Paradigmas de Programação

Sistemas de Tipos

Introdução

Este material contém notas de aula sobre *Sistemas de Tipos*. O objetivo é apresentar conceitos, classificações e características deste componente presente em linguagens de programação.

Sistemas de Tipos

Como já visto na aula de Introdução, um dos elementos básicos, comum a todas as linguagens de programação, é o **sistema de tipos**. O sistema de tipos é um conjunto de regras que atribuem tipo a recursos da linguagem, tais como:

- Condições
- Constantes
- Estruturas de dados
- Expressões
- Parâmetros
- Retorno de métodos
- Variáveis

Os sistemas de tipos existem (ou deveriam existir) para impedir ou reduzir erros e inconsistências nas interações entre recursos da linguagem. Eles decidem, por exemplo, o que fazer em operações com valores de representações diferentes e que ocupam espaços diferentes, tais como casos em que o usuário tenta somar um valor inteiro e um valor de ponto

flutuante com precisão dupla.

Todo *tipo* possui um conjunto de valores e operações possíveis. Na linguagem C#, por exemplo, o tipo `bool` só aceita dois valores: `true` e `false`. Em uma linguagem hipoteticamente criada para fins didáticos, qual pode ser o conjunto de valores e de operações para um tipo que armazena caracteres?

CURIOSIDADE

Quais são, respectivamente, o maior e o menor valor inteiro em Java?

De acordo com a especificação da linguagem, Java utiliza 32 bits para armazenar números inteiros, o que significa que seus 2^{32} valores estarão entre -2^{31} e $2^{31}-1$.

O arquivo **`Main.java`** traz um exemplo que mostra a impressão do menor e do maior valor em Java. Para isso, foram utilizadas as constantes **`Integer.MIN_VALUE`** e **`Integer.MAX_VALUE`**. Ao executar o programa, nota-se que o menor valor é -2147483648 e o maior valor é 2147483647.

Observação: experimente substituir **`Integer.MIN_VALUE`** por **`Integer.MIN_VALUE-1`** e **`Integer.MAX_VALUE`** por **`Integer.MAX_VALUE+1`** e observe o resultado! O que acontece?

Sistemas de Tipos Monomórficos e Polimórficos

Um sistema de tipos pode ser classificado quanto à flexibilidade e à capacidade de manipular diferentes tipos de dados. Nesse sentido, o sistema pode ser **monomórfico** ou **polimórfico**.

- **Monomórfico**: para cada recurso da linguagem deve ser declarado um tipo específico. Na linguagem C, por exemplo, uma assinatura de função válida seria:
    - **`double raizQuadrada(double numero)`**

Neste caso, tanto o parâmetro (`numero`) quanto o valor retornado pela função são do tipo `double`, ou seja, recebe um `double`, realiza um conjunto de instruções e retorna um valor `double`: `double -> double`

- **Polimórfico**: um recurso é dito polimórfico se ele funciona para vários tipos diferentes. Assim, é possível escrever funções genericamente.

Na linguagem Haskell, por exemplo, a declaração da função `segundoValor` seria válida da seguinte maneira:

- **`segundoValor (_, b) = b`**

Neste caso, independentemente do tipo dos parâmetros recebidos, a função retornará sempre o segundo valor. Outro exemplo em Haskell:

- **`identidade x = x`**

Neste caso, independentemente do tipo do parâmetro recebido, a função retornará sempre o próprio parâmetro.

O arquivo Sistema-de-Tipos-Polimorfico.hs contém os exemplos citados da linguagem Haskell. Para executá-lo, basta abrir o terminal do diretório em que o arquivo se encontra e digitar:

```
ghci
:l Sistema-de-Tipos-Polimorfico.hs
```

Depois basta usar as funções como desejar, como nos exemplos a seguir:

```
segundoValor (“IFC”, “BCC”)
segundoValor(99.99, ‘r’)
identidade “IFC”
identidade 90.14
```

Polimorfismo

O Polimorfismo *ad-hoc* refere-se à situação em que um valor é capaz de adotar um entre vários tipos. Há autores que definem polimorfismo *ad-hoc* como sinônimo de polimorfismo de sobrecarga. Outros definem que polimorfismo *ad-hoc* é uma classe que contém os polimorfismos de sobrecarga e de coerção.

Polimorfismo de sobrecarga é aquele em que se usa o tipo e escolhe a definição a ser aplicada. O operador "+", por exemplo, pode ser aplicado entre valores inteiros, reais e, em algumas linguagens, entre cadeias de caracteres (*strings*). Um exemplo pode ser visto na

declaração abaixo:

```
int x = a + b;
```

Como o tipo `int` foi usado, diz-se que o operador "`+`" trabalha com inteiros. Porém, este operador também poderia ser aplicado ao tipo `float` (números reais). Por possuir definições diferentes para tipos diferentes, diz-se que o operador `+` foi sobrecarregado. É comum também ler as expressões "Polimorfismo de sobrecarga" e *overloading*.

O polimorfismo de coerção é aquele em que há conversão explícita do tipo. Nele usa-se a definição e se escolhe o tipo de conversão. O trecho de código a seguir, em linguagem C, exemplifica o uso do polimorfismo de coerção:

```
double x = 2.2 + (double) 2;
```

O trecho de código mostra que o valor inteiro `2` está sendo explicitamente convertido para um número em ponto flutuante com dupla precisão (`double`) antes de ser somado com `2.2`.

Coerção

Também chamada de *casting*, a coerção é, Segundo Moura (s.d.), a conversão implícita de valores de um tipo para outro. Exemplos:

- Converter um valor inteiro para real, como `-2` para `-2.0`.
- Converter uma string para uma lista de caracteres, como `"IFC"` para `['I', 'F', 'C']`.
- Converter valores inteiros para *booleanos* (lógicos), como `1` para `true` (verdadeiro).

Observando as conversões entre tipos citadas, é natural pensar que há tipos cujo conjunto de valores está dentro de outro(s) tipo(s). Por exemplo, se o valor inteiro `0` (zero) for mapeado como falso, e o inteiro `1` (um) como verdadeiro, é possível definir o subtipo lógico ou *booleano* a partir do tipo inteiro, pois ele está contido no tipo inteiro. Em C, por exemplo, é comum ver essa relação entre os tipos inteiros e os tipos *booleanos*. Os valores *booleanos* são representados como inteiros, onde `0` (zero) é considerado falso e qualquer valor diferente de zero é considerado verdadeiro.

Se em uma determinada implementação de uma linguagem de programação o tipo inteiro (`int`) for representado por 32 *bits*, assumimos que ele representa valores entre -2^{31} e $2^{31}-1$. Se, nessa mesma implementação da linguagem, o tipo `short` (ou `short int`) for capaz de representar valores entre -2^{15} e $2^{15}-1$, então o tipo `short` é um subtipo de `int`. Além disso, se um tipo `bool` (lógico) for representado pelos valores `0` e `1`, ele pode ser considerado um subtipo de `short`.

No exemplo dado, se uma variável do tipo `int` receber um valor `short` ou `bool`, diz-se que esta conversão é segura, pois sabe-se que o valor atribuído é sempre um inteiro. Por outro lado, se uma variável do tipo `bool` receber um valor `int`, é necessário verificar em tempo de execução se o valor recebido é do tipo `bool`.

As coerções realizadas pelo sistema de tipos da linguagem JavaScript são bastante peculiares. A seguir estão alguns exemplos a serem testados na ferramenta `Console` do navegador. No caso do Google Chrome, tecle F12 e clique na aba `Console`.

```
true+9
true+10-8*"Ricardo"
true+10-8*"9"
true+10-8*"9" == -61
true+10-8*"9" === -61
typeof "9"
typeof true
NaN == NaN
typeof NaN
```

Para começar a perceber as diferenças entre os sistemas de tipos de diferentes linguagens, pode-se tentar realizar as ações em outras linguagens. Em Python, por exemplo, o terceiro exemplo de JavaScript seria executado da seguinte forma na ferramenta de linha de comando **`python`**:

```
True+10-8*"9"
```

Observação: foi necessário apenas trocar `true` por `True` porque o valor é representado desta forma em Python.

Um exemplo de coerção em PHP está disponível no quadro a seguir. Sugere-se executar o código em https://www.w3schools.com/php/phptryit.asp?filename=tryphp_compiler.

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <title>Sistema de tipos - PHP</title>
    <meta charset="utf-8">
  </head>
  <body>
  <?php
$message1 = "PHP" . " possui tipagem fraca";
echo $message1;
echo "<br>";
$message2 = true + 2;
$message3 = $message2 . " é igual a 3";
echo $message3;
echo "<br>";
echo $message2 + "90";
echo "<br>";
$message3 = 1 + 2 == "3";
echo $message3;
  ?>
  </body>
</html>
```

Classificações

Os sistemas de tipos são tipicamente classificados com base em dois aspectos principais: quanto ao momento em que a verificação de tipos ocorre e quanto à flexibilidade nas conversões entre tipos permitidas. Essas classificações incluem a tipagem estática/dinâmica, que se refere ao momento em que os tipos são verificados (em tempo de compilação ou execução), e a tipagem forte/fraca, que indica a rigidez das regras que governam as operações

entre tipos diferentes.

Tipagem Estática

Em sistemas de tipos com tipagem estática a escolha do tipo ocorre em tempo de compilação, ou seja, estaticamente. A vinculação do tipo pode ser feita de forma explícita ou implícita.

- Explícita: especifica a variável informando o tipo dela.
    - Exemplo em C: `int a, b, c;`
- Implícita: associa o recurso (por exemplo, a variável) a um tipo na primeira vez que ele aparece no código, como se fosse a sua declaração. Isso é feito com base em uma convenção, por exemplo, de nomenclatura. Neste caso, o compilador ou interpretador vincula uma variável a um tipo baseado na forma sintática do nome da variável.
    - Em Fortran, um identificador não declarado explicitamente e que aparece em um programa acaba sendo declarado implicitamente de acordo com a seguinte convenção: Se o identificador começar com uma das letras I, J, K, L, M ou N, ou suas versões em minúsculas, é implicitamente declarado como sendo do tipo `Integer`; caso contrário, será declarado implicitamente como sendo do tipo `Real` (SEBESTA, 2012).

As declarações implícitas para variáveis em linguagens estaticamente tipadas foram praticamente abandonadas. No entanto, ainda são vistas em declarações de funções em linguagens como C. Um exemplo estão disponível no quadro abaixo e no arquivo declaracao-implicita.c:

```
#include <stdio.h>

//void fun(); // descomente esta linha!
int main() {

      fun();

      return 0;
}
```

```
void fun() {
      printf("Apenas um comando de escrita\n");
}
```

O código pode ser compilado com o comando:
`gcc declaracao-implicita.c -o decl`

O código pode ser executado com o comando:
`./decl`

Ao compilar o código com a terceira linha comentada, o compilador `gcc` exibe um aviso (*warning*) de declaração implícita. Como diz a definição de declaração implícita, o recurso (a função) é associado a um tipo convencionado por padrão, que, neste caso, é o tipo `int`. Embora C seja estaticamente tipada, o compilador consegue executar a função porque o endereço dela permanece o mesmo. No entanto, seu comportamento é indefinido e em compiladores mais antigos o programa pode travar. Descomentando a linha em questão, o código é compilado sem avisos.

É importante ressaltar que comportamentos inesperados prejudicam a confiabilidade do programa, pois tornam os erros difíceis de diagnosticar. Uma boa prática para impedir que um código em C seja compilado com declarações implícitas é incluir o parâmetro `-Werror=implicit-function-declaration`, como no comando a seguir:
`gcc declaracao-implicita.c -Werror=implicit-function-declaration -o decl`

Declarações implícitas são vistas em linguagens de tipagem **dinâmica**. Exemplo em Python (pode ser testado diretamente no interpretador interativo `python`):
`x = 5`

Exemplos de linguagens com tipagem estática:

- C
- C++

- Haskell
- Java

Tipagem Dinâmica

Na tipagem dinâmica a vinculação do recurso com seu tipo não é feita por uma instrução de declaração e a escolha do tipo ocorre em tempo de execução. Isso permite, por exemplo, que um mesmo identificador referencie um valor inteiro em um dado momento, depois receba uma cadeia de caracteres (*string*) sem que um erro seja produzido. Portanto, o tipo da variável pode mudar muitas vezes durante a execução do programa.

Exemplo em Python (pode ser testado diretamente no interpretador interativo `python`):

```
x = "Nome"
x
```

Isso retorna:

```
'Nome'
```

Continuando o exemplo:

```
x = 19 * 9.8
x
```

Retorna:

```
186.20000000000002
```

Continuando:

```
x = 19 + 1
x
```

Retorna:

```
20
```

Exemplos de linguagens com tipagem dinâmica:

- JavaScript
- Lisp
- PHP
- Python
- Ruby

Tipagem Forte

Em sistemas de tipos com tipagem forte, restrições impedem operações com resultados anômalos, ou seja, a violação às regras de um tipo resulta em um erro. Um exemplo pode ser o uso de uma operação que o tipo não reconhece, tal como ocorre em C#, que não permite que ocorra uma operação de soma entre `string` e `int`. Esse tipo de operação só pode ser realizada por conversões explícitas.

Exemplos de linguagens com tipagem forte:

- C++
- C#
- Haskell
- Perl
- Python
- Ruby

Tipagem Fraca

Sistemas de tipos com tipagem fraca permitem operações com tipos diferentes em situações de imprevisibilidade, ou seja, sem uma definição de que comportamento ocorrerá. Neles, ocorrem conversões implícitas de tipos.

Exemplos de linguagens com tipagem fraca:

- C
- MATLAB
- JavaScript
- PHP

Observação: a classificação de linguagens em forte/fraca não é algo totalmente aceito na literatura.

Inferência de Tipo

Outra forma de declaração implícita de tipos ocorre por meio da inferência de tipos. Neste caso, o sistema de tipos é capaz de adivinhar, com base no contexto, o tipo de dados. Ou seja, se há uma instrução em que uma variável ainda não declarada explicitamente e não associada a um tipo recebe um valor, o contexto é o tipo do valor atribuído à variável. Este recurso é utilizado por sistemas de tipos estáticos, ou seja, não é possível alterar o tipo do valor atribuído à uma variável depois (como pode ser feito em Python, por exemplo).

Exemplos de linguagens com o recurso de inferência de tipo:

- C#
- ML
- Go
- Kotlin
- Haskell

Um exemplo em C# está disponível no quadro abaixo (SEBESTA, 2012):

```
var sum = 0;
var total = 0.0;
var name = "Fred";
```

Neste exemplo, `sum` é um inteiro (tipo `int`), total é um valor de ponto flutuante (tipo `float`) e name é uma cadeia de caracteres (tipo `string`).

Códigos

Os *links* a seguir levam a códigos que exemplificam as características dos sistemas de tipos das linguagens de programação:

- Tipos.c

- Tipos.cs
- Tipos.java
- Tipos.js
- Tipos.py
- TiposZoados.java

Conclusão

O estudo dos Sistemas de Tipos revela a forma com que os tipos são estruturados nas linguagens de programação. Esses sistemas definem regras e características que atribuem tipos a diversos elementos da linguagem, como condições, expressões, variáveis e estruturas de dados. Sua principal função é garantir a consistência e prevenir erros nas operações entre esses elementos.

Foram abordadas formas de classificação dos Sistemas de Tipos. Quanto à flexibilidade e capacidade de manipular diferentes tipos de dados, o sistema pode ser monomórfico ou polimórfico. A tipagem também pode ser classificada em forte ou fraca, referindo-se à restrição ou flexibilidade na realização de operações entre diferentes tipos de dados. Há também a distinção entre tipagem dinâmica e estática, que diz respeito ao momento em que a vinculação entre variáveis e tipos ocorre.

O material também abordou aspectos como o polimorfismo de sobrecarga e de coerção. Esses conceitos são fundamentais para entender o comportamento de linguagens de programação e criar códigos eficientes e robustos.

Tarefas

1) (POSCOMP 2016, questão 28) Assinale a alternativa que apresenta o nome de uma linguagem de tipagem dinâmica.

a) Java.
b) C.
c) Python.

d) Pascal.
e) C#.

2) (POSCOMP 2018, questão 38) Sobre tipos de dados, é correto afirmar que:
a) Tipos booleanos são valores que são mantidos fixos pelo compilador.
b) O double é um tipo inteiro duplo com menor precisão do que o tipo inteiro.
c) A faixa de valores dos tipos inteiros tem somente dois elementos: um para verdadeiro e outro para falso.
d) Uma conversão de tipos implícita consiste em uma modificação do tipo de dados executado, automaticamente, pelo compilador.
e) Vetores, matrizes e ponteiros são exemplos de tipos de dados primitivos (básicos).

3) Resolva os exercícios da Lista #1.

Referências

MOURA, Hermano Perrelli de. Sistemas de Tipos. Paradigmas de Linguagens Computacionais. Material didático. [S.d.]. Disponível em: https://www.cin.ufpe.br/~if686/aulas/tipos.ppt. Acesso em: 22 fev. 2024.
SEBESTA, Robert W. Concepts of programming languages. Pearson. 10th ed. 2012.