UFRPE
Universidade
Federal Rural
de Pernambuco

Paradigmas de Programação

Variáveis

Kellyton Brito
kellyton.brito@gmail.com
31/10 e 07/11/2017

Diferença entre os grupos de paradigmas

- **Programação descritiva**
  - Foco no COMO é executado
  - Baseado em uma ordem de execução
  - Baseado em:
    - Estados,
    - Variáveis,
    - Atribuições,
    - Contexto,
    - Efeitos colaterais e
    - Ordem de execução
- **Programação declarativa**
  - Foco no QUÊ é executado
  - Programa lido e executado todo de uma vez
  - Não existe conceitos tradicionais de: Estados, Variáveis, Atribuições, Contexto, Efeitos colaterais e Ordem de execução

O que é uma variável?

Uma variável é uma abstração para uma célula de memória

Introdução

- Linguagens como abstrações para a arquitetura de von Neumann (processador e memória)

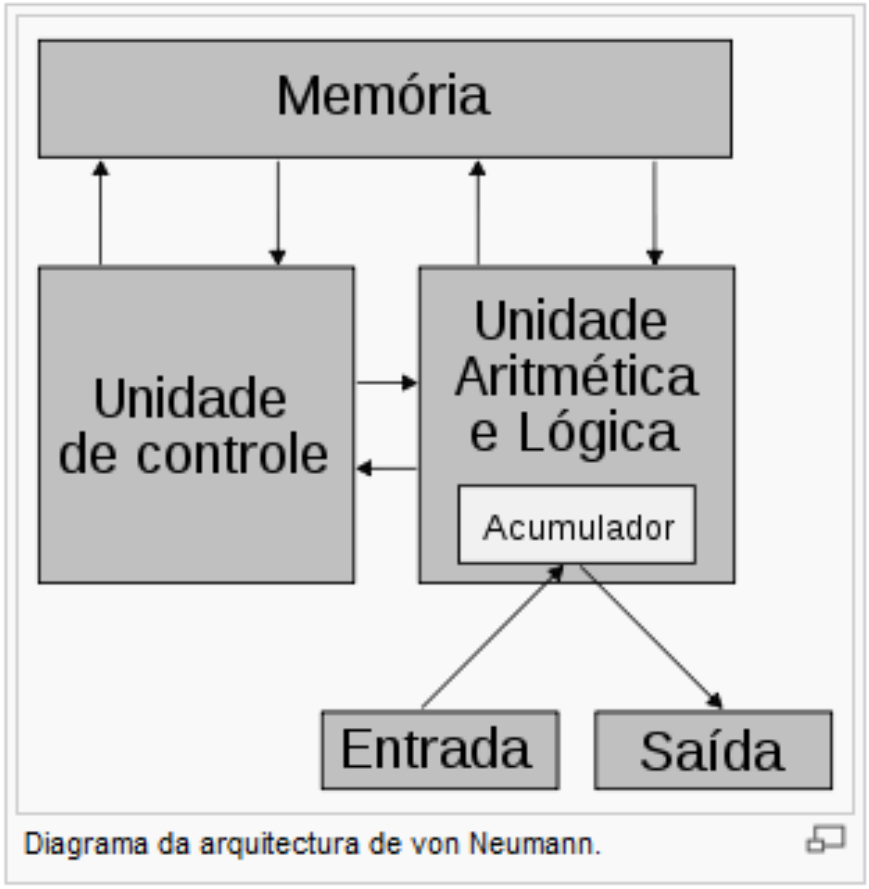


Diagrama da arquitectura de von Neumann.

Introdução

- As ***variáveis*** são ***<u>abstrações</u>*** para as células de memória

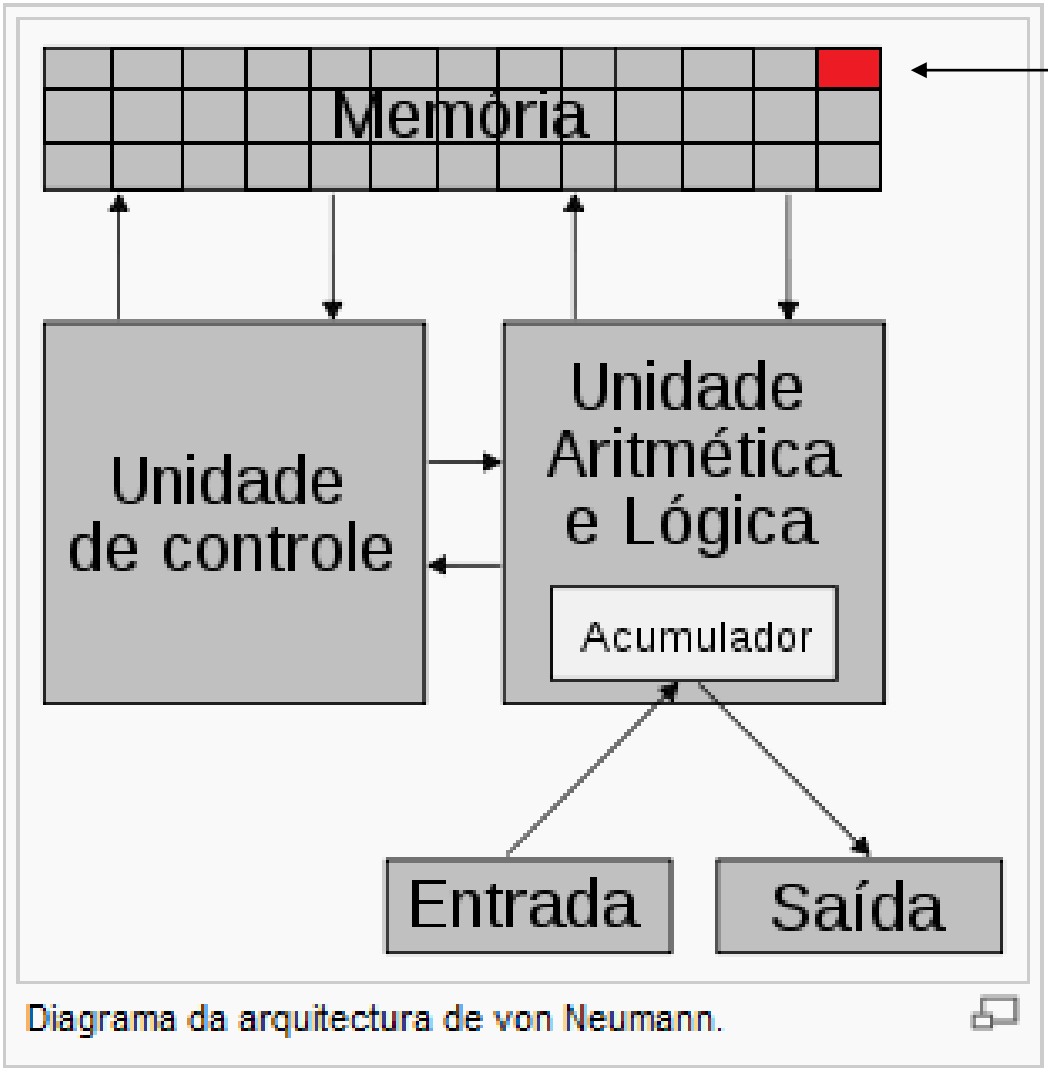


Diagrama da arquitectura de von Neumann.

Introdução

- Possuem um conjunto de propriedades / características
  - Nome
  - Endereço
  - Tipo
  - Valor
  - Escopo
  - Tempo de vida

Variáveis

- Uma abstração de uma célula ou conjunto de células de memória
- É um conjunto de atributos
  - **Nome**
  - Endereço
  - Tipo
  - Valor
  - Escopo
  - Tempo de vida

Nomes ou Identificadores

- Identificam as variáveis, labels, subprogramas e parâmetros, dentre outros.
- Normalmente reconhecidos como um **id** pelo analizador léxico
- Variações quanto a forma:
  - Limites de caracteres
  - Case Sensitive: Diferença entre maiúsculas e minúsculas
  - Normalmente iniciados com letras e seguidos por letras, números ou underscore (_).

Nomes ou Identificadores

- Palavras especiais:
  - Palavras reservadas
    - Não podem ser usadas como nomes
  - Palavras chave
    - Palavras com significado especial
    - Toda palavra chave normalmente é uma palavra reservada
    - Nem toda palavra reservada é chave
      - Reservada apenas para se evitar seu uso
      - **goto** e **const** são reservadas, mas não têm utilidade

Nomes ou Identificadores

- Ex.: Conjunto de palavras reservadas em Java 5

abstract	continue	for	new	switch
assert (3)	default	goto (1)	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum (4)	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp (2)	volatile
const (1)	float	native	super	while

(1) Sem uso na linguagem, (2) a partir da 1.2, (3) A partir da 1.4,

(4) A partir da 1.5

Variáveis

- Uma abstração de uma célula ou conjunto de células de memória
- É um conjunto de atributos
  - Nome
  - **Endereço**
  - Tipo
  - Valor
  - Escopo
  - Tempo de vida

Variáveis

- Endereço
  - O endereço de memória inicial associado à variável
  - A posição final depende do tipo
  - Linguagens permitem aliasing:
    - Múltiplas variáveis podem ser associadas ao mesmo endereço
  - Linguagens tratam comparações de tipos referência como comparações de endereço

Variáveis

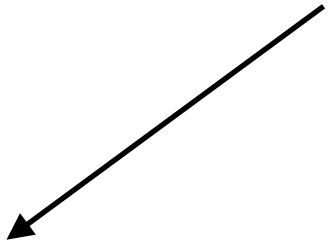
- Comparações de referência como comparações de endereço
- Ex.: O que é impresso ao executar esse código?

```
public void codigo1(){

    String a = new String("Joao");
    String b = new String("Joao");

    if (a == b){
        System.out.println("Sao iguais");
    } else {
        System.out.println("Sao diferentes");
    }

}
```



Variáveis

- Uma abstração de uma célula ou conjunto de células de memória
- É um conjunto de atributos
  - Nome
  - Endereço
  - **Tipo**
  - Valor
  - Escopo
  - Tempo de vida

Variáveis

Tipo

- Define a faixa de valores possíveis
- Define operações possíveis
- Define a quantidade de memória necessária para armazenamento

Ex.: Tipos inteiros em java

Tipo	Tamanho	Valores
byte	8 bits	-128...127
short	16 bits	-32768...32767
int	32 bits	-2147484... 2147483
long	64 bits	-9e18...9e18

Variáveis

Tipo

- Tipos primitivos ou tipos referências

- Primitivos: possuem representação numérica
  - Relação direta com os bytes
  - Normalmente tipos numéricos e char

- Referência: tipos compostos
  - Formados por combinações, composições e agrupamentos de tipos primitivos e (em alguns casos) subprogramas

Variáveis

- Uma abstração de uma célula ou conjunto de células de memória
- É um conjunto de atributos
  - Nome
  - Endereço
  - Tipo
  - **Valor**
  - Escopo
  - Tempo de vida

Variáveis

- Valor
  - O conteúdo da célula(s) de memória associada(s)

Como uma Variável é associada ao seu valor?

Conceito de Binding

- **Binding:** associação
  - Entre um atributo e sua entidade
    - variável <-> célula de memória
  - Entre uma operação e um símbolo
    - “+” <-> Operação de soma
- Bindings diversos ocorrem em momentos diferentes: “*Binding Time*”

Conceito de Binding

- Momento de Design da linguagem
  - Ex.: "*" associado à multiplicação
- Momento de Implementação da linguagem
  - Ex.: Tipos de dados primitivos: int ligado aos possiveis valores
- **Tempo de Compilação**
  - **Ex.: Uma variável ligada a um tipo: "int i;"**
- Tempo de Carga/Load
  - Ex.: Constante alocada a uma célula de memória "*public static final int ia = 2*"
- Tempo de "Linkagen"
  - Ex.: Uma chamada a um subprograma
- **Tempo de Execução**
  - **Ex.: Atribuição de valores em tempo de *execução: a = b***

Tipos de Ligações

- Estática x Dinâmica
  - Antes da execução e não muda ***versus*** pode alterar durante a execução
  - Ex.: Estática
    - Bindings de design e implementação da linguagem
    - Declaração de constantes
  - Ex. Dinâmicas
    - Declarações tradicionais. int i = 2; i = 3; i = 4;

Tipos de Ligações

- Explícitas x Implícitas
  - Declaradas pelo desenvolvedor x definidas automaticamente

- Ligações implícitas são dinâmicas
  - Declaração de acordo com a primeira ocorrência
    - PHP: *$list = 47;*
    - *Python:* *mensagem = "minha mensagem"*
      n = 17

- Inferência de Tipos:
  - Algumas linguagens inferem os tipos de funções
    - Ex.: ML (Metalanguage) infere o tipo de parâmetro e retorno em:

*fun times10 (x) = 10*x;*

Checagem de Tipos

- Processo de garantia de que operandos e operadores são compatíveis

- Considera:
  - **Funções** como operadores e os atributos como operandos
  - **Atribuição** como operadores e os lados da expressão como operandos

Checagem de Tipos

- Tipos compatíveis:
  - Se iguais ou se podem ser convertidos implicitamente
  - Linguagens convertem implicitamente tipos *menores* para tipos *maiores*
    - Ex.: Um int em um float
  - Conversões de tipos maiores para menores devem ser explícitas
    - *int i = (int)j;*

- Ligações dinâmicas apenas permitem checagem dinâmica
  - Maior flexibilidade, maior custo de correção

Compatibilidade de Tipos

- Compatibilidade
  - Nomes: Se tem o mesmo tipo declarado
    - Mais fácil de implementar
    - Mais restritiva
  - Estrutura: Se os tipos tem a mesma estrutura
    - Mais difícil de implementar / debugar
    - Mais possibilidades ao desenvolvedor

- No geral compatibilidade de estruturas é feita se um tipo é subtipo da outra
  - Objetos com herança e interfaces
  - Java+ faz operações envolvendo tipos inteiros diferentes: converte o menor para o maior.

Compatibilidade de Tipos: Exemplo

- O que acontece quando é compilado e executado esse código?

```
public void codigo2(){
    int i = 10;
    double f = 20.5;

    i = f;

    if (i == f){
        System.out.println("Sao iguais");
    } else {
        System.out.println("Sao diferentes");
    }
}
```

Erro de compilação: Não converte implicitamente um tipo maior para um menor

Variáveis

- Uma abstração de uma célula ou conjunto de células de memória
- É um conjunto de atributos
  - Nome
  - Endereço
  - Tipo
  - Valor
  - **Escopo**
  - Tempo de vida

Escopo

- Escopo é o conjunto de instruções onde a variável é visível
- Escopo Global
  - Visível por todo o programa
- Escopo Local
  - Visível dentro do método/função
- Escopo do bloco de execução
  - Visível dentro de loops ou semelhantes

Escopo: Exemplo

- Em linguagens de alto nível, o que acontece quando é compilado e executado esse código?

```
public class Nomes {

    String nome = "Maria";

    public void codigo4(String nome){
        String nome = "Jose";
        System.out.println(nome);
    }

    public static void main(String []args){
        Nomes program = new Nomes();
        program.codigo4("Pedro");
    }
```

Erro de compilação:
Duplicidade de declaração de nome dentro do método "código4"

Escopo: Exemplo

- Em Java, o que acontece quando é compilado e executado esse código?

```
public class Nomes {

    String nome = "Maria";

    public void codigo4(String nome){
                nome = "Jose";
        System.out.println(nome);
    }

    public static void main(String []args){
        Nomes program = new Nomes();
        program.codigo4("Pedro");
    }

}
```

E agora?

Variáveis

- Uma abstração de uma célula ou conjunto de células de memória
- É um conjunto de atributos
  - Nome
  - Endereço
  - Tipo
  - Valor
  - Escopo
  - **Tempo de vida**

Tempo de Vida

- O tempo que uma variável existe

- Semelhante ao escopo
  - Igual, quando não são feitas chamadas de métodos

- Durante uma chamada, o escopo muda, mas a variável continua existindo (tempo de vida)

- Variáveis “static” permanecem existindo mesmo quando o objeto não foi instanciado/criado

Tempo de Vida - Exemplo

- Qual o tempo de vida das duas variáveis ***a*** declaradas no programa abaixo?

```
public void metodo2(){
     int a = 4;
     System.out.println(a);
}

public void metodo1(){
     int a = 2;
     metodo1();
}
```

Nesse código, o escopo do método2 é o *a* interno. Porém o ***a = 2*** continua existindo, apesar de em outro escopo

Leitura até aqui:

Sebesta, Conceitos de Linguagens de Programação.

Capítulos 1 a 5

Dúvidas?