AT Project

The course project is worth 15% of the final grade.

You may work in groups of one, two, three, or four. The complexity of your task does not depend on the group size, which means that forming a group of three will reduce your workload. A group should submit one program and one final report, and presentation. All members of the group must present some part of the project. You may discuss ideas and algorithms with other groups, but you cannot share your code.

Task selection: First, you need to form a group, select a task for your project and post it on the Canvas in the Discussion field. **Do not** choose the topic which is already chosen.

Each group should have a unique project.

Project proposal (Oct. 26, 2 points): You should submit a one-page proposal by 11:59pm on Sunday, Oct. 26. It should describe the selected problem, intended approach to solving it, and methods for evaluating your results.

Final report, slides, and codes (Nov. 20, 9 points): The final report should include the description of your task, summary of results, main conclusions, and discussion of any surprising discoveries (3-5 pages, font 12, double space).

Demonstration and presentation (4 points): The project concludes with a 15-20 min presentation held during the classes.

Submitting your assignment

- Submission via Canvas Assignment.
    - It is your responsibility to submit your report, code, and presentation in a timely fashion.
- All files should be zipped together.
- A report that presents the performance evaluation of your solution.
    - The report should be properly formatted (an academic format style, such as ACM or IEEE being preferred) and contain quantitative data along with you analysis of these data.

Late Submission Policy

- Late work will be not accepted.

Some project Ideas

1. Rascal Meta Programming Language: https://www.rascal-mpl.org/
    a. The goal is to:
    b. Define a simple language with Raskal MPL. It could be the definition of some DSL (domain specific language) or could be some predefined syntax but students should define their own interpretation (compilation) of the code (for example, Java -> SQL, etc.). Language definition contains two parts, one which defines parse tree (concrete syntax) and AST (imploded from parse tree)
    c. Rascal has embed syntax for Java, so another task could be the extension of java syntax with some syntactic sugars.
    d. Code analysis. Instead of defining custom language, extending existing one or defining custom code generation, students can select to perform code analysis - an AST traversal with gatherings of code metrics.
    e. Java code transformations with Rascal MPL.
    f. explain theoretical backgrounds behind Rascal MPL: based on studying https://homepages.cwi.nl/~paulk/publications/rascal-lncs.pdf
        i. The closest aspect to Automata theory is task 1 - defining custom context-free grammar. Other aspects are mostly about AST/parse tree traversal. In proposal students should select concrete task and explain their goal.

2. Regular Expressions:
    1) Using some mainstream language Java, C#, javaScript, python (or classic for regex Perl), develop a bunch of examples of regex applications. It could be even real world data mining examples:
    2) Simple web crawler - request a web page and sue regex to extract emails, phones, addresses, urls, person names (etc. - up to you what do you want to mine). Form an url graph - node should have mined metainfo, links should be based by url. Display the graph in console or graphically. In you presentation describe all your regexes (should be based on corresponding RFCs), compare to html parser approach.
    3) Mail mining (more advanced) - you need to have set up imap/pop client - could be done with google account (probably could be done with USF account). Mine email information using different regexes.
    4) Students should present advanced regex syntax (look-ahead, look-behind and all other features supported by selected host language)

3. SDF3 - https://www.metaborg.org/en/latest/
   Describe the rules of SDF3 on example of your simple DSL. Students should develop a context-free grammar with SDF3 explaining all the features and how they defined production rules. Then, they should develop several examples of programs and present and explain generated ASTs in class. More advanced: implement AST evaluation.

4. Stratego meta programming language. https://www.metaborg.org/en/latest/

Use defined Java grammar in sdf3 and write several code re-factorings (variable, method extraction, Boolean expression transformation etc) with stratego. Students should explain in class their re-factorings and how Stratego works! (Note: Stratego has very trick logic).

5. ANTLR (ANother Tool for Language Recognition) is a powerful parser generator for reading, processing, executing, or translating structured text or binary files. It's widely used to build languages, tools, and frameworks. From a grammar, ANTLR generates a parser that can build and walk parse trees.

6. Lex/Yacc
   Define the grammar of your language in lex/yacc - explain how parsing works in class. How compiler capture lex/syntax errors and recover from
   (Note that in Compilers class students works with these tools)

7. Automata in Compilers (How lexer uses DFA/NFAs. How parser is controlled by DFA)

8. Automata and DNA Computing
   a. T. Krasińskiand, S. Sakowski, "A theoretical model of the Shapiro finite state automaton built on DNA," Theoretical and Applied Informatics, vol. 18, pp. 161-174, 2006.
   b. T. Krasiński and S. Sakowski, Extended Shapiro Finite State Automaton Built on DNA.
   c. Uniwersytet \Lódzki. Wydzia\l Matematyki i Informatyki, 2008.
   d. T. Krasinski, S. Sakowski, and T. Poplawski, "Autonomous push-down automaton built on DNA," Informatica , vol. 36, no. 3, p. 263+, Sep-2012.
   e. Y. Benenson, B. Gil, U. Ben-Dor, R. Adar, and E. Shapiro, "An autonomous molecular computer for logical control of gene expression," Nature , vol. 429, no. 6990, May 2004.

9. Cellular automata as NFAs/DFAs (Game of Life simulator known as Golly).
10. LL, LR, LL(*), GLR, SLR-parsers
11. Rewriting systems
12. Finite-State Transducers
    a. Mealy Machine
    b. Moore Machine
    c. Equivalence of Mealy and Moore machines
    d. Minimization of both
    e. Limitations of Finite-State Transducers
13. Computability/Decidability and Halting problem
14. NP-completeness and reducibility

15. Natural language processing

16. Langton's Ant. Langton's Ant is a two-dimensional universal Turing Machine.

    a. Implementation and visualization of procedure Context-Free Grammar (CFG) to Push-Down Automata (PDA): (Convert CFG to Greibach normal form and then use the procedure CFG to PDA)

17. Implementation of procedure PDA to CFG

18. Implementation and visualization of the procedure NFA into DFA.

19. Implement a membership algorithm for CFG: Convert CFG grammar to Chomsky normal form and implement CYK algorithm

21. Probabilistic Automaton and Stochastic Languages

22. Your own topic