XENON COMPILER

Xenon

Development by Javeritos Inc.

National Autonomous University of Mexico
School of Engineering
Computer Engineering

Compilers
Ing. Norberto Jesús Ortigoza Marqués

Developers
Narváez Marqueda Ricardo André Sebastián
Solano Tavera Francisco Javier
Castillo Reyes Alberto
Zarco Manzanares Daniel Alberto

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

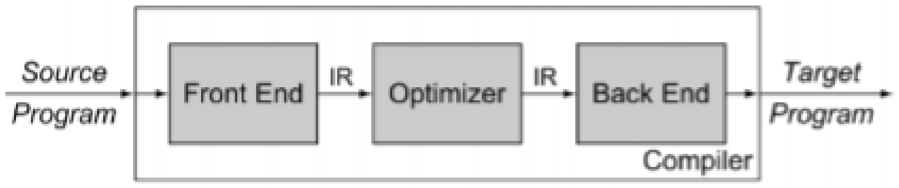
1. Execution must be form command prompt (terminal) based UNIX systems.
2. Executable file must have same name of compiled file and be place in the same directory of source code.
3. With compiler flag -o "executable's name" must be change name of executable file for another tipped from terminal, osseous, 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.

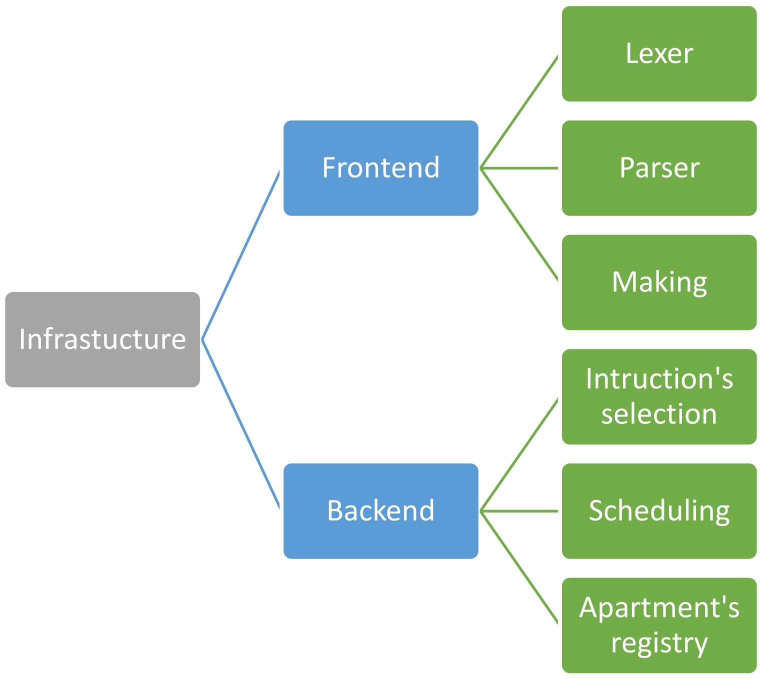
Architecture

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:

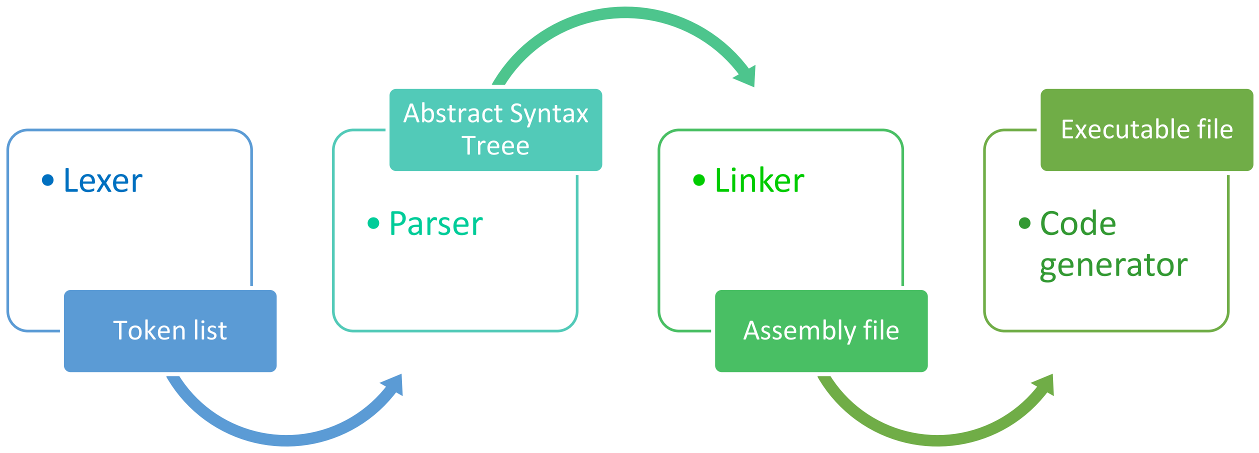


Our compiler has two main branches, frontend and backend. Frontend has many functions, check syntaxis or semantic rules following significance in source code. Also, backend must communicate with assembler or mode of system that communication with the computer.



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	25 to 29 May 2020	Compiler must can operate a binary operation like sum, subtraction, multiplication and division.
Even more binary operator	8 to 12 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 to 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 to 22 February	Divide compiler project on session to developing among team. Division was in modules to weekly session on Saturdays.
	Week 3 24 February to 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 to 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 to 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.
Second release: Unary Operator	Week 6 16 March to 21 March	Discuss design and structure to next phase. Unary operator is bitwise operator, negation operator and support negative numbers. Is fundamental using complement to one to do operations.
	Week 7 23 March to 28 March	Development bitwise operator and execute first test to success get requirements. We make little modifications to document.
	Week 8 30 March to 4 April	Development negation operator with negative number support. Test and registering result on xenon compile's document.
	Week 9 6 April to 11 April	Finish development of second release, where every operator has a success results of test. Register result and add to document results.
Third release: Binary Operator	Week 10 15 April to 22 April	We had planned how can implement new requirements from this third phase.

	Week 11 22 April to 29 April	Development first approach to can operate sum and rest, first development on sum has a success development.
	Week 12 29 April to 6 to May	Development of rest or minus operation, where changing some background operations of sum. Implementation was success.
	Week 13 6 May to 13 May	Development of multiplication operation, having some troubles with recognized flow process. Implementation was success.
	Week 14 13 May to 20 May	Development of division or div operation, some cases was a trouble for make success implementation. Implementation was success.
	Week 15 20 May to 27 May	Test all binary operator adding all cases from past phases. Implementation was success.
Final release: Even more binary operator	Week 16 28 May to 3 June	Add rest of binary operator as comparison, equal, not equal, greater than, less than, greater equal than, etc.
	Week 17 3 June to 10 June	Complete documentation, compiler and roadmap.
Total of weeks	17 weeks	Finish

Conclusions

Narváez Marqueda Ricardo André Sebastián (System Architect)

The purposes of the course were fulfilled by applying the theoretical concepts of a compiler. During development we had the opportunity to work with a new (functional) programming paradigm, which initially caused a bit of conflict due to our previous courses and the paradigms seen in previous courses. As an educational experience, I consider that it was an invaluable contribution to my academic development, and I hope to be able to later work with languages with this paradigm. I do not want to elaborate on Elixir (the applicative part), but I do want to comment that it is a language that "makes you think". Regarding the compilers topic, I learned the phases of a compiler and how the courses that I have been taking during my career were integrated (Structure and programming of computers, formal languages and automata, Data structures and algorithms 1 and 2, programming fundamentals). It was very gratifying to see how "the pieces of the puzzle" were coming together.

In the project subject, I learned to use Git and Github, both of which I knew existed but had never used in a school project, and now I consider them extremely useful tools for my subsequent subjects. In addition to this, due to the modularity of the project, we did not have many problems in implementing our project in a satisfactory way, and again, referencing Git and Github (particularly the second one), it was easy to support my colleagues and provide them with support. when the situation warranted it.
Finally, I think I took many fundamental things with me in this course, not only about compilers, and the reinforcement of knowledge that we had during the course, but that this course "planted a seed" and I am sure that my perspective on programming changed.

Zarco Manzanares Daniel Alberto (Project Manager)

This project was successful cause our teamwork and enthusiasm to moment have in front us a challenge. Firs, the paradigm programming of functional is new for me and this was big challenge. When project start, I need motivation and tips to can development my project's part. And these supports were bringing to me by my team. Every challenge brings us a learn, knowledge and experience for future project and subjects.
My work team was be incredible and very hopeful, thanks of work team, we can successful complete this project. I learn the importance of version tools like Git and GitHub, past subjects, I do not have experience or knowledge about these tools. Elixir is awesome, simple, and fast but is very hard think with another development paradigm because we have a structural and oriented to objects paradigms in our mind since we start our engineering degree. Was a nice experience for my work vision and view of corporative work in outside academic world.
Another experience with high value is to learn a develop any project with modules, or modular perspective, this work form bring more efficiency to project and optimize all scheduling plan.

Solano Tavera Francisco Javier (System Integrator)

This subject was somewhat different from what I was used to in other subjects, and what I can say is that I liked the way of working because it was an interesting experience, from assuming in role that in my case it was being the Integrator, which took me good knowledge because I didn't know how to interact with a very good tool like Git.
I learned to use Git and a new language that was Elixir, which I never imagined was so powerful. I really had not had the pleasure of knowing Elixir and I must admit that it was difficult for me to understand it and to be able to change my mind and perspective of how to program in it.
At first, the project seemed easy to do and understandable, but as the weeks and deliveries went by, it became difficult and complex for me because I consider that there are many

issues that come to court to understand and do the project, in addition to that it is important to have a good foundation and other knowledge for this subject.
Without a doubt, I took a great learning, and not only of elixir, but also of the use of other tools, of how to make a project in a more professional way, of organization and teamwork, of the organization for exhibitions and deliveries, in addition to studying the Compilers in a unique way, with a different format than other teachers. I take a great learning from the teacher who has great knowledge and experience, these learning not only of the subject, but also of other aspects, which are helps to develop as a professional and as a good engineer.
Something important to consider was the development of this semester, because much of it was done remotely and despite that there were no problems, thus demonstrating that, thanks to technologies, it was possible to take advantage of them to achieve the objectives of the matter, so it is also a great teaching that this stage of the pandemic has left us.
Finally, I want to mention that I am pleased that there are teachers who share their knowledge and promote learning with new technologies and tools that can help us later in a workplace, which was the case of teacher Norberto Ortigoza, thank you.

Castillo Reyes Alberto (Tester)

From my perspective as a student, this project was very helpful, since I was able to understand the role of the person responsible for carrying out the tests of a system, software, or project, defining the scope, complying with quality, and above all , from a new programming perspective. By using elixir, I was able to verify a very different thought, since functional programming is one of the programming paradigms that exist in the world of computing.
In the programming of the parser, which is the next module of the lexer, it was a challenge to be able to visualize how it would work given the different phases of the compiler, since even if it is the same structure in the programming, the binary deliveries are extremely different, from the programming and structure of the unary operators, since the former have two children, unlike the latter. This is extremely interesting, since it allowed me to understand how a compiler works, its structure and its different working modules.
The binary operations had greater programming effort, since they require a processing flow and precedence of the tree that is being generated. Being a similar structure, it is different in precedence.