

# **5 Nomes, Amarração, Verificação de Tipos, e Escopo**

---

**Concepts of Programming Languages, 5/e**

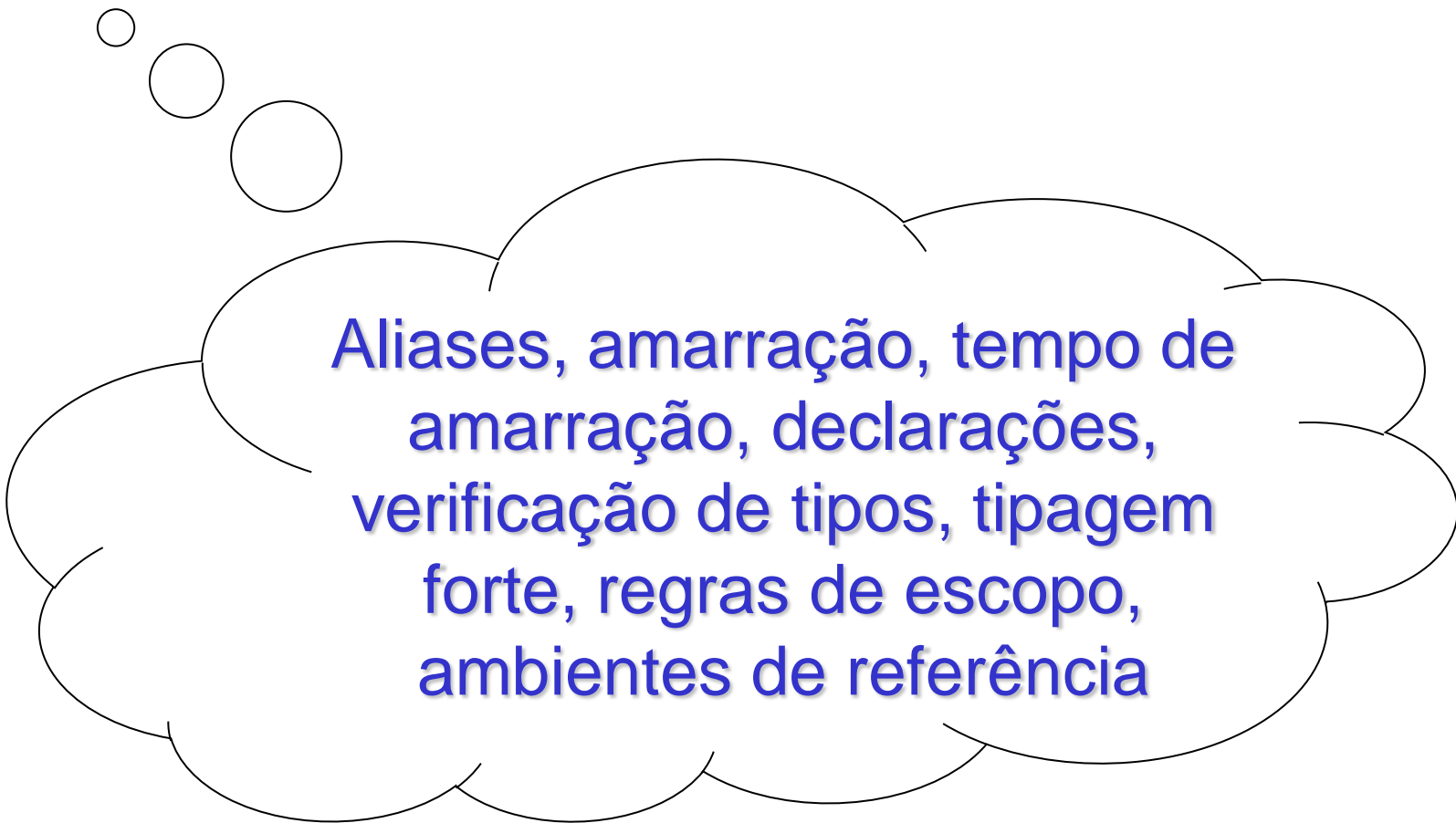
**Robert W. Sebesta**

# Agenda

- Objetivo desta aula: Introduzir os principais aspectos semânticos de variáveis
- Palavras-chave
  - nomes
  - amarração
  - escopo
  - tipos, verificação de tipos

# Variáveis

(nome, endereço, valor, tipo, tempo de vida, escopo)



Aliases, amarração, tempo de amarração, declarações, verificação de tipos, tipagem forte, regras de escopo, ambientes de referência

# Tipo

- Tipo de uma variável: especificação de
  - da classe de valores que podem ser associados à variável,
  - das operações que podem ser usadas para criar, acessar e modificar esses valores[Ghezzi, Jazayeri]
- Exemplo
  - Tipo boolean em Pascal
    - Valores: true e false
    - Operações: and, or e not

# Verificação de Tipos

- Atividade de assegurar-se que os operandos de um operador possuem **tipos compatíveis**
  - generalização
    - subprograma (operador) e parâmetros (operandos)
    - comando de atribuição ( $:=$  operador; lhs e rhs operandos)
  
- Um tipo compatível é
  - Um tipo válido para ser usado por um dado operador, ou
  - Um tipo que pode ser convertido implicitamente em um tipo válido
    - *coerção*: conversão automática
  
- Um *erro de tipo* é a aplicação de um operador a um operando de tipo inadequado.

## Exemplos

```
type range = 1..10;  
int a; real b,c; range d;  
proc x (int p1, real p2) { ... }
```

```
if (a > 0) then ...  
c := a * 99 + b;  
x(d,c);
```

# Verificação de Tipos

- verificação de tipos estática
  - Se a amarração de tipos for estática, a verificação de tipos poderá ser estática (em sua maior parte)
- verificação de tipos dinâmica
  - Se a amarração de tipos for dinâmica, a verificação de tipos deve ser dinâmica
    - APL, JavaScript
- É melhor detectar erros em tempo de compilação, ou seja, estaticamente
  - Custo, confiabilidade

# Verificação de Tipos

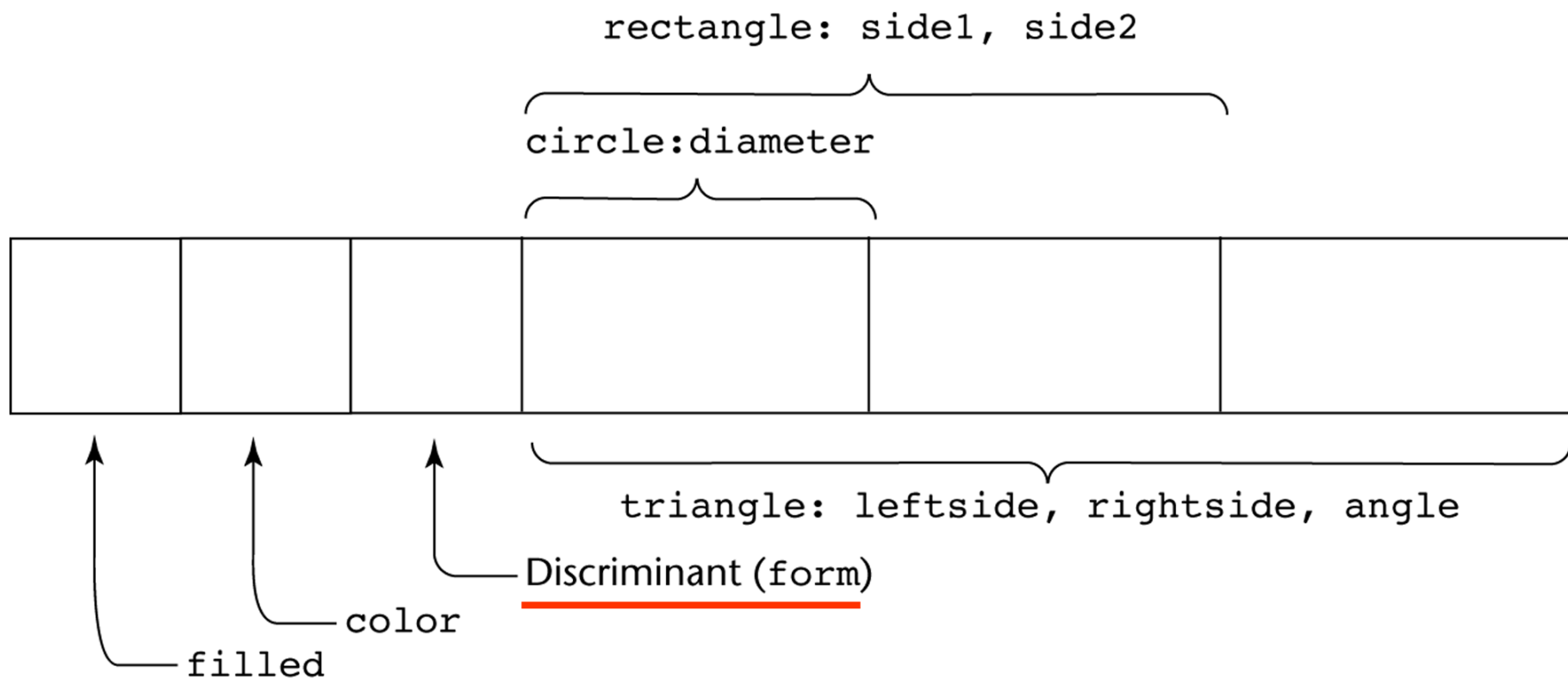
- Áreas Problemáticas

```
type shape = (circle, triangle, rectangle);
  colors = (red, green, blue);
  figure = record
    filled: boolean;
    color: colors;
  case form: shape of
    circle: (diameter: real);
    triangle: (leftside: integer;
              rightside: integer;
              angle: real);
    rectangle: (side1: integer;
               side2: integer)
  end;
```



# Figura:

União discriminada (ou união disjunta ou registro variante)



## Tipificação Forte

- Uma linguagem de programação é dita "fortemente tipificada" se seus **erros de tipos** sempre forem detectados
- Os tipos de todos os operandos podem ser determinados, seja em tempo de compilação ou em tempo de execução

## Tipificação Forte

- FORTRAN 77 não é fortemente tipificada
  - parâmetros formais e reais
  - EQUIVALENCE
- Pascal quase fortemente tipificada
  - registros variantes não são verificados
- C e C++ não são:
  - verificação de tipos de parâmetros pode ser evitada
  - "unions" não são verificados

## Tipificação Forte

- Ada é praticamente fortemente tipificada
  - registros variantes são verificados
  - função `UNCHECKED_CONVERSION` permite suspensão temporária da verificação de tipos
- ML é fortemente tipificada
  - identificadores são amarrados a tipos estaticamente (declaração), ou
  - tipos são reconhecidos via inferência
- Java é fortemente tipificada como Ada

# Tipificação Forte

- Benefícios
  - Permite a detecção de usos de variáveis que resultam em erros de tipo
  - Permite a detecção, em tempo de execução, de uso de valores de tipos incorretos em variáveis que podem armazenar valores de tipos diferentes
- Regras de coerção afetam/enfraquecem tipificação forte
  - C++ *versus* Ada

# Compatibilidade de Tipos

- A idéia de compatibilidade de tipos está relacionada a questões de verificação de tipos
- Métodos para definir compatibilidade de tipos
  - nominal
  - estrutural
- A maior parte das linguagens combina esses métodos

# Compatibilidade de Tipos Nominal

- *Compatibilidade de tipos nominal*
  - Duas variáveis são compatíveis se:
    - Estão na mesma declaração, ou
    - Estão em declarações distintas e usam o mesmo nome de tipo

- Exemplo

type

```
Int1 = Integer;
```

```
Int2 = Integer;
```

var

```
v1: Int1;
```

```
v2, v3: Int1;
```

```
v4: Int2;
```

## Compatibilidade de Tipos Nominal

- *Compatibilidade de tipos nominal* é de fácil implementação porém muito restritiva:
  - Subranges de tipo integer não são compatíveis com o tipo integer. Exemplo:

```
type indextype= 1..100;  
var  
    count : integer;  
    index : indextype;
```

- Parâmetros formais devem ter o mesmo tipo de seus parâmetros reais (Pascal)



## Compatibilidade de Tipos Estrutural

- *Compatibilidade de tipos estrutural*
  - Duas variáveis possuem tipos compatíveis se:
    - Seus tipos possuem estruturas idênticas
- *Compatibilidade de tipos estrutural é mais flexível, porém mais difícil de implementar*

# Compatibilidade de Tipos Estrutural

- Problemas
  - Não se consegue diferenciar tipos com a mesma estrutura:

type

celsius = real;

fahrenheit = real;

# Compatibilidade de Tipos Estrutural

- Perguntas:
  - Registros: são compatíveis se forem estruturalmente compatíveis porém com nomes de campos distintos?
  - Arrays: são compatíveis se possuírem mesmo tipo base e tamanho, porém com esquema de indexação diferente? (ex.: [0..10] e [1..11])
  - Tipos enumerados: são compatíveis se seus componentes forem escritos de modo diferente?

# Compatibilidade de Tipos

- Exemplos
  - FORTRAN, COBOL
    - Compatibilidade de tipos estrutural
  - Pascal (ISO Standard Pascal, 1982)
    - Compatibilidade de tipos estrutural, exceto para parâmetros formais
  - C
    - Compatibilidade de tipos estrutural, exceto para registros e uniões (*equivalência de declaração*)
  - C++
    - Compatibilidade de tipos nominal
    - Compatibilidade de objetos (herança)

# Compatibilidade de Tipos

- Exemplos

- Ada: forma restrita de compatibilidade de nomes (uso de tipos derivados e sub-tipos)

- *Tipos derivados* permitem que tipos com a mesma estrutura sejam tratados como tipos diferentes (incompatível com tipo do qual deriva):

- ```
type celsius is new FLOAT;
```

- ```
type fahrenheit is new FLOAT;
```

- *Subtipos* são versões restritas de um tipo existente (compatível com o supertipo):

- ```
subtype intervalo is INTEGER range 0..99;
```

- Tipos anônimos são únicos e incompatíveis:

- ```
A, B : array (1..10) of INTEGER;
```

## Compatibilidade de Tipos

- A definição original de algumas linguagens não especifica claramente qual o critério a ser adotado
  - Pascal, 1971

## Questões

1. O que é um “tipo” no contexto de LP? Qual a sua importância?
2. O que é “erro de tipo”? E “verificação de tipos”?
3. O que é coerção?
4. O que é “tipificação forte” (strong typing)?
5. O que é compatibilidade de tipos?
6. Qual a diferença entre compatibilidade de tipos nominal e estrutural?
7. O que é equivalência de declaração?