SBVCONC: Construção de Compiladores

Aula 14: Arrays

Bacharelado em Ciência da Computação
Prof. Dr. David Buzatto

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus São João da Boa Vista

Arrays na CPRL

- A CPRL suporta o tipo array de uma dimensão:
  - Os índices são valores inteiros;
  - O índice do primeiro elemento do array é 0;
  - Pode-se declarar arrays de arrays;
- Na declaração de um tipo array são especificados:
  - O nome do tipo array (um identificador);
  - O número de elementos do array, que precisa ser um literal de inteiro ou uma constante;
  - O tipo dos elementos do array;
- **Exemplos:**

```
type T1 = array[100] of Integer;
type T2 = array[10] of T1;
```

Usando Arrays da CPRL

- Para criarmos arrays, precisamos primeiro declarar um tipo array e então declarar uma ou mais variáveis daquele tipo;
- **Exemplos:**

```
type T1 = array[100] of Integer;
type T2 = array[10] of T1;
var a1 : T1;  // contém 100 inteiros, indexados de 0 a 99
var a2 : T2;  // contém 10 arrays de inteiros, indexados de 0 a 9
...

a1[0]     // o inteiro no índice 0 de a1 (o primeiro inteiro)
a2[3]     // o array no índice 3 de a2 (o quarto array)
a2[4][3]  // o inteiro no índice 3 do array no índice 4 de a2
```

Equivalência de Tipos para Arrays

Equivalência de Nome versus Equivalência Estrutural

- Os arrays na CPRL têm o mesmo tipo somente se eles possuem o mesmo tipo. Sendo assim, duas definições de tipo são consideradas diferentes, mesmo que sejam estruturalmente iguais. Essa característica é denominada **"equivalência de nome"** de tipos;
- Dois arrays com o mesmo tipo são compatíveis para atribuição. Dois arrays com tipos diferentes não são compatíveis para atribuição, mesmo quando possuem a mesma estrutura.

Exemplos de Atribuição de Arrays

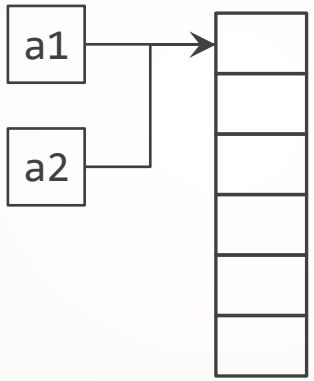
```
type T1 = array[100] of Integer;
type T2 = array[10] of T1;
type T3 = array[100] of Integer;

var a1  : T1;
var a1x : T1;   // a1x tem o mesmo tipo de a1
var a2  : T2;
var a2x : T2;   // a2x tem o mesmo tipo de a2
var a3  : T3;   // a3 não tem o mesmo tipo de a1
...

a1 := a1x;   // permitido (mesmos tipos)
a2 := a2x;   // permitido (mesmos tipos)
a1 := a3;    // *** não permitido na CPRL (tipos diferentes) ***
```

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SÃO PAULO Campus Cubatão - Zona Noroeste

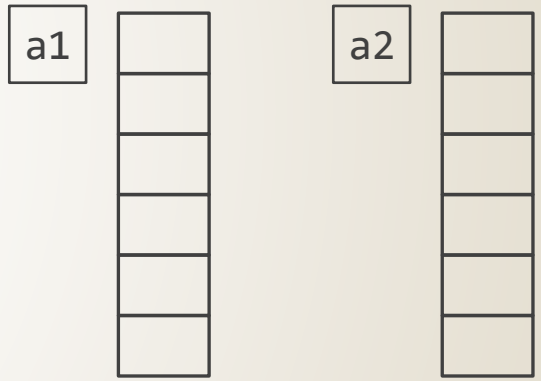
Semântica de Referência versus Semântica de Valor

- A CPRL usa semântica de valor para a atribuição de arrays;
  - Java usa semântica de referência;
- **Exemplo:**

```
a1 := a2;
```



semântica de referência
(a referência é copiada)



semântica de valor
(os valores do array são copiados)

Qual o efeito de modificar o valor de `a2[0]` após a atribuição?

Exemplos Adicionais de Atribuição de Arrays

```
type T1 = array[100] of Integer;
type T2 = array[10] of T1;
var x, y : T2;
...

x := y;                  // atribuição de array (tipo T2)
                         // copia 1000 inteiros (4000 bytes)

x[2] := y[5];            // atribuição de array (tipo T1)
                         // copia 100 inteiros (400 bytes)

x[2][7] := y[5][0]       // atribuição de Integer
                         // copia 1 inteiro (4 bytes)
```

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus São João da Boa Vista

Passando Arrays como Parâmetros

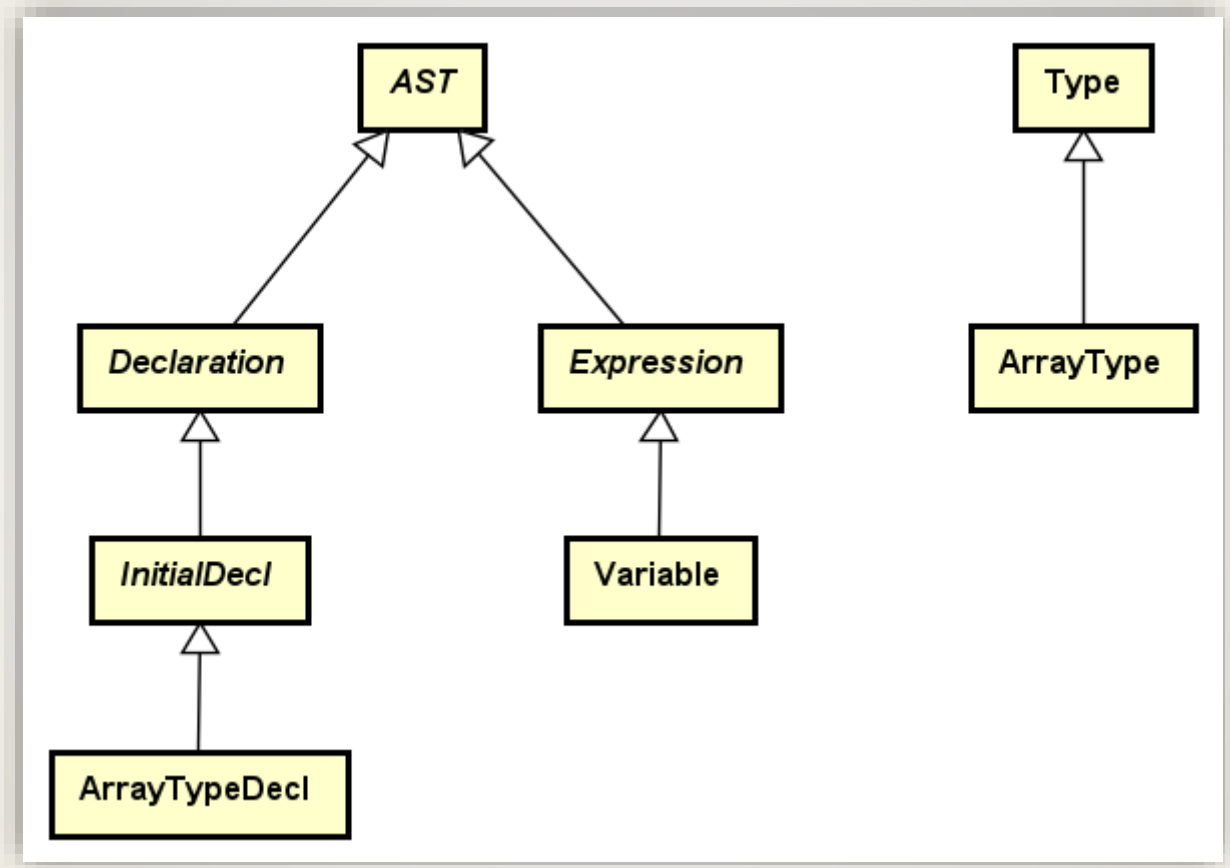
- Assim como os parâmetros de outros tipos (não estruturados), os parâmetros dos tipos de arrays têm semântica similar à da atribuição;
- Passar um array como parâmetro de valor implicará em alocar espaço e copiar o array inteiro;
  - Implica em uso ineficiente de memória se não houver necessidade de copiar o array inteiro;
- Passar um array como parâmetro variável (`var`) implicará em alocar espaço para somente o endereço do array;
  - Tem semântica similar ao que é feito em Java.

Regras Gramaticais Relacionadas aos Arrays

```
initialDecl = constDecl | arrayTypeDecl | varDecl .

arrayTypeDecl = "type" typeId "=" "array" "[" intConstValue "]" "of" typeName ";" .

typeName = "Integer" | "Boolean" | "Char" | typeId .

variable = ( varId | paramId ) ( "[" expression "]" )* .
```

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SÃO PAULO Campus São João da Boa Vista

Classes Relevantes da AST



Métodos de Análise Relevantes
(baseados nas regras gramaticais)

- `InitialDecl parseInitialDecl()`
- `ArrayTypeDecl parseArrayTypeDecl()`
- `Type parseTypeName()`
- `Variable parseVariable()`

Classe `ArrayType`

- Uma declaração de um tipo array cria um novo tipo – o tipo array;
- A classe `ArrayType` encapsula as propriedades de um tipo array;
  - `name` – o nome do tipo array;
  - `numElements` – o número/quantidade de elementos do array;
  - `elementType` – o tipo dos elementos (tipo dos elementos do array);
  - `size` – o tamanho (número de bytes) de uma variável com esse tipo, computado como

```
numElements * elementType.size
```

Endereço de um Array

- O endereço relativo de uma variável com tipo de array é o endereço relativo ao primeiro byte do array;
- O endereço relativo do elemento de um array no índice n é a soma do endereço relativo do array com o deslocamento do enésimo elemento, computado da seguinte forma:

```
relAddr(a[n]) = relAddr(a) + n * elementType.size
```

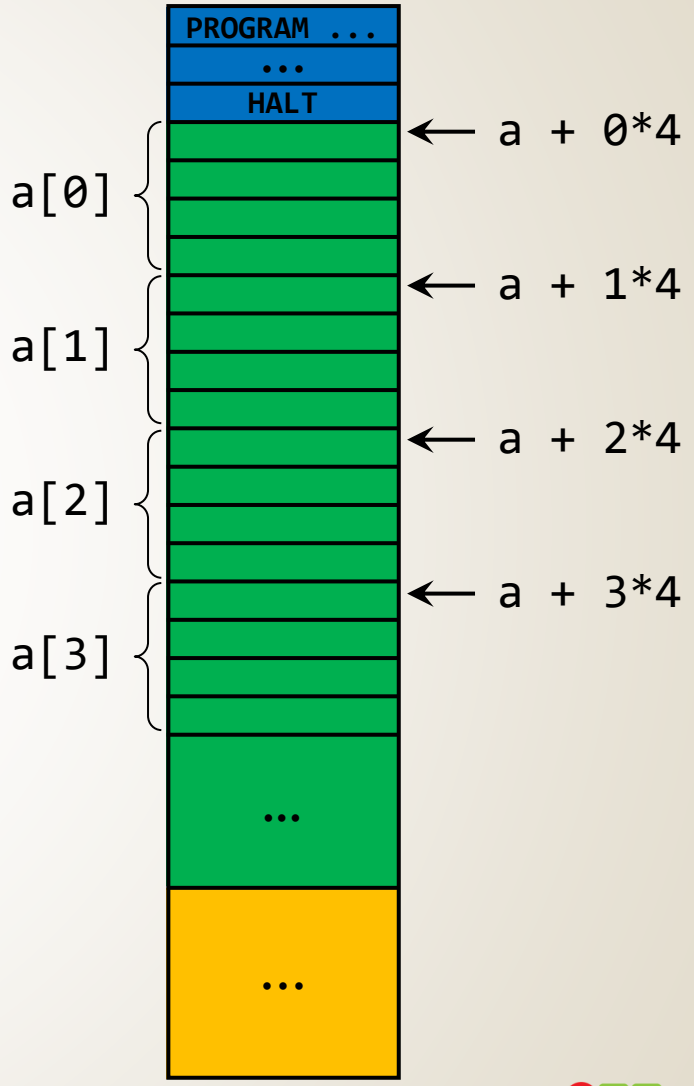
- Note que se o tipo do elemento do array é `Boolean`, então o endereço relativo ao elemento no índice n pode ser simplificado para:

```
relAddr(a[n]) = relAddr(a) + n
```

Exemplo de Índice

```
type T = array[100] of Integer;
var a : T;
```

Se o endereço de memória de `a` for 60, então o endereço de `a[0]` será 60, o endereço de `a[1]` será 64, o endereço de `a[2]` será 68 e assim por diante.



Regras de Restrição para Arrays

- Declaração de Tipo de Array:
  - **Regra de Tipo:** o valor da constante que especifica a quantidade de itens de um array deve ser do tipo `Integer` e o valor associado deve ser um número positivo;
- Variáveis e Valores Nomeados:
  - **Regra de Tipo:** Cada expressão de índice deve ser do tipo `Integer`;
  - **Regra Variada:** O uso de expressões nos índices são permitidas apenas em variáveis de tipos de array.

O Método `checkConstraints()` para a Classe `Variable`

- Considere as declarações abaixo:

```
type T is array[10] of Integer;
var a : T;
```

- **Observação:** `a` tem o tipo `T`, mas `a[i]` tem o tipo `Integer`
- Para cada expressão de índice o método `checkConstraints()` precisa:
  - Verificar se o tipo da expressão é `Integer`;
  - Verificar se o tipo da variável é um tipo de array;
  - Configurar o tipo da expressão com o tipo do elemento do array.

O Método `emit()` para a Classe `Variable`

- Primeiramente, assim como os tipos que não são arrays, o método `emit()` precisa gerar código para deixar o endereço relativo da variável na pilha, isto é, o endereço do primeiro byte do array;
  - Não há necessidade de alterar o código;
- Então, para cada expressão de índice, o método `emit()` precisa:
  1. Gerar código para computar o valor da expressão de índice:

     ```
     expr.emit();
     ```

  2. Gerar código para multiplicar esse valor pelo tamanho do tipo do elemento:

     ```
     emit( "LDCINT " + arrayType.getElementType().getSize() );
     emit( "MUL" );
     ```

  3. Gerar código para adicionar o resultado ao endereço relativo da variável:

     ```
     emit( "ADD" );
     ```

- Como otimização, não é necessário gerar código para o segundo passo acima caso o tipo do array tenha tamanho 1, por exemplo, se o tipo for `Boolean`.

Projeto 7: Implementação da Geração de Código do Compilador da CPRL

- Implemente todos os métodos `emit()` das classes da AST;
- Verifique no projeto o arquivo "Visão Geral das Classes do Projeto.txt" em que as classes que precisam ser modificadas estão listadas;
- As implementações que devem ser feitas estão explicadas em comentários dentro dos métodos `emit()` que precisam ser implementados;
- Há várias classes com a implementação pronta e que podem ser usadas como base para a implementação das que precisam ser complementadas.

Bibliografia

MOORE JR., J. I. **Introduction to Compiler Design: an Object Oriented Approach Using Java**. 2. ed. [s.l.]:SoftMoore Consulting, 2020. 284 p.

AHO, A. V.; LAM, M. S.; SETHI, R. ULLMAN, J. D. **Compiladores: Princípios, Técnicas e Ferramentas**. 2. ed. São Paulo: Pearson, 2008. 634 p.

COOPER, K. D.; TORCZON, L. **Construindo Compiladores**. 2. ed. Rio de Janeiro: Campus Elsevier, 2014. 656 p.

JOSÉ NETO, J. **Introdução à Compilação**. São Paulo: Elsevier, 2016. 307 p.

SANTOS, P. R.; LANGOLOIS, T. **Compiladores: da teoria à prática**. Rio de Janeiro: LTC, 2018. 341 p.