All deliverables in "ELIXIR" language made in "C".

Integration plan: Team "Gold Compilers"

Compiler Foxy.

Contents

Purpose:

This document describes the integration plan for the first part of the C Language Compiler project in Elixir.

Target:

This integration plan shows the necessary software components that were used, in the programming language we are based on "Elixir", that have been used to compile the following program:

```
int main(){
return "constant";
}
```

Risk.

The greatest risk that can arise is not finishing the work on time, due to the lack of time we have, as well as the cohesion of the team. Similarly, it may not work optimally or does not meet the stated requirements.

Complexity.

The first part of this project is not very complex, perhaps the greatest complexity in the first installment is understanding how a compiler works and understanding the elixir language, as well as being in communication with the team.

Previous knowledge.

In this first installment, we still do not have an advanced level of knowledge of how a compiler works, since most of our members do not have any previous knowledge of the elixir language.

First of all:

To start developing compiler integrations the following resources will be needed:

- You must have the repository provided by the teacher: https://github.com/hiphoox/nqcc_elixir
- Review and read part 1, “Write a C compiler” by Nora Sandler: https://norasandler.com/2017/11/29/Write-a-Compiler.html
- Installation of the latest version of the Elixir language.
- Installation of the latest version of Visual Studio Code or failing any other text editor or IDE.
- You must know the commands of basic git commands and commands Ubuntu terminal.

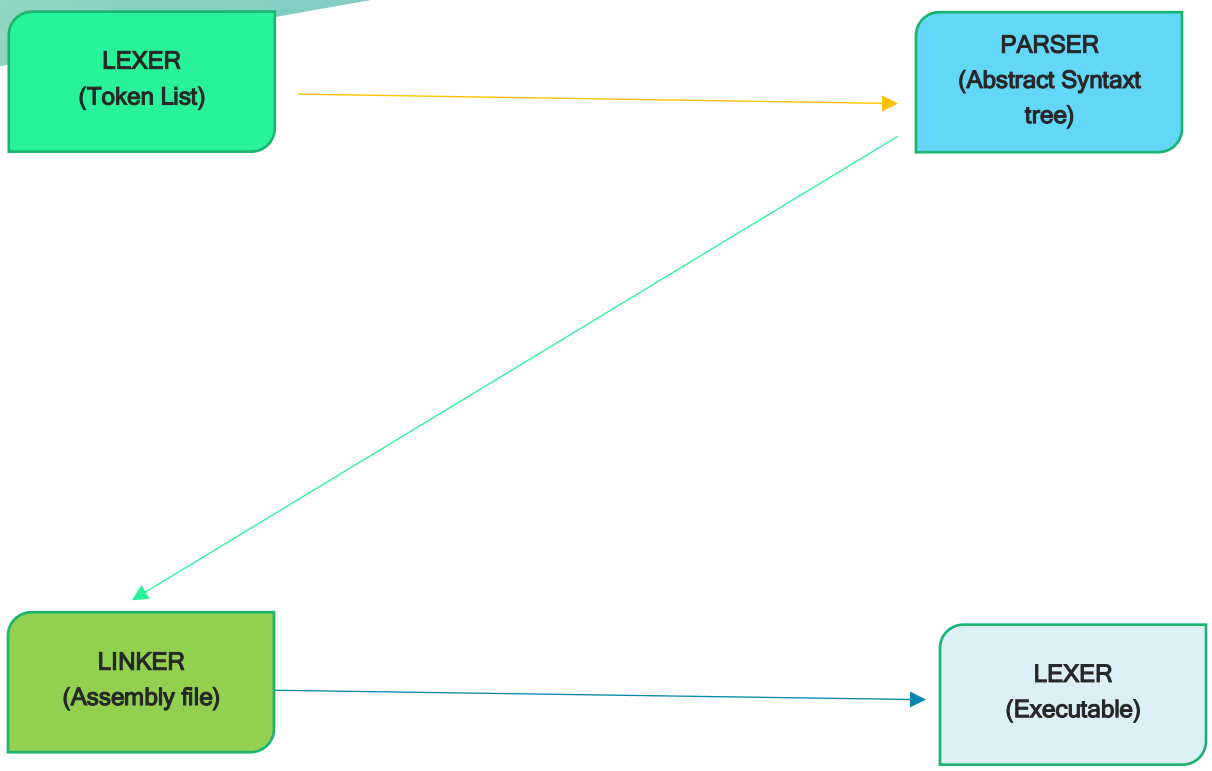
Integration plan.

We will start working from scratch, we will work in a repository to be able to work as a team and in the same way see our progress.

Integrations:	Description:
Lexer:	This integration will validate this: - Validate that list of tokens. - The output will be a list of atom strings tuples. - If there is an error, a list of tuples with the token will be displayed, as well as the wrong column and row.
Parser:	This integration will allow us to establish the following basic functionality:. - Generate an AST with the list of tuples created by the Lexer. - If there is some error, it will display a list of tuples with the token generating the error, the column, and the row.
Code generator:	This integration will allow us to establish the following basic functionality: - Take the AST generated by the Parser to build the code in assembler, from the leaves to the root. - The output will be a string with the representative code in assembler.

Linker:	Linker: is a computer System program that takes one or more object files generated by a compiler or an assembler and combines them into a single executable file, library file, or another 'object' file

How does it work?



Architecture.

This compiler has been designed to be as manageable as possible. By which it means that once it is working, its code should not be altered. To this end, they have been abstracted with a series of actors that simply operate on a series of files for the desired language. So, this architecture only works with this compiler.

What should the compiler do?

The main objective is to execute the following, but in multiple ways:

```
int main(){
return "constant";
}
```

The output should be an executable file, trying to follow the directions below and trying to do it the right way

Flag:	Must do:
-t	It returns the token's list from our compiled file.
-a	It returns the abstract syntax tree from compiled file.
-s	It must generate assembly file.
-c	It compiles de program.
-h	It shows help information about compiler's flags.

Our compiler will also detect if it is missing parentheses, braces, semicolons and if there is a syntax error.

Delivery date of the work:

Integers.	13-08-21	Our compiler must be able to return an integer, compiling a simple snippet code, doing multiple tests.
Unary operators.	13-08-21	Our Compiler must can return a result operated with unary operations like negate result, positive, bitwise and logical negation, doing multiple tests.
Binary Operators.	13-08-21	Our Compiler must can operate a binary operation like sum, subtraction, multiplication and division, doing multiple tests.

Historic:

Author	Description	Version	Date
García Felipe Miguel	First Version	1.0	16-07-21
García Felipe Miguel	Second Version	2.0	26-07-21
García Felipe Miguel	Third version	3.0	13-08-21