Implementation plan

Compiler

for the JiPi programming language

TDP006

Jimmy Dahl
IP1
2008-02-07

Compiler

To translate code written in the JiPi programming language to the Ruby programming language there will be a compiler. A compiler is a program that translates a predefined language to another language, often a language that a computer will understand.

To do this the compiler finds keywords in the specified programming language and generates code for it. This process is split in several parts. The first part is the lexical analyzer that splits the input code into tokens. This part will also count the lines of the code to enable better error codes.

The second part is the parser. The parser is the process of analyzing the sequences of tokens from the lexical analyzer to determine its grammatical structure with respect to a given grammar. The parser will construct a parse-tree which will be transformed to Ruby source code by an evaluator.

Lexical analyzer

The lexical analyzer shall discover and report all keywords, operators, punctuations and identifiers. It shall also discover and report syntactically faulty tokens, but it won't discover syntactically structural faults, like the expression `a = b +.` The lexical analyzer will also remove white spaces like spaces, line-feeds and tabs.

Parser

The parser is to discover the syntactical structure of the code. It will get the tokens from the lexical analyzer and build a syntax tree from it. The syntax tree is a treelike representation of the structure of the program. The "branches" of the tree describes how the structures of the program are linked together. When `if` and `for` statements start and end, and which blocks are kept within.

The parser will also check for syntactically structural faults and throw error codes when encountered. Semantic faults, like the `a = (b == c)` expression, will not be captured here.

Symbol tables

To make it possible to find keywords in the lexical analyzer and build the parse tree in the parser they will have access to two tables of symbols. The first table will contain all the languages keywords and operators and will be used to find the important tokens that the parser will need. The second table will contain all symbols, like variables and function symbols, used in the program that's being compiled. The parse tree tells how these symbols belong together, and together they will form the complete program structure. For symbol tables the compiler will use hashes.

Semantic checker

When the parsing is done the code will be checked for semantic faults. During this process the compiler will check that all operators has the right amount of operands, that the expressions are complete and that the statements are complete and has an end point.

Evaluator

When all other processes are finished the code is ready to be translated to Ruby. The evaluator will use the symbols table and the parse tree to build a complete ruby source code file.

Implementation

To get all these things working I will implement small bites separately, try to make every minor part working well first and then try to make them work together.

Firstly I will implement the numerical part of the language, building a simple calculator compiler only accepting numerical expressions and without functions and control structures. Then I will implement the same thing for string manipulation.

When both work fine I will firstly extend them with control structures and then put them together to one compiler handling both numbers and strings. Then there’s just to implement the function definition and the compiler is complete.

The building of every new compiler part will also be divided in several parts to simplify the development. I will try to make the lexical analyzer, parser, semantics checker and evaluator to work separately.