Ivo Henrique Pedrini, Lorenzo Salume Zimerer, Vinicius de Abreu Bozzi

Prof. Eduardo Zambon

Compiladores

09 de Dezembro de 2021

Checkpoint 1

O seguinte relatório de progresso no trabalho da disciplina de Compiladores, lecionada pelo professor Eduardo Zambon na Universidade Federal do Espírito Santo, tem como objetivo principal dar um retorno sobre o andamento do projeto até o momento. O projeto consiste no desenvolvimento de um compilador simples. O grupo foi contemplado com a linguagem fonte **Pascal** e a arquitetura alvo **MIPS.**

Analisador Léxico

Um dos objetivos do primeiro checkpoint era concluir o desenvolvimento do analisador léxico (scanner) para o nosso compilador.

Para essa tarefa, foi utilizado o Flex : uma linguagem de programação que permite a construção automática de scanners em C.

A Flex serve para dividir a entrada em pequenos tokens, que vão ser, posteriormente, associados entre si, na etapa de análise sintática.

Detalhes da implementação:

➡Comentários foram tratados com *regexes* simples: `\{[^}]*\}` que procura o símbolo { e lê qualquer quantidade de símbolos diferentes de } até chegar ao } que indica o final do

comentário (Foram testadas outras maneiras, mas essa pareceu a mais promissora, pois funcionava bem em comentários com múltiplas linhas)

Vale mencionar que há 3 tipos de comentários; além do mencionado acima, existe a forma (* *), cujo tratamento é semelhante ao supracitado e existe o // que comenta o resto da linha, nos quais apenas foi descartado o restante da linha.

➡Pascal não é case-sensitive. Ou seja, é preciso reconhecer os tokens em qualquer sequência de maiúsculas e minúsculas, portanto, foi necessário agrupar, para cada letra, sua versão em maiúsculo e minúsculo. Isso foi feito utilizando o próprio flex, na forma:

```
A [aA]
```

Desse modo, todas as palavras reservadas e futuros tokens ficaram do tipo:

```
{A}{B}{S}{O}{L}{U}{T}{E}                    { process_token(ABSOLUTE); }
```

➡Para fazer o scanner adaptado ao parser (no módulo 2), onde é preciso retornar algo a cada token, aproveitamos uma linha de código da resolução do professor para o lab2:

```
#define process_token(type) return type
```

Analisador sintático

O outro objetivo do primeiro checkpoint era concluir o desenvolvimento do analisador sintático (parser) para o nosso compilador.

Para esta tarefa, foi utilizado o Bison, com o propósito de converter uma gramática livre de contexto em uma estrutura de dados para melhor associação e processamento posterior.

Como citado na etapa de análise léxica, aqui, os tokens escaneados da entrada vão ser associados entre si, e seus comportamentos serão definidos.

Detalhes da implementação:

➡Para esse trabalho, foram testadas 3 sintaxes (distintas) de Pascal em notação BNF no Bison, visto que não foi encontrada uma versão oficial. Elas apresentavam alguns erros e levavam a acreditar que a gramática estava caindo em alguma recursão infinita e, quando esse problema era "resolvido", outro aparecia, implicando com "regras/não-terminais não utilizados na gramática".

Acabou sendo necessário fazer certa mescla manual dessas BNFs e acrescentar/retirar algumas regras para alcançar o funcionamento devido.

CONCLUSÃO

O trabalho, até o momento, foi de enorme esclarecimento em relação ao funcionamento das partes léxicas e sintáticas de compiladores simples.

A principal dificuldade foi mencionada anteriormente, na parte do Analisador Sintático, no que diz respeito à gramática e certos erros de recursão infinita que foram encontrados ao longo do caminho.

Foi assinalada toda a linguagem fonte, mas a documentação era um pouco duvidosa e misturada, então há alguns tokens **extras** de TurboPascal e Delphi. Dito isso, nem todos os

tokens **extras** foram testados.

Os tokens do Pascal Clássico não definidos no parser foram apenas: file, packed, nil e with. Todos os outros tokens básicos foram testados.