CES41 - Exame: Compilador completo C-

Gabriel Martinz

1 Descrição da Implementação

1.1 Analisadores léxico, sintático e semântico

O analisador léxico foi feito com o auxílio da ferramenta *Flex* (*Fast lexical analyzer*), com as rotinas de análise léxica escritas no arquivo `cminus.l`. Com base nessas rotinas, o *Flex* cria o autômato finito de análise léxica.

A partir da variável `NO_PARSE` no arquivo `main.c` podemos definir se o compilador é um analisador puramente léxico ou se é um analisador sintático. Sendo um analisador puramente léxico, os *tokens* são escritos em `stdout` com a linha do *token* e seu valor.

O analisador sintático foi criado com o auxílio da ferramenta *Bison*, que gera um analisador com base na especificação da gramática em Backus-Naur. A especificação da gramática e a criação da árvore sintática está escrita no arquivo `cminus.y`. O código no arquivo já passou por implementações extras para poder facilitar a criação do analisador semântico. A árvore sintática é escrita em `stdout` quando a variável `NO_PARSE` é assinalada ao valor `FALSE`, entrando no lugar da escrita dos *tokens* do analisador puramente léxico

Para o analisador semântico foi implementado o código para uma tabela de símbolos com contagem de escopo (`symtab.h/.c`) e o código de análise (`analyze.h/.c`). O código de análise contém as rotinas para construir a tabela de símbolos como também possui as rotinas para a checagem de tipos. A tabela de símbolos é escrita em `stdout` a partir das variáveis `NO_PARSE` e `NO_ANALYZE`.

1.2 Gerador de código

A implementação da geração de código foi feita com base na implementação de um código de três endereços.

O código para a geração de código intermediário foi implementado em `code.c` e `cgen.c`. A implementação foi feita baseada em código de três endereços mostrado na aula. Em `code.c` foram implementadas as funções principais de escrita de código, como as variáveis temporárias, enquanto em `cgen.c` foi implementado o controle de fluxo baseado na espécie do nó (*Expression*, *Statement* e *Declaration*).

A saída do gerador de código fica num arquivo separado com a extensão `.tac`, se configurada as variáveis `NO_PARSE`, `NO_ANALYZE` e `NO_CODE`.

2 Resultados Obtidos

As entradas de teste estão contidas na pasta `build`, junto do binário do compilador. As árvores construídas a partir das entradas de teste estão na pasta `outputs` do repositório. O código intermediário gerado está contido também na pasta `build`. O repositório está contido neste link.