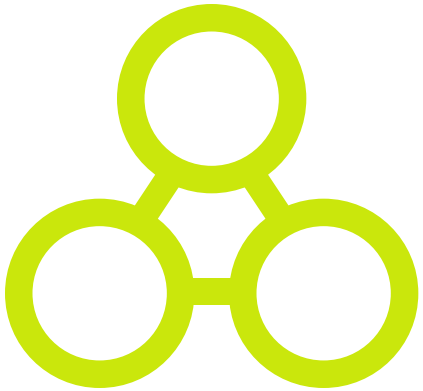


nemo

ontology & conceptual modeling research group

Linguagens de Programação
3 – Valores e Tipos de Dados

Vítor E. Silva Souza

(vitor.souza@ufes.br)

http://www.inf.ufes.br/~vitorsouza

Departamento de Informática

Centro Tecnológico

Universidade Federal do Espírito Santo

Conteúdo do curso

- Introdução;
- Amarrações;
- Valores e tipos de dados;
- Variáveis e constantes;
- Expressões e comandos;
- Modularização;
- Polimorfismo;
- Exceções;
- Concorrência;
- Avaliação de linguagens.

- Estes slides foram baseados em:
  - Slides do prof. Flávio M. Varejão;
  - Livro “Linguagens de Programação – Conceitos e Técnicas” (Varejão);
  - Livro “Linguagens de Programação – Princípios e Paradigmas, 2a edição” (Tucker & Noonan).

Conceitos

- Um **valor** é uma entidade que existe durante uma computação;
- Valor = Dado:

```
3    2.5    'a'    "Paulo"    0x1F    026
```

- São combinados em tipos mais complexos;
- Importância dos dados:
  - Antigamente, computação era “Processamento de Dados”;
  - Muitas vezes uma base de dados é mais cara do que o software que a manipula.

Conceitos

- Um **tipo de dado** é um conjunto cujos valores exibem comportamento uniforme nas operações associadas;
  - Uma operação (a + b) está associada a um tipo se o resultado pertence ao conjunto daquele tipo;
- Linguagens de baixo nível não tem sistema de tipos;
- Possuem cardinalidade (#): número de valores distintos que fazem parte do tipo;
- Em geral possuem número fixo de valores;
  - Em geral, números Reais não são infinitos em LPs!
  - Exceções: fracionários em SmallTalk, `Integer` de Haskell, `BigInteger` em Java;
- Exemplos:
  - `{ true, 25, 'b', "azul" }` não é um tipo;
  - `{ true, false }` corresponde a um tipo.

Tipagem estática vs. dinâmica

- Estática:
  - O tipo de todas as variáveis é fixado em suas definições (em tempo de compilação);
  - Permite detectar mais erros em compilação;
  - Usado por Java, C, Pascal, Fortran, etc.;
- Dinâmica:
  - O tipo de uma variável pode mudar em tempo de execução de acordo com o valor atribuído a ela;
  - Não é o mesmo de não ter um tipo;
  - Usado por Perl, Python, Scheme, etc.

Erros de tipo

- Ocorre quando uma operação é tentada sobre um tipo de valor para o qual não está bem definida;
- Se todos os erros de tipos forem detectados em compilação ou execução, a LP é dita “fortemente tipada”;
  - Java, Ada, Scheme, Perl são, C, C++ não.

```
// Exemplo em C/C++. O que é impresso ao final?
int main() {
  int a = 1000 * 1000 * 1000;
  void* p1 = &a;
  float* p2 = (float*)p1;
  printf("%f\n", *p2);
}
```

0.004724

O custo de um erro

- Axioma da Engenharia de Software:

> Um erro não detectado em uma fase de desenvolvimento custa, em média, uma ordem de magnitude a mais na fase seguinte.

- Exemplo:
  - Se custa R$ 1 durante implementação,
  - Custará R$ 10 na fase de testes
  - E R$ 100 após a fase de testes.
- Um erro no algoritmo de multiplicação de ponto flutuante em modelos iniciais do Intel 386 custou US$ 472 milhões, pois os chips já estavam no mercado.

Importância de um sistema de tipos

Durante a escrita e a depuração, adquiri um grande respeito pela expressividade do sistema de tipos de Simula e pela capacidade do seu compilador de capturar erros de tipos. Observei que erros de tipos quase invariavelmente refletiam um erro tolo de programação ou uma falha conceitual no projeto. [...] Em contraste, tinha descoberto que o sistema de tipos de Pascal era pior do que inútil – uma camisa-de-força que causava mais problemas do que soluções, forçando-me a entortar meus projetos para adaptá-los a um artefato orientado à implementação.

Bjarne Stroustrup, criador do C++

Tipos primitivos

- Não podem ser decompostos em valores mais simples (na visão do programador);
- Costumam ser definidos na implementação da LP;
- Sofrem influência direta do hardware (ex.: int em C, diferente em plataformas diferentes);
- Podem indicar o propósito da linguagem:
  - COMPLEX e reais de precisão variada em Fortran;
  - Strings de comprimento fixo em COBOL.
- Na hora de escolher uma LP para um trabalho qualquer, conheça seus tipos!

Tipo inteiro

- Corresponde a um intervalo do conjunto dos números inteiros;
- Vários tipos inteiros numa mesma LP:
- Normalmente, intervalos são definidos na implementação do compilador:
  - C possui `char` e `int`, com modificadores `signed`, `unsigned`, `short` e `long`. O tipo `int` possui o tamanho da palavra da arquitetura em questão.
- Em Java, o intervalo de cada tipo inteiro é estabelecido na definição da própria LP;
- Ada permite que o programador especifique, mas dá erro caso exceda os limites.

Tipo inteiro

- Tipos inteiros em Java:

Tipo	Tamanho	Alcance
`byte`	1 byte	-128 a 127
`short`	2 bytes	-32.768 a 32.767
`int`	4 bytes	-2.147.483.648 a 2.147.483.647
`long`	8 bytes	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807

- Representação em memória:
  - Tipos *signed* usam complemento de 2 para usarem as operações binários do processador diretamente.
- Qual a cardinalidade de um tipo inteiro?

$2^{(\text{tamanho em bits})}$

Tipo caractere

- Armazenados como códigos numéricos:
  - Tabelas EBCDIC, ASCII e UNICODE;
  - ASCII < Latin1 ISO 8859-1 < UTF-8 < UTF-16 < UTF-32;
- Pascal e Modula-2 oferecem o tipo char;
- Em C, o tipo primitivo char é classificado como um tipo inteiro:

```
char d;
char *p, *q;
d = 'a' + 3;
// ...
while (*p) *q++ = *p++;
```

- Qual a cardinalidade?

Número de caracteres na tabela adotada.

Sobre tabelas de caractere

- Leia “The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!)”, por Joel Spolsky

http://www.joelonsoftware.com/articles/Unicode.html

Tipo booleano

- Tipo mais simples: possui apenas dois valores;
- C/Perl não possuem booleano, mas qualquer expressão numérica pode ser usada como condicional:
  - ≠ zero: verdadeiro; = zero: falso;
- Abordagem de C pode provocar erros:

```
if (c += 1) x = 10; // Erro de digitação, mas compila.
```

- Java inclui o tipo de dado boolean;
- Ocupa geralmente 1 byte (difícil endereçar 1 bit);
- Qual a cardinalidade?

2

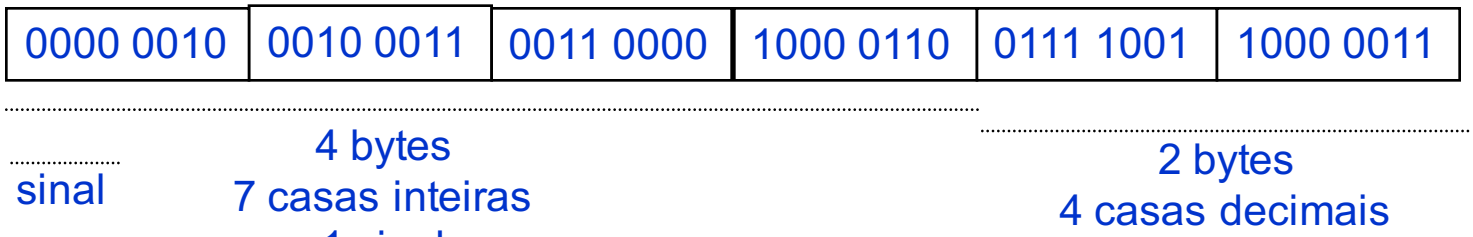
Booleanos em C

- C99 inclui:
  - A biblioteca stdbool.h;
  - O tipo `bool`;
  - Os valores `true` (1) e `false` (0);
- Exemplo:

```
bool b1, b2;
b1 = true;
b2 = false;
if (b1) printf("%d\n", b2); // Imprime 0.
```

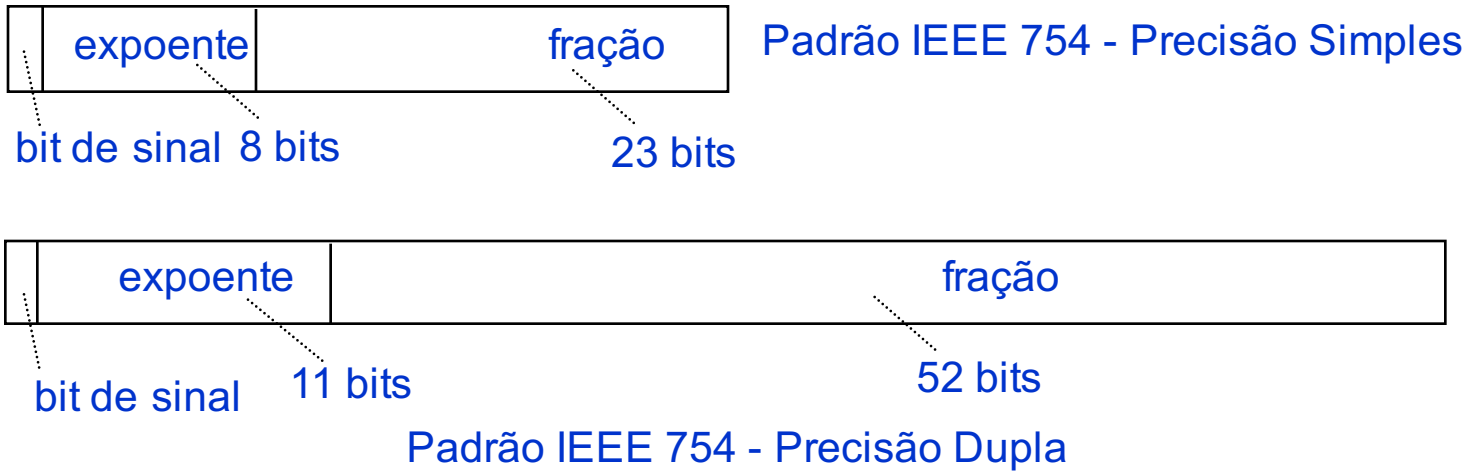
Tipo decimal

- Armazena um número fixo de dígitos decimais:
  - Cada 4 bits representam um valor de 0 a 9 (um dígito do número);
  - Vantagem: precisão;
  - Desvantagem: reduzido intervalo de valores, desperdício de memória, operações complexas;
- COBOL possui este tipo;
- Ex.: que número é este?



Tipo ponto flutuante

- Modela os números reais;
- LPs normalmente incluem dois tipos: `float` e `double`;
- Imprecisão: 3.231 * 100 = 323.100006 (`float` em C).



$$(-1)^s \times 1.m \times 2^{e-127}$$

Tipo ponto flutuante

- Faixa:
  - Float: $\pm 10^{-38}$ a 10^{38}
  - Double: $\pm 10^{-308}$ a 10^{308}
- Precisão:
  - Float: 7 dígitos decimais significativos;
  - Double: 16 " " " ;
- Cardinalidade = ???
  - Inferior a $2^{(\text{tamanho em bits})}$.

Tipo enumerado

- Pascal, Ada, C e C++ permitem que o programador defina novos tipos primitivos através da enumeração de identificadores dos valores do novo tipo:

```
// Exemplo em C:
enum mes_let { mar, abr, mai, jun, ago, set, out, nov };
enum mes_let m1, m2;
```

- Possuem correspondência direta com intervalos de tipos inteiros e podem ser usados para indexar vetores e para contadores de repetições;
- Aumentam a legibilidade e confiabilidade do código;
- Java não suportava tipos enumerados até a versão 5;
- Cardinalidade?

Número de identificadores usados na definição do tipo.

Enums em Java

- Possuem características de classe:

```
public enum StringComparator
                              implements Comparator<String> {
  NATURAL {
     @Override
     public int compare(String s1, String s2) {
        return s1.compareTo(s2);
     }
  },

  INVERSA {
     @Override
     public int compare(String s1, String s2) {
        return -NATURAL.compare(s1, s2);
     }
  };
}
```

Enums em Java

```
public enum Comando {
  AJUDA("?", "Mostra esta lista de comandos."),
  ADICIONAR("adic", "Adiciona um novo contato."),
  LISTAR("list", "Lista os contatos."),
  SAIR("sair", "Sai do programa."),
  DESCONHECIDO("", "");

  private final String nome;
  private final String descricao;

  private Comando(String nome, String descricao) {
     this.nome = nome;
     this.descricao = descricao;
  }

  public String getNome() {
     return nome;
  }                                          /* ... */
```

Tipo intervalo de inteiros

- Em Pascal e Ada, também é possível definir tipos intervalo de inteiros:

```
(* Exemplo em Pascal: *)
type meses = 1 .. 12;
```

- Tipos intervalos herdam as operações dos inteiros;
- Assim como enums, aumentam a legibilidade e confiabilidade do código;
- Cardinalidade?

Quantidade de números no intervalo.

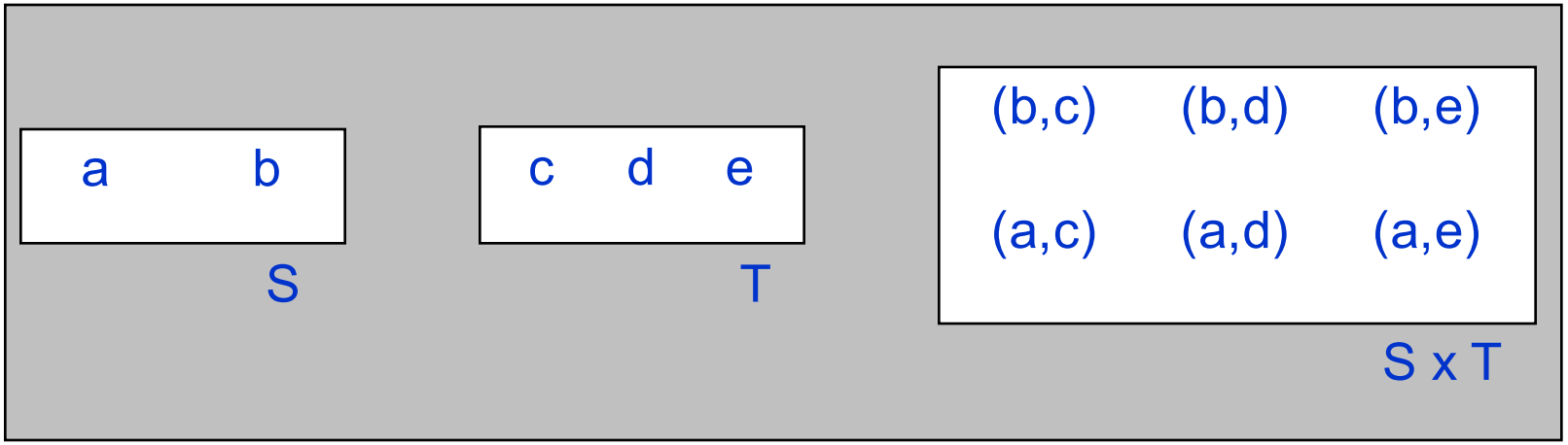
Tipos compostos

- Tipos compostos são aqueles que podem ser criados a partir de tipos mais simples:
  - Ex.: registros, vetores, listas, arquivos, etc.
- Versões iniciais de Fortran, ALGOL e COBOL não permitiam tipos criados pelos programadores;
- Entendidos em termos dos seguintes conceitos:
  - Produto cartesiano;
  - Uniões;
  - Mapeamentos;
  - Conjuntos potência;
  - Tipos recursivos.

Classificação teórica proposta por David Watt em 1990.

Produto cartesiano

- Combinação de valores de tipos diferentes em tuplas:



Produto cartesiano

- São produtos cartesianos os registros de Pascal, Modula-2, Ada e COBOL e as estruturas de C:

```
// Exemplo em C:
struct nome {
  char primeiro [20];
  char meio [10];
  char sobrenome [20];
};

struct empregado {
  struct nome nfunc;
  float salario;
} emp;
```

Produto cartesiano

- Uso de seletores:

```
emp.nfunc.meio
```

- Inicialização em C:

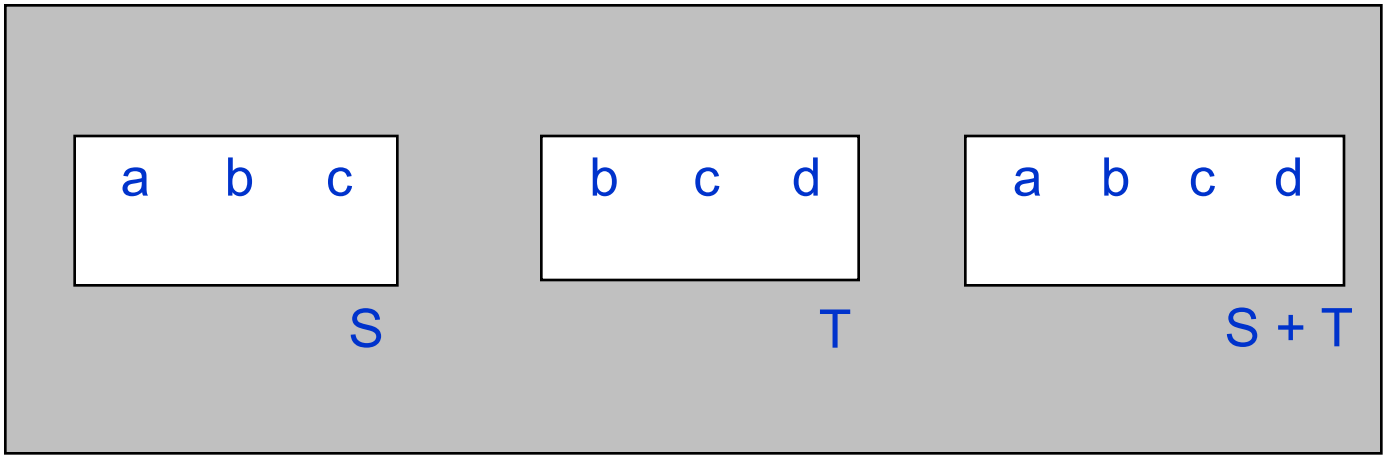
```
struct data { int d, m, a; };
struct data d = { 7, 9, 1999 };
```

- Em LPs orientadas a objetos, produtos cartesianos são definidos a partir do conceito de classe;
- Armazenamento em memória: campos adjacentes, acessando via (endereço, deslocamento);
- Cardinalidade?

$$\#(S1 \times S2 \times \ldots \times Sn) = \#S1 \times \#S2 \times \ldots \times \#Sn.$$

Uniões

- Consiste na união de valores de tipos distintos para formar um novo tipo de dados:



- Podem ser livres ou disjuntas.

Uniões

- Uniões livres:
  - Pode haver interseção entre o conjunto de valores dos tipos que formam a união;
  - EQUIVALENCE de Fortran e union de C;
  - Há possibilidade de violação no sistema de tipos:

```
// Exemplo em C: o que é impresso na tela?
union medida { int centimetros;  float metros; };

int main() {
  union medida medicao;
  float altura;
  medicao.centimetros = 180;
  altura = medicao.metros;
  printf("Altura: %f metros\n", altura);
}
```

Altura: 0.000000 metros

Uniões

- Uniões livres (continuação):
  - Java decidiu não suportar este tipo;
  - Memória: aloca espaço suficiente para o maior tipo.
- Uniões disjuntas:
  - Não há interseção entre o conjunto de valores dos tipos que formam a união;
  - Registros variantes de Pascal, Modula-2 e Ada e a `union` de ALGOL 68.

```
(* Exemplo em Pascal: *)
TYPE   Representacao = (decimal, fracionaria);
       Numero = RECORD CASE Tag: Representacao OF
          decimal: (val: REAL);
          fracionaria: (numerador, denominador: INTEGER);
       END;
```

Uniões

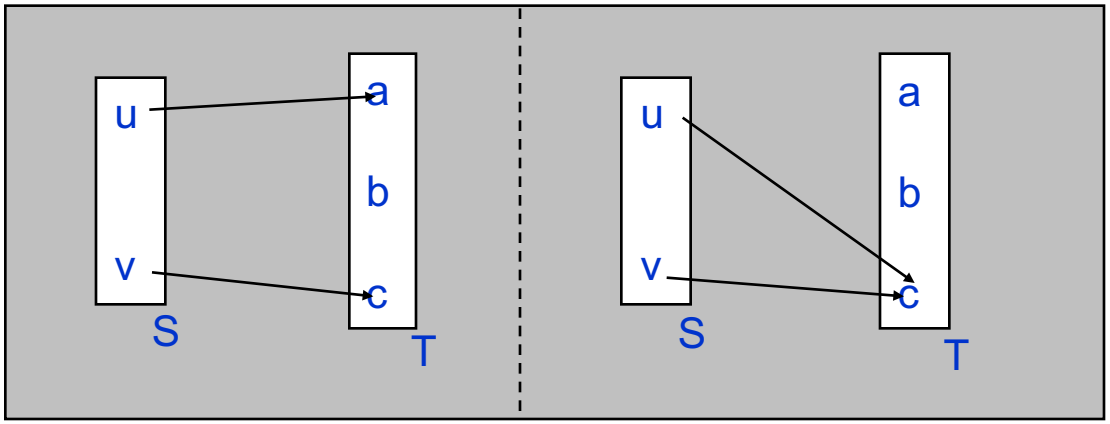
- Algumas linguagens implementam verificação de consistência de tipos união (Ada) enquanto outras não (Pascal);
- Cardinalidade?

$$\#(S1 + S2 + \ldots + Sn) = \#S1 + \#S2 \ldots + \#Sn$$

Pergunta: em LPs Orientadas a Objeto, que recurso poderíamos usar para substituir as uniões disjuntas?

Mapeamentos

- Tipos de dados cujo conjunto de valores corresponde a todos os possíveis mapeamentos de um tipo de dados S em outro T:

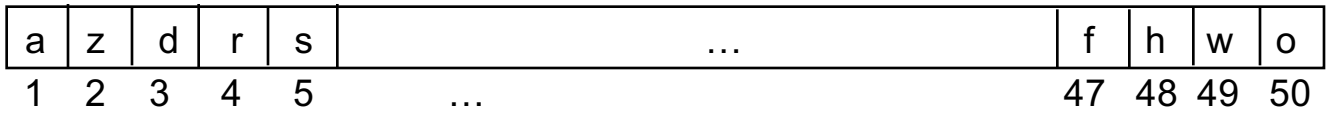


- Funciona como função injetora (todos de S estão incluídos, mas não necessariamente todos de T);
- Cardinalidade: $\#(S \rightarrow T) = (\#T)^{\#S}$.

Mapeamentos finitos

- O conjunto domínio é finito;
- Vetores e matrizes:

```
(* Exemplo em Pascal *)
(* array S of T === (S -> T) *)
A: array [1..50] of char;  (* ([1,50] -> char) *)
```



- O conjunto índice deve ser finito e discreto;
- Verificação de índices: C/C++ vs. Java.

Categorias de vetores

Categoria de vetor	Tamanho	Tempo de definição	Alocação	Local de alocação	Exemplos
Estáticos	Fixo	Compilação	Estática	Base	Fortran 77
Semi-estáticos	Fixo	Compilação	Dinâmica	Pilha	Pascal, C, Modula-2
Semi-dinâmicos	Fixo	Execução	Dinâmica	Pilha	ALGOL 68, Ada, C
Dinâmicos	Variável	Execução	Dinâmica	Monte	APL, Perl

Categorias de vetores

- Exemplos em C:

```
void f() {                  // Estático:
  static int x[10];
}

void f() {                  // Semi-estático:
  int x[10];
}

void f(int n) {             // Semi-dinâmicos (C 99):
  int x[n];
}

void f(int n) {             // Dinâmicos:
  int *x = (int *)malloc(n * sizeof(int));
}
```

Vetores multidimensionais

- Elementos são acessados através da aplicação de fórmulas;
- Posição mat[i][j] =
  - Endereço de mat[0][0] + i x tamanho da linha + j x tamanho do elemento =
  - Endereço de mat[0][0] + (i x número de colunas + j) x tamanho do elemento.

Vetores multidimensionais

- Em Java, vetores multidimensionais são vetores unidimensionais cujos elementos são outros vetores:

```
// Note: cada linha tem um número diferente de colunas.
int[][] a = new int[5][];
for (int i = 0; i < a.length; i++) {
     a[i] = new int[i + 1];
}
```

- O mesmo efeito pode ser obtido em C com o uso de ponteiros para ponteiros;
- APL possui operações de soma, subtração, multiplicação, transposição e inversão de matrizes.

Vetores multidimensionais

- Cardinalidade?
- Exemplo: `int[5][4] mat`:
  - $\{0, \ldots, 4\} \times \{0, \ldots, 3\} \rightarrow int$
  - $(\#int)^{\#(\{0, \ldots, 4\} \times \{0, .., 3\})} =$
  - $(\#int)^{(\# \{0, \ldots, 4\} \times \#\{0, .., 3\})} =$
  - $(\#int)^{(5 \times 4)} =$
  - $(\#int)^{20}$

Mapeamentos através de funções

- Uma função implementa um mapeamento S → T através de um algoritmo;
- O conjunto S não necessita ser finito;
- O conjunto de valores do tipo mapeamento S → T são todas as funções que mapeiam o conjunto S no conjunto T;
- Valores do mapeamento [`int` → `boolean`] em Java:

```
boolean positivo (int n) {
  return n > 0;
}
```

- Outros exemplos: palíndromo, ímpar, par, primo, etc.

Mapeamentos através de funções

- Existe apenas em LPs nas quais funções são cidadãos de 1ª classe:

```
// Exemplo em Java 8:

import java.util.function.*;

public class Teste {
  public static Boolean positivo(Integer n) {
     return (n != null) && (n > 0);
  }

  public static void main(String ... args) {
     Function<Integer, Boolean> fun = Teste::positivo;
     System.out.println(fun.apply(-4));
  }
}
```

Mapeamentos através de funções

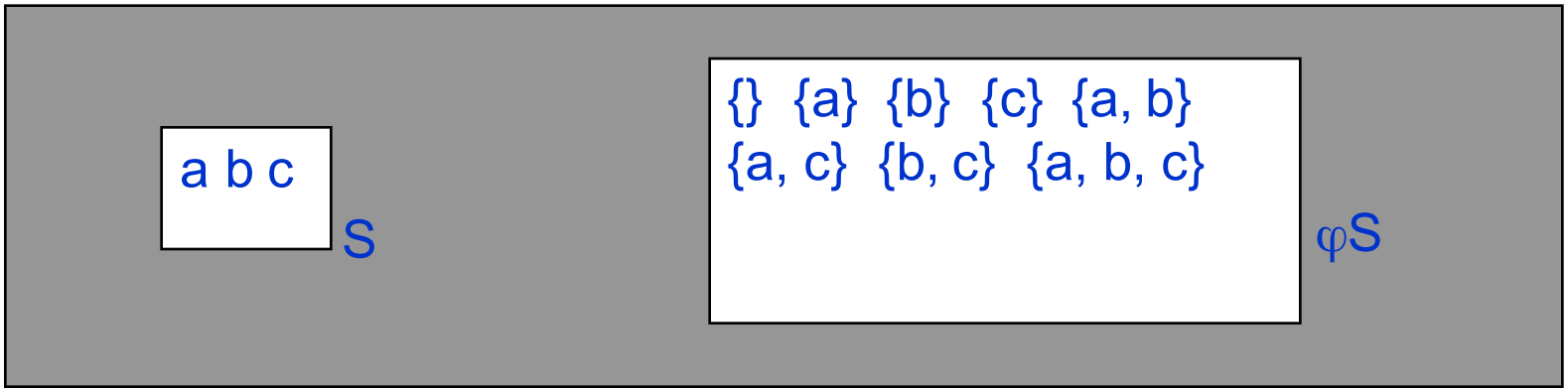
- C utiliza o conceito de ponteiros para manipular endereços de funções como valores:

```
int impar (int n) { return n % 2; }
int negativo (int n) { return n < 0; }
int multiplo7 (int n) { return !(n % 7); }
int conta (int x[], int n, int (*p)(int) ) {
  int j, s = 0;
  for (j = 0; j < n; j++) if ( (*p)(x[j]) ) s++;
  return s;
}

int main() {
  int vet [10]; /* Preencher vetor... */
  printf ("%d\n", conta (vet, 10, impar));
  printf ("%d\n", conta (vet, 10, negativo));
  printf ("%d\n", conta (vet, 10, multiplo7));
}
```

Conjuntos potência

- Tipos de dados cujo conjunto de valores corresponde a todos os possíveis subconjuntos que podem ser definidos a partir de um tipo base S: φ S = {s | s $\subseteq$ S}



Conjuntos potência

- Cardinalidade: $\# \varphi S = 2^{\#S}$
- Operações básicas:
  - Pertinência;
  - Contém;
  - Está contido;
  - União;
  - Diferença;
  - Diferença simétrica;
  - Interseção.

Conjuntos potência

- Poucas LPs oferecem. Muitas vezes de forma restrita;
- Pascal (somente discretos, primitivos e pequenos):

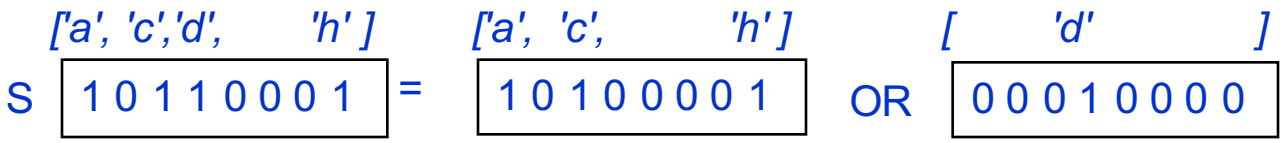
```
TYPE
  Carros = (corsa, palio, gol);
  ConjuntoCarros = SET OF Carros;
VAR
  Carro: Carros;
  CarrosPequenos: ConjuntoCarros;
BEGIN
  Carro := corsa;
  CarrosPequenos := [palio, gol];  (*atribuição*)
  CarrosPequenos := CarrosPequenos + [corsa]; (*união*)
  CarrosPequenos:= CarrosPequenos * [gol];(*intersecao*)
  if Carro in CarrosPequenos THEN  (*pertinencia*)
  if CarrosPequenos >= [gol, corsa] THEN (*contem*)

  (* ... *)
```

Conjuntos potência

- Restrições de Pascal visam permitir implementação eficiente:

```
VAR S: SET OF [ 'a .. 'h' ];
BEGIN
  S := ['a', 'c', 'h'] + ['d'];
END;
```



- Visto como são implementados em Pascal, pode-se usar operações binárias em C para criar conjuntos potência;
- Java possui Set em sua API, porém nem todas as operações mencionadas anteriormente são oferecidas.

Tipos recursivos

- Tipos recursivos são tipos de dados cujos valores são compostos por valores do mesmo tipo:
  - R ::= < parte inicial > R < parte final >
  - Tipo Lista ::= Tipo Lista Vazia | (Tipo Elemento x Tipo Lista).
- A cardinalidade de um tipo recursivo é infinita;
- Isto é verdade mesmo quando o tipo do elemento da lista é finito;
  - Ex.: o conjunto de valores do Tipo Lista é infinitamente grande (não podendo ser enumerado) embora toda lista individual seja finita.

Tipos recursivos

- Definidos a partir de ponteiros ou diretamente:

```
// Em C:
struct no {
   int  elem;
   struct no*
prox;
};
```

```
// Em C++:
class no {
   int elem;
   no* prox;
};
```

```
// Em Java:
class no {
   int elem;
   no prox;
}
```

```
GHCi, version 7.8.3: http://www.haskell.org/ghc/

> let lista = [1, 1, 2, 3, 5, 8, 13]
> lista
[1,1,2,3,5,8,13]
> lista ++ [21, 34, 55, 89, 144, 233]
[1,1,2,3,5,8,13,21,34,55,89,144,233]
```

Tipo ponteiro

- Não se restringe a implementação de tipos recursivos embora seja seu uso principal;
- Ponteiro é um conceito de baixo nível relacionado com a arquitetura dos computadores;
- O conjunto de valores de um tipo ponteiro são os endereços de memória e o valor *nil*;
- Considerados o *goto* das estruturas de dados.

Tipo ponteiro

- Atribuição:

```
int *p, *q, r;  // Dois ponteiros para int e um int.
q = &r;         // Atribui endereço de r a q.
p = q;          // atribui endereço armazenado em q a p.
```

- Alocação:

```
int* p = (int*) malloc (sizeof(int));// Alocação.
```

- Desalocação:

```
free(p);
```

- Derreferenciamento implícito:

```
INTEGER, POINTER :: PTR
PTR = 10
PTR = PTR + 10
```

Tipo ponteiro

- Derreferenciamento explícito:

```
int *p;
*p = 10;
*p = *p + 10;
```

- Aritimética:

```
p++;
++p;
p = p + 1;
p--;
--p;
p = p - 3;
```

- Indexação:

```
x = p[3];
```

Ponteiros genéricos

- Podem apontar para qualquer tipo:

```
int f, g;
void* p;
f = 10;
p = &f;
g = *p;  // erro: ilegal derreferenciar ponteiro p/ void
```

- Servem para criação de funções genéricas para gerenciar memória;
- Servem para criação de estruturas de dados heterogêneas (aquelas cujos elementos são de tipos distintos).

Tipo ponteiro – exemplo

```
// Exemplo em C:
#define nil 0

typedef struct no* listaint;
struct no {
  int cabeca;
  listaint cauda;
};

listaint anexa (int cb, listaint cd) {
  listaint l;
  l = (listaint) malloc (sizeof (struct no));
  l->cabeca = cb;
  l->cauda = cd;
  return l;
}

// Continua...
```

Tipo ponteiro – exemplo

```
void imprime (listaint l) {
  printf("\nlista: ");
  while (l) {
     printf("%d ",l->cabeca);
     l = l->cauda;
  }
}
main() {
  listaint palindromos, soma10, aux;
  palindromos = anexa(343, anexa(262, anexa(181, nil)));
  soma10 = anexa(1234, palindromos);
  imprime (palindromos); imprime (soma10);

  aux = palindromos ->cauda;
  palindromos ->cauda = palindromos ->cauda->cauda;
  free(aux);
  imprime (palindromos); imprime (soma10);
}
```

Problemas com ponteiros

- Baixa legibilidade:

```
p->cauda = q;
```

  - Inspeção simples não permite determinar qual estrutura está sendo atualizada e qual o efeito;
  - O comando acima poderia até mesmo estar criando um loop!

- Possibilitam violar o sistema de tipos:

```
int  i, j = 10;
int* p = &j;  // p aponta para a variavel inteira j
p++;          // p pode não apontar mais para um inteiro
i = *p + 5;   // valor imprevisível atribuído a i
              // ou falha de segmentação
```

Problemas com ponteiros

- Objetos pendentes:

```
int* p = (int*) malloc (10 * sizeof(int));
int* q = (int*) malloc (5 * sizeof(int));
p = q;   // área apontada por p torna-se inacessível
```

  - Provoca vazamento de memória;

- Referências pendentes:

```
int* p = (int*) malloc(10 * sizeof(int));
int* q = p;
free(p);      // q aponta agora para área de
              // memória desalocada
```

Problemas com ponteiros

- Referências pendentes (outro exemplo):

```
main() {
  int *p, x;
  x = 10;
  if (x) {
     int i;
     p = &i;
  }

  // p continua apontando para i, que não existe mais
}
```

  - Se o ponteiro não for inicializado, pode ser considerado também uma referência pendente;
  - Se acessado e o valor default for 0, *segfault*. Se não houver valor default, imprevisível.

Tipo referência

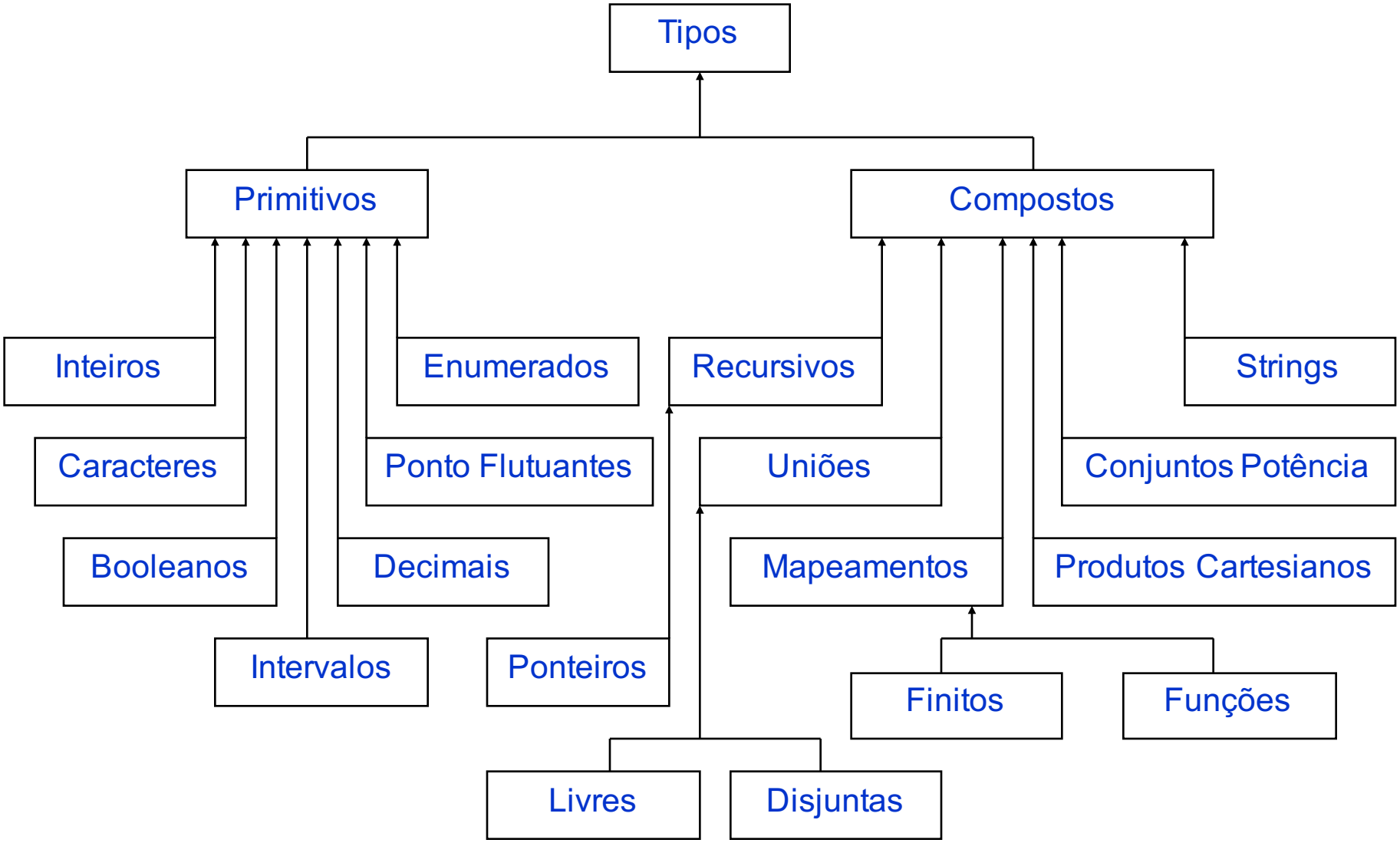
- O conjunto de valores desse tipo é formado pelos endereços das células de memória;
- Todas as variáveis que não são de tipos primitivos em Java são do tipo referência;
- C++ possui ponteiros e referências:

```
int res = 0;
int& ref = res;    // ref passa a referenciar res
ref = 100;         // res passa a valer 100
```

Tipo string

- Valores correspondem a uma sequência de caracteres;
- Não existe consenso sobre como devem ser tratadas;
- Podem ser consideradas:
  - Tipos primitivos (Perl, Snobol, ML);
  - Mapeamentos finitos (C, Pascal);
  - Tipo recursivo lista (Miranda, Prolog, Lisp);
- Três formas comuns de implementação:
  - Estática (COBOL);
  - Semi-Estática (C, Pascal);
  - Dinâmica (Perl, APL, Snobol).

Tipos segundo Varejão



Tipos segundo Tucker & Noonan

- Tipos primitivos (inteiros, ponto flutuante, caracteres, booleanos);
- Tipos compostos:
  - Enumerações (não são primitivos);
  - Ponteiros (idem);
  - Strings;
  - Matrizes (e vetores) = mapeamentos finitos;
  - Registros = produtos cartesianos;
  - Uniões;
- Tipos recursivos;
- Funções como tipos (mapeamento por função).

Conclusões

- Ver os tipos oferecidos por uma LP é um dos primeiros passos no seu estudo;
  - Valores possíveis, como são armazenados, operações disponíveis, etc.
  - Quais tipos podem ser combinados para formar novos tipos;
- Apresentamos uma classificação abstrata, que pode ser usada para estudar qualquer LP.

http://nemo.inf.ufes.br/