**Language Design Proposal Template**

**Student Name(s):** Your name(s)

**Language Name:** The name of your language

**Target Language:** The output language for your compiler. Can be anything (assembly, LLVM bitcode, JVM bytecode, CLR bytecode, JavaScript, etc.). See the project information page for details.

**Language Description:** a description of the language, from a high level. Why this language? What can this language do?

**Key Features:** The key features of this language. The project information has more information about what sorts of things you can put here. If you have too few, I'll suggest additional features you could add to make it complex enough. If you have too many, I'll suggest features to take away or replace. Generally, you should target about 2-3 key features. To be honest, exactly what a "key" feature is here is largely a feeling; put what you want and we can discuss further.

**Planned Restrictions:** is there anything that would make this language impractical to actually use? It's expected that you'll have something here. We don't have enough time to make a fully-fledged compiler and language, only enough time to get a complete start on it. For example, you might generate very inefficient code.

**Suggested Scoring and Justification:**

The percentages each component will be worth in terms of your overall course grade, based on an estimate of how much work you anticipate it will be. These numbers should add up to 80% in total; the remaining 20% is from your language design proposal, user documentation, and presentation.

* **Lexer**: Typically between 2 - 10%. Lexers which only handle reserved words, identifiers, and integers should only be 2-3%. Lexers which handle strings and comments will be worth more. Lexers which handle string interpolation will be worth the most. The more you need to diverge from the example lexers, the more this will be worth.
* **Parser**: Typically between 5 - 20%. Parsers which only handle S-expressions will likely be 5-8%. Parsers for complex grammars will range between 10% and 20%, depending on how much your grammar diverges from the example grammars.
* **Typechecker**: Between 10 - 50%. Typecheckers for C-like languages will likely be closer to 10%. As more features are added, this increases, especially if you add features which are not covered by an example language. Large type-level features like generics and type inference will be worth the most.
* **Code Generator:** Between 10 - 50%. Code generators for low-level targets will likely be closer to 50%, especially if you're translating features that aren't in an example. Code generators for high-level targets might still be worth a lot, as long as the target doesn't natively support the feature, or you intentionally plan to translate it in a way that doesn't use the target language's features.

**Syntax:** EBNF definition of the concrete syntax of your language. It's ok if this changes later. You should annotate non-obvious parts with what they mean. Each one of your key features should be represented somehow in this syntax. For example, if you have structs in your language, there should be syntax to create and access structs.

**Example:** Some example code that uses your language to do something.