In Software Engineering, common recurring problems are solved using Design Patterns. An argument against Design Patterns is they are a sign of weakness in the design of a programming language. Design Patterns are thus used to circumvent the weakness of the language to allow the programmer to express and modularize their code. The problem is to create a programming language which eases or eliminates the need to implement design patterns.

There is many programming languages in existence which do not contain features that eliminate design patterns. Several languages do provide features which eliminate a couple of design patterns, but none of the languages focus on tackling the solution of eliminating design patterns.

- The system will always start operating on the code declared in the main function first.
- When the system reaches a return statement, all code after it is guaranteed to be not executed.
- A function can be called as many times as the user wishes.
- All primitives will allow operators to operate on them.
- A binary operator will return a value after operating on two variables.
- A unary operator will return a value after operating on a variable.
- A branch will control the flow of execution depending on a condition.

The system uses the Pipe and Filter system architecture pattern. The architecture is based on each submodule being fed input and then the output is fed to the next subsystem like a pipe. The Lexer subsystem is fed the source code and outputs tokens which are then the input for the Parser to output an abstract syntax tree.

The C language is still a young and evolving language, but hopes to implement solutions to all design patterns. With the familiarity of the C programming language family, programmers will easily adapt to using this language.

The lexer is an important part of a compiler to transform the source code of a language into tokens. While the parser then takes the tokens and define the syntax of the language and constructs an abstract syntax tree.