CSC 4330: Programming Language Concepts - Project Ideas and Rubrics

The project ideas listed below are intended to serve as inspiration and a reference for your projects. You are welcome and encouraged to propose your own project ideas that align with the learning objectives of the CSC 4330 course. If you have a different project idea in mind, please discuss it with me for approval.

1. Interpreter for a Simple Programming Language

- **Description**: Students can create a basic interpreter for a simple programming language. This language can include fundamental features such as arithmetic operations, conditional statements (if-else), and loops (while, for).
- **Learning Outcomes**: Understanding syntax and semantics, parsing techniques, tokenization, and the execution of code.
- **Tools**: Python, C, or C++.

2. Type Checker for a Mini-Language

- **Description**: Develop a type-checker for a small programming language with features like variables, expressions, and functions. The project can involve defining rules for type inference and checking.
- **Learning Outcomes**: Understanding type systems, static vs. dynamic typing, and implementation of type-checking algorithms.
- **Tools**: Python, C, or C++.

3. Design and Implement a Domain-Specific Language (DSL)

- **Description**: Students can design and implement a simple DSL for a specific domain, such as mathematical expressions, data querying, or graphic generation. They can then write an interpreter or a compiler for this DSL.
- **Learning Outcomes**: Learning about the design of programming languages, syntax definition, parsing, and interpretation.
- **Tools**: Python for implementation; students can use libraries like PLY (Python Lex-Yacc) to handle parsing.

4. Garbage Collector Implementation

- **Description**: Implement a basic garbage collector (e.g., reference counting or mark-and-sweep) for a small interpreter or virtual machine.
- **Learning Outcomes**: Understanding memory management, automatic memory allocation, and garbage collection algorithms.
- **Tools**: C or C++.

5. Concurrency and Multithreading Project

- **Description**: Implement a small program demonstrating the use of threads, synchronization techniques, and handling race conditions. This could involve simulating a real-world scenario, like a multi-user banking system.
- **Learning Outcomes**: Understanding concurrency, synchronization, and thread safety.
- **Tools**: C or C++ (using pthreads or C++11 threads).

6. Building a Mini Compiler

- **Description**: Students can develop a mini-compiler that translates a simple high-level programming language into an intermediate code or directly into assembly code.
- **Learning Outcomes**: Understanding the compilation process, syntax analysis, intermediate code generation, and code optimization.
- **Tools**: C or C++.

7. Syntax Highlighting Editor

- **Description**: Develop a simple text editor with syntax highlighting features for a specific programming language. Students will work on lexing, parsing, and applying syntax rules to highlight text.
- **Learning Outcomes**: Tokenization, syntax parsing, and understanding how programming editors work.
- **Tools**: Python with a GUI toolkit like Tkinter or PyQt.

8. Static Analysis Tool

- **Description**: Implement a basic static analysis tool that can analyze source code to find simple errors or enforce coding standards.
- **Learning Outcomes**: Understanding static analysis, program analysis techniques, and abstract syntax trees.
- **Tools**: Python.

9. Virtual Machine for a Custom Bytecode Language

- **Description**: Create a simple virtual machine that can execute bytecode for a custom-designed bytecode language.
- **Learning Outcomes**: Understanding how virtual machines work, bytecode interpretation, and execution.
- **Tools**: C or C++.

10. Implementing Regular Expression Engine

- **Description**: Build a simple regular expression engine that can parse and match regular expressions with input text.
- **Learning Outcomes**: Understanding automata theory, pattern matching algorithms, and regular expressions.
- **Tools**: Python, C, or C++.

Rubrics

Criteria	Weight	Description
Problem Definition and Complexity	20%	*Clarity and relevance of the problem statement, scope, and objectives of the project. Complexity and challenge level of the problem being addressed.*
Design and Planning	20%	*Effectiveness of the design approach, including architecture, data structures, and algorithms used. Quality of the planning, including project milestones and timeline.*
Implementation and Functionality	30%	*Correctness, completeness, and robustness of the implemented solution. Proper use of programming language features and adherence to coding standards.*
Conceptual Understanding	10%	*Demonstration of a strong understanding of programming language concepts covered in the course. Ability to explain the concepts used in the project.*
Innovation and Creativity	10%	*Creativity in the approach, novel solutions, or innovative use of programming language concepts.*
Documentation	10%	*Quality and clarity of documentation, including comments in code, user manual, and technical report explaining the project's design, implementation, and usage.*
Presentation and Demonstration	10%	*Effectiveness of the project presentation, clarity in explaining the project objectives, approach, and results. Quality of the demo showing the working project.*

Additional Notes:

- **Problem Definition and Complexity**: Projects that tackle more complex or challenging problems will be recognized, provided the problem is clearly defined and within the scope of the course.

- **Design and Planning**: Students should demonstrate thoughtful planning and design choices, including decisions about which data structures, algorithms, and programming paradigms to use.
- **Implementation and Functionality**: The project should function as intended, be free of major bugs, and be built using good programming practices. Proper error handling and testing are expected.
- **Conceptual Understanding**: Students should be able to articulate how their project relates to the programming language concepts discussed in class, such as syntax, semantics, type systems, etc.
- **Innovation and Creativity**: Creative projects that apply concepts in new or unexpected ways will be rewarded. Students are encouraged to think outside the box.
- **Documentation**: Proper documentation is crucial. The project report should include an overview of the project, design choices, implementation details, and how to run the project. Inline comments in the code should explain non-obvious sections.
- **Presentation and Demonstration**: Students will be required to present their project to the class and demonstrate its functionality. The presentation should be clear, concise, and engaging, showcasing both the process and the final product.

Submission Requirements:

- **Project Code**: Submit the complete source code, with appropriate comments.
- **Project Report**: A written report (approximately 5-10 pages) detailing the problem, design, implementation, and how it aligns with programming language concepts.
- **Presentation Slides**: A set of slides used for the project presentation.
- **Demo**: A live demonstration of the project during the presentation.

Project Deadlines:

As mentioned in Course Overview and Lecture Plan (already shared on iCollege)