Aluno: Thiago Brandenburg
Professor: Ricardo Ferreira Martins
UDESC - CCT - Departamento de Ciência da Computação
Matéria: Compiladores - COM0002

Compilador c->java bytecode

Resumo do Trabalho

Objetivos Alcançados:

1. Análise Léxica, Sintática e Semântica
2. Montagem da tabela de símbolos
3. Montagem da Árvore Sintática Abstrata, reconhecimento de estruturas de controle (repetição while e condicional if e if/else), funções build-in e declaradas, declarações de variáveis e operações matemáticas.
4. Montagem parcial do arquivo bytecode (cabeçalho e declaracao de variaveis)

Objetivos Não alcançados:

1. Montagem de demais estruturas no arquivo bytecode: (operações, funções, controle e funções build-in)

Instruções de Compilação:

make build_all : monta o trabalho e cria o compilador, que é o arquivo binario "compilador"

./compilador nome_do_arquivo : executa o compilador no arquivo, gera java_bytecode.j como output

make build_java: executa a compilação do java_bytecode.j pelo jasmin

Descrição dos Arquivos:

lista.h = header contendo a declaração de todas as funções e estruturas usadas no projeto, vale destaque para Nodo_Ast, Nodo_Token e Lista, as quais são as estruturas responsáveis respectivamente pela: Árvore Sintática Abstrata (AST), Nodo da tabela de Símbolos e Tabela de Símbolos em forma de lista.

lista.c = Implementação das funções utilizadas no projeto. relacionadas a tabela de símbolos e a AST.
gerador

arquivo_lex.lex = arquivo do flex, contendo análise léxica

arquivo_bison = arquivo do bison, contendo análise sintática e semântica

INSTRUCOES.txt = arquivo contendo uma versão simplificada do que está explicado aqui

teste.c = exemplo fornecido para testes

compilador = binário gerado pela execução, para manter a formalidade uma versão já compilada dele está inserida na entrega, mas ele pode ser apagado pois é gerado no make build_all

/RASC = pasta contendo rascunhos utilizados

Os demais arquivos são arquivos gerados pelo flex e bison

#Árvore Sintática Abstrata:

Estrutura que representa a “mente” do compilador, se trata em uma representação de todo o código c em forma de uma estrutura de dados do tipo “Árvore” encadeada e alocada dinamicamente. A sua construção foi essencial no compilador e permitiria que ele fosse facilmente adaptado para qualquer outro tipo de assembler que não fosse o java bytecode. A principal vantagem da AST é a possibilidade de acessar instruções do arquivo como se fossem variáveis, facilitando o acesso para geração de código assembler, através de funções.

Neste trabalho, foi utilizado um único tipo de struct para armazenar todos os tipos diferentes de instruções, a struct Nodo_Ast, a alocação foi eficiente pois cada nodo da árvore ocupa apenas 16 bytes, graças ao uso da estrutura union. o tipo de cada nodo é verificado através de um enum, é pela verificação deste parâmetro que diferenciamos um while de uma declaração, por exemplo.

Tabela de Símbolos

A tabela de Símbolos foi implementada utilizando duas structs: Lista e Nodo_Token. A lista é a representação da tabela de símbolos em memória, enquanto os Nodo_Tokens são os símbolos dentro da tabela encadeados. Caso um símbolo corresponda à uma função, possuímos um ponteiro Nodo_ast para podermos armazenar as instruções da funçaõ aqui.

#Detalhes da linguagem tipo C:

1) Toda linha deve terminar em ponto em vírgula, exceto o token terminal
3) O token terminal é FIM, ao entrar com esse token como input, a execução é finalizada, seria o equivalente ao return 0;
4) Não é necessário definir a execução dentro de uma main