Universidade Federal de Goiás
Bacharelado em Ciência da Computação
Compiladores 2025 – 1

Compilador para a Linguagem Goianinha
Especificação das Etapas Finais do Compilador:
Geração da Representação Intermediária,
Análise Semântica
e
Geração de Código

prof. Thierson Couto Rosa

1 Objetivos

Esta etapa do projeto de construção de um compilador para a linguagem Goianinha consiste na geração da representação intermediária do programa fonte, na utilização desta representação para a análise semântica e para a geração do código na linguagem *assembly* do processador MIPS. As seções seguintes descrevem cada uma das fases desta etapa do trabalho.

2 Geração da Representação Intermediária

A representação intermediária corresponde a uma árvore sintática abstrata que é gerada durante a fase de análise sintática (ver Seção 2.8 e 5.3 do livro-texto[1]). Ações semânticas devem ser acrescentadas às produções da gramática no arquivo de entrada para o Bison ou Yacc, formando assim, o esquema de tradução necessário para gerar a árvore abstrata para programas de entrada escritos em Goianinha. A árvore abstrata criada deve ser armazenada na memória principal do computador e servirá de entrada para as fases seguintes que correspondem à análise semântica e à geração de código em *assembly*. Os alunos devem se orientar pelo exemplo de construção de árvore sintática abstrata apresentado na Seção 2.8 do livro-texto.

[1] Alfred Aho et al. Compiladores - Princípios, Técnicas e Ferramentas. segunda edição, Pearson Addison Wesley, São Paulo, 2007.

É importante que a árvore abstrata seja a mais compacta possível, pois durante a fases seguintes será necessário fazer percursos na árvore para detectar erros semânticos e gerar o código em linguagem *assembly*. Os itens léxicos (identificadores, nomes de comandos, operadores, etc) devem ser armazenados na árvore juntamente com a linha onde eles ocorrem no programa fonte, para que as mensagens de erro possam se referir à linha onde o erro foi detectado.

3 Analisador Semântico

O segundo objetivo desta etapa do trabalho corresponde à implementação do analisador semântico do compilador. Entretanto, para que o analisador semântico possa funcionar, é necessário a implementação prévia da tabela de símbolos para que os diversos identificadores e seus atributos que aparecem em um programa fonte possam ser armazenados e utilizados durante a análise semântica e a geração de código.

Aspectos Semânticos da Linguagem Goianinha

Declaração de variáveis e parâmetros formais

As regras de escopo e de tempo de vida das variáveis, funções e parâmetros de funções da linguagem Goianinha são semelhantes às regras da linguagem C, porém há as seguintes restrições:

- Não há o conceito de protótipo de função. Todas funções utilizadas no programa principal devem ser completamente declaradas (cabeçalho e corpo) antes da declaração programa que representa o bloco principal do programa.
- Todas as variáveis declaradas antes da declaração do bloco principal (bloco identificado pela palavra-chave programa) são globais às funções declaradas após as declarações dessas variáveis e também são globais ao bloco principal.
- Não há o conceito de variáveis static como na linguagem C. O tempo de vida de uma variável local a um bloco corresponde à duração do bloco.
- Todo programa em Goianinha ocupa apenas um arquivo e não há compilação separada. Portanto, não há declarações do tipo extern como na linguagem C.
- Assim como na linguagem C, uma variável global ao programa é uma variável declarada fora de qualquer função. Uma variável declarada em um bloco é local a esse bloco e global a todos os blocos internos a esse bloco, em qualquer profundidade de aninhamento de blocos. Entretanto, se uma variável local a um bloco tem o mesmo nome de uma variável global a esse bloco, a variável local se sobrepõe à variável global durante a duração do bloco.

- Os parâmetros formais têm escopo de variável local. Não é permitido uma variável local com mesmo nome de um parâmetro formal no bloco mais externo que forma a função. Entretanto, é permitido criar uma variável com mesmo nome de um parâmetro formal, em um bloco aninhado. Por exemplo:

```
int f(int a, int b){ int a, b; ....} -- não permitido.
int f(int a, int b) {....{int a,b; .....}  .... } -- permitido
```

- Ao contrário da linguagem C, na linguagem Goianinha o número de parâmetros formais de uma função e o número de parâmetros de uma chamada à função deve ser o mesmo.

- A declaração de qualquer nome deve preceder o seu uso em um escopo válido, isto é, um nome pode ser utilizado em um comando somente se ele tiver sido declarado previamente e se o comando estiver no escopo de sua declaração.

Tipos e Checagem de Tipos

A linguagem Goianinha possui apenas dois tipos distintos: int e car. A linguagem não permite operações entre objetos ou expressões de tipo simples (car ou int) distintos. O tipo de cada parâmetro formal deve ser o mesmo de cada parâmetro de chamada correspondente.

Comandos e Expressões

Não existe o tipo lógico explícito na linguagem Goianinha, entretanto nos comandos se e enquanto é necessário avaliar se uma expressão é verdadeira ou falsa para a tomada de uma decisão quanto ao fluxo de execução dos comandos internos a estes dois comandos. Assim como a linguagem C, a linguagem Goianinha trata como verdadeira uma expressão cujo valor é diferente de zero e como falsa, uma expressão cujo valor é zero. Obviamente, essa avaliação é possível somente em tempo de execução do programa fonte. Entretanto, a nível de análise semântica, uma expressão lógica (com operações OR, AND ou NEGAÇÃO) ou relacional (com operadores relacionais – >, <,<=, etc) devem receber tipo int.

Uma expressão retornada por uma função deve ter o mesmo tipo da função. A expressão do lado direito de uma atribuição deve ter o mesmo tipo da variável que recebe a expressão. Os operadores aritméticos devem ser aplicados em expressões do tipo int. Os operadores relacionais devem ser aplicados a operandos de mesmo tipo.

O analisador semântico deve percorrer a árvore abstrata gerada para uma cadeia de entrada e checar, gradualmente, a correção semântica de cada construção presente na árvore, seguindo as regras semânticas apresentadas nesta seção. Qualquer ocorrência de uma construção no programa fonte que não é permitida na linguagem Goianinha deve ser considerada um erro semântico. Nesse caso, o analisador semântico deve reportar o erro semântico encontrado, o número da linha onde ele ocorreu e terminar a compilação.

Geração de Código

Uma vez que o analisador semântico detecta que o programa de entrada (representado internamente pela árvore abstrata) não possui erros semânticos, ele deve ativar o gerador de código. O módulo gerador de código dever percorrer novamente a árvore abstrata e gerar instruções em linguagem assembly, à medida em que caminha na árvore abstrata.

O código será gerado na linguagem assembly do processador MIPS para que possa ser executado no SPIM que é um simulador arquitetura MIPS. Uma descrição da linguagem assembly a ser utilizada pode ser encontrada no Capítulo 3 e no Apêndice A do livro Organização e projeto de computadores, Dad A. Patterson e John L. Hennessy.

4 Informações Sobre Implementação e Entrega

Representação intermediária e análise semântica

O trabalho completo deve ser entregue via tarefa específica disponível no site da disciplina na Plataforma Turing, até o dia, 30/06/2025 às 23:50. Porém, o trabalho deve ser apresentado ao professor nas aulas dos dias 23, 26 e 30 de junho, mesmo que a implementação ainda esteja incompleta.

Especificamente, deve ser entregue o seguinte:

1. O arquivo de especificações para o gerador de analisador léxico utilizado (arquivo de entrada para o Flex/Lex).
2. O arquivo de especificações para o gerador de analisador sintático expandido com ações semânticas necessárias para a geração de árvore sintática abstrata (arquivo de entrada para o Yacc/Bison).
3. O código em C/C++ que implementa a tabela de símbolos.
4. O código em C/C++ do analisador semântico que irá utilizar a árvore abstrata e a tabela de símbolos para detectar a ocorrência de erros semânticos em arquivos contendo programas escritos na linguagem Goianinha.
5. O código me C/C++ que irá utilizar a árvore abstrata e a tabela de símbolos para gerar o código objeto em linguagem *assembly*.
6. Um arquivo Makefile com comandos para:
    - Gerar o código do analisador léxico a partir do arquivo contendo as especificações para o gerador de analisador léxico.
    - Gerar o código do analisador sintático e construtor da árvore abstrata a partir do arquivo contendo as especificações para o gerador de analisador sintático.

- Compilar os arquivos gerados e os escritos na linguagem de programação adotada no projeto (C/C++)
- Gerar o executável (O professor deve ser capaz de obter o executável do trabalho digitando apenas o comando *make* em uma *shell*).

Os itens acima devem estar agrupados em uma pasta, compactada com o utilitário *gzip*. O referido arquivo deve ser submetido na tarefa criada na plataforma Turing, com esse objetivo. **Os códigos gerados devem estar preparados para executarem no sistema operacional Linux (ambiente onde os trabalhos serão avaliados)**.

IMPORTANTE

Cópia de partes de código implicam em nota ZERO na NOTA FINAL do trabalho para todos os alunos envolvidos (os que copiarem e os que cederem o trabalho para cópia).