UAlg FCT
UNIVERSIDADE DO ALGARVE
FACULDADE DE CIÊNCIAS E TECNOLOGIA

Compiladores

2020-2021

Projeto 2 – Análise Sintática

Introdução

No 2.º Projeto pretende-se implementar as regras de *parsing* – regras da gramática - da linguagem, cujo compilador será desenvolvido na cadeira de compiladores ao longo do semestre. A linguagem ALG é baseada na linguagem FIR[1].

Objetivos e requisitos

Partindo do ficheiro de análise lexical implementado no 1.º Projeto de Compiladores, deverão agora usar a ferramenta ANTLR para especificar um ficheiro de regras sintáticas, com o nome *alg.g4* que construa uma árvore de *parsing* para um dado ficheiro da linguagem ALG, de acordo com a especificação que se apresenta na secção seguinte.

As regras gramaticais implementadas têm que estar fatorizadas à esquerda, e não podem ser recursivas à esquerda.

Embora se tenha evitado fazê-lo, durante a especificação da gramática, poderão ser descritas algumas restrições semânticas para melhor ilustrar algumas das componentes da gramática. Certas restrições podem ser implementadas quer através de regras sintáticas, quer através de uma restrição semântica. Um destes casos é a análise e verificação dos tipos das expressões, por exemplo a operação soma só está definida para expressões do tipo Inteiro ou Real. Implementar esta restrição do lado da gramática seria possível, mas iria complicar imenso a mesma. As restrições semânticas não deverão ser implementadas no projeto 2 (serão implementadas no 3.º projeto)

Devem capturar erros lexicais e erros de *parsing* e apresentar ao utilizador do compilador uma informação de erro que seja apropriada e percetível.

Linguagem ALG (especificação da gramática)

Na especificação que se apresenta a seguir, quando é apresentada uma forma ou sequência, utiliza-se elementos a negrito para indicar símbolos terminais da linguagem (tokens) e elementos a itálico para representar símbolos não terminais. No entanto cabe a cada grupo decidir que símbolos não terminais usar, e poderão acrescentar outros símbolos não terminais (ou remover alguns dos apresentados). Os nomes dados aos símbolos não terminais são apenas sugestões, são livres de escolher outros nomes caso o entendam.

Programa

Um programa válido na linguagem ALG corresponde a uma sequência de uma ou mais declarações.

Declarações

Uma declaração pode ser uma declaração de variável seguida de **;** (ponto e vírgula) ou uma declaração de função.

[1] A Linguagem FIR foi uma linguagem usada como projeto da cadeira de Compiladores no Instituto Superior Técnico no ano letivo 2020/2021.

Declaração de variável

Uma declaração de variável pode ser uma declaração simples, ou uma declaração com inicialização.

Um declaração simples corresponde à uma sequência com uma única declaração *tipo identificador* ou a uma sequência com múltiplas declarações de identificadores com o mesmo tipo: *tipo identificador***,** *identificador***, …,** *identificador*

Exemplos:

- Inteiro: **int i;**
- Inteiros: **int i,j;**
- Booleano: **bool b;**
- Real: **float f;**
- Cadeia de caracteres: **string s;**
- Ponteiro para inteiro: **<int> p1;** (equivalente a int* em C)
- Ponteiro para cadeia de caracteres: **<string> s;**

Declaração com inicialização

Uma declaração com inicialização declara e inicializa uma única variável, e tem 2 formas: a forma *tipo identificador* **=** *expressão* e a forma *tipo identificador* **= [***expressão***]**

Exemplos:

- Inteiro (literal): **int i = 3;**
- Inteiro (expressão): **int i = j + 1;**
- Real (expressão): **float x = i – 2.5 + f(3);**
- Ponteiro (literal): **<float> p = null;**

A segunda forma é usada para reservar memória no stack actual para um vector (array) de objectos de um determinado tipo. Por exemplo, <float> p = [5] reserva memória para um array de 5 números reais.

Tipo

Um tipo pode ser um dos tokens correspondentes aos tipos primitivos da linguagem **int**, **float**, **bool** e **string**, ou pode ser um ponteiro para um dos tipos primitivos. Quando é usado um tipo ponteiro, o tipo é colocado entre os tokens < >. Por exemplo, **<float>** representa um ponteiro para um float. Não é permitido a utilização de ponteiros para ponteiros, portanto <<float>> não é um tipo válido.

Expressões

Uma expressão é uma representação algébrica de uma quantidade: todas as expressões têm um tipo e devolvem um valor. Uma expressão pode ser uma expressão simples ou uma avaliação de expressão.

Uma expressão simples corresponde a um literal (literal inteiro, literal ponteiro[2], literal real, literal booleano[3] ou cadeia de caracteres), ou a uma invocação de uma função.

A avaliação de uma expressão corresponde à aplicação de um ou mais operadores a uma ou mais expressões. Existem operadores unários e operadores binários. Por exemplo, se E for uma expressão **(**E**)** é também uma expressão. Se E1 e E2 forem expressões E1 **+** E2 é também uma expressão. A seguinte tabela indica todos os operadores que podem ser usados para avaliação de expressões, e a respetiva prioridade entre eles. Os operadores no topo da tabela têm maior prioridade, e os operadores em baixo têm menor prioridade. Operadores na mesma linha têm a mesma prioridade.

[2] Só existe um literal ponteiro que é o token null.
[3] Um literal booleano corresponde aos tokens true e false.

Tipo de operação	Operadores	tipo operador	Exemplo
Parênteses[4]	**( )**	Unário	**(***E***)**
Indexação de ponteiro[5]	**[ ]**	Binário	*E1***[***E2***]**
Identidade, simétrico, negação, extração de ponteiro[6]	**+, - , ~,?**	Unário	*-E*
Multiplicação e divisão	***, /, %**	Binário	*E1* ***** *E2*
Soma e subtração	**+, -**	Binário	*E1* **+** *E2*
Comparadores	**<,>,>=, <=, ==, !=**	Binário	*E1* **!=** *E2*
E lógico	**&&**	Binário	*E1* **&&** *E2*
Ou lógico	**||**	Binário	*E1* **||** *E2*

Alguns exemplos de expressões válidas:

1 + (2 * f(5))

2 >= p[3+1]

?a+1[2]

((?p[2])[0]&&xpto) == true

Funções

Declarações de funções

Uma declaração de funções corresponde a uma sequência *tipo identificador* **(***argumentos***)** *corpo*. Para além dos tipos normais, uma função pode ser declarada com o tipo **void**, quando não devolve nada. As declarações de variáveis usadas como argumentos de uma função podem ser vazias, uma declaração simples, ou uma sequência de declarações simples separadas por **,** (vírgula).

Para além do caso mais geral, é possível declarar uma função especial com a sequência **int alg(int n, <string> args) corpo** que será usada para iniciar um programa na linguagem alg.

Corpo

Um corpo de uma função é uma sequência *prólogo bloco epílogo*, em que prólogo e epílogo são opcionais. O bloco central é obrigatório. O prólogo corresponde a um bloco precedido do token **@**, e o epílogo corresponde a um bloco precedido do token **>>**.

Um bloco corresponde à sequência **{**[7] *declarações_variáveis instruções* **}**. As declarações de variáveis podem ser vazias, mas tem que existir pelo menos uma instrução dentro de um corpo.

[4] Uma expressão entre parênteses curvos tem o valor da expressão sem os parênteses e é usada para alterar a prioridade dos operadores.
[5] Uma indexação de ponteiros devolve o valor de uma posição de memória indicada por um ponteiro. Corresponde a uma sequência *E1***[***E2***]** em que a primeira expressão é do tipo ponteiro e a segunda é uma expressão do tipo inteiro.
[6] Uma extração de ponteiro corresponde à sequência **?***E* e é usada para retornar o endereço (ponteiro) de uma zona de memória. Por exemplo, se a é uma variável **?a** devolve o ponteiro para essa variável. **?p[2]** devolve um ponteiro para a zona de memória correspondente ao índice 2 do *array* p. O operador ? só pode ser aplicado a expressões que correspondem ao lado esquerdo de uma atribuição.
[7] Deverão adicional os tokens **{** e **}** à linguagem, uma vez que por lapso, não foram incluídos na especificação do 1,º projeto.

Invocação de funções

Uma invocação de funções tem a forma *identificador***(***lista de expressões***)** em que a lista de expressões é uma sequência de expressões (potencialmente vazia) separadas por **,** (vírgula).

Para além da invocação usando um identificador, é possível também invocar as funções especiais **@**, **sizeof**, **write**, e **writeln**. A função @ tem a forma **@()**, ou seja é uma função sem argumentos, e devolve o resultado de ler do standard input; a função sizeof tem a forma **sizeof(***expressão***)** enquanto que a funções **write** e **writeln** recebem qualquer número de expressões como argumento e escrevem o resultado da sua avaliação no ecrã.

Instruções

Uma instrução pode ter várias formas:

- *expressão***;**
- *instrução_controle***;**
- *atribuição***;**
- *instrução_condicional*
- *ciclo*
- *subbloco*

Instrução de controle

Uma instrução de controle é usada para sair de blocos, ou de ciclos, ou para reiniciar um ciclo. Tem as seguintes formas:

- **leave**
- **restart**
- **return**
- **return** *expressão*

Atribuição

Uma instrução de atribuição é usada quando se pretende atribuir um valor a uma zona de memória, e tem a forma *lado_esquerdo* **=** *expressão*

O lado esquerdo de uma atribuição tem que corresponder a uma zona de memória, e, portanto, pode apenas ser um identificador ou uma indexação de ponteiros.

Instrução condicional

Uma instrução condicional corresponde a um if em C, e tem a forma **if** *expressão* **then** *instrução* . Esta forma pode, opcionalmente, ser seguida de **else** *instrução*

Ciclo

Um ciclo é representado pela sequência **while** *expressão* **do** *instrução* , ou pela sequência **while** *expressão* **do** *instrução* **finally** *instrução*.

O segundo caso é usado quando queremos especificar algo extra a fazer depois do ciclo terminar.

Subbloco

Um subbloco serve para organizar um bloco de instruções, no entanto ao contrário do bloco não permite a declaração de variáveis, e não obriga a que haja pelo menos uma instrução. Tem a forma: **{** *instrução* … *instrução* **}**, podendo a sequência de instruções ser vazia.

Condições de realização

O projeto deve ser realizado em grupo, de acordo com as inscrições em grupo do laboratório. Projetos iguais, ou muito semelhantes, originarão a reprovação na disciplina. O corpo docente da disciplina será o único juiz do que se considera ou não copiar num projeto.

O 2.º projeto vale 20% da nota final da componente prática. Devido ao facto de ser relativamente simples, esta segunda parte também não será avaliada automaticamente, e, portanto, não será necessária a submissão via *Mooshak*. Deverão submeter ambos os ficheiros alg.g4 e algLexer.g4 com as regras sintáticas e lexicais obrigatoriamente por via eletrónica, através da tutoria, até às **23:59** do dia **23/04/2021**.

Os alunos terão de validar o código juntamente com o docente durante o horário de laboratório correspondente ao turno em que estão inscritos, na semana de 26 a 30 de Abril. Será feita uma breve discussão com cada grupo, e serão feitos pedidos para alterações ligeiras nas regras sintáticas. Embora a realização do projeto seja em grupo, a nota do projeto é individual e dependerá da prestação de cada elemento na discussão do projeto.