Linguagens de Programação

Tipos

Carlos Bazilio
carlosbazilio@id.uff.br
https://carlosbazilio.github.io/cursos/paradigmas/

Definições

- Classificação que categoriza conjuntos de valores com comportamentos similares
- Tipos de dados primitivos: tipos de dados que **não são definidos** em termos de outros tipos
- Exemplos:
  - Tipo Numérico (Inteiro, Ponto Flutuante, Decimal, ..)
  - Tipo Booleano (valores verdade)
  - Tipo Caracter

Cadeia de Caracteres ou String

- O valor usualmente é uma sequência de caracteres
- Apesar de usualmente não ser um tipo primitivo, é considerado um tipo básico no projeto de linguagens
- Operações comuns:
  - Extração de parte da cadeia, Concatenação, Cálculo do tamanho, Busca e/ou substituição de parte da cadeia, Comparação, etc.

Cadeia de Caracteres ou String

- Uma outra operação comum em cadeias é a busca por padrões
- A linguagem SNOBOL foi precurssora nesse tipo de operação
- Uma outra linguagem importante nesse aspecto é a linguagem Perl
- Nela foram incorporados recursos, os quais são comumente chamados expressões regulares
  - Exemplo: /[A-Za-z][A-Za-z\d]*/
- Acima define-se regras para identificadores

Tipos Ordinais
Enumeração

- Define faixa de valores, a qual usualmente é associada a um conjunto de inteiros
  - Ex (ADA): type DIAS is {Seg, Ter, Qua, Qui, Sex};
- Uma questão de projeto é quanto a ocorrência de um valor em mais de uma enumeração (literal sobrecarregado)
  - type LETRAS is {'A', 'B', 'C', ... 'Z'}
  - type VOGAIS is {'A', 'E', 'I', 'O', 'U'}
- As linguagens normalmente proibem estas ocorrências, ou forçam uma diferenciação explícita

Tipos Ordinais
Sub-faixa

- Um tipo similar à enumeração, normalmente uma sub-sequência
  - Ex (Pascal): **type** maiuscula = 'A' .. 'Z'; indice = 1..100;
- No caso de Ada, esses tipos são sub-tipos de tipos já existentes
  - Ex: **subtype** FINAL **is** DIAS **range** Qua .. Sex;
- Neste caso, as operações definidas para o tipo pai (DIAS, no exemplo) também valem para o filho (FINAL); ou seja, o filho herda as operações do pai

Coleções

- Permite a manipulação de conjunto de valores
- A linguagem pode permitir o acesso por índices ou de forma iterativa, como numa lista encadeada
- Coleções com acesso por índice:
  - Vetores, Matrizes, etc
- Coleções acessadas de forma iterativa:
  - Listas, Pilhas, Filas, Mapas, Conjuntos, etc
- Há linguagens que oferecem tipos mistos (Java, por exemplo)

Coleções

- Algumas possibilidades de implementação:
  - Estática: alocada antes da execução e com tamanho bem definido (Fortran 77)
  - Fixa e Dinâmica na Pilha: alocação feita durante a a execução, mas também com um tamanho já definido (Pascal e C)
  - Dinâmica na Pilha: mesmo que o anterior, mas com o tamanho sendo definido durante a execução (Fortran 90)
  - Dinâmica no Heap: totalmente flexível, podendo ser alocada a qualquer momento e tendo seu tamanho modificado ao longo da execução (C, C++, ...)

Coleções Indexadas Características

- Usualmente, o número de índices suportados por uma linguagem é indeterminado
- Operações comuns:
  - Inicialização múltipla
    - Ex (C): char *nomes[] = {“fulano”, “ciclano”, “beltrano”};
    - Ex (ADA-like): nomes : array (1..3) of STRING := (1 => “fulano”, 2 => “ciclano”, others => “”);
  - Atribuição, Comparação, Concatenação, Operações Aritméticas (soma, média, ..), Mín/Max, Ordenação ..
- Índices numéricos são necessários para acessar os elementos

Coleções Indexadas Características

- Quanto a implementação, o esforço computacional para uso destas estruturas reside na manipulação de memória
  - Por exemplo, em C, o acesso *vet[indice]* representa a expressão *(*(vet + indice))*, onde o * é o operador que dereferencia um endereço de memória
- Esta complexidade aumenta com o aumento no número de índices, por exemplo, na manipulação de matrizes

Matrizes Associativas

- Diferente das matrizes comuns, essas matrizes associativas (mapas, dicionários, tabelas) utilizam índices diferentes de números (chaves)
- A cada chave utilizada, que pode ser uma string por exemplo, é associado um valor
- Por exemplo, em Perl:
  - %salarios = (“fulano” => 10000, “ciclano” => 5000);
  - $salarios{“beltrano”} = 8000;
  - delete $salarios{“ciclano”};
- Essas estruturas são bastante dinâmicas e elegantes

Tipo União ou Registro com Variante

- Suponha que queremos criar um único tipo registro para manipular diferentes informações
- Estas informações possuem campos em comum e campos exclusivos
- O tipo união possui essa flexibilidade

```
type
    forma = (circulo, triangulo, retangulo);
    cores = (vermelho, verde, azul);
    figura = record
        preenchido: boolean;
        cor: cores;
        case aspecto: forma of
            circulo: (diametro: real);
            triangulo: (le, ld: integer; angulo: real);
            retangulo: (l1, l2: integer);
        end;
var minhafigura: figura;
```

Tipo União

- O exemplo em Pascal fornecido anteriormente pode ser utilizado da seguinte forma:

```
var minhafigura: figura;
{Inicialização dos campos comuns da variável minhafigura}
minhafigura.aspecto := triangulo;
case minhafigura.aspecto of
      circulo: writeln('E um circulo com diametro: ', minhafigura.diametro);
      triangulo: ...;
      retangulo: ...;
end;
```

- Ou seja, precisamos definir qual o tipo de figura manipulada
- Observe que erros podem ocorrer caso se defina um figura de um aspecto e utilize campos de outro

Tipo União

- Em termos de implementação, é alocado para o tipo a área de memória correspondente à maior variante
- Isto propicia um uso mais eficiente da memória
- Em C, esta funcionalidade é obtida através do recurso denominado *union*
- Em linguagens OO (C++, Java, etc), estes recursos são elegantemente substituídos

Conversão de Tipos

- Suponha a atribuição:

```
int a = 10;
float b = 7.5;
a = b;
```

- Nesta atribuição teremos perda de valor, uma vez que o tipo *int* só armazena inteiros
- Algumas linguagens/compiladores aceitam naturalmente esta atribuição, enquanto outros emitirão um alerta; também existem os que proibirão

Conversão de Tipos

- Na situação em que há um alerta, este pode ser eliminado através das operações de conversão explícita (*casting*)

```
int a = 10;

float b = 7.5;

a = (int) b;
```

- Desta forma, o programador indica para o compilador que está ciente da possível perda de valor

Classificação de Linguagens com respeito a Tipos

- As linguagens podem ser classificadas em:
  - Tipagem Forte
  - Tipagem Fraca
- Quanto ao momento em que as verificações são feitas, podemos ter:
  - Tipagem Estática
  - Tipagem Dinâmica

Sugestões de Tipos (Type Hints) em Python

```
def greeting(name: str) -> str:
    return 'Hello ' + name
```

- Vantagens: capturar certos erros, documentação, melhora de IDEs e linters, construção e manutenção do sistema
- Desvantagens: esforço de inserção, adaptado para versões recentes de Python, penalidade na inicialização
- Fonte: https://realpython.com/lessons/pros-and-cons-type-hints/

Ponteiros, Tipos Recursivos e Tipos Genéricos

- Demonstrar exemplo de lista genérica em C

Tipos Opcionais

- Utilizados para encapsular valores nulos ou indefinidos
- Importante para se declarar, já na assinatura de operações, que parâmetros/retornos podem ser nulos
- Busca tornar códigos mais seguros
- Algumas linguagens sofrem a crítica de tentarem implementar o conceito e tornar o código mais verboso e ainda sujeito a erro: https://homes.cs.washington.edu/~mernst/advice/nothing-is-better-than-optional.html

Exemplo de Uso de Optional em TypeScript

```
let foo = {bar : {baz: 40}} // JSON - JavaScript Object Notation

// let foo = null

let x = foo?.bar.baz

// let x = foo === null || foo === undefined ? undefined : foo.bar.baz

console.log(x)
```

https://playcode.io/typescript/