Universidade Federal de Campina Grande – UFCG

Centro de Engenharia Elétrica e Informática - CEEI

Departamento de Sistemas e Computação – DSC

Disciplina: Compiladores

Professor: Franklin Ramalho

Fase 2 – Analisador Semântico para Compilador

Linguagem Fonte: Java

Linguagem Destino: Assembly

```
push    ebp
mov     ebp, esp
movzx   ecx, [ebp+arg_0]
pop     ebp
movzx   dx, cl
lea     eax, [edx+edx]
```

Equipe

Adalberto Teixeira

Andrey Menezes

Augusto Queiroz

Campina Grande-PB, Outubro/2011

CONTEÚDO

INTRODUÇÃO

Nesse relatório descrevemos os resultados da segunda fase do projeto de compilador para a linguagem OurJava.

ANALISADOR LÉXICO

Para o desenvolvimento dessa nova fase, nosso analisador léxico teve que passar por algumas mudanças para melhor se ajustar ao projeto como um todo. Logo abaixo será especificado o que foi modificado:

- Foram incluídas estruturas de dados para auxiliar na organização dos atributos;
- O tipo byte foi adicionado à gramática.

ANALISADOR SINTÁTICO

Nosso analisador sintático também teve que passar por algumas mudanças para poder se ajustar ao projeto. Abaixo expomos as modificações:

- Foi criado um conjunto de estruturas de dados para ser utilizado como base para todas as checagens semânticas. Essas estruturas simulavam os contextos, global e local, guardando os tipos declarados, nome, tipos, tamanho do array, etc;
- Inserção de tipos de retorno para as regras do analisador sintático;
- Foi inserido código C nas regras, permitindo assim, as checagens semânticas utilizando atributos sintetizados (apenas, não conseguimos utilizar atributos herdados);
- Várias regras foram alteradas, pois percebemos que eram desnecessárias;
- Foram criadas algumas variáveis globais para permitir o acesso a informações sobre contextos específicos

ANALISADOR SEMÂNTICO

Nessa seção são expostas todas as checagens que foram realizadas no analisador semântico de modo a garantir que a linguagem *OurJava* tivesse as características requisitadas no início do projeto pelo professor, como também mantivesse a correspondência com a Especificação da linguagem Java[1].

Aqui estão expostas as checagens realizadas divididas por tópicos:

- Funções e procedimentos
  - Permite overload na declaração de funções e procedimentos
  - Não permite a criação de duas funções/procedimentos que possuam o mesmo nome e mesmo tipos e sequência de parâmetros
- Comandos condicionais
  - Se as expressões para as condições são sempre booleanas
- Comandos iterativos
  - Se as expressões para as condições são sempre booleanas
- Comando de atribuição
  - Checa se os tipos são relacionados (alargamento implícito)
  - Permite casting
  - Permite a criação de operações de incremente e decremento
- Expressões aritméticas e booleanas
  - Os tipos dos operandos são avaliados quanto à possibilidade da operação
- Expressões relacionais
  - Os tipos dos operandos são avaliados quanto à possibilidade da operação
  - Operadores de precedência são utilizados
- Literais
  - Variáveis
    - Não permite múltiplas declarações de uma mesma variável.
    - Duas variáveis são iguais se estão no mesmo contexto (local, dentro de métodos, ou global, escopo da classe) e tem o mesmo nome
    - Ao acessar uma variável, checa se ela já foi declarada
    - A distinção de contexto é realizada através do identificador da classe (nome da classe, pois toda classe é estática), caso este não seja especificado, procura-se a variável no contexto local, depois no global
    - Quando tem o atributo final não são atualizadas
  - Tipos
    - Pode ser feito o alargamento implícito e casting explícito dos tipos definidos no projeto (essas operações seguem a Especificação de Java)
- Arranjos
  - Permite-se criar arranjos multidimensionais
  - Checa se o tipo e os níveis de arrays da definição são iguais aos da declaração
- Chamada de funções e procedimentos

[1] http://java.sun.com/docs/books/jls/download/langspec-3.0.pdf

    - Checa métodos com a mesma assinatura (nome e parâmetros) e caso não encontre realiza conversões implícitas e tenta casar com os outros métodos
- Sequenciadores de escape
    - Declaração de labels e verificação da existência
    - Para o return, foi verificado se o tipo do retorno é equivalente ao do método em análise

COMO EXECUTAR O COMPILADOR

Com o flex e bison devidamente instalados, basta executar pela linda de comando o **make** dentro da pasta "lex-sint-sem". Isso irá gerar o executável "java_compiler", para executar o compilador basta passar o arquivo .java com entrada de nosso executável, então nosso comando de execução fica assim: **./java_compiler < teste.java** .

Dentro do cd existe uma pasta executável, onde pode se encontrar o nosso executável, o mesmo gerado pelo código acima, para executa-lo basta utilizar o comando:

./java_compiler < teste.java

O código java deve conter um main, apenas uma classe pode ser especificada, todo os métodos e variáveis de contexto global devem ser static.