Linguagens de Imperativa – Parte 1: variáveis

Prof. Arnaldo Candido Junior

UNESP – IBILCE

São José do Rio Preto, SP

Introdução

- As primeiras linguagens versões de linguagens imperativas não eram estruturadas
  - Sem subprogramas (funções e procedimentos)
  - Programação feita utilizando comandos GoTo (não é uma boa prática)
  - Exemplos: primeiras edições de Basic e Fortran
- Linguagens imperativas modernas: são **estruturadas**

Introdução (2)

- Linguagens imperativas são usadas até hoje
  - Sistemas embarcados
  - Softwares de grande porte como Kernels
    - Ordens de magnitude maior que sistemas comerciais populares
  - Principal representante hoje: linguagem C

Introdução (3)

- **Foco**: questões de projeto e recursos das linguagens linguagens imperativas estruturadas
- Recursos e estruturadas vistos neste material
  - Também aplicáveis aos outros paradigmas que estudaremos

Questões de Projeto

- 1 Identificadores são sensíveis ao caso ou não?
  - Obs: Identificadores são nomes de variáveis, funções, classes, etc
- 2 Qual a regra para criar identificadores?
  - É comum em linguagens modernas que comecem por letra ou “_”
  - Seguidos por letras, números ou “_”

Questões de Projeto (2)

- 3 Quais são as palavras reservadas?
  - Exemplos: if, while, class, etc
  - Aquelas que não podem ser usadas como nomes de variáveis, funções, etc

Variáveis

- Formalmente, variáveis são definidas por uma sêxtupla contando:
  - 1 Nome
  - 2 Endereço
  - 3 Tipo
  - 4 Valor
  - 5 Tempo de vida
  - 6 Escopo

Variáveis (2)

- **Nome**: é a forma de se identificar a variável no programa
- **Endereço**: e a posição de memória ocupada
- **Tipo**: define quais valores são permitidos para a variável em questão. Exemplo: inteiro, string, etc
- **Valor**: é o valor armazenado na memória em um dado instante do tempo. Exemplo: 3, “três”, 3.14, ‘3’, etc

Variáveis (3)

- **Tempo de vida** (lifetime): quanto tempo a variável fica na memória antes de ser desalocada?
- **Escopo**: onde a variável é visível.
- **Importante**: tempo de vida != escopo
  - A variável não fica visível durante todo o tempo.
  - Exemplo: uma função chamada não tem acesso as varáveis locais da função que a chama

Vinculação

- **Vinculação**: na prática, associação entre duas coisas. Exemplo em Java:
  - count = count + 5
- **Tipo**: int. Vinculado a variável durante a compilação do programa
- **Possíveis valores**: 2^{-32} até $2^{32} – 1$. Vinculado no projeto da linguagem

Vinculação (2)

- **Comando do operador** “+”: soma. Vinculada em tempo de compilação (também representa concatenação em strings)
- **Representação do literal** 5: 0101. Vinculado em tempo de compilação (número binário)
- **Valor da variável** count. Vinculado em tempo de execução

Vinculação (3)

- Vinculação de tipos
  - **Tipagem estática**: em tempo de compilação, antes do programa rodar
  - **Tipagem dinâmica**: em tempo de execução, após o programa rodar

Escopo

- Escopo estático: visibilidade da variável é conhecida em tempo de compilação
- Escopo dinâmico: visibilidade da variável é conhecida só em tempo de execução.

Escopo (2)

- Exemplo: escopo dinâmico em JavaScript:

```
function f() {
  var y = x;
  var z = 3;
}
```

- A variável *x* precisa estar definida no momento f é chamada, mas pode ser definida em diferentes locais
- Obs: não confundir escopo dinâmico com variável dinâmica (a seguir)

Escopo (3)

- Variáveis estáticas: ficam na memória durante toda a execução do programa. Exemplo: variáveis globais
- Variáveis dinâmicas em pilha: ficam na pilha até a variável sair de escopo. Exemplo: variáveis locais
- Variáveis dinâmicas em heap: ficam no heap até serem desalocadas. Exemplo em C: malloc

Blocos e variáveis

- Blocos são sequências de comandos que compartilham o mesmo escopo
- Exemplo em C: definidos entre “{“ e “}”
- Normalmente combinados com estruturas de fluxo (if, while, função, etc)
- Mas podem aparecer independentemente dessas estruturas

Blocos e variáveis (2)

- Podem ser aninhados e podem ter suas próprias variáveis locais
- Em algumas linguagens (ex.: C89): variáveis declaradas no início do bloco
- Em outras linguagens (ex.: C99): variáveis podem ser declaradas em qualquer local
- Ambiente de referência: todas as variáveis que são visíveis em um determinado bloco

Tipos de dados

- Definição: um tipo é definido por uma coleção de valores e operações sobre esses valores
- Tipos primitivos (numéricos, booleanos, string, ...)
- Matrizes (arrays)
- Registros (structs)
- Tuplas e listas
- Uniões

Tipos primitivos

- Tipos numéricos (inteiro, ponto flutuante, complexo, ...)
- Booleanos (ou booliano ou lógico)
- Caractere
- Strings (cadeias de caracteres)

Tipos numéricos

- Inteiros
- Ponto flutuante
- Complexo (ex.: Python)
- Decimal
- Existem outros (ex.: racional em Lisp)

Tipos numéricos: inteiros

- Exemplo em C: possuem vários tamanhos
- Estão sujeitos a estouro (overflow)
- Na maioria das linguagens: trade-off custo pesa mais que trade-off confiabilidade
  - Não são feitos checagens de overflow

Tipos numéricos: inteiros (2)

- Podem ser de dois subtipos: signed (positivos e negativos) ou unsiged (só positivos)
- Tipo unsiged: um bit representa o sinal.
  - Conversão de número positivo para negativo: complemento de dois
  - Passo 1: inverter bits
  - Passo 2: somar um

Tipos numéricos: inteiros (3)

- 8 bits: (unsigned) char (representa um byte)
- 16 bits: (unsigned) short
- 32 bits: (unsigned) int
- 64 bits: (unsigned) long
- 128 bits: long long
- **Obs**: algumas linguagens antigas oferecem o tipo nibble (meio byte)

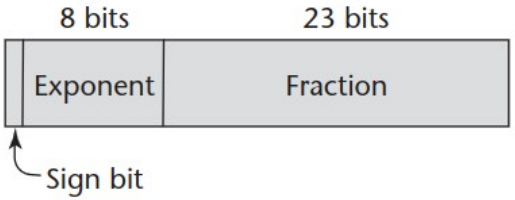
Tipos numéricos: inteiros (4)

- Exemplo para o tipo char em C

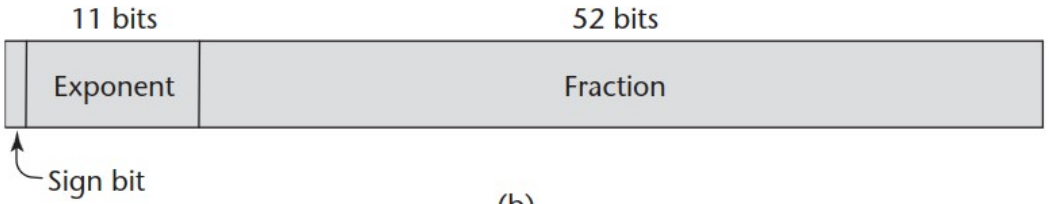
```
0b 0000 0000:    0
0b 0000 0001:    1
0b 0000 0010:    2
0b 0000 0011:    3
(…)
0b 0111 1111:  127
0b 1000 0000: -128
0b 1000 0001: -127
0b 1000 0010: -126
(…)
0b 1111 1111:   -1
```

Tipos numéricos: ponto flutuante

- Seguem o padrão IEEE Floating-Point Standard 754: 32 ou 64 bits
- Três partes: sinal, expoente e mantissa (fração)



(a)



(b)

Tipos numéricos: complexo e decimal

- Complexo: comum em linguagens para aplicações científicas. Exemplo Python: x = 0 + 1j; x*x = -1
- Decimal: comum em linguagens para aplicações comerciais. Exemplo: Visual Basic
  - Não sobre perda de precisão quando
  - Também chamado de tipo Moeada

Tipo booleano

- Assuem os valores verdadeiro ou falso. As vezes assuem terceiro valor as vezes: nulo (ex.: Python, Linguagens SQL)
- Em C tratado como um tipo numérico (falso = 0; verdadeiro != 0)
- Em Java: variáveis booleanas são agrupadas quando possível
  - Em uma situação ideal, 8 variáveis ocupam um único byte (um bit por variável)

Tipo Caractere

- Representa um único caractere
- Linguagem C: originalmente pensada para ASCII
  - Possuía limitações para manipular caracteres acentuados
  - Biblioteca recomendada: wchar.h
- Idealmente, terminal e arquivos fontes devem usar a mesma **codificação** (a seguir...)

Tipo Caractere: ASCII

- **ASCII** (American Standard Code for Information Interchange) ou ISO-IR-006
  - Alfabeto latino, números arábicos, símbolos especiais e pontuações
  - Cada caracter ocupa um byte e o primeiro bit deve ser zero
  - Valores maiores que 127 são reservados para as aplicações
  - https://en.wikipedia.org/wiki/ASCII

Tipo Caractere: ISO

- **ISO/IEC** ou **ECMA-94**: família de codificações para diversos alfabetos
- **ISO-8859-1** (Windows ANSI, Latin 1): caracteres latinos com acentos e especiais (ex.: cedilha)
- ISO-8859-7: inclui caracteres gregos
- ISO-8859-8: inclui caracteres hebraicos
- ISO-8859-15: Latin-1 com caractere para o Euro
- Entre outros

Tipo Caractere: ISO (2)

- Caracteres ocupam um byte
- ISO é superset do ASCII
  - Um arquivo ASCII pode ser corretamente visualizado como ISO
  - Mas um arquivo ISO não vai, necessariamente, ser corretamente visualizado como ASCII

Tipo Caractere: Unicode

- **Padrão Unicode**: proposto para substituir o padrão ISO
- Representa a grande maioria dos alfabetos (latino, árabe, grego, ...), silabários (hiragana, katakana) e ideogramas (kanji)
- Inclusive de línguas mortas como hieroglifos (egípcio arcaico) e cuneiforme (sumério arcaico):
  - 

Tipo Caractere: Unicode (2)

- Contém muitos caracteres especiais, incluindo emojis/emoticons coloridos:
- Na versão 15, contém ~150.000 caracteres
- Cade caractere contém um identificador único, chamado de **CodePoint**
  - Que pode ser representado em disco de diferentes maneiras (detalhes a seguir)

Tipo Caractere: Unicode (3)

- UTF-7: representa qualquer CodePoint como uma sequência de caracteres ASCII
  - Usado principalmente em e-mails
  - Combinado com Base64 para representar dados binários

Tipo Caractere: Unicode (4)

- **UTF-8**: principal codificação hoje em dia, amplamente usada na Web
- Caracteres ocupam de 1 a 4 bytes
- Também é um superset ASCII:
  - Por definição qualquer arquivo ASCII é ISO-8859-1...
  - … e UTF-8 ao mesmo tempo!

Tipo Caractere: Unicode (5)

- Codepoints em UTF-8

First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
U+0000	U+007F	0xxxxxxx			
U+0080	U+07FF	110xxxxx	10xxxxxx		
U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx	
U+10000	U+10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

Tipo Caractere: Unicode (6)

- UTF-16: a maioria dos caracteres ocupam 2 bytes, embora alguns ocupem 4
  - LE (little endiam): byte mais significativo à esquerda
  - BE (big endiam): byte mais significativo à direita
- UCS-32: usa os próprios CodePoints, ocupando sempre 4 bytes

Tipo Caractere: Unicode (7)

Caracter	UTF-8	UTF-16	UCS-32
'A'	0x41	0x0041	0x00000041
'z'	0x7A	0x007A	0x0000007A
'ç'	0xC3 0xA7	0x00E7	0x000000E7
'α'	0xCE 0xB1	0x03B1	0x000003B1
' '	0xF0 0x9F 0x98 0x80	0xD83D 0xDE00	0x0001F600

Tipo string

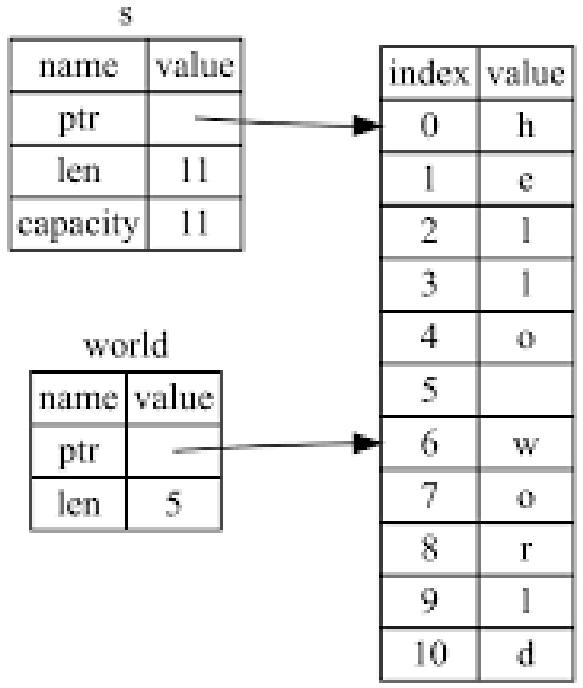
- Strings são sequências de caracteres
- Linguagens modernas tendem a usar UTF-8 (ou Unicode em geral) em strings e caracteres
- Quebras de linha: variam conforme S.O.
  - “\n” (0a): Unix e derivados
  - “\r” (0d): MacOS
  - “\r\n” (0d 0a): Windows

Tipo string (2)

- Linguagens fornem quebras de linha multiplataforma. Exemplo Java:
  - System.lineSeparator() e System.out.println()
- Algumas linguagens, como Python, permitem UTF-8 inclusive para declarar variáveis
  - Boa prática: código compatível com ASCII (unicode só em comentários, strings e chars)

Tipo string (3)

- Fatiamento (slices): fragmentos de strings
  - Uso em Python
    s = “hello world”
    world = s[6:11]
  - Representação Rust:



Tipo string: segurança em C

- Em C, strings são terminadas pelo caracter nulo (\0, ASCII 0)
- C privilegia trade-off custo sobre confiabilidade
- Historicamente, isso possibilitou muitos ataques, inclusive em servidores da Internet

Tipo string: segurança em C (2)

- Extravasamento de buffer ou estouro de buffer
  - Fornecer mais caracteres que o código está preparado para ler
  - Pode inclusive permitir a injeção de código
    - Ao sobrescrever o return address (ver tópico 01, slide 17)

Tipo ordinal: enumerações

- Permitem associar uma relação de ordem a valores constantes
- Normalmente, facilmente convertas de/para inteiros
- Úteis para deixar o código mais legível
- Exemplo em C:

```
enum dias {dom, seg, ter, qua, qui, sex, sab};
```

Matrizes

- Contém elementos do mesmo tipo (geralmente)
  - Principalmente nas linguagens estaticamente/fortemente tipadas:
  - Por isso, também chamadas de: tipo de dado composto homogêneo
  - Linguagens modernas: oferecem slices (fatiamento), como em strings

Matrizes (2)

- Matriz na computação: aquilo que, na matemática é chamado de tensor
  - vetor, matriz, tensor de ordem 3, ordem 4, etc
- Matriz na matemática: aquilo que na computação é chamado de matriz bidimensional. Ex.:

```
M = [[10, 20, 30],
     [40, 50, 60]]
```

Matrizes (3)

- Matrizes são agrupadas em dois grandes grupos
  - Retangulares: são como as matrizes matemáticas. Ex.:

```
M = [[10, 20, 30],
     [40, 50, 60]]
```

  - Irregulares (jagged). Ex.:

```
M = [[10, 20, 30],
     [40, 50]]
```

Matrizes (4)

- Matriz de tamanho fixo: não mudam de tamanho uma vez alocadas:
- Matrizes dinâmicas: podem mudar de tamanho
  - Quando falta espaço em uma região de memória, são copiadas para outra
  - Em C: feito com realloc
  - Em Java: automaticamente pelas ArrayLists

Matrizes em C

- Strings e matrizes bidimensionais não são ortogonais
  - Vetores de palavras é uma matrizes de caracteres são representados da mesma forma
- Ponteiros e matrizes 2D também não são ortogonais
  - M[3] é um ponteiro que aponta para a quarta linha de uma matriz 2D

Matrizes em C (2)

- Acesso a índices: a maioria das linguagens favorece o trade-off confiabilidade sobre custo
  - Índices de arrays são verificados em tempo de execução para evitar acessos indevidos
  - Em java: OutOfBoundsException
  - Em: C favorece custo, reduzindo a confiabilidade dos programas

Matrizes associativas

- Matrizes cujos índices não são do tipo inteiro (ex.: índices do tipo string)
- Dependem de uma função de hash (espalhamento)
- Também chamadas de:
  - Hash (ex.: HashMap em Java)
  - Dicionários (ex.: Dict em Python)
  - Tabelas associativas (as vezes em SQL)

Matrizes associativas (2)

- Exemplo em Python

```
rgb = {"vermelho": "F00",
       "azul": "00F",
       "amarelo": "FF0",
       "cinza": "777"}
rgb["amarelo"]
```

Registros

- Também chamados de: tipo de dados composto heterogêneo
- Contém valores de diferentes tipos
- Em C: structs

```
struct pessoa {
  char nome[50];
  int cpf;
  double salario;
};
```

Tuplas

- Semelhantes à registros, mas valores não são nomeados
- Deestruturação: uma forma conveniente de acessar elementos de uma tupla
- Obs: em linguagens dinamicamente tipadas, tuplas, listas e matrizes tendem a ser essencialmente o mesmo conceito
  - Exemplo a seguir

Tuplas (2)

- Em Python existem duas versões:
  - Tuple: imutável
    cor = (“azul”, 0.0, 0.0, 16.0)
  - List: mutável
    cor = [“azul”, 0.0, 0.0, 16.0]
- Deestruturando
  (nome, valor_r, valor_r, valor_b) = cor
- Acessando elmento: cor[0]

Tuplas (3)

- Em Rust, tuplas podem ser nomeadas
  - struct Pair(i32, f32);
  - struct Point(i32, f32);
  - Obs: uma variável do tipo Point não pode ser atribuida a um tipo Pair, mesmo que são membros são os mesmos

Listas

- Conceito importante linguagens funcionais
- Tipo nativo da linguagem
- Listas podem ser aninhadas
- Exemplo Lisp:
  (A (B C) D)

Uniões

- Podem guardar diferentes tipos de valores, apenas um por vez
- Vantagem: economia de espaço; desvantagem: propenso a erros
- Pouco usado nas linguagens modernas

Uniões (2)

- Linguagem Ceylon: usado para passagem de parâmetros
- Exemplo em C:

```
union Numero {
    int i;
    float f;
    double d;
};
```

Ponteiros

- Acesso indireto a valores em outras posições de memória
- Ponteiros são muito propensos a erros
- Dangling Pointers (ponteiro solto; ponteiro selvagem)
  - Apontam a posições ainda não alocadas ou cuja a aloção já foi liberada
- Na prática, só usado em linguagens de médio nível (C, Zig), kernels e sistemas embardados

Referências

- Referências: alternativa aos ponteiros
- Uma forma mais controlada de acessar posições de memória de forma indireta.
- Exemplo: Rust