Projeto de Compilador E2 de **Análise Sintática**

Prof. Lucas Mello Schnorr schnorr@inf.ufrgs.br

1 Introdução

A segunda etapa consiste em construir um analisador sintático utilizando a ferramenta de geração de reconhecedores bison. O arquivo tokens.h da etapa anterior desaparece, e deve ser substituído pelo arquivo parser. y (fornecido, mas que deve ser modificado para atender a esta especificação) com a declaração dos tokens. A função principal deve estar em um arquivo main.c, separado do arquivo scanner.1 (léxico, da etapa 1) e do parser.y (sintático, por codificar na etapa 2). A solução desta etapa deve ser composta de arquivos tais como scanner.1, parser.y, e outros arquivos fontes que o grupo achar pertinente (devem ser compilados usando o Makefile que deve executar flex e bison). No final desta etapa o analisador sintático gerado pelo bison a partir da gramática deve verificar se a sentença fornecida – o programa de entrada a ser compilado – faz parte da linguagem ou não.

2 Funcionalidades Necessárias

2.1 Definir a gramática da linguagem

A gramática da linguagem deve ser definida a partir da descrição geral da Seção "A Linguagem", abaixo. As regras gramaticais devem fazer parte do parser.y, arquivo este que conterá a gramática usando a sintaxe do bison.

2.2 Relatório de Erro Sintático

Se a análise sintática tem sucesso, o programa retorna zero. Esse é o comportamento padrão da função yyparse(), chamada pela função main do programa. Caso contrário, deve-se retornar o valor retornado pela função yyparse() e imprimir uma mensagem de erro informando a linha do código da entrada que gerou o erro sintático e informações adicionais que auxiliem o programador que está utilizando o compilador a identificar o erro sintático identificado. Não encerre o programa de maneira não estruturada (evite chamar exit). O compilador deve parar ao encontrar o primeiro erro sintático. Use

%define parse.error verbose no cabeçalho do arquivo parser.y para obter a mensagem de erro com mais detalhamento.

2.3 Remoção de conflitos gramaticais

Deve-se realizar a remoção de conflitos Reduce/Reduce e Shift/Reduce de todas as regras gramaticais. Estes conflitos devem ser resolvidos através da reescrita da gramática de maneira a evitá-los. Os conflitos podem ser melhor identificados e compreendidos através de uma análise cuidadosa do arquivo parser.output gerado automaticamente quando o bison é executado com a opção —report—file. Sugere-se fortemente um processo construtivo da especificação em passos, verificando em cada passo a inexistência de conflitos. Por vezes, a remoção de conflitos exige uma revisão mais profunda da gramática.

3 A Linguagem

Um programa na linguagem é composto por uma lista de funções, sendo esta lista opcional.

3.1 Definição de Funções

Cada função é definida por um cabeçalho e um corpo. O cabeçalho consiste no nome da função, o caractere igual '=', uma lista de parâmetros, o operador maior '>' e o tipo de retorno. O tipo da função pode ser float ou int. A lista de parâmetros é composta por zero ou mais parâmetros de entrada, separados por TK_OC_OR. Cada parâmetro é definido pelo seu nome seguido do caractere menor '<', seguido do caractere menos '-', seguido do tipo. O corpo da função é um bloco de comandos.

3.2 Bloco de Comandos

Um bloco de comandos é definido entre chaves, e consiste em uma sequência, possivelmente vazia, de comandos simples cada um **terminado** por ponto-e-vírgula. Um bloco de comandos é considerado como um comando único simples, recursivamente, e pode ser utilizado em qualquer construção que aceite um comando simples.

3.3 Comandos Simples

Os comandos simples da linguagem podem ser: declaração de variável, atribuição, construções de

fluxo de controle, operação de retorno, um bloco de comandos, e chamadas de função.

Declaração de Variável: Consiste no tipo da variável seguido de uma lista composta de pelo menos um nome de variável (identificador) separadas por vírgula. Os tipos podem ser int e float. Uma variável pode ser opcionalmente inicializada caso sua declaração seja seguida do operador composto TK_OC_LE e de um literal.

Comando de Atribuição: O comando de atribuição consiste em um identificador seguido pelo caractere de igualdade seguido por uma expressão.

Chamada de Função: Uma chamada de função consiste no nome da função, seguida de argumentos entre parênteses separados por vírgula. Um argumento pode ser uma expressão.

Comando de Retorno: Trata-se do token return seguido de uma expressão.

Comandos de Controle de Fluxo: A linguagem possui uma construção condicional e uma iterativa para controle estruturado de fluxo. A condicional consiste no token if seguido de uma expressão entre parênteses e então por um bloco de comandos obrigatório. O else, sendo opcional, deve sempre aparecer após o bloco do if, e é seguido de um bloco de comandos, obrigatório caso o else seja empregado. Temos apenas uma construção de repetição que é o token while seguido de uma expressão entre parênteses e de um bloco de comandos.

3.4 Expressões

Expressões tem operandos e operadores, sendo este opcional. Os **operandos** podem ser (a) identificadores, (b) literais e (c) chamada de função ou (d) outras expressões, podendo portanto ser formadas recursivamente pelo emprego de operadores. Elas também permitem o uso de parênteses para forçar uma associatividade ou precedência diferente daquela tradicional. A associatividade é à esquerda (portanto implemente recursão à esquerda nas regras gramaticais). Os **operadores** são os seguintes:

- Unários prefixados
 - inverte o sinal
 - ! negação lógica
- Binários infixados
 - + soma
 - subtração
 - * multiplicação
 - / divisão
 - % resto da divisão inteira
 - operadores compostos

As regras de associatividade e precedência de operadores matemáticos são aquelas tradicionais de linguagem de programação e da matemática. Pode-se usar esta referência da Linguagem C. Para facilitar, abaixo temos uma tabela com uma visão somente com os operadores de nossa linguagem. Recomenda-se fortemente que tais regras sejam incorporadas na solução desta etapa através de construções gramaticais, evitando as diretivas %left %right do manual do bison.

Precedência	Op.	Aridade
1	_	Unária
	!	Unária
2	*	Binária
	/	Binária
	용	Binária
3	+	Binária
	_	Binária
4	<	Binária
	>	Binária
	TK_OC_LE	Binária
	TK_OC_GE	Binária
5	TK_OC_EQ	Binária
	TK_OC_NE	Binária
6	TK_OC_AND	Binária
7	TK_OC_OR	Binária