XENON COMPILER

Xenon

Development by Javeritos Inc.

National Autonomous University of Mexico
Faculty of Engineering
Computer Engineering

Compilers
Ing. Norberto Jesús Ortigoza Marqués

Developers
André Marqueda
Javier Solano
Alberto Castillo
Daniel Zarco

Xenon

C language Compiler

Compiler is an informatic program that traduce a program written on any high-level development language into an executable file. The compiler's building involucrate the division of process. Generally, these phases have grouped on two tasks: analysis of source program that, is involucrate in the frontend design, and two, synthesis of source object program involucrate in the backend.

1. Frontend: Part of process that realize the source program analysis, checking her validation, generate derivation tree, adding lexical analysis phase, and middle generation code.
2. Backend: Part to generating machine code, specifically the platform's architecture, part to the frontend's result.

This division allow generate machine code on self-backend to various development languages different among their, frontend have function of source program analysis to concrete development language, supervising machine source code generation on different platforms.

Main project's requirement

Main objective is compiling a source file written on C language with this content:

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

The output must be an executable file.

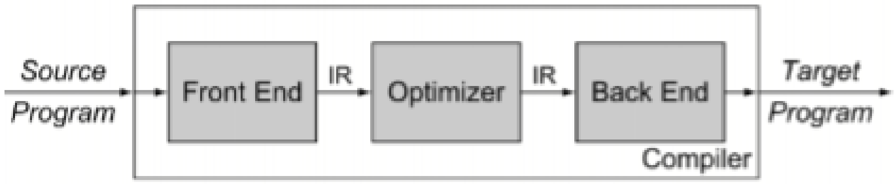
Secondary project's requirements

1. Execution must be form command prompt (terminal) based UNIX systems.
2. Executable file must have same name of compiled file and be place in the same directory of source code.
3. With compiler flag -o "executable's name" must be change name of executable file for another tipped from terminal, osseous, personalized name given by the user.
4. With compiler flag -s, must generating assembly file only, don't generated executable file.
5. With compiler flag -t, return a token's list from compiled file.
6. With compiler flag -a return Abstract Syntax Tree from compiled file.
7. With compiler flag -h, the compiler shows help information about compiler's flags.
8. Compiler must detect losses element like parenthesis, braces and brackets and syntax errors.
9. We give to user an instruction's set to operate and manipulate the compiler in user's manual. Also, give to user one document with Nora Sandler's test.
10. Version management on Git-hub platform.

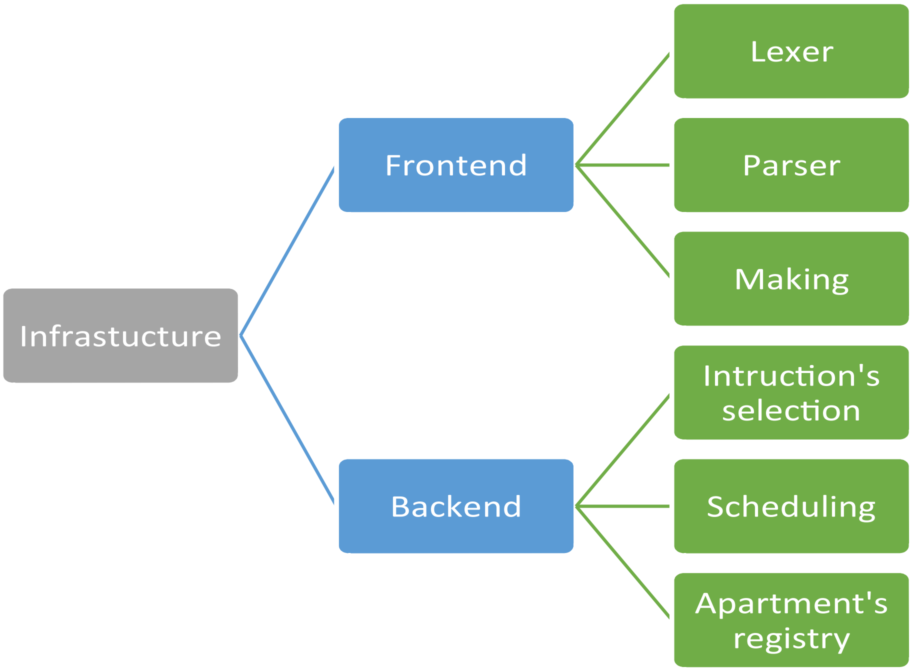
Architecture

Compiler objective is to preserve source code when compile process is running, that process is denominated semantic aspect, besides, if is possible, upgrade input program as output, be more efficient in possible, take up all tool of environment.

Now, compiler's structure was defined, following diagram shows the compiling process:



Our compiler has two main branches, frontend and backend. Frontend has many functions, check syntaxis or semantic rules following significance in source code. Also, backend must communicate with assembler or mode of system that communication with the computer.



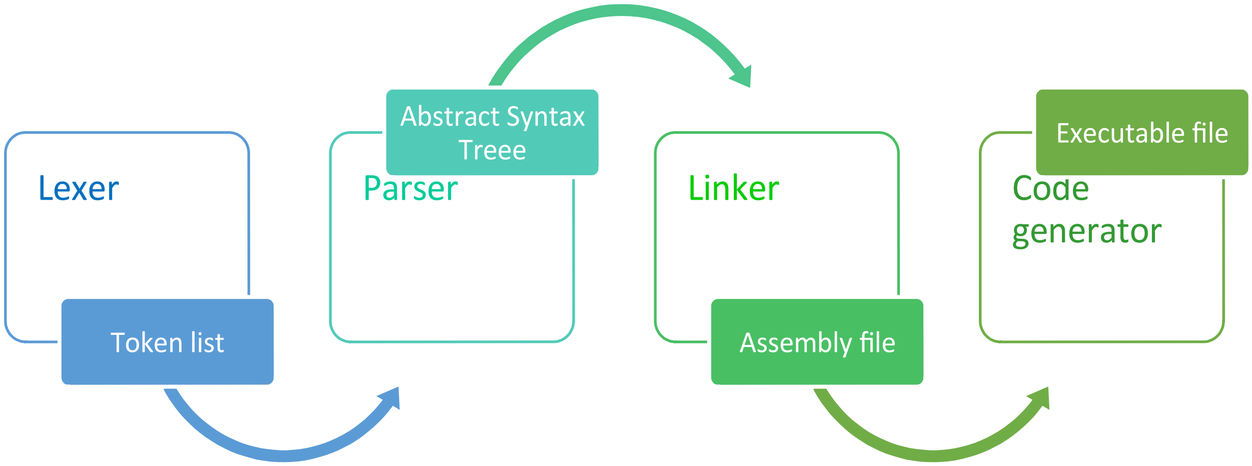
Implementation

Implementation is based on standard building compiler, lexical analysis on lexer, this part, doesn't matter order in the program, only will check lexical components among characters with coherent significant in specific development language, returning a list of elements without black spaces, denominated **tokens** contained in a single list.

Continues compiler process, one token's list was returned, the parser received list, starting syntactical process following on **Abstract Syntax Tree** (AST), choosing this structure to can easily manipulate syntactical information of source code.

Next phases of compiler, AST must have complete level to extract and organize structure of operations, comparisons and results. This's hierarchical relationship.

Finally, on code generator has generate assembly code task, leading assembly code generation as linker.



Architecture's process	Output
Lexer: is the process of converting a sequence of characters (such as in a computer program or web page) into a sequence of tokens (strings with an assigned and thus identified meaning).	**Token list:** list of characters associated to atomic word called token. Every token has the identifier function
Parser: is the process of analyzing a string of symbols, either in natural language, computer languages or data structures, conforming to the rules of a formal grammar.	**Abstract Syntax Tree**: is a tree representation of the abstract syntactic structure of source code written in a programming language. Each node of the tree denotes a construct occurring in the source code
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	**Assembly file**: The assembler is used to convert an assembly file into an object file (.obj extension). The assembly optimizer and the compiler are used to convert, respectively, a linear assembly file and a C file into an object file. The linker is used to combine object files, as instructed by the linker command file (.cmd extension), into an executable file

Code generator: is the process by which a compiler's code generator converts some intermediate representation of source code into a form (e.g., machine code) that can be readily executed by a machine	**Executable file**: referred to as an executable, causes a computer "to perform indicated tasks according to encoded instructions",[1] as opposed to a data file that must be parsed by a program to be meaningful

Work's plan

Phase's name	Release's date	Scope
Integers	17 March 2020	Compiler must can return an integer, osseous, compiling simple fragment code.
Unary operators	14 April 2020	Compiler must can return a result operated with unary operations like negate result, positive, bitwise and logical negation.
Binary Operators	12 May 2020	Compiler must can operate a binary operation like sum, subtraction, multiplication and division.

Even more binary operator	4 June 2020	Compiler must can return the result written in source code ass mix of back phases. Also must support and operate logical operation like ass AND, OR, >, <, =, <=, >=, etc.
		int main( ) {
		return 3+4 <= 4 || 1&&2 != 3 > -6;
		}

Scheduling Plan

Phase	Date	Activities
First release: Integers	Week 1 10 February at 15 February	Install Elixir, Git and Visual Studio and make first test to start learning and be more familiar. Test how configurate git, create repository, do push and pull.
	Week 2 17 February at 22 February	Divide compiler project on session to developing among team. Division was in modules to weekly session on Saturdays.

	Week 3 24 February at 29 February	Complete first phase, called lexer and developing other phases, starting parser developing. Writing Business Requirements Document take up Nora Sandler’s web. Check and revision by Ing. Norberto Ortigoza.
	Week 4 2 March at 7 March	Scheduling work plan again by recommendations of Ing. Norberto Ortigoza. Division compiler modules to individual team’s integrant and build the input and outputs from every module.
	Week 5 9 March at 14 March	Complete developing of all module’s compiler. Testing compiler and write documentation like Project document, presentation, user’s manual and add more detail to Business Requirement Documents from Noberto’s recomdations.