Prof. Alberto Costa Neto alberto@ufs.br

Linguagens de Programação



Departamento de Computação Universidade Federal de Sergipe

- Tipos permitem que dados sejam descritos de forma efetiva
 - Previnem operações sem sentido (ex: 5 * true)
 - Programas mais confiáveis e eficientes
- Na prática, toda linguagem é tipada
- Checagem de Tipo classifica linguagens
 - Estaticamente Tipada (tempo de compilação)
 - Dinamicamente Tipada (tempo de execução)

- Linguagens Estaticamente Tipadas
 - Variáveis e Parâmetros possuem tipo fixo
 - Os tipos das expressões podem ser deduzidos, e portanto checados, na compilação
 - (+) Programas mais eficientes
 - (+) Erros são descobertos mais cedo
 - (-) Flexibilidade reduzida para o programador

```
Pascal
```

```
function ehPar(n: Integer): Boolean;
begin
ehPar := (n mod 2 = 0)
end

Maioria das
Linguagens de Alto Nível
```

Verificação Estática de Tipos

```
c int par (int n) {
    return (n % 2 == 0);
}
main() {
    int i = 3;
    par (i);
}
```

Ações do Compilador

- Operandos de % devem ser inteiros, logo n deve ser inteiro
- Os operandos de == devem ser de um mesmo tipo. Como
 0 e o resultado de % são inteiros, a op. é válida
- O tipo de retorno de par deve ser inteiro
- O parâmetro real de par deve ser inteiro



- Linguagens Dinamicamente Tipadas
 - Somente os valores possuem tipo fixo
 - Variáveis podem ter tipos diferentes em tempos diferentes e portanto não se sabe o tipo de uma expressão no momento da compilação

Lisp

```
(defun mult-tres (n)
  (* 3 n))
(mult-tres 7)
(mult-tres "abacaxi")
```

Normalmente, Linguagens Interpretadas

Linguagens Dinamicamente Tipadas

- (+) Maior Flexibilidade para o programador
- (-) Programas menos eficientes
- (-) Erros são descobertos tardiamente

```
Lisp (defun segundo (l) (car(cdr l)))
```

```
(segundo (1 2 3) )
(segundo ( (1 2 3) (4 5 6)))
(segundo ("manga" "abacaxi" 5 6))
```

car: retorna o primeiro elemento da lista

cdr: retorna a lista sem o primeiro elemento



- Linguagens Fortemente Tipadas
 - -Uma LP é **fortemente tipada** se:
 - Erros de tipo sempre são detectados
 - Os tipos de todos os operandos têm que ser determinados em tempo de compilação ou execução
 - Detecta, em tempo de execução, quando valores incorretos são atribuídos a variáveis que podem armazenar mais de um tipo (uniões disjuntas)

[Compatibilidade]

- Checagem de tipos requer definição de equivalência de tipos
- Formas de equivalência
 - Equivalência Estrutural
 - Equivalência por Nome

- Equivalência Estrutural [Compatibilidade]
 - T¹ ≡ T² se, somente se T¹ e T² possuem o mesmo conjunto de valores
 - Pode ser verificada checando as estruturas dos tipos
 - Exemplo
 - Se T = A x B e T' = A' x B', então T ≡ T' se, somente se,
 A ≡ A' e B ≡ B'. (produtos cartesianos)
 - Ex: Integer x Boolean ≡ Integer x Boolean
 - Se T = A → B e T' = A' → B', então T = T' se, somente se, A = A' e B = B'. (mapeamentos array)
 - Ex: Integer → Boolean = Integer → Boolean

[Compatibilidade]

- Equivalência por Nome
 - T¹ = T² se, somente se T¹ e T² foram definidas no mesmo local
 - (+) Fácil de implementar
 - () Altamente restritiva

```
type T1 = file of Integer;
T2 = file of Integer;
var f1: T1;
f2: T2;
procedure p (var f: T1); ...
```

```
p (f1) --YES: f e f1 são EN
p (f2) --NO: f e f2 ñ são EN
```

[Compatibilidade]

Equivalência por Nome X Estrutural

```
typedef float quilometros;
typedef float milhas;
quilometros converte (milhas m) {
    return 1.6093 * m;
}
main() {
    milhas s = 200;
    quilometros q = converte(s); // ambas
    s = converte(q); // estrutural apenas
}
```

[Compatibilidade]

Equivalência por Declaração

```
type tipo1 = array[1..10] of integer;
tipo2 = array[1..10] of integer;
tipo3 = tipo2;
```

- não há equivalência por estrutura tipo1≠ tipo2
- tipo3 \equiv tipo2
 - mostra que equivalência de nomes nem é utilizada
 - Equivalência de Declaração

[Compatibilidade]

- Algumas Linguagens
 - Haskell: equivalência por estrutura permitindo declarações de tipos novos (newtype e data)
 - C: equivalência estrutural exceto para estruturas definidas no mesmo arquivo (usa equivalência de declaração).
 - C++: equivalência por nome. Note que typedef não define um tipo novo (só sinônimo)
 - Java: equivalência por nome

[Princípio da Completude]

Algumas linguagens restringem o uso de alguns valores em alguns casos

- Ex: Funções em Pascal
 - não podem ser devolvidas como resultados de funções
 - não podem ser atribuídas
 - não podem ser utilizadas como componentes de um registro
 - Valor de "Segunda Classe"
- Valores de tipos primitivos podem ser usados sem estas restrições
 - Valores de "Primeira Classe"

Sugestões de Leitura

- Concepts of Programming Languages (Robert Sebesta)
 - Seções 6.10, 6.11 e 6.12
- Programming Language Concepts and Paradigms (David Watt)
 - Seção 2.5
- Linguagens de Programação (Flávio Varejão)
 - Seção 7.1 (menos 7.1.4)