UNIVERSIDADE FEDERAL DE LAVRAS 30-05-2016
GCC-130 Compiladores

# Construção de um compilador para a linguagem C-.

**Carlos Henrique Pereira [1], Bruno Lucinda Ribeiro[1], Gabriel Rodrigues Oliveira[1].**

*Keywords : lex, C-, lexical analysis.*
*Versão : 1.0*

## 1. Resumo

Este trabalho foi proposto pelo professor Ricardo Terra Nunes Bueno Villela docente da Universidade Federal de Lavras e é refente aos estudos feitos na disciplina Compiladores (GCC130) ministrada por ele até a presente data.

O trabalho consiste em construir um compilador para uma linguagem preestabelecida (C-) com o intuito de aumentar a participação prática dos alunos no conteúdo aprendido na disciplina.

## 2. Introdução

Este documento será modificado e arquivado em versões ao termino de cada etapa de desenvolvimento proposta.

O trabalho proposto dividi-se em três partes fundamentais, a construção análise léxica, análise sintática e a análise semântica.

Na construção do analisador léxico utilizamos o *lex* [1] que possui rotinas implementadas que constituem no mínimo automato finito determinístico (AFD) resultante da declaração das expressões regulares.

## 3. Análise Léxica

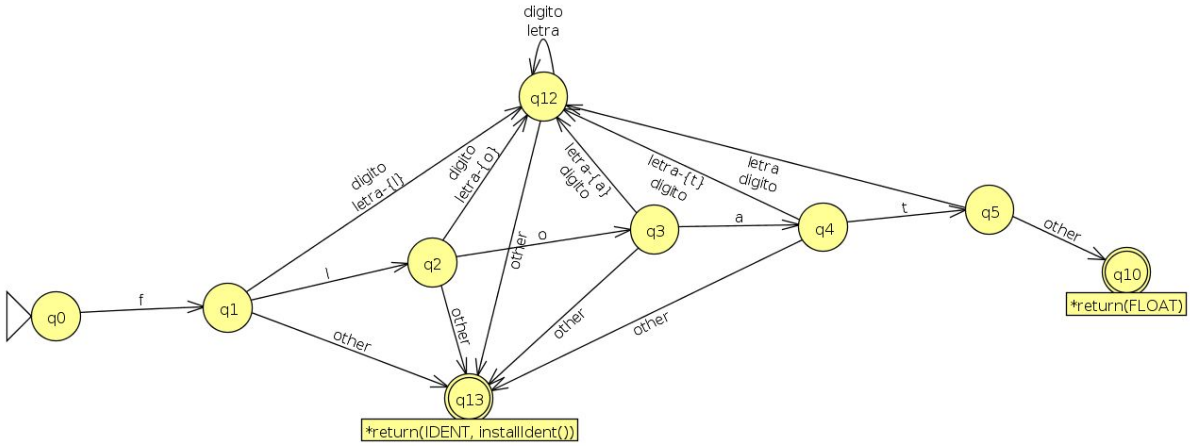
A análise léxica informalmente pode ser compreendida como a identificação de padrões em um código fonte através de regras bem definidas.

O as regras são definidas  através de expressões regulares, que indicam quais são as características fundamentais da linguagem em questão.

O grupo desenvolveu diversos autômatos individuais para cada *token*, ao término unimos os autômatos individuais em um único autômato principal, onde esta descrito toda as regras formais da especificação da linguagem.

## 3.1 Autômatos para reconhecimento dos *tokens*

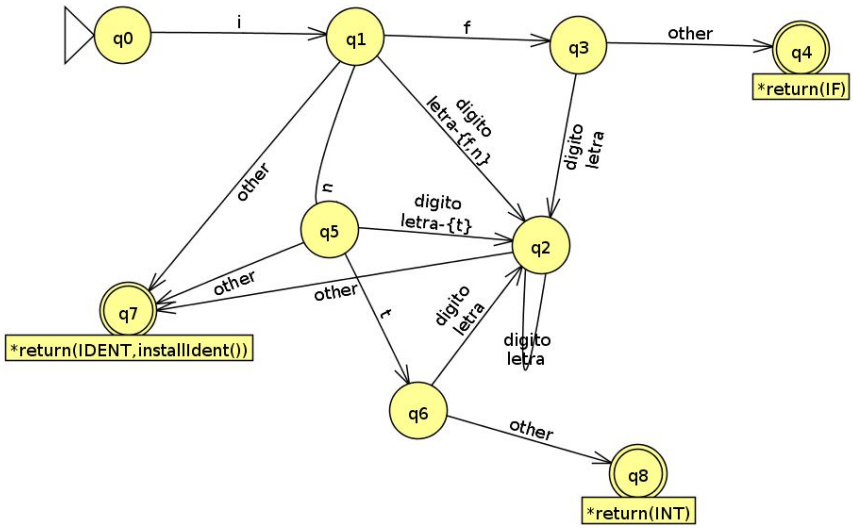
Nesta seção serão apresentados os autômatos utilizados para o reconhecimento dos *tokens*. Abaixo está um autômato completo para o reconhecimento do *token* FLOAT.



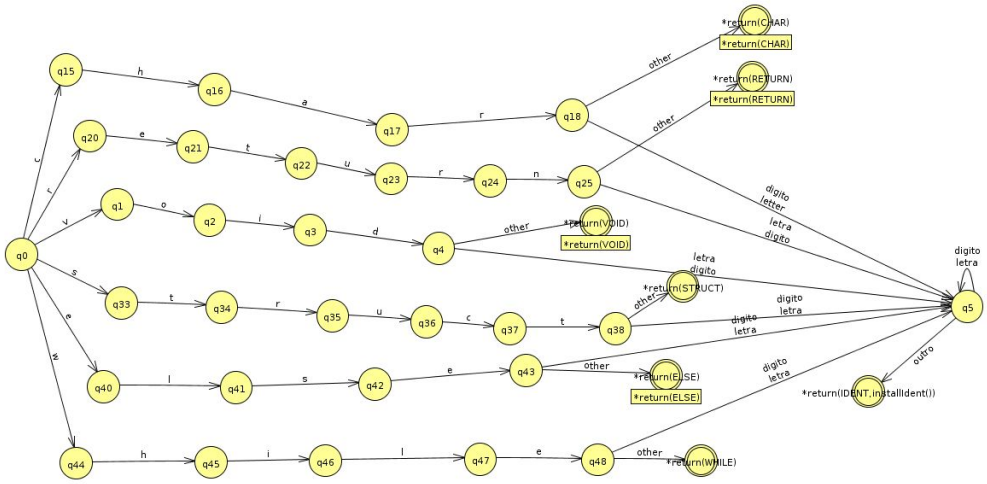

Percebe-se que o lexema deste *token* pode ser o prefixo de um lexema do *token* IDENT. Nota-se também que um prefixo do lexema do token FLOAT pode ser o prefixo de um lexema do token IDENT.

Por causa dessas características o autômato desse *token* torna-se um pouco complicado. E todas as palavras reservadas da linguagem compartilham dessa mesma característica com o *token* IDENT.

Abaixo está um autômato completo para o reconhecimento dos *tokens* IF e INT.

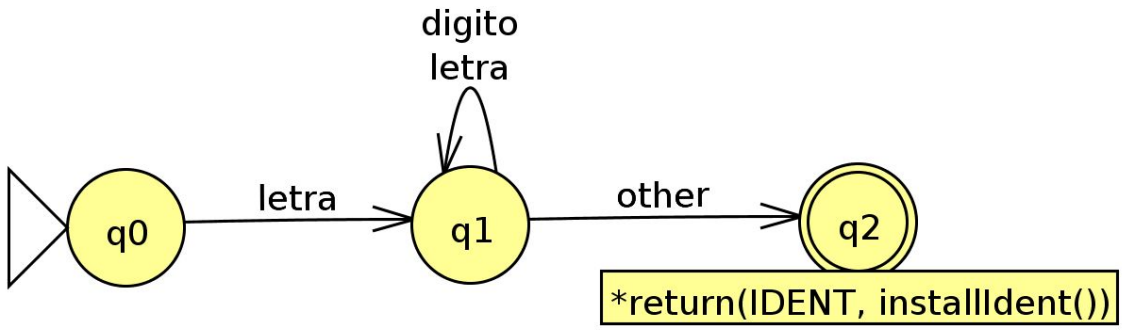



Esses dois *tokens* (IF e INT) possuem em seus lexemas um prefixo em comum, "i". Como comentado anteriormente, essa característica dos prefixos dos lexemas das palavras reservadas com os possíveis lexemas do *token* IDENT complica o autômato, por isso abaixo é apresentado um autômato simplificado dos outras palavras reservadas. Porém deve compreender que o lexema, ou um prefixo do mesmo, de um *token* de uma palavra reservada pode também ser um lexema, ou um prefixo do mesmo, de um IDENT.



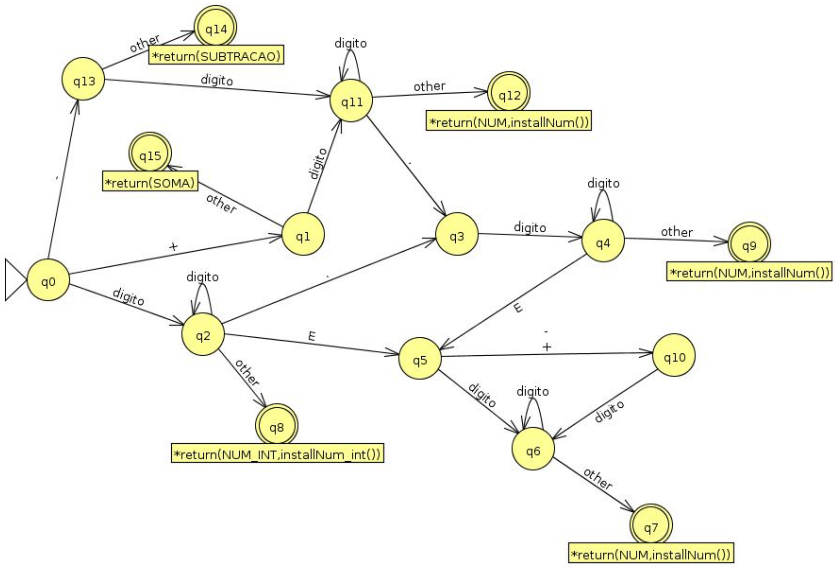

Acima está o autômato para o reconhecimento dos tokens das palavras reservadas ainda não apresentadas (CHAR, RETURN, VOID, STRUCT, ELSE, WHILE).

Abaixo é apresentado o autômato simplificado para o reconhecimento do *token* IDENT, pois não é mostrado que se começar com uma letra que é inicial de uma palavra reservada ele entraria em um autômato mais complicado.



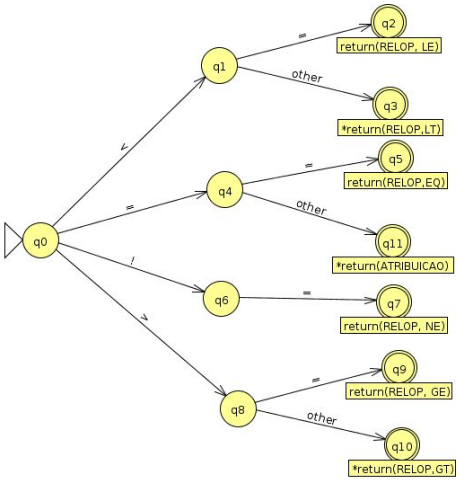

Relembrando as letras iniciais das palavras reservadas são: f,i,c,r,v,s,e,w; portanto se um lexema de um IDENT começar com uma dessas letras o autômato iria seguir partes dos autômatos completos das palavras reservadas que foram mostrado anteriormente.

Abaixo está um autômato para reconhecimento dos *tokens* NUM, NUM_INT, SUBTRACAO e SOMA.



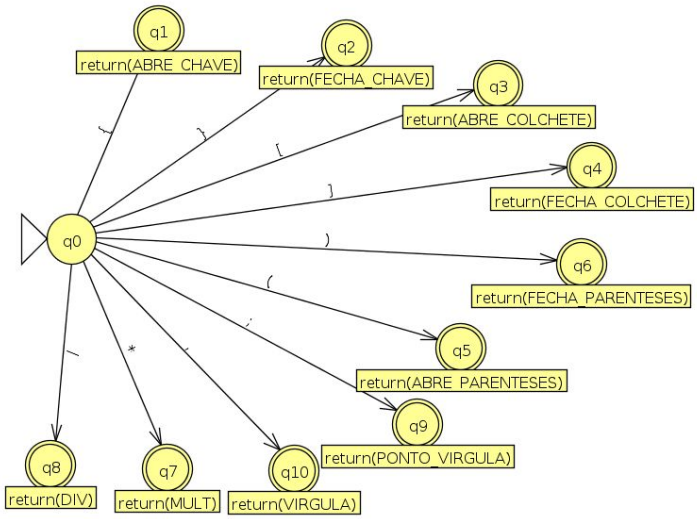

Percebe-se que os sinais de + e - podem pertencer tanto ao lexema de um NUM_INT quanto podem ser os lexemas dos *tokens* SOMA ou SUBTRACAO.

Abaixo o autômato para o reconhecimento do *token* RELOP e do token ATRIBUICAO.




Nota-se que para o token RELOP (operador relacional) são possíveis 6 lexemas: “<=”, “<”, “==”, “!=”, “>=” e “>” que são representados, respectivamente, pelos 6 seguintes valores LE, LT, EQ, NE, GE e GT.

Abaixo o autômato para o reconhecimento dos *tokens* restantes: ABRE_CHAVE, FECHA_CHAVE, ABRE_COLCHETE, FECHA_COLCHETE, ABRE_PARENTESES, FECHA_PARENTESES, PONTO_VIRGULA, VIRGULA, MULT e DIV.




## 6. Referências


[1] The Lex Project. 1975–2016. Disponível em : <http://dinosaur.compilertools.net/lex/index.html>.

[2] Lesk, M.E.; Schmidt, E. "Lex – A Lexical Analyzer Generator". Retrieved August 16, 2010.

[3] Levine, John R.; Mason, Tony; Brown, Doug (1992). lex & yacc (2 ed.). O'Reilly. pp. 1–2. ISBN 1-56592-000-7.

[4] The ANSI C project. 1989 - 2016. Disponível em : <http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=57853/>

[5]The Ubuntu Project *(release 15.4)*. 2005 - 2016. Disponível em : <http://www.ubuntu.com/>

[6] The GCC Project. GCC - GNU Compiler Collection. 1990–2016. Disponível em : <https://gcc.gnu.org/gcc-4.9/>.