ssdavis/40/p1/SeansSrc Thursday morning. In any case, you may wish to refer to my code if you do not understand some of the references in the instructions. Here are the steps to make the conversion. Your program should compile without warnings, and run perfectly after each complete step.

1. Rename all .c files to .cpp files, and rewrite the Makefile accordingly.
2. Replace all calls using stdio.h with calls using iostream, fstream, and iomanip.
   - 2.1. Eliminate the #include <stdio.h>, and replace it with #includes of iostream, fstream, and/or iomanip. Add "using namespace std;" wherever these header files are used.
   - 2.2. printf() calls should be replaced with lines using cout and the functions of iomanip described on pages 758-759 in the text.
   - 2.3. File operations should use an ifstream object named inf. getline() will be quite useful.
3. Replace #includes of string.h with cstring, and stdlib with cstdlib. Add "using namespace std;" where needed.
4. Replace calls to malloc() with calls to new. If you code has any calls to free(), then replace them with calls to delete.
5. Make the typedef struct Reader a class.
   - 5.1. Change the typedef in reader.h to a Reader class. Remember to have a semi-colon after the final '}'.
   - 5.2. Make read(), and fetch() public member functions of the Reader class.
     - 5.2.1. You should remove the Reader pointer from their parameter lists.
     - 5.2.2. In reader.h, move their prototypes into the Reader class under a "public:" heading.
     - 5.2.3. In reader.cpp, add "Reader::" in front of their function names.
       - 5.2.3.1. Remove the Reader pointer from their parameter lists.
       - 5.2.3.2. Within the functions, remove all "reader->", since the private data is directly available (a simple search/replace works great!).
     - 5.2.4. In main(), just call them using the Reader object declared in main(), e.g. "reader.read(&registers, argv[1]);"
   - 5.3. After everything compiles and runs perfectly again, as an experiment, delete the semi-colon after the final '}' of the class definition to see what obscure error you get. Then put the semi-colon back.
6. Make the typedef struct Decoder a class.
   - 6.1. Use the steps described for the Reader class as a guideline.
   - 6.2. Make parse(), and execute() public member functions of the Decoder class.
   - 6.3. Make all the assembly language functions private member functions of the Decoder class.
     - 6.3.1. Do not use "this" to make their calls from execute(). There is no need to since the calls are already in a member function, and the object is understood implicitly.
     - 6.3.2. Change pushl() so that it subtracts four from esp, and then places the *operand1* information at the location specified by the esp.
7. Make the typedef struct Registers a class, using the steps of the Reader class as a guideline.
   - 7.1. Move the address() code from main.cpp to registers.cpp.
   - 7.2. Move the address() prototype and enum from main.h to registers.h.
     - 7.2.1. Eliminate main.h and any mention of it in any file.
     - 7.2.2. Make the enum a typedef with the name RegName;
   - 7.3. Make address(), initialize(), and print() public member functions of the Registers class.
   - 7.4. Add a public accessor method, get() which takes a RegName, and returns an int. If the parameter is out of range return 0.
     - 7.4.1. Replace all accesses to the regs in other files with calls to get().
   - 7.5. Add a public mutator method, set(), which takes a RegName, another int, the value for the indicated register, and returns void. If the first parameter is out of range do nothing.
     - 7.5.1. Replace all attempts to change the regs in other files with calls to set(). Note that code that used "+=" will now involve a call to set(), and get()!

8. Make the typedef Instruction a class with a default constructor, a destructor, two accessor methods, and two mutator methods. You will now need an instruction.cpp file to hold all of the implementation code for these functions.
   8.1. The default constructor should set info to NULL. This will be called when Reader creates its array of Instruction. NULL is defined in <iostream>.
   8.2. The destructor should use delete to de-allocate info if it is non-NULL. This will be called when the program terminates and destroys the array of Instruction in Reader.
   8.3. The accessor method, getInfo(), should return a const char*.
   8.4. The accessor method, getAddress(), should return an int.
   8.5. One mutator method, setInfo(), should take a const char*, dynamically allocate enough space for the string, and then copy it into info. This will permit you to move your dynamic memory allocation in the Reader::read() function into this method. The function should return void.
   8.6. The other mutator method, setAddress(), should take an int, and return void.

Further specifications:

1.1. You may assume that all input will be valid, and not require any form of range checking.
1.2. Your Makefile file must contain five pairs of compiling lines, and one pair of linking lines. You must use g++ with the –g –Wall –ansi options for compiling and linking. You will lose one point for each warning.
1.3. You will find test?.c, test?.S, and my own executable in ~ssdavis/40/p3.