Notes On the Assembler Project

Statement of the project

The term project requires that you create an assembler and emulator for the VC1620 computer. Note: this is similar to the IBM 1620 that was popular in the 1960s. Let's look this machine up on the Web.

The program must work as a necessary condition for you to pass this course. That is, if you don't have a working program you will fail the course. The program must run in Visual Studio. It must at the very least translate the example as specified in: Typical Output Note: I will run every submitted program to make sure it works.

Incompletes are not given in this course unless you have a very substantial reason.

Design Process for a Small Project

One of the goals for assembler project is to give a picture of the steps necessary to develop a relatively small piece of software. This is similar to your efforts in industry. Note: Most of the software development you will doing will be to create/modify pieces of a larger project. (Agile development uses this approach.)

1. Get the requirements for the project.
    1. That is a complete definition of the project. **You might have to write this yourself**. That does sound strange. We will talk about it.
    2. This might take several rewrites as you interact with the originator of the request. I do this all the time in my consulting. This saves a lot of pain later. Why???
    3. Having a project requirements document will save you a considerable amount of time by truly knowing what you will be trying to accomplish.
2. Gain an understanding of what the code has to do. **How do you do this?**

3. Develop a list of the issues and problems involved in building the project.

4. Identify the classes needed to implement the project.
    1. These are the basic conceptual elements that we identified in the previous step.
    2. A sloppy choice here can lead to a **lot** of problems later.
    3. Be aware of additional resources that you may require. Like 3rd party libraries.

5. As you code the project, you should be willing to change the design based on the knowledge that you have gained in writing the code. **I have seen big problems on the senior project when students were unwilling to change their design.** We will talk more about coding the project later.

The project

The term project for this semester to the create an assembler and emulator for the VC1620 computer. This is a "pretend" computer that has some similarities to the IBM 1620 of the 1960s and 1970s.

1. An assembler is a program that translates a program written in assembly language (symbolic machine language) into machine language.

2. An emulator is a program that imitates a specified machine. So, if you give it a machine language program for that machine, it will be able to run it. Emulators are important in the support of apps creation. What does this mean?

3. Note: the VC1620 computer is quite different from the VC5000 of last semester.

The Design Process for creating an Assembler for the VC1620 computer - we will first get the requirements for the project.

1. We will discuss the VC1600 machine language from the handout. VC1620.
2. We will discuss the format of an assembly language for the VC1620 computer. Use the VC1620 Assembler handout. This is part of getting the requirements for the project.

3. Take a look at the output that is expected for this project. Typical Output. This is part of getting the requirements for the project. **Note: your project must be able to successfully generate this output in order to be accepted. You should do the emulator, but you will not automatically fail the course if you do not finish the emulator. But, it will hurt your grade if you don't complete it.**

   At this point we have the requirements for the project. start here - 10/27/2021

4. To understand the concepts of creating an assembler, go through the steps of translating an assembler language program. We will do this together in class.

5. Create a list of the elements of the project. My list is below from when I did this alone. We will probably have a somewhat different list when doing this in class.
   1. File access. Opening, reading lines, rewinding the source file, and reading the lines again.
   2. Parsing input line into label, op code and operands. Must also identify when there is no label. Need to identify when there are not enough operands. Need to handle assembler op codes.
   3. Symbol table. Be able to record, sort, display and access data from a symbol table.
   4. Translating numeric operands into numeric values
   5. Code generation. Generate machine language instructions from Op code and operands.
   6. Error reporting. Complete check of syntax. **Errors should be reported after the offending statement.** The translation should continue even if there are errors. When an error occurs, as much of the offending statement as possible should be translated. Address and op codes which cannot be translated should be replaced by "?"s.
   7. Compute the address of the next instruction given the current instruction and its address.
   8. Identify the type of instruction.
   9. Determine if a string is numeric and convert strings containing numeric data into integers.
   10. Op code lookup.
   11. Note that the assemble process breaks down into two passes. Pass I to establish the location of the labels and Pass II to generate the code. We do not have to report error messages until Pass II since we will be going through the same code twice.
   12. Must build emulation. This is a relatively independent part of the assembler part of the project. Later, we will have a discussion in class. It does not require much effort.

6. Error processing is a major part of an real-world project. **All errors should be reported in the second pass, immediately after the offending statement has been displayed. You will not stop translating after you reported the first error.** All errors will be reported and the assembler will translate as much as it can after the error. We will list errors together. Below is a list that I wrote up.
   1. Multiply defined labels. (Reported after the offending statements.)
   2. Undefined label. Namely, a symbolic operand does not have a matching a label.
   3. Syntax error in construction of the label or operands. For example, the operand of a DS must be numeric and those of an ADD instruction must be symbolic. Labels and operand must meet the format given in the specifications.
   4. Extra statement elements.
   5. Illegal operation code.
   6. Insufficient memory for the translation.
   7. Missing end statement or the end statement is not the last one in the program.
   8. Constant too large for VC1620 memory.
   9. Missing or extra operands

7. Error processing for the emulation will be simple. Just report the location where you find the first instruction that cannot be executed, indicate the problem, and terminate.
8. In writing a C++ project of this size the first step in the program design is to determine the classes needed for this effort. A common mistake of entry level programmers and some academics is to have too many classes. Some programmers do not create enough classes. See Class Design for the C++ class definitions for the assembler. We are matching the classes to the functionality discovered earlier. There may be more classes added later.
9. The next step is to develop class definitions for each of the needed classes. This will be followed by a top down implementation of the Assembler. (Why top down?????) We give a quick look and then come back after some preliminary material.
10. **Testing should be done using a debugger and other programming tools.**

Submission of Project

You should submit your project via email. The .h and .cpp files should be submitted as attachments. Please do not zip them. Many email systems reject these. Please do not wait until the end of the semester to work on this project. This project does take a lot of time and I do not give incompletes.

Programming Tools (Later possibly after the presentations.)

Note: we have already discussed almost all of these or will be doing so in the presentation.

Debugger

In order to do effective software development, it is necessary to use development tools. I assume that you are comfortable with Visual Studio. This will supply us with an editor, a C++ compiler, a debugger, online help, and a profiler. It is a development environment. If you live in the UNIX world, Eclipse which uses the GNU C++ compiler and the GNU debugger provide a fine development environment. I will demo the Eclipse for C++ development later in the semester.

Libraries

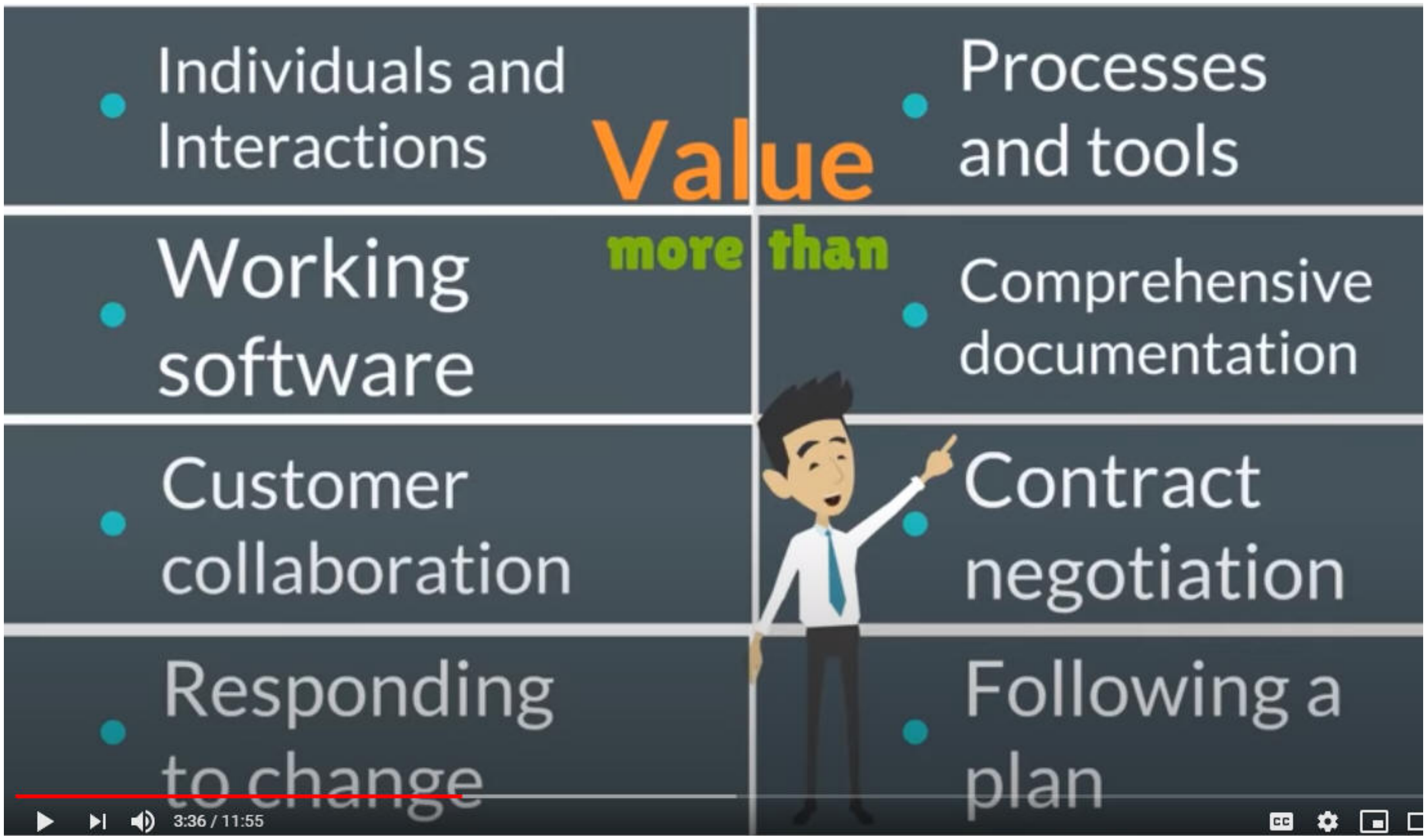
No boss is happy if you rewrite existing code. You should make as much use as possible of existing code. C and C++ both have standard libraries. STL is supported in C++. Also, there are libraries that may be purchased or free(e.g. Boost) and libraries that your company has produced. As an example, consider the fact that you will have be able to search the symbol table and display it in alphabetical order. STL provides tools to do this. The standard C library also supplies this functionality. See qsort and bsearch for a description a sort and a search function. If there is time, I will describe these because their prototypes are unusual and good to know the weirdness. Also, you should try to take full advantage of other built-in features of C++.

Coding Standards

Almost every well managed company will have a set of programming standards which the programmers are expected to follow. I have created some professional standards that I expect to follow in your project. Despite the fact that they might seem to require a lot of extra effort, these coding standards will assist you in completing your project on time.

We will talk a little about Agile development if it is not in one of the student presentations

Below are some initial ideas



See: https://www.youtube.com/watch?v=Z9QbYZh1YXY for more ideas.