Universidade Federal de Alagoas

Instituto de Computação

Antonio Manoel de Lima Neto

Blaise

13 de Julho de 2018

Universidade Federal de Alagoas

Instituto de Computação

Antonio Manoel de Lima Neto

Blaise

Trabalho Acadêmico solicitado pelo Prof. Alcino Dall Igna Júnior, com vistas na obtenção parcial de nota da disciplina Compiladores do Curso de Bacharel em Ciência da computação - UFAL

13 de Julho de 2018

Sumário

1. Nomes, Vinculações, Verificação de Tipos e Escopo

1.1. Formas de nome

Relativo às questões de projeto, temos:

- Os identificadores podem possuir qualquer comprimento.
- A grafia dos identificadores deve ser composta de todos os caracteres constituintes em ordem textual, ignorando sensitividade. Deve ter comprimento maior ou igual a 1, iniciado com caractere e seguido por caracteres, dígitos ou o caractere de conexão.
- Nenhum identificador deve ter a mesma grafia de qualquer outra palavra-símbolo.
- O único caractere de conexão permitido é o "_".
- É case-sensitive.
- As palavras-chave (*keywords*) são reservadas, e serão listadas a seguir:

KEYWORD	**DESCRIÇÃO**
const	Define um valor imutável, seja inteiro, real, char ou cadeia de caracteres.
integer	Descreve uma variável que agrega um valor de número inteiro com valor positivo ou negativo.
real	Descreve uma variável que agrega um valor de número de ponto flutuante.
char	Descreve uma variável onde o conteúdo tem comprimento com restrição à apenas um caractere.
string	Descreve uma variável onde o conteúdo tem comprimento sem restrição de um caractere.
boolean	São basicamente do tipo inteiro. Possuem dois possíveis valores pré-definidos.
if/then	Execução de bloco com condicional sempre booleana.
else	Uma condição opcional pode seguir o *if*, sendo executada quando a expressão booleana é falsa.

for	Um comando de repetição que permite computações repetidas um número específico de vezes.
while	Um comando de repetição que permite computações repetidas até uma condição testada ser satisfeita.
read/readln	Responsável por fazer leitura de entrada através do teclado em uma variável.
write/writeln	Responsável por realizar a apresentação do conteúdo de uma variável.
not	Negação lógica.

1.2. Variáveis

A declaração e conversão devem ser explícitas. As variáveis são dinâmicas na pilha, e seguem as regras de compatibilidade de tipo nominal.

Coerções suportadas
Real para inteiro
Inteiro para real
Char para inteiro

1.3. Escopo

O escopo é do tipo estático e os blocos são delimitados pelos termos **begin** e **end**. Identificadores podem ser locais ou globais e existem regras para acessar identificadores que não estão declarados localmente.

Além disso, também há devidas regras para lidar com procedimentos aninhados dentro de outros procedimentos. Qualquer coisa declarada em um bloco que contenha procedimentos aninhados é **não-local** aquele procedimento aninhado (e.g. Identificadores globais são não-locais com respeito a todos os outros blocos, exceto ao programa principal.) O processo de acessar um identificador declarado fora de seu bloco é considerado um **acesso não-local**.

As regras são como a seguir:

1. O escopo de um identificador inclui todas as declarações dentro do bloco que contém sua definição. Isto inclui blocos aninhados, exceto pelo que é denotado na regra 2.
2. O escopo de um identificador não se estende a nenhum bloco aninhado que contenha um identificador definido localmente com a mesma grafia.
3. O escopo de um parâmetro é idêntico ao escopo de uma variável local no mesmo bloco.

```
program scope_example;
var
      A1: Integer; (* variável global *)
procedure bloco1(A1: Integer); (* Previne acesso a variável
global A1 *)
      var B1, B2 : Integer; (* variável local ao bloco 1 *)
      begin;
      ..
      end;
procedure bloco2;
      var
            A1, (* Previne acesso a variável global A1 *)
            B2 :  (* variável local ao bloco 2, não conflita
      com B2 do bloco 1 *)
                  Integer;
      begin;
      ..
      end;
```

1.4. Constantes nomeadas

São variáveis declaradas com a palavra reservada ***const*** onde o valor, uma vez definido, não pode ser alterado.

```
const <identifier> := <valor>;
```

```
const
      nemesis := 'starrrrrrrrs!';
      jill := 'what the!';
```

2. Tipos de Dados

Tipo numérico: Há números inteiros e de ponto flutuante, variáveis desse tipo devem ser declaradas com as palavras reservadas *integer* e *real*, respectivamente. As operações suportadas são: adição (+), subtração (-), multiplicação(*) e divisão (/).

Já os caracteres utilizam a codificação ASCII e devem ser declaradas com a palavra reservada *char*. Cadeias de caracteres são definidas como na linguagem Pascal, utilizando arranjo unidimensional e finalizadas com o caractere especial "/0".

3. Expressões e instruções de atribuição

3.1. Operadores

Operadores aritméticos

Ordem	Operação	Operador	Associatividade	Operandos	Resultado
1º	Parênteses	( )	Não associativo	-	-
2º	Multiplicação	*	Esquerda	inteiro ou real	inteiro ou real
2º	Divisão	/	Esquerda	inteiro ou real	real
3º	Adição	+	Esquerda	inteiro ou real	inteiro ou real
3º	Subtração	-	Esquerda	inteiro ou real	inteiro ou real

Operadores relacionais

Operação	Operador	Associatividade	Operandos	Resultado
Menor que	<	Não associativo	inteiro, real, booleano, char	booleano
Maior que	>	Não associativo	inteiro, real, booleano, char	booleano
Menor ou igual que	<=	Não associativo	inteiro, real, booleano, char, array	booleano
Maior ou igual que	>=	Não associativo	inteiro, real, booleano, char	booleano
Igual	=	Não associativo	inteiro, real,	booleano

			booleano, char, array	
Diferente	<>	Não associativo	inteiro, real, booleano, char, array	booleano

Operadores lógicos

Operação	Operador	Associatividade	Resultado
Conjunção	and	Esquerda	booleano
Disjunção	or	Esquerda	booleano
Negação	not	Direita	booleano

Operadores binários

Operação	Operador	Associatividade	Operandos	Resultado
bitwise not	~	Não associativo	inteiro	inteiro
bitwise and	&	Não associativo	inteiro	inteiro
bitwise or	|	Não associativo	inteiro	inteiro

Concatenação de Cadeias de Caracteres

A concatenação de cadeias de caracteres será dada pelo operador (+). Será criada uma nova cadeia de caracteres com o tamanho total das duas anteriores somadas e atribuído o conteúdo de ambas.

Ordem de Precedência

Precedência	Operador
Mais Alta	~, not
	*, /, and, div
	+, - (binário)
	<, >, <=, >=, ==, !=
	&

	|, + (operador de concatenação)
Mais baixa	= (operador de atribuição)

4. Instrução

Os arranjos unidimensionais devem ser declarados de acordo com a seguinte sintaxe:

```
<variável>: array [<valor-inicial>..<valor-final>] of <tipo>;
```

A estrutura condicional mais simples, frequentemente usada para mudar o fluxo da execução de um programa, tem a sintaxe a seguir:

```
if condição then S;
```

Onde a condição é um booleano ou condição relacional, já a estrutura condicional de duas ou mais vias é da seguinte forma:

```
if condição then S1 else S2;
```

A estrutura de laço com controle lógico permite a repetição de computações até a condição ser satisfeita. A validação é realizada ao início de cada iteração, e a sintaxe é da seguinte forma:

```
while(condição) do S;
```

A estrutura de laço controlada por contador, onde o tipo da variável controladora do laço é um inteiro e a verificação da condição é feita uma vez a cada início da iteração.

```
for <variável> := <valor-inicial> to <valor-final> do
```

O comando de entrada é utilizado para receber dados digitados pelo usuário. Os dados recebidos são armazenados em variáveis. Esse comando é representado pela palavra-chave **Read** ou **ReadIn**. Ambos desempenham a mesma função, a única diferença é que após a entrada de dados com **Read** o cursor fica na mesma linha, e no caso do **ReadIn** o cursor vai para a próxima linha. A mesma diferença é válida para **write** e **writeln**.

Função	Sintaxe
Entrada	**read**/**readln** (nome-da-variável, nome-da-variável-2, ...);
Saída	**write**/**writeln**(nome-da-variável);

Os tipos, por sua vez, serão representados da seguinte maneira:

5. Atribuição

O operador (:=), associativo à *direita*, será utilizado para atribuição.

```
x := 200 + 100;
y := num * x;
```

6. Funções

Para declaração de funções identifica-se através da palavra reservada **function** seguida do nome da função. Os parâmetros são especificados com seus tipos dentro de parênteses, separados por (;). Para indicar o tipo do retorno da função deve-se adicionar um último parâmetro após os parênteses com uma palavra reservada de tipo de dados. Um exemplo da sintaxe é dada a seguir;

```
function nome-da-função(param1: tipo; param-n: tipo): tipo-de-retorno;
```

7. Exemplos

- Alô Mundo

```
program Hello;
begin
     writeln ('Hello World.');
end.
```

- Série de Fibonacci

```
program Fibonacci;
function Fibonacci(n: Integer): Integer;
var
     i, u, v, w, Result: Integer;
begin
     if n <= 0 then
          write('0');
          exit(0);
     if n = 1 then
```

```
            write('0, 1');
            exit(1);
        u := 0;
        v := 1;
        write('0, 1')
    for i := 2 to n do
    begin
        w := u + v;
        u := v;
        v := w;
        write(', ');
        write(w);
    end;
    Result := v;
end.
```

- Shell Sort

```
const

     read(MaxN);

type

     TArray := array[0..Max] of Integer;

function shellsort(var A : TArray; N : Integer);

var
     i, j, step, tmp: Integer;
begin
     step := N div 2;
     while step > 0 do
     begin
          for i := step to n do
          begin
               tmp := A[i];
               j := i;
               while (j >= step) and (A[j-step] > tmp) do
               begin
                    A[j] := A[j=step];
                    dec(j, step);
               end;
          A[j] := tmp;
          end;
     step := step div 2;
     end;
end.
```