Compiladores

2 Ano do Curso Engenharia Informática

Trabalho Prático

Relatório de Desenvolvimento

Bruno Medeiros	Joel Pinto	Jorge Pereira
71337	70773	14023

23 de Janeiro de 2021

Resumo

No mundo da computação quando falamos em linguagem, falamos de linguagens de programação, compiladores e a sua importância na interação computadores-pessoas. Uma das partes mais importantes para trabalhar num compilador e a sua análise sintática, acompanhada de toda a sua gramática que se pretende que seja rigorosa e sem ambiguidades. A realização deste projeto prático de Compiladores tem como objetivo a implementação de um analisador sintático (usando Yacc/-Lex) capaz de verificar uma versão simplificada da linguagem C- naive C. No final do trabalho foi possível verificar o sucesso a implementação deste mini compilador através da sua execução em exemplos de blocos de código em linguagem C apresentados.

Conteúdo

Capítulo 1

Introdução

1.1 Enquadramento Teorico

A(s) definição(ões) de linguagem variam de acordo com o sentido que se pretende dar, por exemplo, o português é uma linguagem natural que tem como principal objetivo a comunicação entre pessoas. No entanto, no meio da computação quando falamos em linguagem, falamos de linguagens de programação. Uma linguagem de programação é uma notação para escrever programas, sendo que um programa é uma sequência de instruções que devem ser executadas por um computador. As linguagens de programação existem, a princípio, para que o programador comunique ao computador as tarefas que devem ser realizadas, portanto, uma linguagem precisa, pois, o computador não pode fazer julgamentos e resolver ambiguidades. É impossível estudar compiladores sem estudar as linguagens de programação. Já o contrário é possível: podemos estudar linguagens sem conhecer nada sobre compiladores. **Mas o que são compiladores?** As pessoas e computadores funcionam de forma diferente, o que leva à existência de linguagens de programação com diferentes níveis (baixo nível versus alto nível). Um compilador é um programa que traduz programas escritos em uma linguagem, chamada de linguagem-fonte, para outra linguagem, a linguagem-destino. Normalmente, a linguagem-fonte é uma de alto nível, e a linguagem de destino é uma linguagem de máquina de algum processador, ou algum outro tipo de linguagem de baixo nível que seja executada diretamente por uma plataforma existente.

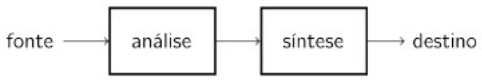

Figura 1.1: Estrutura básica de um compilador.

A estrutura básica de um compilador (Figura 1.1.) divide-se me duas partes principais: a primeira analisa o programa-fonte para verificar sua corretude e extrair as informações necessárias para a tradução; a segunda utiliza as informações coletadas para gerar, ou sintetizar, o programa na linguagem de destino. É o modelo de análise e síntese; a fase de análise também é chamada de vanguarda do compilador (front-end) e a de síntese é conhecida como retaguarda (back-end).

O programa-fonte é, inicialmente, um conjunto de caracteres; a tarefa da fase de análise léxica é agrupar esses caracteres em palavras significativas para a linguagem, ou tokens. Em seguida, a análise sintática deve, através do conjunto e ordem dos tokens, extrair a estrutura gramatical do programa, que é expressa em uma árvore sintática. A análise semântica, ou análise contextual, examina a árvore sintática para obter informações de contexto, adicionando anotações à árvore com estas informações. A fase final da análise é a transformação da árvore com anotações, resultado de todas as fases anteriores, no código intermediário necessário para a síntese.

1.2 Geradores de Analisadores Sintáticos em Uso

Existem diversos programas de computadores desenvolvidos com o propósito de facilitar a criação de analisadores sintáticos. Esses programas em geral consistem em receber como entrada uma gramática e retornarem como saída um programa que, é um analisador sintático para essa gramática. Neste projeto o gerador de analisadores sintático utilizado para a criação de um analisador sintático capaz de verificar uma versão simplificada da linguagem C- naive C, foi o Yacc/Lex.

1.2.1 Gerador Analisador Léxico - Lex

Para a realização do trabalho pratico foi utilizada a ferramenta Lex para a escrita de um *script* que gerasse um analisador léxico definindo expressões regulares para descrever padrões para os tokens. A estrutura dos scripts de Lex, são constituídos por três seções - definições ou declarações; regras de tradução e rotinas auxiliares (em C) - separadas por dois sinais de percentagem (Figura 1.2.).

```
{definições}
%%
{regras}
%%
{rotinas auxiliares}
```

Figura 1.2: Estrutura de programa Lex e Yacc.

A primeira seção serve para importar *header files* ou então para definir variáveis que serão usadas mais tarde no programa, como por exemplo, um contador de tokens ou criação de uma tabela com o tipo e descrição do tokens. A seguir temos a seção das regras, onde são definidas expressões regulares para cada tipo de token que vai ser lido, neste caso as expressões regulares são capazes de identificar números inteiro e reais, identificadores, etc. A última seção é onde se implementa rotinas auxiliares, normalmente em código C. Por norma contém um programa principal para compilar a saída Lex como um programa independente e obter um output.

1.2.2 Gerador Analisador Sintatico - Yacc

Yacc (*Yet another compiler compiler*) foi desenvolvido nos laboratórios da AT&T e tem sido utilizado no sistema Unix como o seu gerador de analisadores sintáticos. Inicialmente o Yacc

não tinha um método padrão para análise léxica, mas Lesk e Schmidt criaram o Lex, gerador analisador léxico, que permitiu esta parceria na geração de analisadores sintáticos. O Yacc recebe um programa-fonte em linguagem Yacc contendo uma gramática e gera as tabelas necessárias para que o método LALR (Look-Ahead LR parser) seja executado. A estrutura do script em código Yacc é dividida também em três secções (Figura 1.2.) declaração, regras de tradução e rotinas de suporte em C - separadas por dois símbolos de percentagem. A secção das definições ou declarações é usada para definirmos parâmetros em linguagem C, como por exemplo header files ou variáveis globais, assim como para a definição de parâmetros para o nosso analisador sintático, como por exemplo a declaração dos tokens da gramática, definidos também no ficheiro de Lex. Na secção das regras gramaticais são colocadas as produções da gramática e a ação semântica associado a cada regra. De uma forma simples, é através destas regras gramaticais que é decidido que qualquer bloco de código recebido, em tokens, está de acordo com as especificações da linguagem. Por último, a secção das rotinas de suporte em linguagem C serve para chamar? o parser assim como se define a função responsável pelos erros sintáticos.

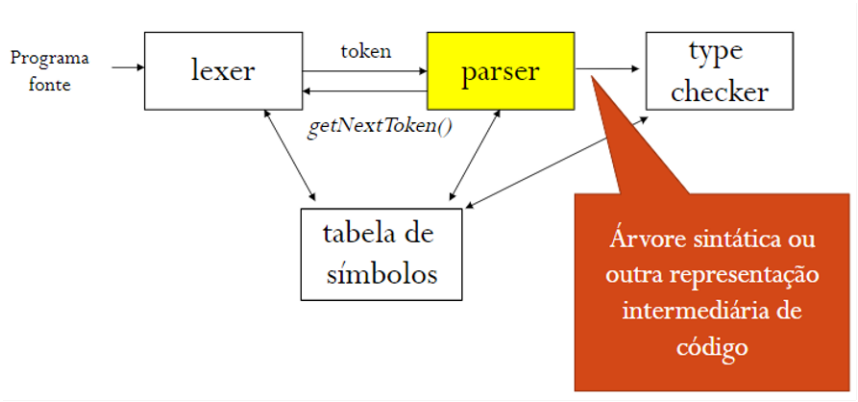

Figura 1.3: Relação do parser com o analisador léxico.

A análise sintática é uma das fases de compilação de uma linguagem de programação, mais propriamente a segunda fase. É responsável por definir como se formam frases corretas a partir de tokens que foram obtidos anteriormente na análise léxica, criando assim uma estrutura de dados na forma de uma árvore sintática. Nesta fase, o parser/analisador sintático verifica se as tokens que foram geradas pelo analisador léxico seguem as regras gramaticais que foram definidas. Por fim, também detecta e reporta erros sintáticos no caso de,o conjunto de tokens não pertencer à gramática da linguagem definida. Tal como foi mencionado, o analisador léxico e sintático trabalham em sequência, uma vez que o analisador léxico aceita como entrada uma sentença (uma lista de tokens) e o parser constrói para a sentença a sua árvore gramatical (Figura 1.3.).

Duas técnicas básicas para a construção de analisadores sintáticos são a construção ascendente ou a construção descendente. Na construção ascendente (bottom-up), a construção da árvore sintática começa de baixo para cima tal como o próprio nome indica. Na construção descendente (top-down), a construção da árvore sintática será feita da raíz até às folhas. Alguns princípios associados a estas

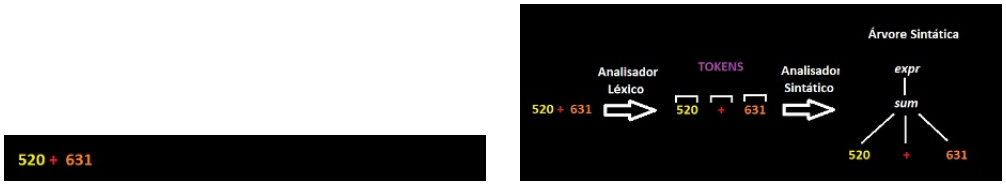

Figura 1.4: Exemplo de uma análise sintática.

técnicas de parsing são:

1. O uso da derivação mais à esquerda ou mais à direita.
2. O tipo de algoritmo de implementação usado.

Através de um exemplo, suponhamos que a seguinte sentença é aceite como entrada pelo analisador léxico e seria analisada da forma observada na Figura 1.4, apresentando a sua árvore sintática.

1.3 Descrição Informal do Problema

O principal objetivo deste projeto prático, no âmbito da Unidade Curricular de Compiladores, é desenvolver e implementar um analisador sintático, usando as ferramentas LEX (Flex) e YACC (Bison). Este, tem de ser capaz de verificar uma versão simplificada da linguagem C e fazer o tratamento de erros, tanto léxicos como sintáticos. A programação do código é feita em C. O desenvolvimento e implementação do analisador sintático, designado por mycc, será realizado em duas fases:

1. Implementação do analisador léxico, capaz de fazer o reconhecimento dos tokens específicos da linguagem C - naive C. Nesta fase iremos usar a ferramenta LEX (Flex).
2. Implementação do analisador sintático, capaz de analisar as definições, instruções e comentários específicos da linguagem C - naive c. Nesta fase iremos usar a ferramenta YACC (Bison).

O analisador sintático irá poder analisar sintaticamente programas que contenham:

1. Números inteiros e reais: identificados pela expressões regulares, [0-9]+ e [0-9]+"."[0-9]+.
2. Identificadores: identificados pela expressão regular, [a-zA-Z][a-zA-z0-9]*.
3. Declaração de variáveis: int, float, char, void, double.
4. Instruções de atribuição e instruções: while, for, if, else, printf.
5. Estruturas de dados: struct.
6. Operadores: $=,\ =, ,\ , ==, !, =, Outroscomandos - \#include$
7. Comentários - especificado por /* ... */ (comentário multilinha) ou por // (comentário singular).

1.4 Estrutura do Documento

O relatório do projeto prático de Compiladores, no qual se pretendia a implementação de um analisador sintático, chamado mycc, foi estruturado em 4 capítulos. No capítulo 1, fez-se um enquadramento teórico relativamente aos assuntos do projeto, assim como se descreveu qual a problemática do mesmo. No capítulo 2, descreve-se como foi pensado e concebido o projeto, apresentando os ficheiros para analise do léxico e analise sintática. Já no capítulo 3 foram apresentados alguns dos testes feitos ao analisador sintático. Por fim, o capítulo 4 temos a conclusão onde apresentamos se os objetivos foram conseguido e o que poderíamos fazer para melhorar este projeto.

Capítulo 2

Concepção do Código do Projeto

Neste projeto o gerador de analisadores sintático utilizado para a criação de um analisador sintático capaz de verificar uma versão simplificada da linguagem C- naïve C, foi o Yacc/Lex.

2.1 Ficheiro lex.l

Para a realização do trabalho pratico foi utilizada a ferramenta Lex para a escrita de um script (ficheiro lex.l - em anexo no ficheiro zip enviado) que gerasse um analisador léxico definindo expressões regulares para descrever padrões para os tokens. No ficheiro lex.l, na secção das definições foi adicionado um pré-processador (include "y.tab.c") para o ficheiro parser.y aceder aos tokens criados no analisador léxico. Na parte das regras foram definidas expressões regulares para a identificação de tokens como por exemplo, números inteiros e reais, identificadores, etc. A última secção foi implementada uma rotina em linguagem C. Caso particular do nosso trabalho que criasse um ficheiro chamado output.txt que tivesse a lista de todos os tokens encontrados de um dado programa que foi recebido como input pelo analisador léxico e que exibisse uma mensagem a dizer o total de tokens lidos entre outras coisas. Desta forma foi-nos possível verificar que o analisador léxico encontrava as tokens do nosso programa em naive C.

O ficheiro lex.l criado para realizar a listagem dos tokens a partir de blocos de código C e que permitia obter um output desse mesmo tokens, foi alterado para ser utilizado em conjunto com o ficheiro de Yacc criado, parser.y. Neste caso foi retirada a rotina em linguagem C na última secção, adicionado include "y.tab.c"assim como return nos tokens necessários.

2.2 Ficheiro parser.y

Usando a ferramenta Yacc foi criado o ficheiro parser.y (em anexo no ficheiro zip enviado), no qual na secção das definições ou declarações foram definidos por exemplo, parâmetros em linguagem C, como pré-processadores importantes para correr o ficheiro parser.y. Na secção das regras gramaticais são colocadas as produções da gramática e a ação semântica associado a cada regra. De uma forma simples, é através destas regras gramaticais que é decido que qualquer bloco de código recebido,

em tokens, está de acordo com as especificações da linguagem. Foi também nesta secção que por exemplo se fez a declaração dos tokens da gramática, definidos também no ficheiro de Lex. Por último, a secção das rotinas de suporte em linguagem C serve para "chamar"o parser.y mas também para criar funções em linguagem C responsável pela apresentação pelos erros sintáticos.

Capítulo 3

Resultados

3.1 Testes Realizados e Resultados

Os geradores de analisadores sintáticos funcionam de maneira bastante similar aos geradores de analisadores léxicos, como foi visto no Capitulo 1. Para gerar um analisador sintático, usamos a ferramenta geradora passando como entrada uma especificação da estrutura sintática da linguagem que queremos analisar a saída do gerador é um analisador sintático na forma de código em alguma linguagem de programação (no nosso caso, um arquivo na linguagem C). Esse analisador recebe um fluxo de tokens na entrada e gera uma árvore sintática na saída.

Depois de efetuado o código dos ficheiros lex.l (analisador léxico-em anexo no ficheiro zip enviado) e parser.y (analisador sintático-em anexo no ficheiro zip enviado) é importante a sua testagem para verificar se o nosso mini compilador **mycc** é capaz de verificar uma versão simplificada da linguagem C ? naive C. Assim, os comandos usados no projeto para executar cada um dos ficheiros lex.l e parser.y estão apresentados na Figura 3.1..

```
lex lex.l
yacc -d parser.y
-Wconflicts-sr]
gcc y.tab.c -ll -o mycc
./mycc main.c
```

Figura 3.1: Comando usados para executar o analisado sintático.

O analisador sintático **mycc** vai receber um ficheiro em naive C como input, analisá-lo e retornar um output, com uma mensagem a mostrar se a análise ocorreu com sucesso ou houve erros sintáticos. No caso de haver erros irá mostrar a linha onde ocorreu esse erro. Nos exemplos de programas (Figura 3.2.) em linguagem naive C, é possível verificar a existência de erros que foram colocados de maneira propositada. Isto serve para verificar se o analisador sintático consegue detetar e reportar estes mesmos erros. Neste caso existe um erro, logo o output irá mostrar uma mensagem de erro, indicando a respectiva linha.

Figura 3.2: Exemplo de um input em C e o respetivo output.

Tambem foi possível observar (Figura 3.3.) não foram encontrados erros o que significa que o parser consegue detectar números inteiros e reais, identificadores, declaração de variáveis, instruções de atribuição e instruções , estrutura de dados, operadores, comentários e outros comandos.

```c
#include <stdio.h>

struct Pessoa { // Cria uma STRUCT para armazenar os dados de uma pessoa

    float Peso;   // define o campo Peso
    int Idade;    // define o campo Idade
    float Altura; // define o campo Altura
};

void main(){

        int num1=1; //variavel numero
        float num2=2.0;
        double num3=2.35;
        char letra = 'x';

        /*Ciclo que incrementa
        o num1 a cada ciclo.*/

        while(num1 <= 5 )
        {
                if(num1 == 2){
                        printf("%d",num1);  //resultado esperado "2"
                        num++;
                }
                else{
                        num1++;
                }
        }

        int i;

        for(i=0;i<10;i++){
                printf("%d",i);  //resultado esperado "1 2 3 4 5 6 7 8 9 10"
        }
}
```

```
Analise Sintatica completa
```

Figura 3.3: Exemplo de um input em C e o respetivo output

Capítulo 4

Conclusão

O estudo das linguagens de programação é uma das áreas principais da ciência da computação, sendo os compiladores, implementações rigorosas de linguagens de programação, logo de extrema importância o seu estudo. Na realização deste projeto prático de Compiladores, foi possível entender de uma maneira geral, o processo de compilação de uma linguagem, assim como aprender a criar uma vasta quantidade de elementos essenciais à construção de um analisador sintático, tais como uma gramática, expressões regulares, entre outros. Com a conclusão do nosso analisador sintático os aspetos que achamos que podem ser melhorados no futuro, são por exemplo, expandir a nossa gramática, uma vez que a linguagem proposta era uma versão simplificada da linguagem C, com isto seria possível incluir mais funcionalidades na linguagem. Também poderia ser adicionada uma tabela de símbolos e de constantes que permitam uma melhor organização e identificação das tokens.