Desenvolvendo um Compilador em C para a linguagem (.hen)

Henrique R. Favaro[1]

[1]Departamento de Informática – Universidade Estadual de Maringá (UEM)
Maringá, PR – Brazil

ra115408@uem.br

Resumo. *A geração de código de máquina, durante a criação de programas por desenvolvedores, nem sempre é transparente e presente na escrita de código destes, ficando à cargo dos compiladores realizarem esta tarefa, traduzindo os códigos escritos em diferentes linguagens, para uma linguagem que a máquina entenda. Além disso, diversas etapas são necessárias durante o processo de compilação do código, visando garantir o completo funcionamento deste, também sendo responsabilidade dos compiladores a realização dessas etapas. O presente estudo aborda a criação de um compilador, desenvolvido com a linguagem C, para a compilação de códigos escritos na linguagem .hen, criada durante o desenvolvimento do trabalho, apresentando as facilidades, dificuldades e detalhes do projeto.*

1. Detalhes do Projeto

Esta sessão apresenta os detalhes e ferramentas empregadas durante a criação e desenvolvimento do compilador como um todo, características de cada ferramenta e sua função essencial dentro de cada fase. As fases de um compilador completo podem ser observadas na Figura(1).

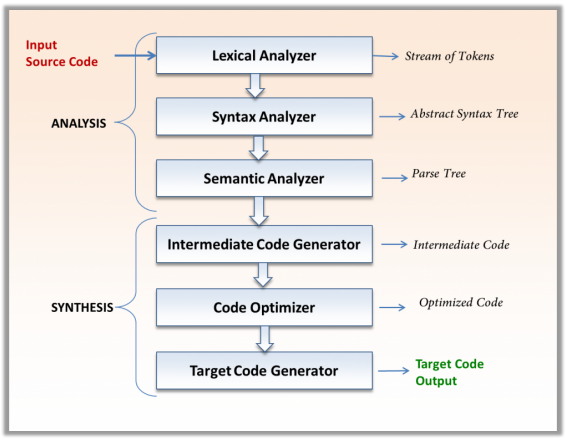

Figura 1: Estrutura de um Compilador Completo

1.1. Análise Léxica

O projeto teve seu início com estudos sobre as primeiras fases de compilação, sendo elas Análise Léxica e Sintática, podendo-se entender as necessidades para cada uma delas, bem como os desafios à serem enfrentados. Dessa forma, para a primeira análise (léxica), optou-se pelo uso da ferramenta Flex[1], a qual pode ser configurada para escanear o código de entrada, gerando tokens conforme as regras definidas.

Para que a ferramenta seja capaz de escanear o código de entrada e gerar tokens como saída, um código C é construído automaticamente, tendo como base as regras definidas no arquivo *(lexer.l)*, então o código de entrada é analisado e os tokens são gerados, como apresentado pela Figura(2).

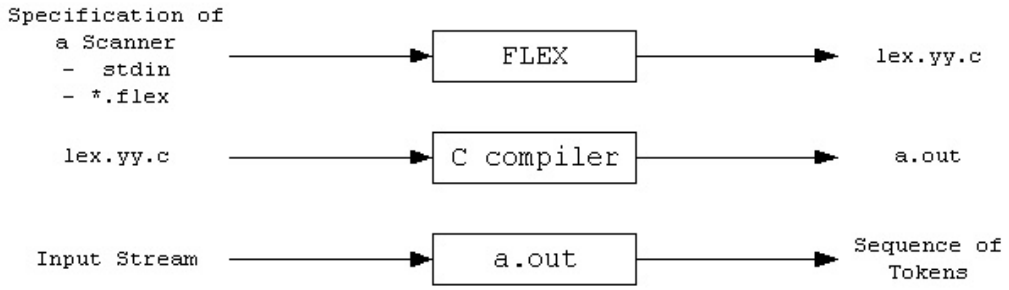

Figura 2: Diagrama de Funcionamento - Flex

Com todos os tokens gerados, a análise léxica pôde ser realizada por completo, onde identificam-se caracteres que não fazem parte da linguagem definida para o compilador, gerando-se erros quando presentes no código de entrada mas sem alguma relação com as regras definidas.

1.2. Análise Sintática

Em sequência, com os tokens gerados para cada cadeia de caracteres, foi possível o início da análise Sintática, onde busca-se verificar a integridade das palavras, ou seja, verificar se a ordem dos tokens gerados está correta para cada linha do código. Nesta etapa, a ferramenta Bison[2] foi empregada, ficando à cargo de receber regras do arquivo *(parser.y)* e gerar o código para as tabelas necessárias, o qual é utilizado para verificação dos tokens recebidos da análise léxica, como apresentado pela Figura(3).

Com ambas as ferramentas instaladas e configuradas, tem-se uma estrutura essencial para a base do compilador, podendo-se estender suas funcionalidades com a adição de novas funções e tabelas para as análises futuras.

[1]https://github.com/westes/flex
[2]https://www.gnu.org/software/bison/

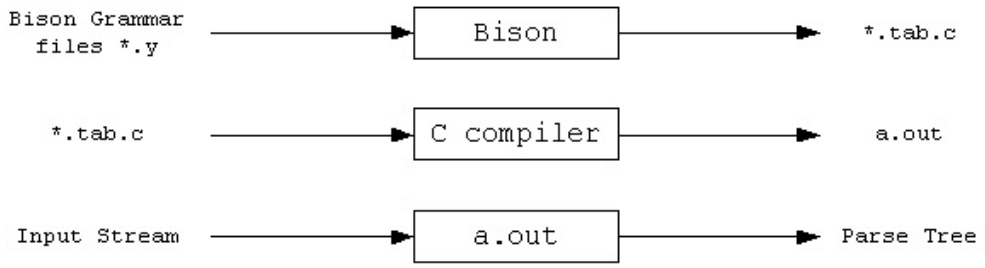

Figura 3: Diagrama de Funcionamento - Bison

1.3. Análise Semântica

Dessa forma, seguindo com a estrutura necessária para um compilador, uma tabela *hash* foi criada, onde os identificadores de variáveis e funções são adicionados, contendo também seu escopo pertencente e as linhas do código onde ocorrem, possibilitando assim o início do desenvolvimento da Análise Semântica. A tabela1 representa a estrutura empregada para esta etapa.

id	***type***	***scope***	***lines***
x	int	0	1
...	...	...	...

Tabela 1: Tabela *Hash* Empregada

Para a análise semântica, entende-se como necessário garantir o correto uso da linguagem, ou seja, todos os tokens, agora que verificados quanto sua escrita correta (análise léxica), ordem congruente (análise sintática), necessitam ser compatíveis quanto ao tipo durante agregações, definições únicas para evitar conflitos em endereços de memória, uso adequado de variáveis após sua definição inicial, chamada de funções com parâmetros corretos, entre outros aspectos.

Para esta análise, foi possível a verificação de variáveis declaradas em duplicidade, ou seja, uma variável pode ser declarada de maneira única dentro de determinado escopo, bem como a verificação da importação de uma classe quando a mesma é utilizada dentro dos blocos de função, gerando-se erros semânticos quando estes erros acontecem.

As variáveis podem ser inseridas na tabela *hash* toda vez que são definidas *(int X;)*, por exemplo, o que possibilita a consulta futura durante a definição de outra variável de mesmo identificador, evitando definição duplicada em mesmo escopo, como já comentado anteriormente. Ao trocar-se de escopo, todos os identificadores presentes na tabela são apagados, evitando conflitos entre diferentes escopos, sendo mantidos apenas aqueles que englobam o escopo analisado.

1.4. Orientação à Objetos

Ainda, para uma melhor utilização da linguagem criada, foram empregadas estruturas para Orientação à Objetos, ficando possível a criação de classes dentro de arquivos separados, as quais podem ser importadas dentro de um arquivo principal para utilização de suas funções e variáveis. Também foram empregados tokens para controle de variáveis e métodos Públicos ou Privados à classe, restando apenas a adição de verificações semânticas para esses.

2. Facilidades

Durante o desenvolvimento, as etapas que apresentam maior facilidade de desenvolvimento, são as Análises Léxica e Sintática. Por se tratarem e análises onde há ferramentas para auxiliar, são mais práticas de desenvolver e de melhor entendimento, pois apenas necessitam da criação de regras e estas irão gerar todo o código C complexo que realmente realiza a análise do código de entrada.

Dessa forma, o desenvolvimento do compilador como um todo teve suas duas análises iniciais desenvolvidas com grande completude, podendo-se verificar diversos pontos e apresentar erros de maneira consistente quando o usuário insere caracteres inválidos ou em ordem incorreta, como esquecer um (;) ao final de uma linha, ou, esquecer a palavra reservada (int) quando vai realizar a definição de uma variável;

3. Dificuldades

Já quanto à Orientação à Objetos, esta não está completa, sendo possível apenas verificar os tokens para a criação de uma classe e verificar sintaticamente a mesma, porém a análise semântica das classes foi uma etapa muito custosa no desenvolvimento, chegando à ser finalizada uma única funcionalidade que verifica a importação da classe antes do uso, restando diversos pontos à serem finalizados, como a verificação das funções, parâmetros, chamada de função totalmente correta e afins.

Conseguinte, ainda relacionado à análise semântica, como o compilador possui uma tabela de símbolos *(hash)*, própria, isso dificultou o desenvolvimento completo desta análise, pois a criação da AST *(Abstract Syntax Tree)* foi iniciada, com todas as funções e estruturas de dados criadas manualmente, porém prejudicou-se o funcionamento das análises iniciais já finalizadas, de maneira que optou-se pela remoção desta etapa e uma polimento das fases iniciais.

De maneira geral, as maiores dificuldades encontradas foram durante o desenvolvimento de fases onde não havia a presença das ferramentas, em relação às estruturas de dados aceitas pela linguagem C, o que dificultou o desenvolvimento do restante do compilador, ficando completas as análises Léxica e Sintática. A finalização da análise semântica ficaria como um trabalho futuro, uma vez que seja necessária a finalização do compilador para utilização em produção da Linguagem (.hen).

Por fim, a geração de código intermediário não foi iniciada, pois a AST deveria estar presente para que fosse possível a geração de códigos em LLVM (Low Level Virtual Machine), consequentemente não sendo possível a implementação das fases de otimização de código e geração de código assembly.

Referências

AHO, A. V., LAM, M. S., SETHI, R., and ULLMAN, J. D. (2007). *Compiladores: princípios, técnicas e ferramentas, 2ª edição*. Editora Pearson.