Tipos de Dados

- Abstrações que servem para modelar características da informação
  relacionada ao problema do mundo real em questão

- O ideal é um meio-termo na quantidade de tipos disponíveis em uma LP:

  - poucos tipos básicos
  - flexibilidade para a composição de tipos mais sofisticados pelo programador de acordo com o problema

Tipos Primitivos

- Aqueles que não são definidos em termos de outros tipos de dados

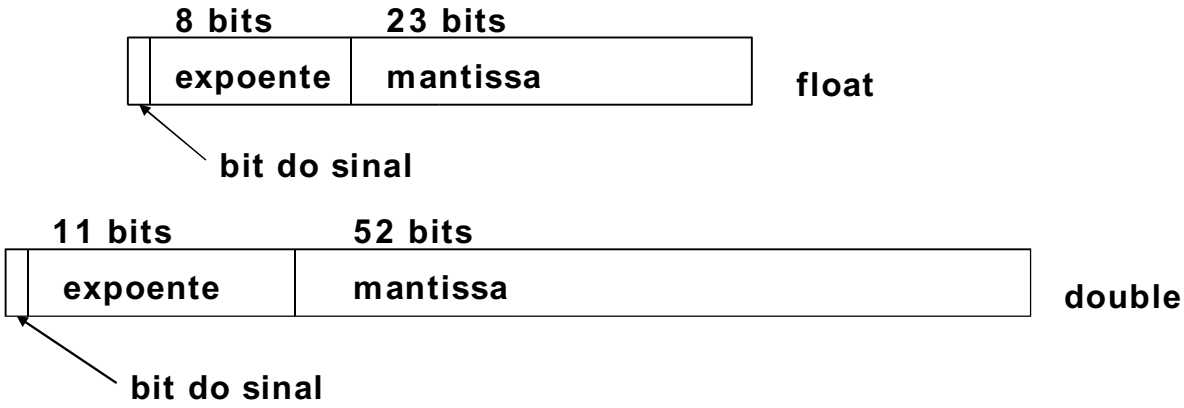
Tipos Numéricos

- Inteiro
  - short integer, integer, long integer, etc.
  - representação no h/w: cadeia de bits (sinal e magnitude, complemento de dois para negativos)

- Ponto Flutuante
  - float, double
  - aproximação para os reais (partes fracionárias não finitas, números não finitos em binário)
  - perda de exatidão em operações aritméticas

Tipos Primitivos

- Padrão IEEE 754 para ponto flutuante



- Decimal
  - para aplicações de comércio
  - representação em h/w: número de dígitos decimais e pos decimal são fixos
  - código binário para cada dígito decimal (normalmente, 1

Tipos Primitivos

- Booleano (lógico)
  - o mais simples dos tipos primitivos
  - importante para legibilidade (ex. flags); exceção: C
  - representação em h/w: representável por um único bit, mas usualmente implementado como um byte para facilidade de acesso às posições de memória

Tipos Primitivos

- Caractere
  - 1950 – 1970: EBCDIC (Extended Binary Coded Decimal Interchange Code)
    - 5 bits por caractere
  - 1970 – 2000: ASCII (American Standard Code for Information Interchange)
    - 7 bits por caractere; 8 no ASCII estendido
  - 1990 – : UNICODE
    - usado em Java
    - 16 bits 2^{16} caracteres possíveis = 65536
    - caracteres e ideogramas de línguas com alfabeto diferente do romano (grego, japonês, mandarim, tailandês, etc.)
    - símbolos matemáticos também

Tipos Cadeia de Caracteres

- Valores são sequências de caracteres

- Operações típicas:

  - Atribuição
  - Concatenação
  - Referência a substrings
  - Comparação (= , > , etc.)
  - Casamento de padrões (pattern matching)

Tipos Cadeia de Caracteres

- Questão de projeto da LP:
  - É um tipo primitivo?
    - melhor para redigibilidade

  Ou um array de caracteres?
    - acesso a elementos através subscritos (índices)

      - FORTRAN 77, FORTRAN 90 e BASIC:
        - tipo primitivo
        - atribuição, comparação, concatenação
      - Ada:
        - tipo não primitivo
        - atribuição, comparação, concatenação, e referência a substrings definidas
          - `NOME1 := NOME1 & NOME2;`
          - `NOME(2:4)`

Tipos Cadeia de Caracteres

- C/ C++ :
  - tipo não primitivo: array de `char`
  - biblioteca de funções padrão (string.h) com operações sobre strings
    - `strcpy, strcmp, strlen,` etc.

- Pascal:
  - tipo não primitivo
  - pode ser manipulado como primitivo se declarado como array de `char` com atributo `packed` (empacotado)

Tipos Cadeia de Caracteres

- Java:
  - como tipo primitivo: classe `String`
    - valores são cadeias constantes

    como tipo não primitivo: classe `StringBuffer`
    - valores em posições específicas podem mudar
    - tratados como arrays dinâmicos de caracteres
  - com métodos para operações sobre seus objetos

    (ex. length, < , > , append)

Tipos Cadeia de Caracteres

- SNOBOL4:
  - tipo primitivo
  - linguagem de manipulação de strings para o desenvolvimento de editores
  - operações de casamento de padrões
    - ex.

```
LETRA = 'abcdefghijklmnopqrstuvxz'
PADPALAV = BREAK(LETRA) SPAN(LETRA) . PALAVRA
```

Tipos Cadeia de Caracteres

- Perl:
  - tipo primitivo
  - também com operações de casamento de padrões
    - inspiradas nas operações de implementação de um editor de linhas UNIX (ed)

    - ex. expressão de representação de um padrão:

```
/[A-Za-z][A-Za-z\d]+/
```

descreve a formação típica de nomes (ex. de variáveis) em LPs:
- strings que começam com uma letra, seguida de uma ou mais letras ou dígitos

Tipos Cadeia de Caracteres

- Questão de projeto da LP:
  - O tamanho das cadeias é estático ou dinâmico?
    - para dinâmico, alocação dinâmica de memória necessária

    - FORTRAN 77, FORTRAN 90, COBOL, Ada, Pascal (packed arrays)
      - estático: especificado na declaração
        - ex. `CHARACTER (LEN = 15) NOME;`
        - pode haver desperdício de espaço

Tipos Cadeia de Caracteres

- C e C++
  - dinâmico de tamanho máximo limitado
    - ex. `char *s;`

```
s = (char*)malloc(10 * sizeof(char));
s = "cadeia";   ou
      s = "cadeia\0";
```

- tamanho real é deteminado pelo caracter "null" no final do string, mas espaço restante não é liberado

- SNOBOL4, JavaScript e Perl
  - dinâmico: tamanho variável sem máximo
    - máxima flexibilidade
    - custo de gerenciamento dinâmico de memória

Tipos Cadeia de Caracteres

- Java
  - estático: objetos constantes
    - ex.: `String str = "abc";`

```
str = "efgh";
```

  - dinâmico:

ex.:

```
x = new StringBuffer(2).append("a").append("b").append("c");
```

Tipos Ordinais

- Elementos discretos com uma relação de ordem entre eles
  (como nos inteiros)

- Definidos pelo usuário: tipos enumeração e subfaixa

Tipos Enumeração

- Todos os valores possíveis são enumerados pelo usuário como constantes simbólicas

  ex. `type DIAS_DA_SEMANA is (S, T, Q, I, X, Sab, Dom);`

- Implementados internamente através de associação de cada constante simbólica a um inteiro não-negativo
  (ex.: 1ª. cte associada a 0, 2ª. a 1, etc.)

Tipos Ordinais Enumeração

- Bom para:
  - legibilidade
  - manutenibilidade: fácil adicionar/atualizar os valores dos tipos
  - eficiência: o compilador pode selecionar e usar uma representação compacta eficiente (ex. inteiros de pequena magnitude)

  - confiabilidade: restringe valores e operações só ao necessário (verificação de faixa de valores, antecessor/sucessor, posição na enumeração de valores, etc.)

    - não vale para C e C++ que tratam os valores enumerados como inteiros quaisquer

      ex.: `enum cursosICE (Fisica, Matematica, Computaçao);`

Tipos Ordinais Enumeração

- Pascal

```
type cores = (vermelho, alaranjado, amarelo, verde,
              azul, anil, violeta);
var  cor: cores;
…
cor := azul;
…
if cor > verde;
…
for cor := vermelho to violeta do … ;
…
```


Tipos Ordinais Enumeração

- Ada – permite uma mesma constante simbólica em mais de uma declaração de tipo enumeração num mesmo ambiente de referenciamento

```
type letras is ('A','B','C','D','E','F','G','H','I','J','L','M',
                'N','O','P','Q','R','S','T','U','V','X','Z');
  type vogais is ('A','E','I','O','U');

for letra in vogais'('A')..vogais'('U') loop
```

- Java – classes que implementam a *interface* Enumeration

Tipos Ordinais

Tipos Subfaixa

Uma subsequência contígua de um tipo ordinal (da LP ou definido pelo usuário), ex. `1..10` é uma subfaixa do tipo inteiro

- Operações são as do tipo ordinal pai, exceto atribuição de valores fora da subfaixa

- Implementados internamente como o tipo pai + código inserido pelo compilador para garantir atribuições dentro da subfaixa

- Ainda mais legibilidade e confiabilidade

Tipos Ordinais Subfaixa

- Ex.: Pascal

```
type maiuscula = 'A'..'Z';
     indice = 1..100;
        pos = 0..MAX;
```

Ada

```
subtype DIAS_UTEIS is DIAS_DA_SEMANA range S..X;
```

- C, C++, Java: sem recursos "built-in" para tipos subfaixa; podem ser criados pelo programador através de funções, classes, métodos, etc.

Tipos Array (vetores e matrizes)

- Um *array* é um agregado homogêneo de elementos de dados de um certo tipo onde cada elemento é identificado por sua posição com relação ao primeiro

- Modelam coleções de valores que são do mesmo tipo e devem ser processados da mesma maneira

Tipos Array

Índices

- *Indexação* é ...

  `mapeia(nome_array, valor_índice)` → `elemento`

  - em tempo de execução, deve produzir endereços de elementos

- *Sintaxe*
  - índices entre [ ] – Pascal, Modula, C/ C++ , Java
  - índices entre ( ) – FORTRAN, PL/ 1, Ada
    - ex.: `soma = soma + c(i)`

    - adotados por falta de opção de caracteres ou para uniformidade com chamada a funções
    - programadores e compiladores usam informações de contexto para diferenciar

Tipos Array

Índices (cont.)

- *Tipos*
  - FORTRAN, C, Java: somente inteiro (`enum` é tratado como inteiro)
  - Pascal, Ada, Modula: tipos ordinais (`int`, `boolean`, `char`) ou definidos pelo usuário
  - vinculação dinâmica não é comum (ex. JavaScript)

- *Limite inferior*
  - especificado pelo programador na maioria das LPs
  - default 1 em FORTRAN 77 e 90
  - 1 em FORTRAN I, II e IV
  - implicitamente definido como 0 em C, C++, Java

- *Checagem de faixa de índices*
  - importante para confiabilidade
  - presente em Pascal, Ada, Java

Tipos Array

Arrays qto à vinculação do índice a uma faixa de valores e vinculação de armazenamento

1. *Estático*

   - vinculações estáticas do:
     - índice a sua faixa de valores; e
     - array a espaço de armazenamento (memória)

   - ex. FORTRAN 77, alguns arrays em Ada

   - vantagem: eficiência na execução (sem alocação/desalocação)

Tipos Array

2. *Pilha- dinâmico fixo*

- vinculação estática do índice a sua faixa valores; e
- vinculação dinâmica (em tempo de execução da declaração) do array a espaço de armazenamento

- ex. variáveis locais em Pascal, C/ C++ (as não declaradas `static`)

```
int colecao[5][4];
```

- vantagem: eficiência na utilização de memória – dois arrays em procedimentos distintos podem usar o mesmo espaço, desde que não estejam ativos ao mesmo tempo

Tipos Array

3. *Pilha- dinâmico*

- vinculações dinâmicas da faixa de índices e de espaço de armazenamento
- permanecem os mesmos durante o tempo de vida do array

- ex. Ada

```
declare
   CONJ : array (1..N) of FLOAT;
   begin
   ...
   end;
```

- vantagem: flexibilidade

Tipos Array

4. *Heap-dinâmico*

- vinculações dinâmicas da faixa de índices e de espaço de armazenamento
- podem mudar durante o tempo de vida do array
- exs.:
  - FORTRAN 90

```
INTEGER, ALLOCATABLE, ARRAY (:,:) :: MAT
```

(declara `MAT` como um array dinâmico bidimensional de inteiros)

```
ALLOCATE (MAT (10, NUM_DE_COLS))
```

(aloca espaço para `MAT` com 10 linhas e `NUM_DE_COLS` colunas)

```
DEALLOCATE MAT
```

(desaloca o espaço de `MAT`)

Tipos Array

- Java

```
int [ ] lista1;    // lista1 eh uma variavel de
                   // referencia inicializada com 'nil'
...

lista1 = new int [100];
```

- C (funções `malloc` e `free`), C++ (operadores `new` e `delete`) quando definidos ponteiros para os arrays

- APL, Perl e JavaScript:
  - arrays crescem implicitamente qdo atribuídos elementos além do último corrente
  - encolhem com a atribuição do agregado vazio `()`

- vantagem: mais flexibilidade

Tipos Array

Número de dimensões

- FORTRAN I: até 3
- FORTRAN IV em diante: até 7
- linguagens modernas não limitam o número de dimensões
- abstração poderosa! Não limita-se a dimensões espaciais

Tipos Array

Mais vinculação de armazenamento

- memória em hardware é usualmente linear
- portanto, é necessário linearizar estruturas multidimensionais
  - por linha
  - por coluna (FORTRAN)
- é mais eficiente fazer acesso seqüencial a elementos da matriz na ordem em que estão armazenados

Tipos Array

Inicialização

- FORTRAN 77 – ao mesmo tempo que alocação de memória (estática)

```
INTEGER LISTA(3)
DATA LISTA /0, 5, 5/
```

- C/ C++ – (pilha-dinâmico fixo, heap-dinâmico)
  - faixa de índices é implicitamente deteminada pelo tamanho da lista de inicialização

```
int lista [] = {2, 4, 6, 8};
char cidade [] = "manaus";
char *cidades [] = {"manaus", "manacapuru",
                    "itacoatiara"};
```


Tipos Array

- Ada – (pilha-dinâmico)

```
lista : array (1..5) of integer := (1, 3, 5, 7, 9);

tabela : array (1..14, 1..2) :=
                    (1 => (24, 10), 2 => (10, 7),
                     3 =>(12, 30), others => (0, 0));
```

- Pascal e Modula-2 não permitem inicialização na declaração

Tipos Array

Operações

- arrays inteiros como operandos (não elementos do array)

- maioria das LPs:
  - atribuição de array para array (A ← B) com as mesmas faixas de índice e número de dimensões
  - atribuição de lista de valores (A ← {1,...,n})

- Ada
  - concatenação (operador &) entre arrays unidimensionais e entre array unidimensional e elemento
  - relação de (des)igualdade (operadores = , /=)

Tipos Array

- FORTRAN 90
  - operações aritméticas, relacionais e lógicas
  - funções para operações como multiplicação, transposição, produto escalar

- APL
  - principal característica; a mais poderosa
  - todas as acima, mais operadores para transposição, inversão, operadores compostos (ex. `+.x`)

- Java
  - métodos

Tipos Array

Fatias

- uma fatia (*slice*) é uma sub-estrutura referenciável de um array
- referencimento depende do número de dimensões do array
- recursos em algumas linguagens

- exs.:
  - FORTRAN 90

```
INTEGER VET(1:10) , MAT(1:3, 1:3), CUBO(1:3, 1:3, 1:4)
```

`VET(3:6)`, `VET(2:10:2)` fatia da posição 2 à 10 de 2 em 2 posições

```
MAT(1:3, 2), MAT(2:3, 1:3)
CUBO(1:3, 1:3, 2)
```


  - Ada só permite fatias com elementos

Tipos Array

Arrays associativos

- uma coleção não ordenada de valores de dados indexados por uma coleção, de mesmo tamanho, contendo outros valores chamados chaves

- eficiente para acesso a elementos específicos do array

- ao contrário dos não-associativos que tem índices regulares, as chaves dos arrays associativos precisam ser armazenadas

- cada elemento é uma par <chave, valor de dado>

- disponíveis em Java e Perl

Tipos Array

- ex.: Perl
  - nomes de arrays associativos começam com %
  - são heap- dinâmicos

  - atribuição

```
%cres = ("Ana" => 87, "Pedro" => 79, "Carla" => 70);
```

  - referência a um elemento

```
$cres{"Pedro"} = 80;
```

  - remoção de um elemento

```
delete $cres{"Carla"};
```

  - para tornar o array vazio

```
%cres = ();
```

  - outros operadores: `exists`, `keys`, `values`, `each`

Tipos Registro

- Um *registro* é um agregado de elementos de dados possivelmente heterogêneos em seus tipos, sendo cada elemento identificado por um nome

- Diferentes dos arrays, que têm elementos homogêneos
  - elementos (ou campos, membros) não são referenciados por índices, mas por seus nomes (ou identificadores)

- Disponíveis em todas as linguagens modernas amplamente utilizadas

Tipos Registro

- Registros podem ser aninhados
- Exs. de **definições**:

```
01 REG-EMP.
   02 NOME.
      05 PRIMEIRO          PICTURE IS X(20).
      05 MEIO              PICTURE IS x(10).
      05 ULTIMO            PICTURE IS X(20).
   02 REM-HORA             PICTURE IS 99V99.
```

(Cobol – note os números de nível)

```
MOVE CORRESPONDING REG-EMP TO REG-SEGURADO.
```

(uma atribuição possível)

```
type endereco = record rua: string [30];
                       numero: integer;
                       cep: real
                end;
var cidadao: record nome: string [30];
                    ender: endereco;
                    cpf: real;
                    sexo: char
             end;
```

(Pascal)

Tipos Registro

```
struct data {
       int dia;
       int mes;
       int ano;
       int dia_ano;
       char nome_mes[3];
} d;
```

(C)

```
struct pessoa {
       char nome[20];
       char endereco[30];
       long cpf;
       double salario;
       struct data nascimento;
} funcionario;
```

```
struct data d = {7, 9, 2004, 250, "set"};
```

(uma atribuição)

Tipos Registro

Referências a campos de um registro

- Exs.:
  - Cobol: nome_do_campo OF nome_do_registro_1 OF ... OF nome_do_registro_n
    (de dentro para fora)

```
MEIO OF NOME OF REG-EMP
```

  - maioria das LPs: notação com ponto (de fora para dentro)
    - Pascal: `cidadao.endereco.cep`

- estas são *referências totalmente qualificadas*: o capo específico e todos os registros aninhados envolvidos são explicitamente nomeados

Tipos Registro

Referências a campos de um registro (cont.)

- *referências elípticas* são as que explicitam o nome de campo mas podem omitir nomes dos registros aninhados envolvidos (até todos)
  - condição: referência resultante não é ambígua no ambiente de referenciamento
    - exs.:
      - Cobol:

```
MEIO
MEIO OF NOME
MEIO OF REG-EMP
```

      - Pascal:

```
with cidadao do
begin
   nome := 'Jose';
   rua := 'dos Bares';
   numero := 0;
   cep := 69060000
   cpf := 99999999999;
   sexo: 'M'
end;
```

  - prejudiciais para legibilidade

Tipos União

- Uma variável do tipo união pode armazenar valores de diferentes tipos em diferentes momentos da execução
  - verificação de tipos dinâmica

- Permitem manipular diferentes tipos de dados numa mesma área de armazenamento

- É alocado espaço de armazenamento suficiente para o maior tipo

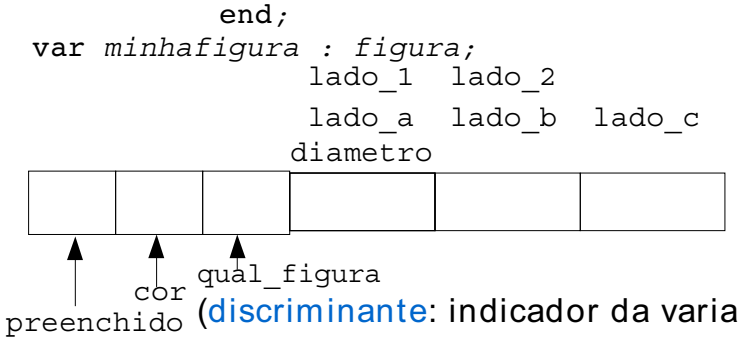
- É como se o tipo da variável fosse a "união" de todos os tipos que ela pode armazenar

Tipos União

- Ex. Pascal: integra uniões a registros criando *registros variantes*

```
type forma = (circulo, triangulo, retangulo);
     cores = (vermelho, verde, azul);
     figura =
        record
           preenchido : boolean;
           cor : cores;
           case qual_figura : forma of
                circulo: (diametro: real);
                triangulo: (lado_a : integer;
                            lado_b : integer;
                            lado_c : integer);
                retangulo: (lado_1: integer;
                            lado_2: integer)
           end;
var minhafigura : figura;
```

parte variante do registro



(discriminante: indicador da variante corrente)

Tipos União

- Ex. Pascal (cont.)

```
case minhafigura.qual_figura of
   circulo: writeln('Eh um circulo; seu diametro eh:',
                    minhafigura.diametro);
   triangulo: writeln('Eh um triangulo; seus lados sao:',
                      minhafigura.lado_a, minhafigura.lado_b,
                      minhafigura.lado_c);
   retangulo: writeln('Eh um retangulo; seus lados sao:',
                      minhafigura.lado_1, minhafigura.lado_2)
end;
```

  - situação possível:

```
minhafigura.diametro | 2,73 |

medida_fig := minhafigura.lado_c;
```

(mudança de variante sem mudança de discriminante!)

  - inconsistência entre a variante corrente e o discriminante
  - para confiabilidade, o discriminante deve sempre indicar qual a variante corrente armazenada : Ada

Tipos União

- Ex. Ada

  - o discrimante está sempre sintonizado com a variante atual
  - toda referência à variável união inclui o discriminante
  - também registro variante como Pascal

```
FIGURA_1: FIGURA;

FIGURA_1 := (PREENCHIDO => true,
             COR => AZUL,
             QUAL_FIGURA => RETANGULO,
             LADO_1 => 12;
             LADO_2 => 3);
```

Tipos União

- Outras LPs
  - uniões livres
    - verificação de tipos não é feita:
      - responsabilidade do programador
      - por isso, não fortemente tipadas

    - C e C++: `union`
      - ex.:
```
union u_tag {
        int ival;
        float fval;
        char *sval;
      } u;
```

    - FORTRAN: `equivalence`
    - não têm que ser parte de registros (ou estruturas)

  - Java não dispõe de tipos união

Tipos Conjunto

- Um tipo conjunto é aquele cujas variáveis podem armazenar coleções não-ordenadas de valores distintos de algum tipo ordinal chamado seu *tipo básico*

- Usados para modelar conjuntos matemáticos

- Pascal é a única LP imperativa amplamente utilizada que inclui um tipo conjunto (também presente em Modula)
  - operações:
    - união (+), interseção (*), differença (-), relacionais (=, <>, `A >= B` (A contém B), `A <= B` (A está contido em B), `X in A` (X é um elemento de A))
  - pode ter tamanho máximo, dependendo da implementação
    - empobrece redigibilidade e portabilidade

Tipos Conjunto

- Tipo conjunto em Pascal (cont.)
  - declaração:

```
type T = set of Tipobase;
```

  - se o tipo base possui n elementos então o tipo `set of Tipobase` contém 2^n elementos
    - ex.: `var Digset : set of 1 .. 3;`
      possíveis valores para `Digset`:
      `[ ],[1],[2],[3],[1,2],[1,3],[2,3],[1,2,3]`

Tipos Conjunto

- Tipo conjunto em Pascal (cont.)
  - *ex.:*

```
program Bingo;
type Pedra = 1..90;
     Saco = set of Pedra;
var Bingo : Saco;
    Aux : Pedra;
begin
  Bingo := [1..90];
   while Bingo <> [ ] do
    begin
        Aux := Geranum;  (* função gera num. aleat. em 1..90 *)
        if Aux in Bingo
          then
            begin
                Bingo := Bingo - [Aux];
                writeln(Aux)
            end
    end
end.
```

Tipos Conjunto

- Outras linguagens:
  - Ada: não inclui conjuntos, mas define o operador `in` para tipos enumeração
  - Java: classe de operações sobre conjuntos
  - outras: conjuntos simulados como arrays
    - pouco prático pos exige manipulação elemento a elemento