Architecture of nqcc:Compis Compiler

A compiler for a subset of C language.

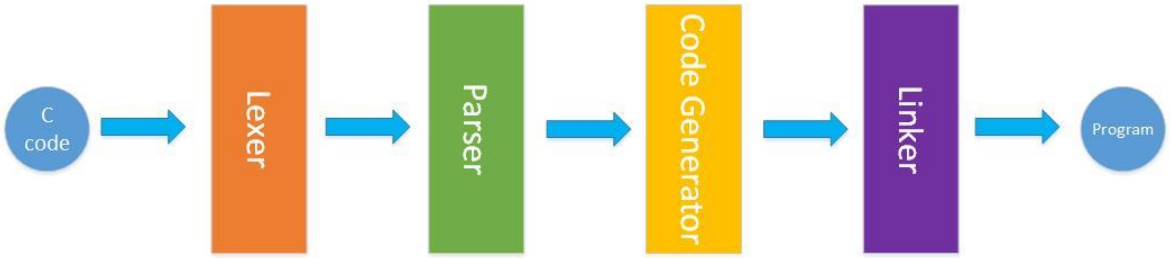
The Compis Software Company

Version control

Date	Version	Author	Organization	Description
24/03/19	1.0	Miramontes Sarabia Luis Enrique	Compis Software Company	Definition of the architecture for the first delivery.

Architecture definition

The nqcc:Compis compiler is designed following the structure as shown in the diagram below.



Where the Lexer will generate a token list that will be the input of the Parser. The Parser will output an Abstract Syntax Tree that is going to be required by the code generator to create the assembler code for the linker to generate the output program.

This being the most basic architecture of a compiler and it does not includes a module for code optimization.

The compiler will be developed in Python 3.6.5 and having as target Unix and Unix-like operative systems, in this case OSX 10.11.4 El Capitan.

Defining the modules

1. **Lexer**: The lexical analysis will be taken care by this module, as input it takes the following subset of the C language.

```
int main()
{
        return 2;
}
```

As output, the Lexer module will generate a list containing the following possible tokens:

- keyword_main
- keyword_return
- keyword_int
- parentheses_open
- parentheses_close
- bracket_open
- bracket_close
- semicolon
- (integer, numeric_value)*

*Where numeric_value will take the value of the number recognized by the lexical analysis.

2. **Parser**: The parser will be on charge of the syntax analysis for the compiler, as input it will take the token list previously generated by the Lexer, as output the Parser will generate an Abstract Syntax Tree implemented using the treelib module for Python.

3. **Code Generator:** The code generator will take care of the creation of the assembler code; in this case, the AST generated by the Parser is employed by the code generator and will be traversed using Deph-First and Post-order to generate the assembler code.

4. **Linker:** This module of the compiler just takes the assembler generated by the code generator module and uses the GNU Compiler Collection (gcc) to generate the program and also delete the assembler code.

Code standards

For this project, PEP-8 is going to be employed as possible with the exception of wildcard imports that are going to be used to import all the functions of the corresponding modules to be used in the compiler main script (compiler.py).

Code metrics and inspection tools

For the code metrics, the tool of choice is Pylint, a tool that contains checkers for PEP8 code style compliance, design, exceptions and many other source-code analysis tools.

In addition, to make quicker the verification and automatize the correction of the code autopep8 is utilized, making quicker the adaptation to PEP8 format.

The results of the code metrics evaluated with Pylint are the following for each module of the compiler.

Module	Score
Compiler.py	3.72/10
Lexer_module.py	7.1/10
Parser_module.py	5.41/10
Linker_module.py	5/10
Code_generator.py	5.65/10

Note: The compiler.py module as consequence of the wildcard imports obtained the lowest score as expected.