Linguagens de Programação
Conceitos e Técnicas

Valores e Tipos de Dados

Prof.

Conceituação

- Valor

  3 2.5 'a' “Paulo” 0x1F 026

- Tipo

  {true, 25, 'b', “azul” } não corresponde a um tipo

  { true, false } corresponde a um tipo

Tipos Primitivos

- Não podem ser decompostos em valores mais simples
- Costumam ser definidos na implementação da LP
  - Sofrem influência direta do hardware

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
  - C99
    - unsigned long long e long long int (>= 64 bits)
  - Representação
    - binária, complemento a dois (hardware)
    - vetor de dígitos (software)

Tipo Inteiro

- Representação
  - Hardware
    - binária
    - complemento a dois
      - único zero
      - aritmética sem levar em conta sinal
      - conversão: inverter e somar 1
  - Software
    - vetor de dígitos
      - limitação é capacidade de armazenamento

Tipo Inteiro

- Python
  - Um "único" tipo inteiro
    - Transparente a transição de representações
    - Versão 2: tipos int e long (L): 2**1024
    - Versão 3: são unificados em int
    - Representação: complemento a dois + vetor de digitos
- Em JAVA, o intervalo de cada tipo inteiro é estabelecido na definição da própria LP

Tipos Inteiros em JAVA

Tipo	Tamanho (bits)	Intervalo	
		Início	Fim
byte	8	-128	127
short	16	-32768	32767
int	32	-2.147.483.648	2.147.483.647
long	64	-9223372036854775808	9223372036854775807

Tipo Inteiro

- Operações
  - C, C++, Java (trunca em direção a zero)
    - 7/-3 = -2 -7/3 = -2 -7/-3 = 2
    - 7%-3 = 1 -7%3 = -1 -7%-3 = -1
  - Python (trunca em direção a infinito negativo)
    - 7/-3 = -3 -7/3 = -3 -7/-3 = 2
    - 7%-3 = -2 -7%3 = 2 -7%-3 = -1
  - Exponenciação
    - Python: usa operador ** :: 2**10 = 1024
    - C: usafunções da biblioteca padrão

Tipo Booleano

- Tipo mais simples
  - Possui apenas dois valores
- C não possui o tipo de dado booleano, mas qualquer expressão numérica pode ser usada como condicional

  Valores ≠ zero ⇒ verdadeiro

  Valores = zero ⇒ falso
  - Abordagem de C pode provocar erros

```
if (c += 1) x = 10;
```

Tipo Booleano

- Abordagem comum em C

```
#define FALSE 0
#define TRUE 1
typedef int Bool;
Bool b = 37;
if (b == TRUE)          // falso
if (b += 1) x = 10;     // permanece
```

- C++ e C99 incluem o tipo bool

```
#include<stdbool.h>     // necessario incluir
bool b = 37;            // conversao implicita
if (b == true)          // verdadeiro
if (b += 1) x = 10;     // permanece
```

Tipo Booleano

- JAVA inclui o tipo de dado *boolean*

```
boolean b = 37;          // erro – nao ha conversao
boolean b = true;
if (b == true)           // verdadeiro
if (b += 1) x = 10;      // erro – operacao invalida
```

- Em Python o tipo bool é uma especialização do tipo inteiro
  - True = 1 e False = 0
  - False, 0, None, '' são todos valores falsos
  - Mesmos problemas de C

Tipo Caractere

- Armazenados como códigos numéricos
  - Tabelas EBCDIC, ASCII e UNICODE
- UTF (Unicode Transformation Format)
  - UTF-8, UTF-16, UTF-32
    - Tipos de codificação Unicode
    - UTF-8 usa até 4 bytes para representar caracteres Unicode

Tipo Caractere

- ADA oferece os tipos *Character, Wide_Character, Wide__ Wide_Character*

```
ascii_val:= Character'Pos(Meu_Char);
Meu_Natural:=Meu_Char;  -- erro de compilação
```

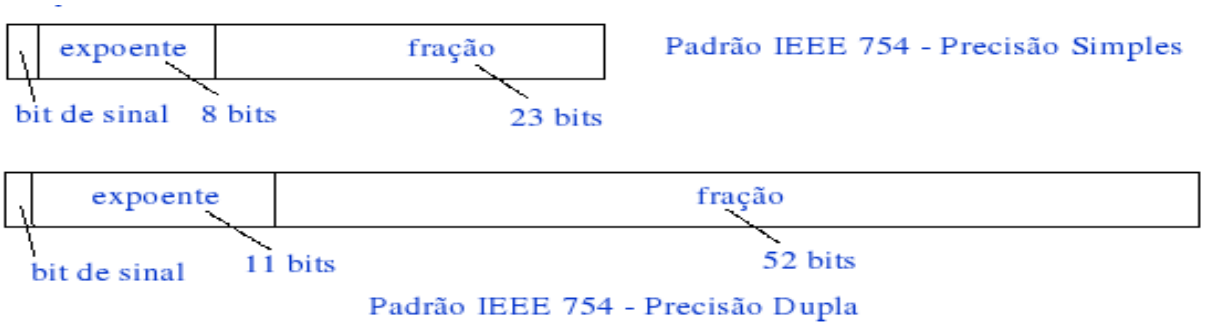
- Em C, o tipo primitivo *char* é classificado como um tipo inteiro

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

- C também possui o tipo wchar_t (>= 2 bytes)

Tipo Ponto Flutuante

- O tipo primitivo ponto flutuante modela os números reais
- LPs normalmente incluem dois tipos de ponto flutuante: *float* e *double*



Tipo Ponto Flutuante

- Cuidado com Precisão
  - *C*

```
float t = 5.0;
for (int i = 0; i < 50; i++) {
   t = t - 0.1;
}
printf ("t = %f\n", t);      // t = 0.000003
```

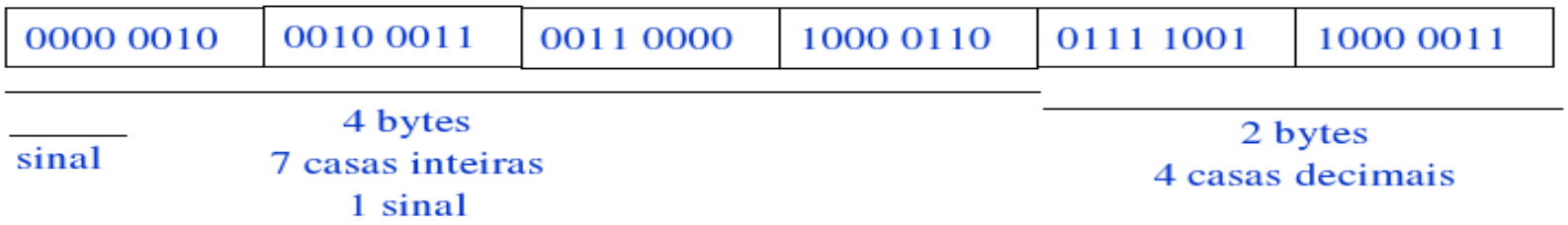
Tipo Ponto Flutuante

- Cuidado com Precisão
  - Python

```
t = 5.
for i in range(50):
   t = t - 0.1
print 'Float:', t
# Float: 1.02695629778e-15
```

Tipo Ponto Fixo

- Armazena um número fixo de dígitos decimais
  - Usa codificação BCD – Bynary Coded Decimal
  - COBOL possui



Tipo Ponto Fixo

- Python

```
from decimal import *
t = Decimal('5.')
for i in range(50):
   t = t - Decimal('0.1')
print 'Decimal:', t
# Decimal: 0.0
```

Tipo Complexo

- C99

```
#include <complex.h>
#include <stdio.h>

int main(void){
    complex c = 5 + 3*I;
    printf("%g + %gi\n", creal(c), cimag(c));
    return 0;
}
```

Tipo Complexo

- Python

```
c = 3 + 4j
print 'c =', c
print 'Parte real:', c.real
print 'Parte imaginária:', c.imag
print 'Conjugado:', c.conjugate()
```

- Saída

```
c = (3+4j)
Parte real: 3.0
Parte imaginária: 4.0
Conjugado: (3-4j)
```

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

```
enum mes_letivo { mar, abr, mai, jun, ago, set, out, nov };

enum mes_letivo 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 inclui o tipo enumerado de C e C++

Tipo Enumerado

- ADA

  type Cor is (Branca, Vermelha, Verde, Azul, Preta);

- Java (1.5)
  - Podem ter atributos e métodos

    public enum Naipes { Ouro, Espada, Paus, Coração }

Tipo Intervalo de Inteiros

- Em PASCAL e ADA, também é possível definir tipos intervalo de inteiros

```
type meses = 1 .. 12;
```

- Tipos intervalos herdam as operações dos inteiros

Tipos Compostos

- Tipos compostos são aqueles que podem ser criados a partir de tipos mais simples

  registros, vetores, listas, arquivos

- Entendidos em termos dos conceitos

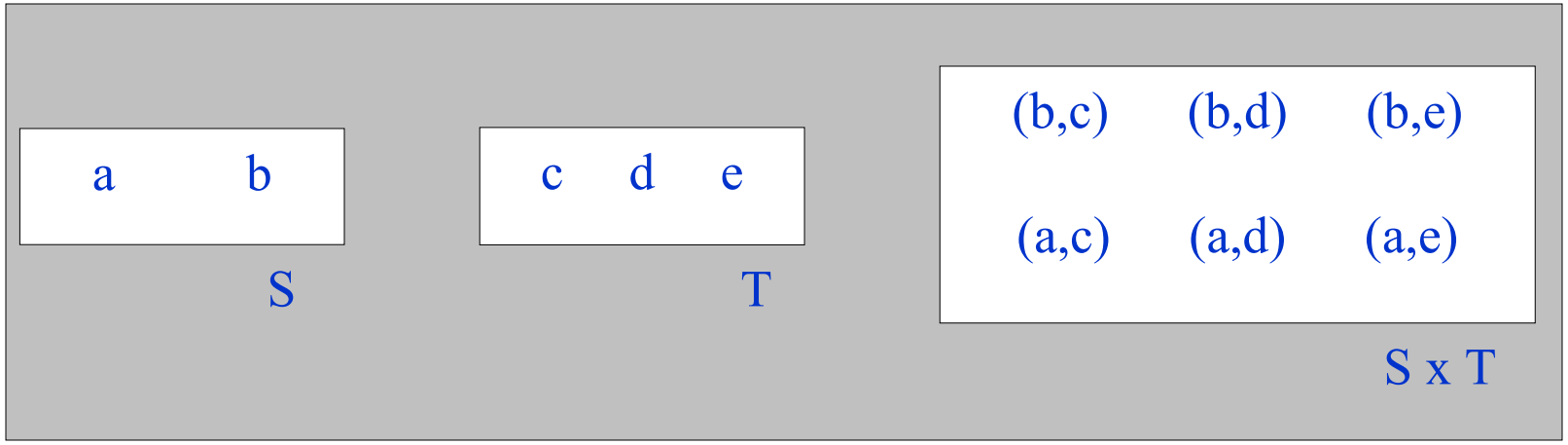
  produto cartesiano, uniões, mapeamentos, conjuntos potência e tipos recursivos

- Cardinalidade

  número de valores distintos que fazem parte do tipo

Produto Cartesiano

- Combinação de valores de tipos diferentes em tuplas



Produto Cartesiano

- São produtos cartesiano os registros de PASCAL, MODULA 2, ADA e COBOL e as estruturas de 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
  - JAVA só tem class
- Cardinalidade

  #(S1 x S2 x ... x Sn) = #S1x #S2x ...x #Sn

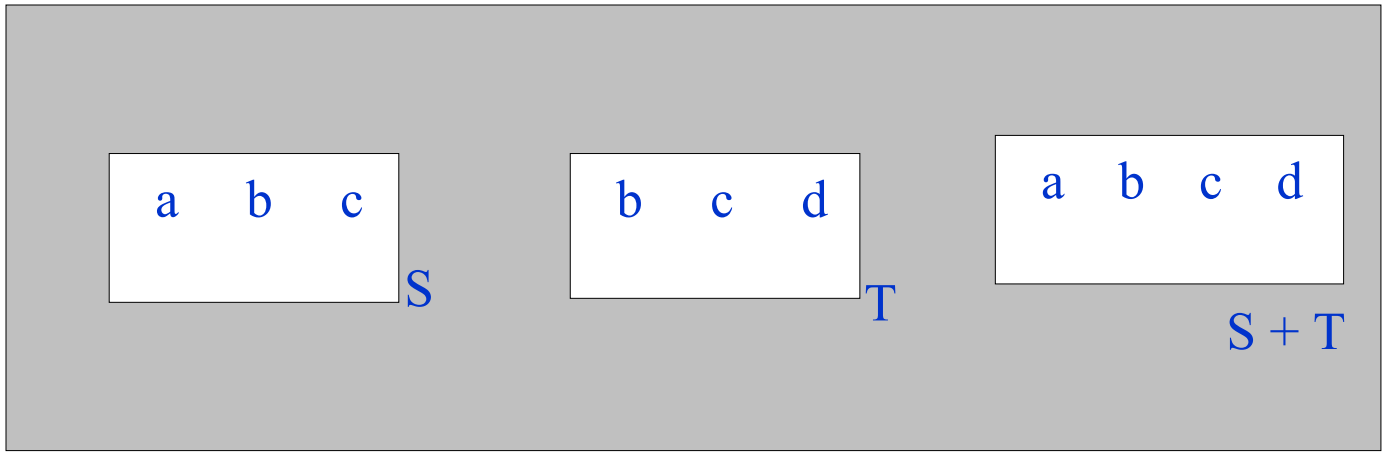
Produto Cartesiano

- Python
  - classes
  - tuplas
    - Imutáveis – não pode apagar, aumentar ou atribuir valores aos seus elementos

```
t = ([1, 2], 4)
t[0].append(3)  # t => ([1, 2, 3], 4)
t[0] = [2, 5]     # erro – não pode atribuir
```

Uniões

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



Uniões

- Uniões Livres
  - Pode haver interseção entre o conjunto de valores dos tipos que formam a união
  - Há possibilidade de violação no sistema de tipos

```
union medida {
  int centimetros;
  float metros;
};
```

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

Uniões

- 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

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

  - Cardinalidade

$$\#(S_1 + S_2 + \ldots + S_n) = \#S_1 + \#S_2 \ldots + \#S_n$$

Uniões

- Uniões Disjuntas

```
type COMBUSTIVEL is (GAS, GASOLINA, ALCOOL, FLEX);
type VEICULO (Motor: COMBUSTIVEL := FLEX) is
    record
      Ano : INTEGER range 1888..1992;
      Rodas : INTEGER range 2..18;
      case Motor is
        when GAS    => Cilindros: INTEGER range 1..16;
        when GASOLINA  => Consumo : INTEGER range 5..22;
                          Aditivada : BOOLEAN;
        when ALCOOL => Injecao : BOOLEAN;
        when FLEX  => Velocidade : INTEGER range 10..150;
      end case;
    end record;
```

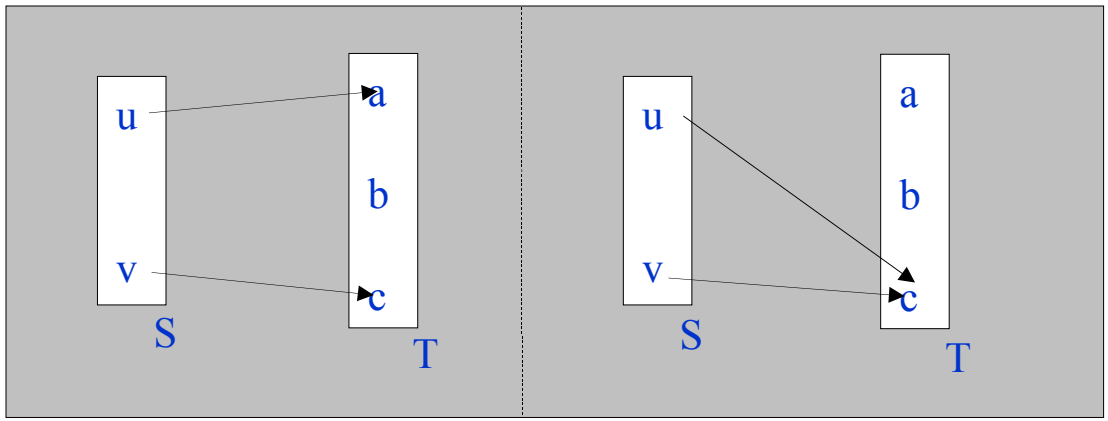
Uniões

- Uniões Disjuntas

```
Fiesta, Corolla, Uno : VEICULO;
Fit : VEICULO(GASOLINA);
begin
   Corolla := (GAS, 1956, 4, 8);
   Uno := (ALCOOL, 1985, Injecao => TRUE, Rodas => 4);
   Uno.Motor := GAS;            -- Erro: nao altera tag separado
   Uno.Ano := 1986;
   Uno.Injecao := FALSE;
   Fit.Ano := 2007;             -- Automatico – GASOLINA
   Fit.Rodas := 4;
   Fit.Consumo := 21;
   Fit.Aditivada := TRUE;
   Fiesta.Velocidade := 100;    -- Assume padrão – FLEX
```

Mapeamentos

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



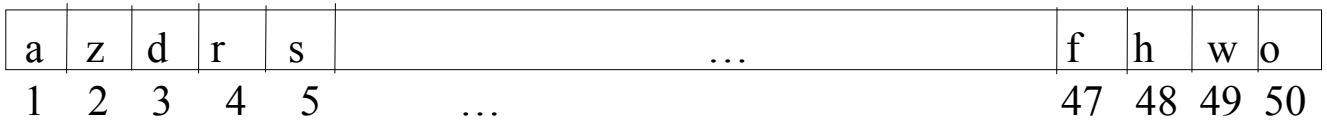
- Cardinalidade: $\#(S \rightarrow T) = (\#T)^{\#S}$

Mapeamentos Finitos

- O conjunto domínio é finito
- Vetores e matrizes

  array S OF T; (S → T)

  A: array (1..50) of Character; A: ([1,50] → Character)



- O conjunto índice deve ser finito e discreto
- Verificação de Índices em C e JAVA

  int v[7];

  V[13] = 198;

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	PYTHON

Categorias de Vetores

- Estáticos (C)

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

- Semi-Estáticos (C)

```
void f () {
    int x[10];
}
```

- Semi-Dinâmicos (Padrão C99)

```
void f (int n) {
    int x[n];
}
```

- Dinâmicos (Python)

```
p = [2, 3, 4]
p += [15, 16]
```

Vetores Semi-Dinâmicos

- Se alocados na pilha, deslocamento de variáveis posteriores é obtido em função do tamanho dinâmico do vetor, não mais sendo definido em tempo de compilação.

  dr = n * sizeof(tipo_elemento) + dc

  - acesso menos eficiente
- Se alocados no monte, necessário desalocar quando fim da função ou através de coletor de lixo

Vetores Dinâmicos

- Podem ser implementados em C, C++ e JAVA através do monte
- Necessário alocar nova memória, copiar conteúdo quando vetor aumenta de tamanho e desalocar memória original (realloc)
- Java não requer desalocação por causa da existência do coletor de lixo
- Programação em Python abstrai desses detalhes

Operações

- Python
  - Atribuição

```
p = [2, 3, 4]                    # p => [2, 3, 4]
```

  - Concatenação

```
p = [2, 3, 4] + [5, 6, 7]        # p => [2, 3, 4, 5, 6, 7]
```

  - Anexação

```
p += [8, 9]                      # p => [2, 3, 4, 5, 6, 7, 8, 9]
```

  - Fatiamento

```
p = p   [2: 5]                   # p => [4, 5, 6]
```

  - Comparações

```
p < [4, 6]                       # True
```

Operações

- Python
  - Pertinência

```
5 in p                  # True
```

  - Igualdade

```
p == [2, 3, 4]          # False
```

  - Indexação

```
p [2] = 7                   # p => [2, 3, 7]
```

  - Multiplicação

```
p = [2, 3]
p *= 3                  # p => [2, 3, 2, 3, 2, 3]
```

  - Muitas outras

Vetores Associativos

- Também chamados de dicionários ou mapeamentos
- Estrutura de dados onde os índices são uma coleção de chaves únicas (podendo ser de tipos distintos) associadas a valores (que também podem ser de tipos distintos)
- Normalmente implementadas como tabelas hash ou árvores balanceadas

Vetores Associativos

- Python
  - Chaves devem ser de tipos imutáveis (inteiros, strings, tuplas, ...)
  - Atribuição

```
d = {1: 'flavio', 'noite': 32.4, (3, 5): [1, 2, 3]}
d[1] = 'igor'
d['igor'] = 10
```

  - Indexação

```
x = d['noite']          # x => 32.399999999999
y = d[(3,5)]            # y => [1, 2, 3]
```

  - Comparação

```
d < {100: 1, 200:2}     # False
```

Vetores Multidimensionais

- Elementos são acessados através da aplicação de fórmulas

  posição mat [i] [j] =

  endereço de mat [0][0] + i × tamanho da linha + j × tamanho do elemento =

  endereço de mat [0][0] + (i × número de colunas + j) × tamanho do elemento

Vetores Multidimensionais

- Em JAVA vetores multidimensionais são vetores unidimensionais cujos elementos são outros vetores

```
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

Vetores Multidimensionais

- Python

```
mat = [ [1, 2, 3], [4, 5, 6], [7, 8, 9]]
x = mat [1] [0]                                  # x = 4
```

- Cardinalidade

```
int mat [5][4];
```

Conjunto de valores

$\{0, \ldots, 4\} \times \{0, \ldots, 3\} \rightarrow int$

$(\#int)^{\#(\{0, \ldots, 4\} \times \{0, .., 3\})} =$

$(\#int)^{(\#\{0, \ldots, 4\} \times \#\{0, .., 3\})} =$

$(\#int)^{5 \times 4} =$

$(\#int)^{20}$

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 valores do mapeamento: palindromo, impar, par, primo

Mapeamentos Através de Funções

- Python

```
def minmax(test, *args):
    res = args[0]
    for arg in args[1:]:
        if test(arg, res):
            res = arg
    return res
def menor(x, y): return x < y
def maior(x, y): return x > y
print minmax(menor, 4, 2, 1, 5, 6, 3)
print minmax(maior, 4, 2, 1, 5, 6, 3)
```

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;
}
```

Mapeamentos Através de Funções

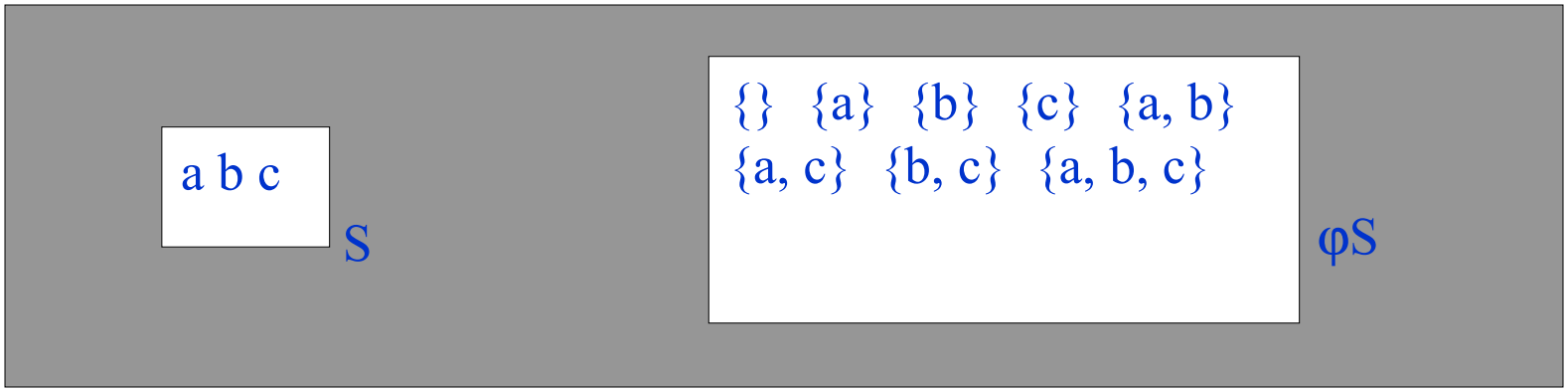
```
main() {
   int vet [10];
    printf ("%d\n", conta (vet, 10, impar));
   printf ("%d\n", conta (vet, 10, negativo));
   printf ("%d\n", conta (vet, 10, multiplo7));
}
```

- JAVA não trata funções (métodos) como valores
- Pode-se empregar algoritmos diferentes para implementar um mesmo mapeamento
- Algumas vezes, vetores e funções podem ser usados para implementar o mesmo mapeamento finito

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

$\varphi S = \{s \mid s \subseteq S\}$



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

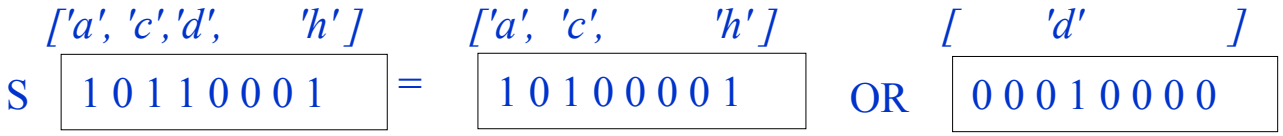
```
TYPE
      Carros = (corsa, palio, gol);
      ConjuntoCarros = SET OF Carros;
VAR
     Carro: Carros;
     CarrosPequenos: ConjuntoCarros;
BEGIN
     Carro:= corsa;
     CarrosPequenos := [palio, gol];                    /*atribuicao*/
     CarrosPequenos:= CarrosPequenos + [corsa];  /*uniao*/
```

Conjuntos Potência

```
CarrosPequenos:= CarrosPequenos * [gol];      /*intersecao*/
if Carro in CarrosPequenos THEN               /*pertinencia*/
if  CarrosPequenos >= [gol, corsa] THEN       /*contem*/
```

- Restrições de PASCAL visam permitir implementação eficiente

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



Conjuntos Potência

- Python

```
s1 = set(range(3))
s2 = set(range(10, 7, -1))
s3 = set(range(2, 10, 2))
s1s2 = s1.union(s2)
s1s2.difference(s3)
if 1 in s1:
    print '1 pertence a s1'
if s1.issuperset([1, 2]):
    print 's1 inclui 1 e 2'
```

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
- O conjunto de valores do tipo listas é infinitamente grande (não podendo ser enumerado) embora toda lista individual seja finita

Tipos Recursivos

- Em Python o tipo Lista é oferecido pela Linguagem
  - Admite listas homogêneas e heterogêneas
  - O conceito de vetor é englobado pelo de listas

```
l = [1, 1.4, 1+1j, "Qq tipo", ["lista dentro de lista"]]
l[3]="Entendi!"
k = []
len(l)
l.append(35)
l.reverse()
l.pop()
l.remove(1+1j)
l.sort()
```

Tipos Recursivos

- Tipos recursivos podem ser 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;
};
```

Tipos Ponteiros

- 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_

Tipos Ponteiros

```
#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;
}
```

Tipos Ponteiros

```
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);
```

Tipos Ponteiros

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

- Atribuição

```
int *p, *q, r;  // dois ponteiros para int e um int
q = &r;   // atribui endereco de r a q
p = q;    // atribui endereco armazenado em q a p
```

- Alocação

```
int* p = (int*) malloc (sizeof(int));
```

Tipos Ponteiros

- Desalocação

```
free(p);
```

- Derreferenciamento explícito

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

- Derreferenciamento implícito

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

Tipos Ponteiros

- Aritmética (C)

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

- Indexação (C)

```
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)

Problemas Com Ponteiros

- Baixa Legibilidade

  p->cauda = q;

  Inspeção simples não permite determinar qual estrutura está sendo atualizada e qual o efeito

- Possibilitam violar o sistema de tipos

```
int  i, j = 10;
int* p = &j;   // p aponta para a variavel inteira j
p++;       // p pode nao apontar mais para um inteiro
i = *p + 5;   // valor imprevisivel atribuido a i
```

Problemas Com Ponteiros

- Objetos Pendentes

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

  Provoca vazamento de memória

- Referências Pendentes

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

int* p;
*p = 0;
```

Problemas Com Ponteiros

- Referências Pendentes

```
main() {
    int *p, x;
    x = 10;
    if (x) {
        int i;
        p = &i;
    }
    // p continua apontando para i, que nao existe mais
}
```

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

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

Tipo String

- Valores correspondem a uma seqüência de caracteres
- Podem ser consideradas
  - Tipos primitivos (Python)

```
s = 'string'
type(s)      # => type ('str')
```

  - Mapeamentos finitos (C)

```
char nome[25];
```

  - Tipo recursivo lista (Haskell)

```
"A string" == [ 'A',' ','s','t','r', 'i','n','g']
```

Tipo String

- Três formas comuns de implementação
  - Estática (COBOL)
    - tamanho fixo

```
01 NOME                PIC X(25).
```

  - Semi-Estática (C)
    - tamanho variável, mas o máximo é fixo

```
char nome[25] = {'S', 'T', 'R', '\0'};
```

  - Dinâmica (Java)
    - tamanho variável

```
String nome = "Fabiana";
nome += " Batista";
```

- ADA possui as três formas

Tipo String

- Podem ser
  - Mutáveis (C)

    ```
    nome[1] = 'I';
    ```

  - Imutáveis (Python, Java)

    ```
    nome[6] = 'o'
    ```

- Operações
  - Python
    - Concatenação

      ```
      s = 'bom' + ' ' + 'dia'
      ```

    - Anexação

      ```
      s += '!'
      ```

Tipo String

- Operações
  - Python
    - Substring

      'bom' in s
    - Interpolação

      print 'tamanho de %s => %d' % (s, len(s))
    - Iteração

      for ch in s: print ch
    - Multiplicação

      print 3 * s

Tipo String

- Operações
  - Python
    - Igualdade

      'bom dia!' == s
    - Fatiamento

      s[4:7]
    - Maiusculização

      s.upper()
    - Muitas outras

Imutabilidade

- Python
  - tuplas e strings
    - Desestímulo a programação com efeitos colaterais
    - Tratamento de strings similar a número
      - tipos elementares
    - Desempenho
      - Tipos imutáveis são "hashable" e podem ser armazenados em dicionários e conjuntos através de função hash
      - Facilita a comparação de strings (compara endereços)

Hierarquia de Tipos