

# Fundamentos de Linguagens de Programação

## Nomes, variáveis, vinculações e escopos

Nelson Carvalho Sandes

Centro de Ciências Tecnológicas - CCT  
Universidade Federal do Cariri

2021

# Tópicos

## 1 Introdução

## 2 Nomes

## 3 Variáveis

## 4 Vinculação

- Vinculação de tipos
- Vinculação de armazenamento e tempo de vida

## 5 Escopos

# Tópicos

## 1 Introdução

## 2 Nomes

## 3 Variáveis

## 4 Vinculação

- Vinculação de tipos
- Vinculação de armazenamento e tempo de vida

## 5 Escopos

# Introdução

- Na ciência da computação, programação imperativa é um paradigma de programação em que existem estados (variáveis) que podem ser modificados através de ações.
- Uma variável é caracterizada por uma coleção de atributos (ex. nome e tipo).
- Nesse contexto, as variáveis são alocadas em **regiões da memória**.
- Em particular, iremos dar foco em duas regiões da memória: *stack* e *heap*.

# Introdução

## Memória

- De um modo informal, podemos dizer que a memória **stack** é uma região da memória que é utilizada para alocar espaço para variáveis de funções. Quando a função é finalizada, as variáveis correspondentes à ela são desalocadas da memória *stack*.
- A memória **heap** é uma região da memória que é reservada para variáveis que são alocadas e desalocadas explicitamente (na linguagem C) pelo programador.
- É importante frisar que existem outras regiões da memória responsáveis por guardar valores de variáveis globais/estáticas além do código a ser executado.

# Memória

## Stack

```
#include <stdio.h>
#include <stdlib.h>

int quadrado(int x) {
    return x*x;
}

int soma_quadrado(int a, int b) {
    int z = quadrado(a + b);
    return z;
}

int main() {
    int a = 4;
    int b = 8;
    int total = soma_quadrado(a, b);
    printf("Total: %d", total);
    return 0;
}
```

# Introdução

## Heap

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    int* p = (int*)malloc(sizeof(int));
    int* vet = (int*)malloc(3*sizeof(int));
    *p = 10;
    vet[0] = 1;
    vet[1] = 2;
    vet[2] = 3;
    printf("Vet[2] = %d \n", vet[2]);
    printf("*p = %d \n", *p);
    free(p)
    free(vet);
    return 0;
}
```





# Nomes

## Definição

- Um **nome** é uma cadeia de caracteres usada para identificar alguma entidade do programa.
- O termo **identificador** também é muito utilizado como sinônimo de nome.
- Cada linguagem de programação descreve um formato para os seus nomes.
  - Limitação de caracteres.
  - Sensibilidade à capitalização.

# Nomes

## Palavras especiais

- **Palavras especiais** são usadas para tornar os programas mais legíveis ao nomearem as ações à serem realizadas.
- Uma *palavra-chave* é uma palavra especial apenas em **alguns contextos**.
  - Em FORTRAN, a palavra *Integer*, quando encontrada no início da sentença e seguida por um nome, é considerada uma palavra-chave que indica que a sentença é declarativa. Ex:
    - Integer Apple
  - Em outro contexto, a palavra Integer não é considerada especial. Ex:
    - Integer = 4

# Nomes

## Palavras especiais

- Uma **palavra reservada** é uma palavra especial que **não** pode ser usada como nome.
- Para tornar programas menos confusos, geralmente as palavras reservadas são mais usadas do que as palavras-chave nas linguagens de programação. Por exemplo, seria possível escrever algo do tipo em FORTRAN:
  - Integer Real
  - Real Integer
- Também é preciso ter cuidado com a escolha das palavras reservadas. Por exemplo, COBOL usa palavras reservadas que são muito comuns de serem utilizadas por programadores: *LENGHT*, *COUNT* e *BOTTOM*.





# Variáveis

## Atributos das variáveis

- Tipo
  - O **tipo** determina a faixa de valores que uma variável pode assumir.
- Valor
  - O **valor** é o conteúdo da memória que está associado à variável.
- Antes de definir o escopo e tempo de vida de uma variável, é importante definir conceitos relacionados à **vinculação**.
- Após isso, voltaremos para o tópico de escopos e definiremos o tempo de vida de uma variável.



# Vinculação

## Definição

- Uma **vinculação** é uma associação, como, por exemplo, um atributo e uma entidade ou entre uma operação e um símbolo.
- Podemos dizer que uma vinculação é **estática** se ela ocorre pela primeira vez antes do tempo de execução do programa e permanece inalterada durante a execução.
- Se a vinculação ocorre em tempo de execução ou pode ser alterada ao longo da execução do programa, dizemos que a vinculação é **dinâmica**.



# Vinculação

## Vinculação de tipos

- Antes da variável ser referenciada em um programa, ela deve ser vinculada a um tipo de dados.
- A vinculação com os tipos podem ser estáticas ou dinâmicas.
- Na vinculação estática de tipos, vamos ter declarações explícitas e declarações implícitas.
- Uma **declaração explícita** é uma sentença em um programa que lista nomes de variáveis e especifica que elas são de um tipo.
- Já na **declaração implícita**, se associa tipos às variáveis por meio de convenções.

# Vinculação de tipos

## Vinculação estática

- Por exemplo, em FORTRAN é possível ter declarações tanto explícitas como implícitas. Ex:
  - Integer Apple;
- Por outro lado, se a declaração não for feita de forma explícita, é possível fazer uma vinculação de tipo de forma implícita.
- Se o identificador começar com as letras I, J, K, L, M ou N (contando com versões minúsculas), ele é declarado como Integer, caso contrário ele será considerado do tipo Real.
- Declarações implícitas podem ser prejudiciais para a confiabilidade.

# Vinculação de tipos

## Vinculação dinâmica

- Na vinculação dinâmica de tipos, a vinculação ocorre quando é atribuído algum valor para a variável.
- Por exemplo, quando fazemos " $a = 10$ " na linguagem Python, o tipo inteiro é associado à variável " $a$ " em tempo de execução.
- Isso fornece muita flexibilidade para o programador. Por outro lado, diminui a capacidade de detecção de erros.

# Vinculação de armazenamento e tempo de vida

- Para poder definir o tempo de vida de uma variável, é preciso entender o conceito de alocação e desalocação.
- A **alocação** ocorre ao se vincular uma variável à um conjunto de células disponíveis na memória.
- A **desalocação** ocorre quando as células de memória vinculadas à uma variável se tornam disponíveis novamente.
- O **tempo de vida** de uma variável se inicia quando ela é vinculada à um conjunto de células e se encerra quando ela é desvinculada.

# Vinculação de armazenamento e tempo de vida

- **Variáveis estáticas** são vinculadas à células de memória antes da execução do programa e permanecem vinculadas a essas mesmas células até o final da execução do programa.
- Variáveis globais em C são um exemplo de variáveis estáticas.
- É possível criar variáveis estáticas em C usando o especificador **static**.
- É importante não confundir com o especificador **static** no contexto de uma classe em orientação à objetos.

## Vinculação de armazenamento e tempo de vida

- As **variáveis dinâmicas na pilha** são aquelas cujas vinculações de armazenamento são criadas a partir da elaboração de suas instruções de declaração, mas cujos tipos são estaticamente vinculados (Elaboração: processo de alocação e de vinculação de armazenamento que ocorre em tempo de execução)
- As **variáveis dinâmicas no monte explícitas** são células de memória sem nome (abstratas) alocadas e desalocadas por instruções explícitas em tempo de execução, especificadas pelo programador.
- As variáveis dinâmicas no monte explícitas podem apenas ser referenciadas por ponteiros ou variáveis de referência.

# Vinculação de armazenamento e tempo de vida

- **Variáveis dinâmicas do monte implícitas** são vinculadas ao armazenamen- to no monte apenas quando são atribuídos valores a elas.
- Por exemplo, em Javascript é possível fazer a seguinte declaração:
  - *heighs* = [74, 84, 86, 90, 71];
- Apesar de ser uma abordagem flexível, ela pode dificultar a detecção de erros do código.





# Escopo

- O **escopo** de uma variável é a faixa de sentenças nas quais ela é visível.
- Uma variável é **visível** em uma sentença se ela pode ser referenciada nessa sentença.
- Uma variável é **local** a uma unidade ou bloco de programa se ela for declarada lá.
- As variáveis **não locais** de uma unidade ou bloco de programa são as variáveis que não foram declaradas nesse bloco, mas são visíveis a ele.

# Escopo

## Escopo Estático

- No **escopo estático**, o escopo das variáveis são determinados **antes** do tempo de execução do algoritmo.

```
procedure Big is
  X : Integer;
  procedure Sub1 is
    X : Integer;
    begin -- de Sub1
    ...
    end; -- de Sub1
  procedure Sub2 is
    begin -- de Sub2
    ...X...
    end; -- de Sub2
  begin -- de Big
  ...
  end; -- de Big
```

# Escopo

## Blocos

- Além dos escopos gerados por funções, também existem os escopos gerados por **blocos**.
- Por exemplo, na linguagem C, as chaves - `{}` - são utilizadas para dar início e fim à um bloco de instruções.
- Variáveis criadas nesse bloco são consideradas locais a ele. Além disso, a questão de visibilidade é a mesma em relação aos escopos de funções.

# Escopo

## Escopo Estático

```
#include <stdio.h>

int main() {
    int count = 0;
    int i = 0;
    while (i < 5) {
        int count = 0;
        count++;
        printf("%d \n", count);
        i++;
    }
    printf("----- \n");
    printf("%d \n", count);
    return 0;
}
```

## Escopo Global

- Algumas linguagens permitem a criação de variáveis fora do escopo das funções.
- Apesar disso, geralmente, essas variáveis são visíveis para as funções.
- Exemplo de linguagens que permitem isso: C, C++, Python, PHP...

# Escopo

## Escopo Dinâmico

- O **escopo dinâmico** é determinado em tempo de execução.
- Ele é baseado na sequência de chamadas das funções/procedimentos, não em seu relacionamento espaciais uns com os outros.
- Vamos considerar mais uma vez o exemplo da linguagem Ada, em que temos os procedimentos *Big*, *Sub1* e *Sub2*.
- Imaginem na execução um cenário em que *Big()* é chamado, após isso ele chama *Sub1()* e *Sub1()* chama *Sub2()*. A variável *X* utilizada em *Sub2()* estará relacionada com o *X* de *Sub1()* ou de *Big()*?

# Escopo

## Escopo Dinâmico

```
procedure Big is
  X : Integer;
  procedure Sub1 is
    X : Integer;
    begin -- de Sub1
      ...
    end; -- de Sub1
  procedure Sub2 is
    begin -- de Sub2
      ...X...
    end; -- de Sub2
  begin -- de Big
    ...
  end; -- de Big
```