CC410 System Programming Project Ideas:

Responsible use of online resources:

Please cite the web sites that you download examples and resources from. Copying partial contents from the Internet is alright to learn from. If you submit fully copied projects, you will receive a zero mark. Please add more features, or change the behaviour of the function, or the interface at least. Please remember that you will display a demo of your project running and explain how you wrote the source code. If the project doesn't run, you will loose a lot of marks. Similarly, if you fail to explain how the source code was written, you will loose marks.

General Rules:

- A group can be formed from 3 or 4 students per project.
- Groups are not allowed to repeat ideas, so first come first serve bases, once a group comment on the project's thread with a choice, the idea can not be chosen by another group.

Grading Criteria:

The project is worth 10% of your final mark. These marks are graded individually by asking each student in the presentation for their contribution and testing their understanding. These are broken into:

- 5 marks for correctness (no compilation or run-time errors).
- 4 marks for the application of course concepts, where feasible, (such as: assembly code, instruction formats, addressing modes, assembler functions, loaders and linkers functions, macroprocessors, and compilation)
- 1 marks for the interface, presentation, documentation, and teamwork.
- 5 bonus marks for all other concepts you self-study outside the learning objectives of the course.

Submission Details:

All project files are zipped and submitted named as "proj_LeaderStudentID.zip", where "LeaderStudentID" is replaced by a leader team member chosen by all team members. The zip file should contain:

- All source code used to develop the project, either as a jar file, or a folder of the project packages and files.
- A written report (soft-copy and hardcopy) according to the instructions written in the guide and example shown on moodle.

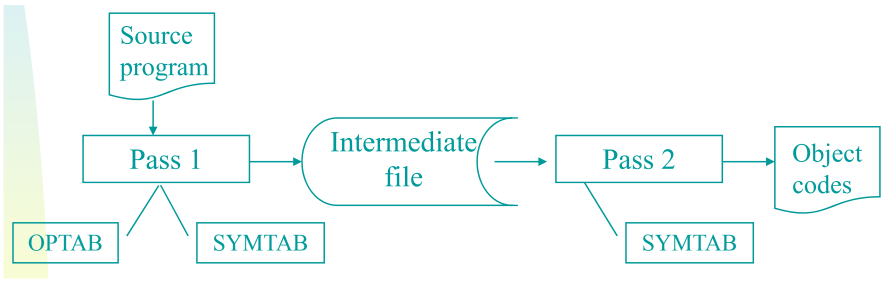
Note:

- Your project will be rejected if it does not comply with the above submission requirements.

- When you submit your project you get 5 marks until you prove that it is your own work, during a demo in Week 15.

1. **Project code: CC410-01: Assembler**

Use the pseudo-code in Figure 2.4 to implement 2-pass assembler using any programming language of your choice. Choose a platform, such as the SIC machine, Intel, AMD, ... etc, and use the appropriate OPTAB (Appendix A for SIC and SIC/XE), and the appropriate instruction format and addressing modes. The assembler should take an input a source code written in assembly language, produce the SYMTAB an intermediate file in pass 1, then the object file (machine code) in pass 2. Test your program using some test assembly programs, and do manual reverse engineering for the generated machine code, to prove the correctness of the implementation.



2. **Project code: CC410-02: Loader**

Implement an absolute and relocating (relative) loader using the pseudo-code in Figure 3.2 and the code in Figure 3.3. Use a machine code example as input and show how your project will simulate its loading in memory.

3. **Project code: CC410-03: Linking Loader**

Implement a 2-pass linking loader using the pseudo-code in Figure 3.11. Use a machine code example as input and show how your project will simulate its linking and loading in memory.

4. **Project code: CC410-04: Macro-processor**

Implement a one-pass macro-processor using the pseudo-code in Figure 4.5. Use a source code file as input to test how your project will expand and process the code.

5. **Project code: CC410-05: Design a Language Grammar**

Design a programming language of your choice:

- Start with expressions and operators, work upwards to statements, then to functions/classes etc.
- Decide a list of what punctuation is used for what.
- Define syntax for referring to variables, arrays, hashes, number literals, string literals, other built-in literal.
- Define your data naming model and scoping rules.
- Check whether your grammar makes sense focus on a level (literal/variable, operator, expression, statement, function etc.) and make sure that punctuation and tokens from other levels interspersed or appended/prepended is not going cause an ambiguity.
- Finally write it all out in EBNF and run it through ANTLR or similar.

6. Project code: CC410-06: Implement a lexical analyzer for a language

Using an existing programming language, or work with the group working on project CC410-05, to design the lexical analyser for the language. You can use tools such as lex or flex for tokenisation. The input should be a source code using the chosen language, and the output should be the list of tokens and word analysis in a systematic presentation.

7. Project code: CC410-07: Implement a parser for a language:

Implement a parser (Phrase Analyser) for an existing language of your choice or use the one designed in Project CC410-05. You can also use the lexical analysis output from the group working on project CC410-06 while designing your parser. Use Yacc or bison for your development. The input should be a source code using the chosen language, and the output should be the syntax errors