FIRST DELIVERY C LANGUAGE COMPILER IN ELIXIR

INTEGRATION PLAN

TEAM ASSEMBLY

Elaborated:

Integrator, José Enrique Hernández Zamora

Revision history

Date	Version	Description	Author
16/FEB/2020	1.0	Draft copy creation	José Enrique Hernández Z.
24/FEB/2020	2.0	Purpose and scope	José Enrique Hernández Z.
29/FEB/2020	3.0	Starting point and integration mode	José Enrique Hernández Z.
1/MAR/2020	4.0	Risk, Maturity and Complexity	José Enrique Hernández Z.
8/MAR/2020	5.0	Definition of integration 1, 2 3 y 4	José Enrique Hernández Z.
14/MAR/2020	6.0	Integration plan	José Enrique Hernández Z.

Contents

Introduction

Purpose

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

Scope

This integration plan shows the necessary software components, in the Elixir programming language, that were used to compile the following program:

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

Risk

The greatest risk that can arise is not finishing the job on time, due to a lack of many or few components.

The next important risk is to finish the project, but that does not work optimally or does not meet the stated requirements.

Complexity

The first part of this project is not very complex, but it is important to keep in mind in order to obtain a greater maturity in the next installments, as well as to reduce their complexity.

Maturity

In this first installment, we still do not have an advanced level of maturity (but it is sufficient) because none of the project participants has used the Elixir programming language before.

Start point

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

- Installation of the latest version of the Elixir language (version 1.10.2).
- Installation of the latest version of Visual Studio Code (version 1.42.1) or failing any other text editor or IDE.
- Review and read part 1, Write a C compiler, by Nora Sandler (web address attached to the references).
- Management of basic git commands.
- Review of the github nqcc elixir repository provided by the teacher (web address attached in the references).
- The project will start from scratch, but with the support of the teacher's repository.

Integration mode

We will use simultaneous implementation because we will start the project from scratch, but we will also base ourselves on progressive implementation, that is, we will use the provided repository to support us, because the maturity we have in this first installment is only enough.

Integrations

Integration one (Reader)

The first integration will allow us to establish the following basic functionality:

- Read the file in .c format and transform it into a string to pass it to the Lexer.
- Generate two maps, one with atom-expression tuples and the other with possible ASTs. These maps will serve integration three and four.

Integration two (Lexer)

The second integration will allow us to establish the following basic functionality:

- Divide the string received by the Reader into a list of tokens.
- Validate that list of tokens.
- The output will be a list of atom-string tuples.
- If there is some error, a list of tuples with the wrong token, column, and row will be displayed.

Integration three (Parser)

The third integration will allow us to establish the following basic functionality:

- Receive a possible AST from the Reader to optimize.
- 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.

Integration four (Code Generator)

The fourth 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.

Integration five (Writer)

The fifth integration will allow us to establish the following basic functionality:

- Take the string with the assembly language representative code and write it in a text file.
- The output will be a file ready to be converted into object code.

Integration six (Invoker)

The sixth integration will allow us to establish the following basic functionality:

- Provide the path and the necessary arguments to gcc to generate the compiled object code.

References

http://repositorio.utn.edu.ec/bitstream/123456789/527/19/04%20ISC%20156%20PLAN%20DE%20INTEGRACION.pdf

https://www.isotools.org/2014/04/09/como-desarrollar-un-plan-de-integracion/

https://www.isotools.org/2014/04/16/que-contenidos-debe-incluir-un-plan-de-integracion/

https://norasandler.com/2017/11/29/Write-a-Compiler.html

https://github.com/hiphoox/nqcc_elixir

https://www.isotools.org/2014/04/09/como-desarrollar-un-plan-de-integracion/