Trabalho Final - Compiladores

Bruno Rafael dos Santos, Gabriel Anselmo Ramos

29 de Julho de 2022

Resumo: O presente trabalho trata-se do desenvolvimento de um compilador, usando as ferramentas flex, bison e jasmin afim de compilar uma linguagem de nossa autoria baseada na linguagem de referência passada pelo professor, criada na disciplina de Compiladores (COM0002), ministrada pelo Prof. Ricardo Ferreira Martins. O processamento da linguagem se inicia com analise léxica, no qual é feita a identificação de tokens com o auxilio do flex, na fase seguinte temos a verificação da gramatica no qual fazemos a analise sintática usando o bison, e na fase final enfatizada nessa etapa do trabalho é feita a análise semântica e a conversão da linguagem de alto nível para a linguagem de máquina.

1 Geração do cabeçalho

Como parte padrão da tradução há a escrita do cabeçalho do arquivo output.j qual ocorre através da respectiva regra da gramática com seus devidos comandos associados:

```
programa:
PROGRAM_TOKEN { generateHeader(); generateMainHeader(); }
ID_TOKEN
DOTCOMMA_TOKEN
corpo
END { generateMainFooter(); }
```

Note que a função $generateHeader()$ irá gerar todo o cabeçalho e rodapé inicial, a função $generateMainHeader()$ irá gerar o cabeçalho da função main e a função $generateMainFooter()$ irá gerar o rodapé da mesma.

Obs.: no cabeçalho da função main são dados os limites 100 locais de armazenamento para a pilha de execução (".limit stack 100") e 100 locais de armazenamento para a heap (".limit locals 100") para garantir espaço em ambas as estruturas de armazenamento durante a execução do programa.

2 Implementação dos comandos da linguagem

Os principais comandos implementados na linguagem criada para o presente trabalho estão definidos da seguinte forma:

1. **Atribuições:**

```
atribuicao:
variavel
TWODOTS_EQUAL_TOKEN
expressao_simples { atributeIntVariable($1); }
```

2. **Comando if-else:**

```
condicional:
IF_TOKEN
PLEFT_TOKEN
condicao_contraria
PRIGHT_TOKEN
THEN_TOKEN
CBLEFT_TOKEN { $1 = count_label; onlyLabelForIf($1); }
lista_de_comandos
CBRIGHT_TOKEN { onlyGoTo($1 +1); onlyLabel($1); count_label+=2; }
comando_else { onlyLabel($1 + 1); }
```

3. **Comando while:**

```
comando_while:
WHILE_TOKEN { $1 = count_label; onlyLabel($1); count_label+=2; }
PLEFT_TOKEN
condicao_contraria { onlyLabelForIf($1 + 1); }
PRIGHT_TOKEN
CBLEFT_TOKEN
lista_de_comandos
CBRIGHT_TOKEN { onlyGoTo($1); onlyLabel($1 + 1); }
```

4. **Comando do while:**

```
comando_do_while:
DO_TOKEN { $1 = count_label; onlyLabel($1); count_label+=2; }
CBLEFT_TOKEN
lista_de_comandos
CBRIGHT_TOKEN WHILE_TOKEN
PLEFT_TOKEN condicao_contraria { onlyLabelForIf($1 + 1); }
PRIGHT_TOKEN { onlyGoTo($1); onlyLabel($1 + 1); }
```

5. **Comando for:**

```
comando_for:
FOR_TOKEN
PLEFT_TOKEN
atribuicao
DOTCOMMA_TOKEN { $1 = count_label; onlyLabel($1); }
condicao_contraria { onlyLabelForIf($1 + 1); count_label+=2; }
DOTCOMMA_TOKEN
atribuicao
PRIGHT_TOKEN
CBLEFT_TOKEN
lista_de_comandos
CBRIGHT_TOKEN { onlyGoTo($1); onlyLabel($1 + 1); }
```

Obs.: todas as funções até agora mencionadas além das estruturas de dados foram implementadas nos arquivos *Compiler.c* e *Compiler.h*.

3 Problemas a serem resolvidos

Durante a implementação da etapa de tradução de código houveram as seguintes dificuldades:

1. **verificação de tipos:** quanto a esta funcionalidade que por sua vez não foi implementada, a maior barreira para tal foi a realização das operações aritméticas com diferentes tipos devido a necessidade de uma estratégia para conversão de tipos durante as reduções das regras relacionadas as expressões matemáticas.

2. **for:** o comando for atualmente implementado possui um defeito ainda a ser tratada que é a realização do incremento antes da primeira iteração, visto que tal incremento em nosso código ocorre por meio da regra de atribuição qual por sua vez escreve o código de máquina antes do código gerado pela regra de lista de comandos (dado que a variável de atribuição está à esquerda da variável de lista de comandos na regra gramática).

4 Utilização do compilador

Na pasta contendo o arquivos flex e bison, o código que se deseja rodar deve ser escrito no arquivo teste.txt (a não ser que se altere o comando descrito anteriormente para o terminal). Assim basta executar o arquivo rodar.bat na pasta do trabalho (caso a execução seja em sistema Windows) ou executar diretamente os seguintes comandos no terminal:

```
flex Trabalho02.lex
bison -d Trabalho02.y
gcc Trabalho02.tab.c lex.yy.c Compiler.c -o Trabalho02 -lm
type teste.txt | .\Trabalho02.exe
java -jar C:\Users\bruni\Downloads\jasmin-2.4\jasmin.jar output.j
java test
```