Tipos de dados

Prof. Hugo de Paula

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Departamento de Ciência da Computação

Sumário

1. Valores e tipos
   - Valores e tipos
   - Tipos primitivos
     - Tipos enumerados
     - Tipos subfaixa
   - Tipos compostos
     - Produto cartesiano
     - União disjunta
     - Mapeamento
     - Conjunto potência
   - Tipos recursivos
2. Sistemas de tipos
   - Verificador de tipos
   - Tipagem estática de dinâmica
   - Equivalência de tipos

Valores e tipos

- **Valor**: tudo aquilo que pode ser avaliado, armazenado, e passado como parâmetro
  - Valores são agrupados em tipos
- **Tipo**: conjunto de valores e de operações que podem ser realizadas sobre os mesmos
  - *Matematicamente*: baseados na teoria dos conjuntos
  - *Computacionalmente*: definem também quantidade de memória e forma de codificação
    - Na matemática: $\mathbb{Z} = [-\infty, +\infty]$
    - Em computação: $int = [MIN\_INT, MAX\_INT]$
- Linguagens de Programação devem incluir um conjunto de entidades que representam tipos de dados simples e prover mecanismos para a construção de novos tipos (abstração de dados)
- Tipos podem ser: primitivos, compostos e recursivos

Tipos primitivos

- **Tipo primitivos**:
  - Valores são atômicos (não decomponíveis)
  - Exemplos: inteiros, reais, caractere, lógicos, enumerados
- Tipos pré-definidos: tipos fornecidos pela LP, a partir dos quais todos os outros tipos são construídos
  - Normalmente identificados por palavras reservadas da linguagem

Exemplo: Tipos primitivos do Pascal

Tipo	*sizeof()*	Descrição
char	8-bit	codificação ASCII
integer	16/32-bit	número inteiro (tamanho pode variar com a implementação)
real	32-bit	número real
boolean	1-bit	tipo lógico (true, false)

Exemplos de tipos primitivos pré-determinados

Exemplo: Tipos primitivos do Java

Tipo	*sizeof()*	Valor mínimo	Valor Máximo
char	16-bit	Unicode 0	Unicode $2^{16} - 1$
byte	8-bit	-128	$+127$
short	16-bit	-2^{15} $-32,768$	$+2^{15} - 1$ $-32,767$
int	32-bit	-2^{31} $-2,147,483,648$	$+2^{31} - 1$ $2,147,483,647$
long	64-bit	-2^{63} $-9,223,372,036,854,775,808$	$+2^{63} - 1$ $9,223,372,036,854,775,807$
float	32-bit	32-bit IEEE 754 floating-point numbers	
double	64-bit	64-bit IEEE 754 floating-point numbers	
bool	1-bit	true or false	
void	—	—	—

Tipos primitivos: tipo complexo

Número complexo

Matematicamente, um número é escrito na sua forma $a + bj$. Os elementos *a* e *b* são números reais em que o valor de *a* representa a parte real e o valor de *b* representa a parte imaginária de um número complexo.

- Algumas linguagens suportam o número complexo como um tipo primitivo, apesar de possuir partes "decomponíveis". Ex: Fortran, Python e C99.
- Cada valor é representado por dois elementos **float**.
- Forma literal no Python: `(10 + 2j)`.

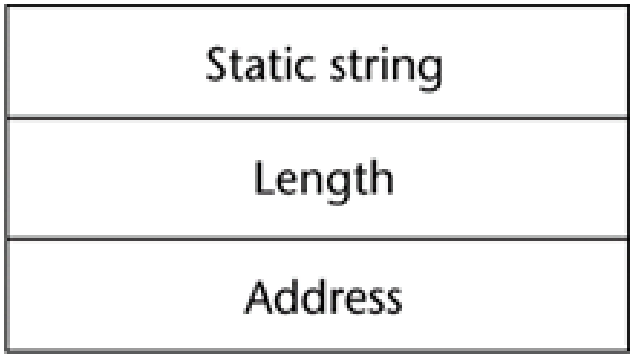
Tipos primitivos: **boolean** (lógico)

- Apenas dois elementos: **true** e **false.**
- Pode ser implementado como bit, mas normalmente é implementado como byte.
- Possui operações lógicas associadas: conjunção (AND, $\&\&$, $*$), disjunção (OR, $||$, +) e negação (NOT, $!$, $\tilde{}$), ou exclusivo (XOR, $\hat{}$).
- Mas operações relacionais sobre tipos numéricos também produzem valores lógicos: $>, <, \leq, \geq, \equiv, \neq$.
- Linguagem C usa inteiros para representar valores lógicos: 0 (**false**), $\neq 0$ (**true**).

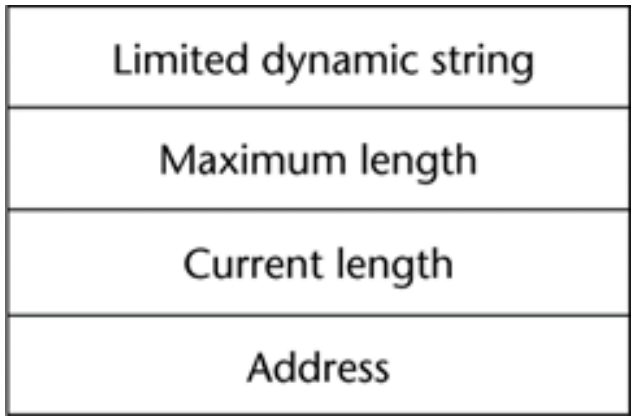
Tipos primitivos: strings

- Questões polêmicas de projeto:
  - Deve ser um tipo primitivo ou uma cadeia de caracteres?
  - Devem possui comprimento estático ou dinâmico?
- Operações típicas: atribuição, cópia, comparação, concatenação, referência para substring e casamento de padrões.
- Strings estáticas: Java (classe String)
- String dinâmicas de comprimento limitado: C e C++. Utilizam um caractere especial que indicam o final da string (ex.: *nulll terminated string*).
- Strings dinâmicas: Perl e JavaScript. Não possuem máximo.

Tipos primitivos: strings



Descritor de strings estáticas em tempo de compilação.



Descritor de strings dinâmicas de comprimento limitado em tempo de execução.

Construtores de tipos primitivos: tipos enumerados

- **Tipos Enumerados**: conjuntos cujos elementos são listados e enumerados explicitamente
- Na maioria das linguagens o conjunto é ordenado (ou seja, a ordem de enumeração é importante)
- Operadores mais comuns são sucessor e antecessor

Exemplo em C++ / C

```
enum dia_semana {dom, seg, ter, qua, qui, sex, sab};
dia_semana hoje, ontem;
ontem = seg;
hoje = ontem + 1;

enum cores { vermelho = 1, verde = 256, azul = 65536};
```

Construtores de tipos primitivos: tipos enumerados

Exemplo em Pascal

```
type
      Dia = ( seg, ter, qua, qui, sex, sab, dom )

var
      hoje, amanha : Dia;

begin
      hoje := seg;
      if ( hoje = ter ) ...
      amanha := succ(hoje);
end.
```

Construtores de tipos primitivos: tipos subfaixa

- **Tipos subfaixa**: subconjuntos de tipos primitivos cujos elementos são especificados definido o limite inferior e o limite superior dos elementos (intervalo)

Exemplo em Pascal

```
type
     Dias_do_Mes    = 1..30;
     unsigned_byte  = 0..255;
     signed_byte    = -128..127;
     string         = packed array [1..255] of char;
var
     x : 1..10;
     y : 'a'..'z';
```

- C e Java não suportam tipos subfaixa, mas o seu comportamento pode ser simulado com *Iterators*

Tipo ponteiro

- Um tipo ponteiro possui uma faixa de valores que corresponde a endereços de memória, e um valor especial *nil* (normalmente representado pelo zero).
- Provê endereçamento indireto.
- Usado para gerenciamento dinâmico de memória.
- Região de memória em que o armazenamento é criado dinamicamente é chamado de *heap*.
- Tipo do ponteiro indica forma de codificação da região apontada.
- Em C: ponteiro **void** pode apontar para região de qualquer tipo.

Tipo ponteiro

- Operações fundamentais: atribuição e dereferenciação.
  - Atribuição: direciona o ponteiro para um endereço de memória.
  - Dereferenciação: retorna o valor armazenado no local apontado pelo valor do ponteiro.

Ex. em C:

Declaração	**Semântica de valor**	**Semântica de referência**
int i	i	&i
int *ptr	*ptr	ptr
int fun()	fun()	fun

Tipos compostos

- Valores são compostos a partir de valores mais simples
- Uma vez que tipos são conjuntos, operações sobre conjuntos podem ser utilizadas para construir novos tipos a partir dos tipos existentes (abstrações de tipos de dados)
- Essas operações são chamadas de construtores de tipos
- Construtores de tipos compostos:
  - Produto Cartesiano, União Disjunta, Mapeamento, Conjunto Potência

Produto cartesiano

- *Matematicamente*: $A \times B = \{(x, y) | x \in A, y \in B\}$
  - O produto cartesiano de n conjuntos $S_1, S_2, \ldots, S_n$, denotado por $S_1 \times S_2 \times \ldots \times S_n$, é um conjunto cujos elementos são *n-tuplas* ordenadas, onde $s_i \in S_i$
- *Computacionalmente*: São vistos por linguagens de programação como campos com nomes simbólicos: tuplas, registros, estruturas.
  - Diferença entre produto cartesiano e um registro:
    - No registro, os campos possuem nomes
    - No produto cartesiano, os campos são identificados pela sua posição

Produto cartesiano

Exemplo em C++

```
typedef struct {
    int dia;
    int mes;
    int ano;
} Data;

Data natal;

natal.dia = 25;
natal.mes = 12;
natal.ano = 00;
```

- Na Programação Orientada para Objetos, as classes são formas de implementação de produtos cartesianos

Produto cartesiano

Exemplo em Pascal

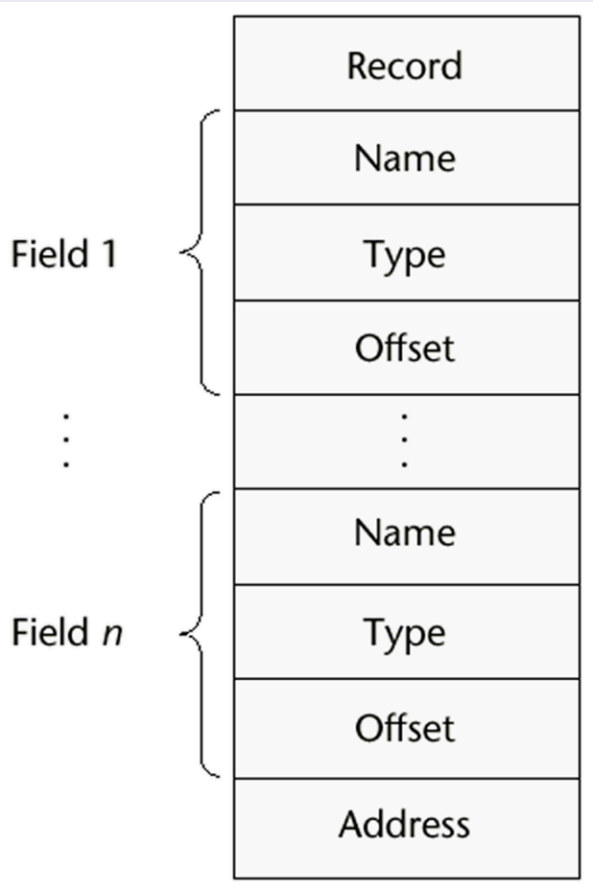
```
type
      Data = record
            dia : integer;
            mes : integer;
            ano : integer;
      end;

var
      natal : Data;

begin
      natal.dia := 25;
      natal.mes := 12;
      natal.ano := 00;
end.
```

Produto cartesiano

- Memória:
  - $\text{sizeof}(A \times B) = \text{sizeof}(A) + \text{sizeof}(B)$
- Cardinalidade:
  - $\sharp(A \times B) = \sharp A \times \sharp B$
- Exemplo do registro **Data**, considerando que o tipo *int* possui 4 bytes
  - $Data = int \times int \times int$
  - $sizeof(Data) = sizeof(int) + sizeof(int) + sizeof(int) =$ 12 *bytes*
  - $\sharp Data = 2^{32} \times 2^{32} \times 2^{32} = 2^{96}$

Tipo de dados Registro

Estrutura típica de um tipo de dados Registro



União

- Estrutura cujos campos compartilham uma área de memória.
- *Matematicamente*: $A + B = \{x | x \in A \text{ ou } x \in B\}$
- *Computacionalmente*: Implementa-se união disjunta. Mesmo que dois conjuntos possuam valores análogos, os valores do tipo A serão sempre considerados distintos dos valores do tipo B.
  - Exemplo: Seja $T = \{a, b, c\}$, então $T + T = \{esq\ a, esq\ b, esq\ c, dir\ a, dir\ b, dir\ c\} \neq T$
  - Na matemática $T \cup T = T$,

União livre

- C/C++, cria uma união não discriminada.
- União livre é **unsafe**, pois não permite checagem de tipos.

Exemplo de união livre em C++

```
enum tipo_dados {inteiro, real, logico};
struct Constante {
    char *nome;  tipo_dados  tipo;
    union {                 // valor_int, valor_real e
        int val_int;        // valor_log compartilham a
        float val_real;     // mesma área de memória
        bool val_log;       // tam. da área = sizeof(float)
    } valor;
};
Constante c;
strcpy(c.nome,"pi"); c.tipo = real;
c.valor.val_real = 3.1415926;
```

União discriminada

- Em Pascal: registro variante

Exemplo de união discriminada em Pascal

```
type
    Cores= (vermelho, verde, azul);
    Formas= (circulo, triangulo, retangulo);
    Figura = record
        preenchida: boolean;
        cor: Cores;
        case tipo: Formas of    { tag ou discriminante }
            circulo:    (diametro: real);
            triangulo:  (lado: real; altura: real);
            retangulo:  (lado1: real; lado2: real);
        end
    end;
```

União disjunta

- Memória:
  - $\text{sizeof}(A + B) = max(\text{sizeof}(A), \text{sizeof}(B))$
- Cardinalidade:
  - $\sharp(A + B) = \sharp A + \sharp B$
- Uniões são úteis para reduzir a alocação de memória quando diferentes dados não são necessários simultaneamente
- Uniões não são necessárias em linguagens orientadas para objeto: pode-se utilizar herança
  - Em C#, no entanto, é possível se simular união com *StructLayout*

União disjunta

Exemplo em C# de StructLayout

```
using System.Runtime.InteropServices;

[StructLayout(LayoutKind.Explicit)]

struct UniaoSimulada {
    [FieldOffset(0)]
    public byte val_byte;
    [FieldOffset(0)]
    public int val_int;
    [FieldOffset(0)]
    public float val_float;
}
```

Mapeamento

- *Matematicamente*: $m : A \rightarrow B = \{y = f(x) | x \in A, y \in B\}$
- Um mapeamento é uma função de um conjunto finito de valores A em valores de um tipo B.
- Mapeamentos em LP: implementados de duas formas
  - Por meio de arranjos.
  - Por meio de abstrações de função.

Mapeamento: arranjos

Exemplo em C

```
bool impar[10] = {0, 1, 0, 1, 0, 1, 0, 1, 0, 1};
```

- Um arranjo representa um mapeamento finito, onde o conjunto do índice é enumerável.
- Muitas linguagens possuem arranjos multidimensionais.
- O conjunto do índice deve ser discreto.

Mapeamento: arranjos

Exemplo em Pascal

```
type
    matriz = array [1..10, 0..20] of integer;
var
    impar : array [1..100] of Boolean;
```

- Matematicamente:
  - $\text{matriz} : [1;10] \times [0;20] \rightarrow \mathbb{Z}$
  - $\text{impar} : [1;100] \rightarrow \{V;F\}$

Mapeamento: arranjos

- Tipos dos índices dos arranjos:
  - Fortran e C: apenas int.
  - Java: tipos inteiros (byte, short, int long, unsigned int, etc..)
- Verificação de índice fora da faixa:
  - Fortran, C, Perl: Não verificam faixa dos índices.
  - Java, C#: especificam checagem de faixa de índice.

Variáveis Arranjos

- Quando o conjunto de índices de um arranjo é determinado?
- Arranjos Estáticos:
  - Conjunto de índices determinado em tempo de compilação
  - Exemplos: Pascal e C++

Exemplo em Pascal

```
var
vetor : array [1..10] of integer;
```

Variáveis Arranjos

- Arranjos Dinâmicos:
  - Conjunto de índices determinado em tempo de execução, no momento da criação do arranjo
  - Uma vez determinado, conjunto de índices não pode ser alterado
  - Exemplo: Java

Exemplo em Java

```
int [] x;
int y = Console.readInt();
x = new int [y];
```

Variáveis Arranjos

- Arranjos Flexíveis:
  - Conjunto de índices pode variar durante a execução
  - Exemplo: Perl

Exemplo em Perl

```
@fib = ();
@fib = (0, 1, 1, 2, 3, 5);
@fib = (0, 1, 1, 2, 3, 5, 8, 13, 21, 34);
```

Variáveis Arranjos

- Arranjos Flexíveis em Bash:
  - Conjunto de índices não precisa ser consecutivo ou contínuo
  - Partes do vetor não precisam ser inicializadas

Exemplo em Bash

```
area[13]=10
area[32]=TEXTUAL

echo "area[13] = ${area[13]}"
echo "Conteúdo de area[32] é ${area[32]}."

echo -n "area[46] = "
echo ${area[46]}          # não imprime nada

for i in "${area[@]}"; do
echo "${i}"
done
```

Slices (fatiamento)

Slice

Uma fatia (*slice*) é uma subestrutura de um arranjo. Serve como mecanismo de referenciação.

- Úteis quando a linguagem provê operações sobre arranjos.
- Ex: Python

```
vetor = [2, 4, 6, 8, 10, 12, 14, 16]
matriz = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

print(vetor[3:6])  # é um arranjo de 3 elementos.
print(matriz[0][0:2]) # 2 primeiros elementos da 1a linha da matriz.
```

Descritores de arranjos

Array
Element type
Index type
Index lower bound
Index upper bound
Address

Descritor de arranjo 1-D em tempo de compilação.

Multidimensioned array
Element type
Index type
Number of dimensions
Index range 1
⋮
Index range *n*
Address

Descritor de arranjo N-D em tempo de compilação.

Mapeamento: abstrações de funções

Exemplo em Pascal

```
function impar(n: integer): boolean;
begin
    impar:= (n mod 2) = 1
end;
```

- Abstração de função $\neq$ função computacional
- A abstração de função implementa uma função através de um algoritmo. $\rightarrow$ deve haver representação analítica
  - uma função computacional pode acessar e alterar valores de variáveis não-locais
  - uma abstração de função não pode produzir efeito colateral e deve possuir transparência referencial

Mapeamento

- Memória:
  - $\text{sizeof}(A \to B) = \sharp(A) \times \text{sizeof}(B)$
- Cardinalidade:
  - $\sharp(A \to B) = \sharp B^{\sharp A}$
- Cálculo de memória não faz sentido para abstrações de funções
- Em abstrações de funções o tamanho é fixo, definido pelo código compilado da função, mas exige processamento a cada mapeamento realizado

Tipos funcionais e tipos procedimentais

- Tipos de funções genéricas e tipos procedimentais podem ser criados em algumas linguagens
- Exemplo: em C, um tipo funcional (tipo função) de inteiro para inteiro pode ser declarada como
  - `typedef int (*funcInt)(int);`

Exemplo em C

```c
int quadrado(int x) { return x*x; }
funcInt f = quadrado;

int avaliar(funcInt g, int val) {
   return g(val);
}
...
printf("\%d\n", avaliar(f, 3)); // imprime 9
```

Conjunto potência

- *Matematicamente*: $\mathcal{P}(S) = \{X | X \subseteq S\}$
- Em uma LP, como Pascal:

Exemplo em Pascal

```
type
      Cor = (vermelho, azul, verde, amarelo);
      ConjCores = set of Cor;
      Letras = set of char;
var
      conj1  : ConjCores;
      vogais : Letras;  ch: char;
begin
      conj1  := [vermelho, verde];
      vogais := ['a', 'e', 'i', 'o', 'u'];
      if vermelho in conj1 then .....
      if ch in vogais then ....
end.
```

Conjunto potência

- Memória:
  - $\text{sizeof}(\mathcal{P}(S)) = [0..\sharp S \times \text{sizeof}(S)]$
- Cardinalidade:
  - $\sharp\mathcal{P}(S) = \sum_{n=0}^{\sharp S} \binom{\sharp S}{n} = 2^{\sharp S}$

Tipos recursivos

- Um tipo composto T é dito recursivo se possuir componentes que são do próprio tipo T
- Normalmente, utiliza ponteiros (ou apontadores)

Exemplo em C

```
struct no {      // nodo é um tipo recursivo
   char info;
   struct no *esq, *dir;
};

struct elem {
   elem a1;  // Erro, tipo recursivo apenas com ponteiro
   elem *a2; // OK, ponteiros
};
```

Sistemas de Tipos

- **Tipos**: conjunto de valores com operações em comum.
- **Sistema de Tipos**: regras que regulam atribuição de tipos a várias partes de um programa.
- **Verificador de Tipos**: verifica se programas obedecem às regras do sistema de tipos.

Checagem de tipos (*Type Checking*)

- Generaliza o conceito de operandos e operadores para incluir subprogramas e atribuições.
- **Checagem de tipos** (*Type Checking*): é a atividade que garante que os operandos de um operador são de tipos compatíveis.
- Um tipo é compatível se ele é um tipo previsto para o funcionamento da operação, ou se existe alguma regra na linguagem que permite que ele seja implicitamente convertido para um tipo legítimo do operador.
  - A conversão automática de tipos é chamada de coerção.
- **Erro de tipo**: é causado pela aplicação de um operador a operandos de tipos inapropriados.

Sistema de tipos

- Linguagem Estaticamente tipada:
  - Toda variável possui um tipo
  - Verificação de tipos ocorre em tempo de compilação
  - Exemplo:
    - bool impar (int n) { .... }
  - Linguagens: Pascal, C, C++, Java
- Linguagem Dinamicamente tipada:
  - Somente valores possuem um tipo fixo
  - Variáveis podem assumir valores de tipos diferentes
  - Verificação de tipos ocorre em tempo de execução
  - Exemplo:
    - (defun impar (n) (....))
  - Linguagens: Lisp, Smalltalk

Sistema de tipos

Exemplo de tipagem dinâmica em JavaScript

```
<html>   <head>
<script type="text/javascript">
   function startTime() {
      var today = new Date();
      var h = today.getHours();
      var m = today.getMinutes();
      var s = today.getSeconds();

      m = checkTime(m);
      s = checkTime(s);
      document.getElementById('txt').innerHTML = h + ":" + m + ":" + s;
      t = setTimeout('startTime()',500);
   }
   function checkTime(i) {
      if (i < 10)   i = "0" + i;
      return i;
   }
</script>
</head>
<body onload="startTime()">
   <div id="txt"></div>
</body>   </html>
```

Tipagem forte

- Se todas as associações de tipos são estáticas, então as checagens de tipo podem ser estáticas.
- Se as associações de tipo são dinâmicas, a checagem de tipos deve ser dinâmica.
- A linguagem de programação é fortemente tipada (*strong typing*) se os erro de tipos são sempre detectados.
- Vantagem da tipagem forte: permite detecção de mau uso de variáveis, que podem resultar em erros de tipo.
  - C e C++ não são fortemente tipadas: parâmatros não são checados e unions não são checadas.
  - Java e C# são fortemente tipadas (com exceção de *type casting* explícito).
  - Perl, Ruby, Python, Rust e F# são fortemente tipadas.

Tipagem forte vs tipagem fraca

Elementos que enfraquecem a tipagem?

- Regras de coerção.
- *Type Casting*.
- União livre.
- Operações aritméticas sobre ponteiros.

Segurança de tipos (*type-safety*)

É um conceito mais abrangente que tipagem forte. Em uma linguagem estaticamente tipada, garante que o eventual valor de uma expressão será sempre um membro legítimo do tipo estático da expressão.

Equivalência de tipos

- **Equivalência Estrutural**: duas variáveis têm tipos compatíveis se possuem a mesma estrutura.
- **Equivalência por Nomes**: duas variáveis têm tipos compatíveis se possuem o mesmo nome de tipo.

Exemplo em Pascal

```
type
     par1 = record  a, b: integer;  end;
     par2 = record  a, b: integer;  end;
var
     p: par1;  q, r: par2; .....
     p := q;          //  (a)
     q := r;          //  (b)
```

- Equivalência de tipos estrutural: (a) e (b) corretas
- Equivalência de tipos por nomes: apenas (b) correta

Equivalência estrutural

Equivalência Estrutural: $T \equiv T'$ se e somente se T e T' têm o mesmo conjunto de valores. Verificação é feita comparando a estrutura dos tipos. Regras

- T e T' são ambos primitivos, então $T \equiv T'$ sse T e T' forem idênticos. Ex: $Integer \equiv Integer$
- $T = A \times B$ e $T' = A' \times B'$, então $T \equiv T'$ sse $A \equiv A'$ e $B \equiv B'$. Ex: $Char \times Real \equiv Char \times Real$.
- $T = A + B$ e $T' = A' + B'$, então $T \equiv T'$ sse $A \equiv A'$ e $B \equiv B'$ ou $A \equiv B'$ e $B \equiv A'$. Ex: Char + Real $\equiv$ Real + Char.
- $T = A \rightarrow B$ e $T? = A' \rightarrow B'$, então $T \equiv T'$ sse $A \equiv A'$ e $B \equiv B'$. Ex: $Integer \rightarrow Real \equiv Integer \rightarrow Real$
- Caso contrário, T não é equivalente a T'.

Equivalência por nomes

Equivalência por Nome: $T \equiv T'$ se e somente se possuem o mesmo nome de tipo.

Exemplo:

```
1        type
2                T1  =  file  of  Integer;
3                T2  =  file  of  Integer;
4        var
5                f1 : T1;          f2  :  T2;
6
7        procedure  p(var  f  :  T1);
```

p(f1) é válido? p(f2) é válido?

Usam a mesma declaração de tipos?

- Tipos subfaixa de inteiros não são equivalentes a tipos inteiros.
- Parâmetros formais devem possui o mesmo tipo que os parâmetros reais, ou de chamada.