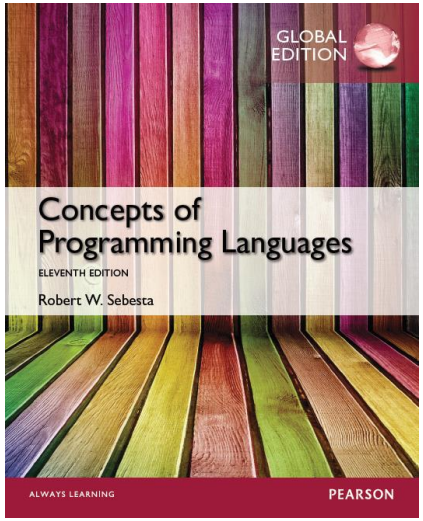
Modelos de Linguagem de Programação

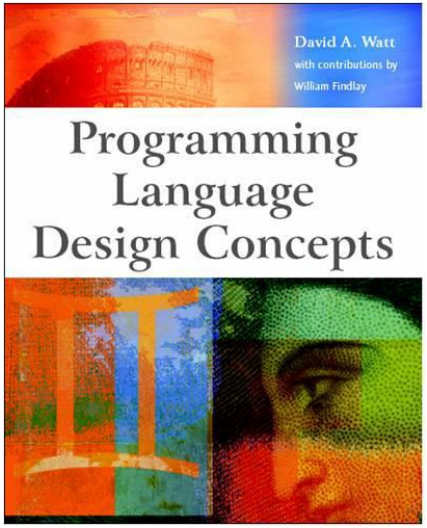
Unidade 5 - Nomes, Vinculações, Escopos e Tipos de Dados

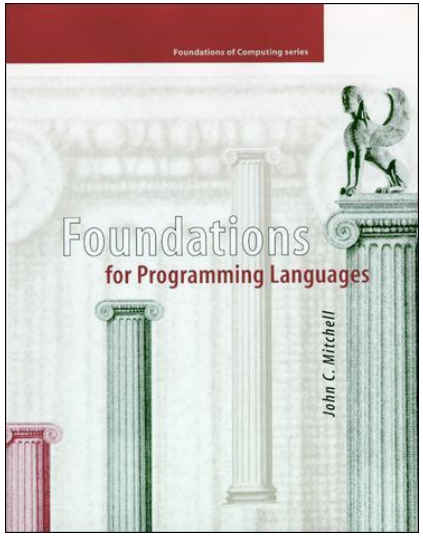
Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUSP
aparecidovfreitas@gmail.com

Bibliografia

- Sebesta, Robert W. Concepts of Programming Languages – Eleventh Edition
- Watt, D. Programming Language Design Concepts. John Wiley and Sons, 2004.
- Mitchell, J. Foundations for Programming Languages, MIT Press, 1996.

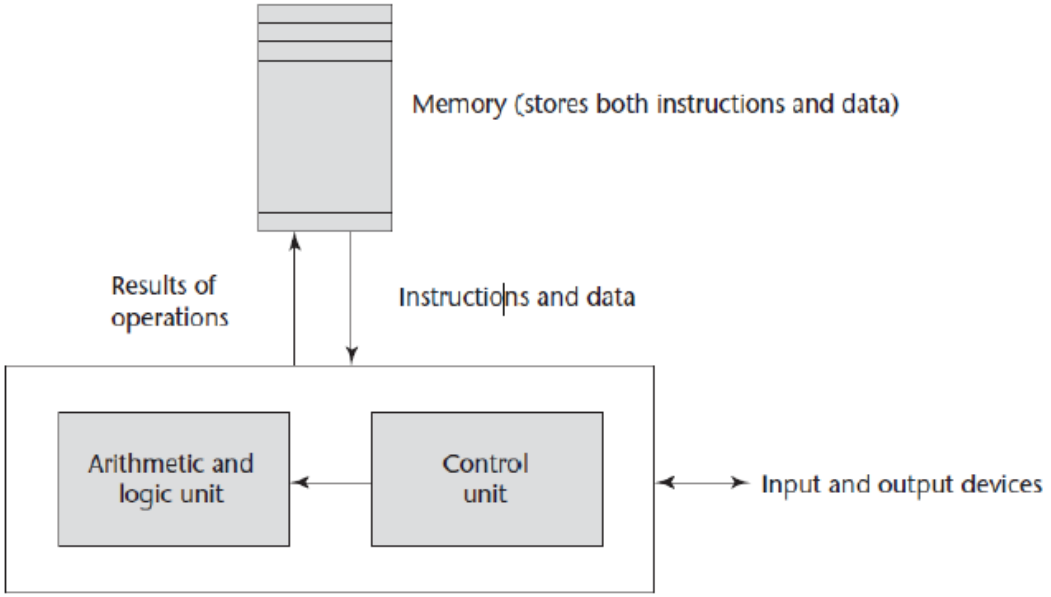






Introdução

- Linguagens de programação imperativas são abstrações da arquitetura de **Von Neumann**:
  - **Memória:** armazena instruções e dados;
  - **Processador:** fornece informações para modificar o conteúdo da memória.
    - As abstrações para as células de memória da máquina são as **variáveis**;



Nomes

- Um **nome** (ou identificador) é uma cadeia de caracteres usado para identificar alguma entidade em um programa.

- **Questões de projeto:**
  - Qual é o tamanho máximo de um nome?
  - Caracteres especiais podem ser usados em nomes?
  - Os nomes fazem distinção entre minúsculo e maiúsculo?
  - As palavras especiais são palavras reservadas ou palavras-chave?

Formato dos Nomes

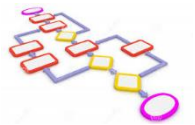
- Geralmente nomes devem iniciar com letras (a..z) ou _ (underline) e seguidos por letra _ ou dígitos.

- Em algumas linguagens os nomes são case sensitive:
  - Exemplo: iprj, Iprj e IPRJ são nomes distintos;
  - Afeta a legibilidade;

Tamanhos de Nomes

- Tamanhos para nomes estabelecidos por algumas linguagens de programação:
  - **FORTRAN**: máximo 6;
  - **COBOL**: máximo 30;
  - **FORTRAN 90 e ANSI C**: máximo 31;
  - **Ada**: sem limite*;
  - **C++**: sem limite*;

* Implementadores frequentemente estabelecem limite.



Variáveis

- Variável é uma abstração de uma **célula de memória**;
- Surgiram durante a mudança das linguagens de programação de baixo para alto nível;
- **Pode ser caracterizada por 6 atributos:**
  - **Nome:** identificador;
  - **Endereço:** localização da memória a ela associado;
  - **Tipo:** intervalo de possíveis valores e operações
  - **Valor:** o que está armazenado na variável num determinado momento;
  - **Tempo de vida:** tempo durante o qual a memória permanece alocada para a variável;
  - **Escopo:** partes do programa onde a variável é acessível;

Nomes: Palavras Reservadas

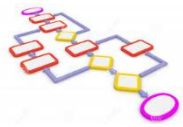
- **Palavra reservada:** palavra que não pode ser usada como um nome definido pelo usuário
  - Quanto maior a quantidade de palavra reservada, maior a dificuldade do usuário definir nomes para as variáveis

Escopo de Variáveis

```
#include <stdio.h>

int res;

int fat(int n)
{
  int f;
  if (n == 0)
    f = 1;
  else
    f = n * fat(n-1);
  return f;
}
int main(void)
{
  res = fat(3);
  printf("Fatorial = %d \n", res);
  return 0;
}
```

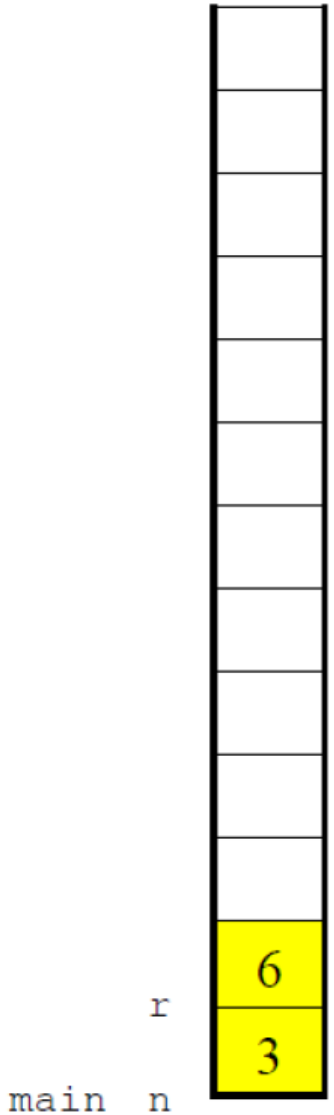
Escopo de f

Escopo de res



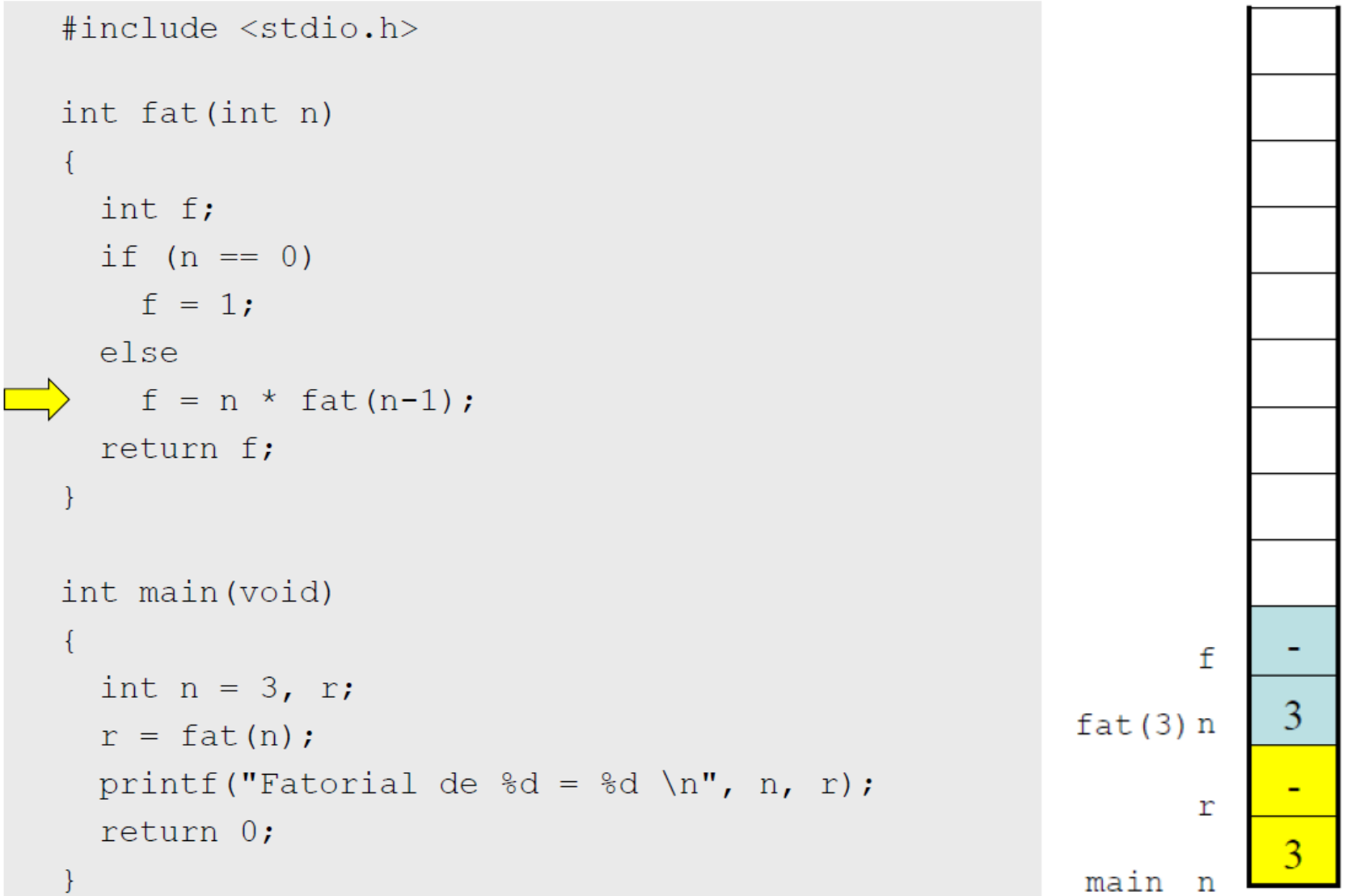
Tempo de Vida de Variáveis

```
#include <stdio.h>

int fat(int n)
{
  int f;
  if (n == 0)
    f = 1;
  else
    f = n * fat(n-1);
  return f;
}

int main(void)
{
  int n = 3, r;
  r = fat(n);
  printf("Fatorial de %d = %d \n", n, r);
  return 0;
}
```



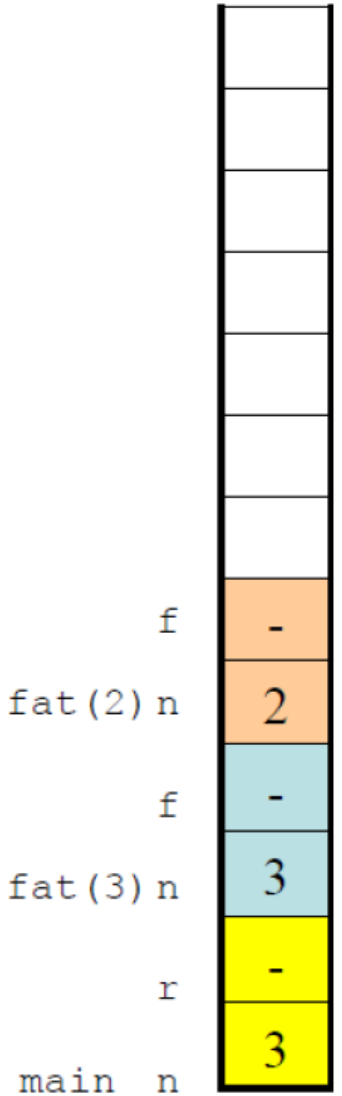
Tempo de Vida de Variáveis

```
#include <stdio.h>

int fat(int n)
{
  int f;
  if (n == 0)
    f = 1;
  else
    f = n * fat(n-1);
  return f;
}

int main(void)
{
  int n = 3, r;
  r = fat(n);
  printf("Fatorial de %d = %d \n", n, r);
  return 0;
}
```



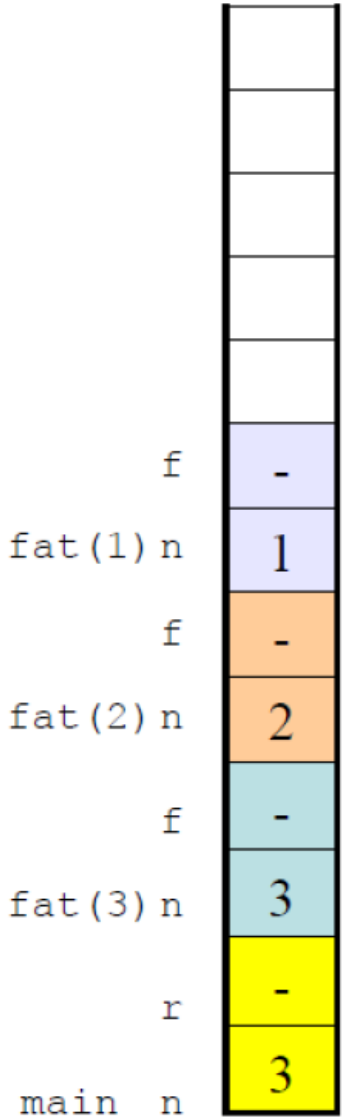
Tempo de Vida de Variáveis

```
#include <stdio.h>

int fat(int n)
{
  int f;
  if (n == 0)
    f = 1;
  else
    f = n * fat(n-1);
  return f;
}

int main(void)
{
  int n = 3, r;
  r = fat(n);
  printf("Fatorial de %d = %d \n", n, r);
  return 0;
}
```



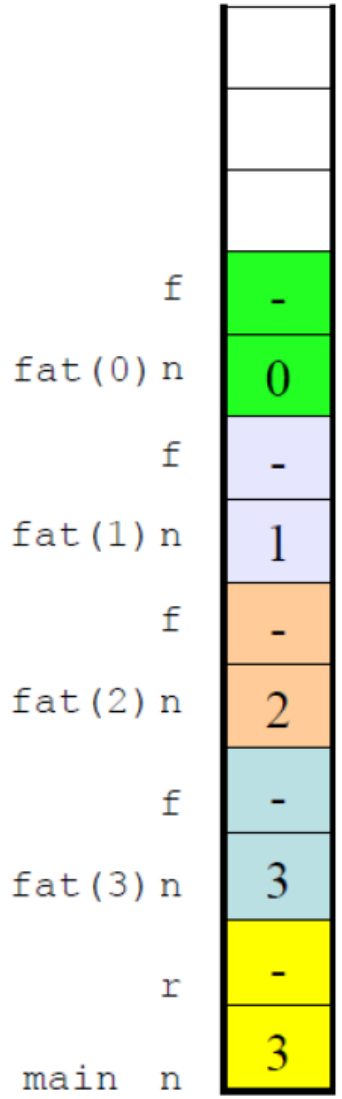
Tempo de Vida de Variáveis

```
#include <stdio.h>

int fat(int n)
{
  int f;
  if (n == 0)
    f = 1;
  else
    f = n * fat(n-1);
  return f;
}

int main(void)
{
  int n = 3, r;
  r = fat(n);
  printf("Fatorial de %d = %d \n", n, r);
  return 0;
}
```



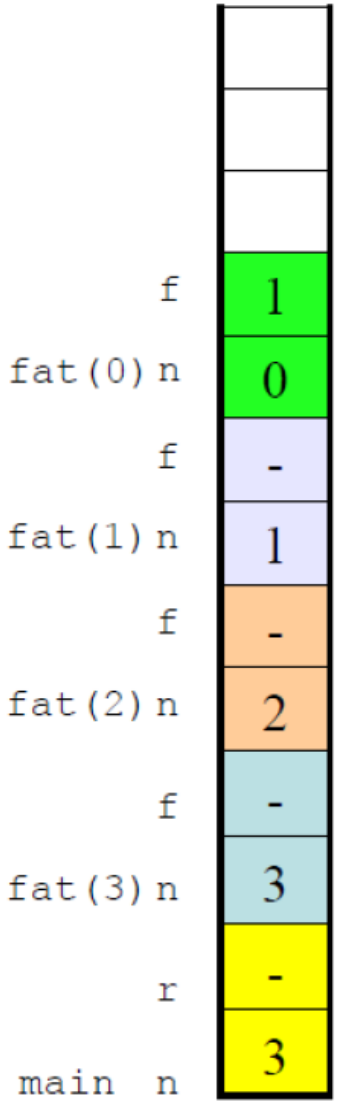
Tempo de Vida de Variáveis

```
#include <stdio.h>

int fat(int n)
{
  int f;
  if (n == 0)
    f = 1;
  else
    f = n * fat(n-1);
  return f;
}

int main(void)
{
  int n = 3, r;
  r = fat(n);
  printf("Fatorial de %d = %d \n", n, r);
  return 0;
}
```



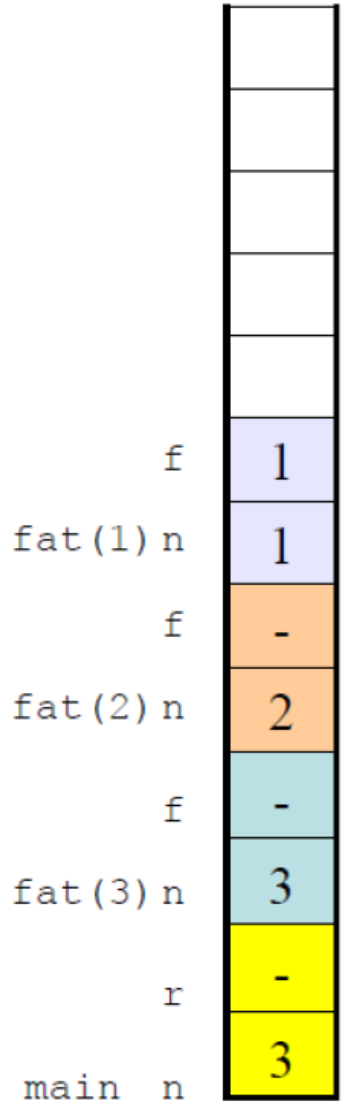
Tempo de Vida de Variáveis

```
#include <stdio.h>

int fat(int n)
{
  int f;
  if (n == 0)
    f = 1;
  else
    f = n * fat(n-1);
  return f;
}

int main(void)
{
  int n = 3, r;
  r = fat(n);
  printf("Fatorial de %d = %d \n", n, r);
  return 0;
}
```



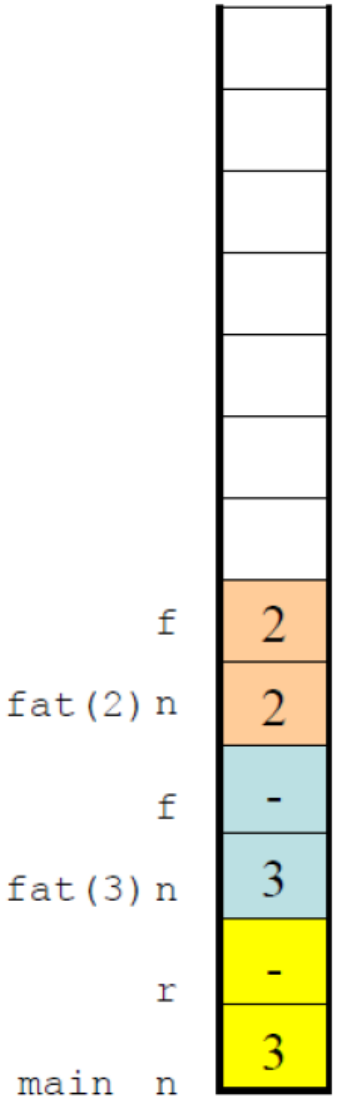
Tempo de Vida de Variáveis

```
#include <stdio.h>

int fat(int n)
{
  int f;
  if (n == 0)
    f = 1;
  else
    f = n * fat(n-1);
  return f;
}

int main(void)
{
  int n = 3, r;
  r = fat(n);
  printf("Fatorial de %d = %d \n", n, r);
  return 0;
}
```



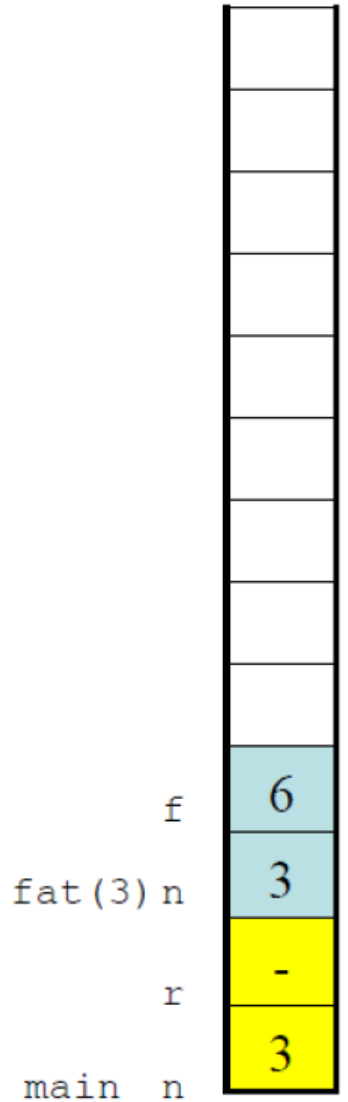
Tempo de Vida de Variáveis

```
#include <stdio.h>

int fat(int n)
{
  int f;
  if (n == 0)
    f = 1;
  else
    f = n * fat(n-1);
  return f;
}

int main(void)
{
  int n = 3, r;
  r = fat(n);
  printf("Fatorial de %d = %d \n", n, r);
  return 0;
}
```



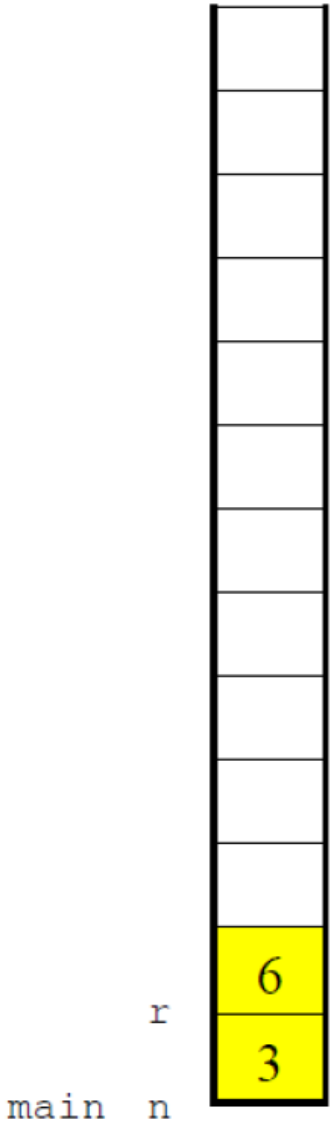
Tempo de Vida de Variáveis

```
#include <stdio.h>

int fat(int n)
{
  int f;
  if (n == 0)
    f = 1;
  else
    f = n * fat(n-1);
  return f;
}

int main(void)
{
  int n = 3, r;
  r = fat(n);
  printf("Fatorial de %d = %d \n", n, r);
  return 0;
}
```



Tempo de Vida de Variáveis

```
#include <stdio.h>

int fat(int n)
{
  int f;
  if (n == 0)
    f = 1;
  else
    f = n * fat(n-1);
  return f;
}

int main(void)
{
  int n = 3, r;
  r = fat(n);
  printf("Fatorial de %d = %d \n", n, r);
  return 0;
}
```



O conceito de Vinculação

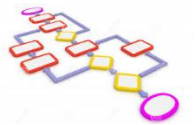
- Binding corresponde à uma associação entre uma variável e seu tipo ou valor ou entre uma operação é um símbolo.

- Exemplo:

```
const int z = 0;
char c;

int func()
{
  const float c = 3.14;
  bool b;
  ...
}

int main()
{
  ...
}
```

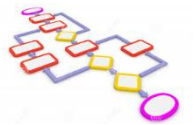
c: constante 3.14
b: variável boleana
z: constante 0
func: função

c: variável char
z: constante 0
func: função



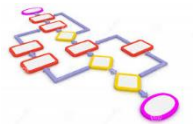
Binding

- O tempo em que ocorre a vinculação (**binding**) é um proeminente conceito de **Semântica** de Linguagens de Programação;
- Bindings podem tomar efeito em tempo de projeto da linguagem de programação, em tempo de implementação da linguagem, em tempo de compilação, em tempo de load, em tempo de linkagem ou ainda em tempo de execução.
- Por exemplo, o símbolo (*) é usualmente atribuído à operação de multiplicação em **tempo** de **projeto** da linguagem;
- Um **tipo** de dado, tal qual **int** em C, está associado a um conjunto possível de valores em tempo de **implementação** da linguagem;
- Uma **variável** pode ser associada à uma **célula** de memória em tempo de **load** em memória;



Binding

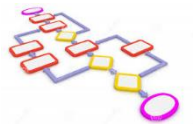
- O tempo em que ocorre a vinculação (**binding**) é um proeminente conceito de **Semântica** de Linguagens de Programação;
- Bindings podem tomar efeito em tempo de projeto da linguagem de programação, em tempo de implementação da linguagem, em tempo de compilação, em tempo de load, em tempo de linkagem ou ainda em tempo de execução.
- Por exemplo, o símbolo (*) é usualmente atribuído à operação de multiplicação em **tempo** de **projeto** da linguagem;
- Um **tipo** de dado, tal qual **int** em C, está associado a um conjunto possível de valores em tempo de **implementação** da linguagem;
- Uma **variável** pode ser associada à uma **célula** de memória em tempo de **load** em memória;



Binding – Exemplo
contador = contador + 5;

- O **tipo** de dado da variável **contador** é definido em tempo de compilação;
- O conjunto possível de valores da variável **contador** é estabelecido em tempo de projeto do compilador (implementação da linguagem);
- O significado do operador + é definido em tempo de compilação, quando os tipos de seus operandos estiverem também definidos;
- A representação interna do **literal 5** é definida em tempo de compilação;
- O valor da variável **contador** é definido em tempo de execução (runtime) com o comando: **contador** = **contador** + 1;

Observação: Um completo entendimento dos tempos de binding para os atributos de um programa é um pré-requisito para o entendimento da Semântica de Linguagens de Programação.



Binding de Atributos à Variáveis

- Um binding é **Estático** se ele ocorre primeiramente ANTES de iniciar a execução do programa e PERMANECE ao longo de toda a execução do programa;

- Se o binding ocorre primeiramente durante a execução do programa (runtime) ou pode mudar no curso de execução do programa, é chamado Binding **Dinâmico**;

- Esse ponto é essencial para distinguir **Linguagens Estáticas** de **Linguagens Dinâmicas**.

Binding de Tipo Estático

- Com binding **estático** existe uma declaração **explícita** no construto que especifica o **tipo** particular de dados da variável;
- Pode-se também efetivar o binding de tipo estático por meio de uma declaração **implícita** (por meio de alguma **convenção** ao invés de uma declaração explícita);
- No caso de uma declaração **implícita**, a primeira ocorrência do nome da variável no programa definirá seu tipo de dado;
- Tanto declaração explícita quanto implícita cria **bindings de tipo estático**;
- Exemplo de declaração implícita: Em Perl qualquer nomo que começa com **@** é um array.

Binding de Tipo Estático – Inferência de Tipos

- Uma outra forma de definir declaração implícita é por inferência de tipos (contexto ou escopo);
- Exemplo: C#

var (Referência de C#)

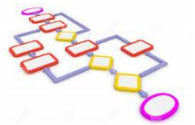
Começando com o Visual C# 3.0, as variáveis que são declaradas no escopo do método podem ter um "tipo" implícito `var`. Uma variável local de tipo implícito é fortemente tipada, como se você mesmo tivesse declarado o tipo, mas o compilador determina o tipo. As duas declarações a seguir de `i` são funcionalmente equivalentes:

C#

```
var i = 10; // Implicitly typed.
int i = 10; // Explicitly typed.
```

Binding de Tipo Dinâmico

- Com binding de tipo dinâmico, o tipo de uma variável não é especificado por um construto de declaração, nem pode ser determinado por alguma convenção imposta no nome da variável;
- Ao invés disso, a variável é ligada a um tipo de dado somente no instante em que uma atribuição de algum valor é feito à variável em tempo de execução;
- Adicionalmente, o tipo de uma variável pode ser diversas vezes alterado durante a execução do programa;
- Nesse caso, o binding de uma variável a um tipo de dados é temporário, podendo assim, ser alterando em tempo de execução;
- Exemplos: Python, Ruby, JavaScript, PHP, Lisp;



Tempo de Vida

- O tempo de vida de uma variável é o tempo durante o qual ela está vinculada a uma célula de memória.

- Os tipos de variáveis são classificadas em:
  - Variáveis estáticas;
  - Variáveis dinâmicas na pilha;
  - Variáveis dinâmicas no monte (heap) explícitas;
  - Variáveis dinâmicas no monte (heap) implícitas;

Variáveis Estáticas

- Vinculadas a células de memória antes da execução, permanecendo na mesma célula de memória durante toda a execução do programa.
  - Exemplo: variáveis estáticas em C

- **Vantagem:**
  - Eficiência (endereçamento direto);
  - Não há sobrecarga em tempo de execução para alocação e liberação.

- **Desvantagem:**
  - Falta de flexibilidade (sem recursão);
  - Armazenamento não compartilhado entre variáveis.

Variáveis Estáticas

- Exemplo em C:

```
void func()
{
  static int x = 0;
  x++;
  printf("%d\n", x);
}

int main()
{
  int cont;
  for (cont = 0; cont < 100; cont++)
  {
    func();
  }
}
```

- Exemplo em Java: public static int contador = 0; //variáveis de classe

Variáveis Dinâmicas de Pilha

- Vinculações são criadas quando suas sentenças de declaração são efetuadas, mas o tipo é estaticamente vinculado.
  - Exemplo: em C, as declarações de variáveis que aparecem no início de um método são elaboradas quando o método é chamado e as variáveis definidas por essas declarações são liberadas quando o método completa sua execução.

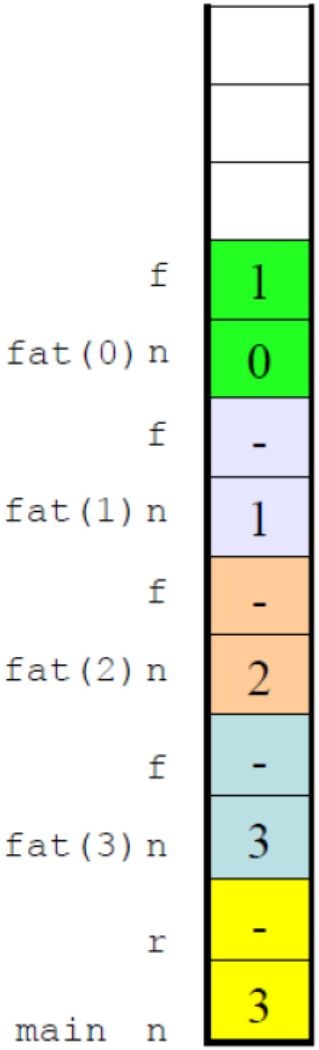
- **Vantagem:**
  - Permitem recursão e otimizam o uso de espaço em memória.

- **Desvantagens:**
  - Sobrecarga de alocação e liberação.

Exemplo em **Java**: Todas as variáveis definidas em métodos são por default, **variáveis dinâmicas de Pilha**;

Variáveis Dinâmicas de Pilha

```
#include <stdio.h>

int fat(int n)
{
  int f;
  if (n == 0)
    f = 1;
  else
    f = n * fat(n-1);
  return f;
}

int main(void)
{
  int n = 3, r;
  r = fat(n);
  printf("Fatorial de %d = %d \n", n, r);
  return 0;
}
```



Variáveis Dinâmicas na Heap (Explícitas)

- Variáveis dinâmicas do monte (heap) explícitas são células de memória não nomeadas (abstratas), que são alocadas e liberadas por instruções explícitas, especificadas pelo programador, que tem efeito durante a execução.
  - Variáveis referenciadas através de ponteiros.

- **Vantagem:**
  - Armazenamento dinâmico.

- **Desvantagens:**
  - O programador é responsável pelo gerenciamento da memória (não confiável).

Variáveis Dinâmicas na Heap (Explícitas)

- Exemplo em C:

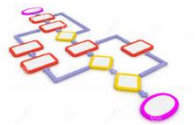
```
int *valor;
valor = (int*) malloc(sizeof(int));

...

free(valor);
```

Variáveis Dinâmicas na Heap (Implícitas)

- Variáveis dinâmicas do monte implícitas são vinculadas ao armazenamento no monte (heap) apenas quando são atribuídos valores a elas.
  - Exemplo em JavaScript:

```
highs = [74, 84, 86, 90, 71];
```

- **Vantagem:**
  - Flexibilidade.
- **Desvantagens:**
  - Sobrecarga em tempo de execução.



Verificação de Tipos de Variáveis

- Verificação ou Checagem de Tipo é uma atividade de garantia de que operandos e operadores são de tipos compatíveis
  - Tipo compatível é um tipo que é permitido para um operando segundo as regras da linguagem. Pode ser que haja a conversão automática para garantir a compatibilidade.

- Uma linguagem é fortemente tipada se erros de tipo são sempre detectados.
  - A linguagem java é fortemente tipificada.

Escopo de Variáveis

- O escopo de uma variável é a faixa de instruções na qual a variável é visível.
  - Uma variável é visível em uma instrução se puder ser referenciada nessa instrução.
- **Variáveis locais** vs **variáveis globais**.

- É possível definir escopos através de blocos. Exemplo em C:

```
int a, b;
if (a > b){
  int temp;
  temp = a;
  a = b;
  b = temp;
}
```

Blocos e Escopo

- Exemplo:

```
void sub()
{
  int count;
  . . .
  while (. . .)
  {
    int count;
    count++;
    . . .
  }
  . . .
}
```

- Válido em C e C++
- Ilegal em Java e C#

Blocos e Escopo

ED_IV ▸ src ▸ uscs ▸ Escopo_01 ▸ main(String[]) : void

```
1  package uscs;
2
3  public class Escopo_01 {
4
5      public static void main(String[] args )  {
6
7          int count  = 0;
8
9          int i = 10;
10
11         while (i > 0 )  {
12
13             int count = 0;
14
15             i = i -1;
16
17
18         }
19
20
21     }
22 }
```

Java

Tipos de Dados

- Um **tipo de dado** define uma coleção de dados e um conjunto de operações predefinidas sobre esses dados.
- É importante que uma linguagem de programação forneça uma **coleção apropriada de tipos de dados**.
- **Sistema de tipos:**
  - Define como um tipo é associado a uma expressão;
  - Inclui regras para equivalência e compatibilidade de tipos;
- Entender o sistema de tipos de uma linguagem de programação é um dos aspectos mais importantes para entender a sua **semântica**.

Tipos de Dados Primitivos

- Os **tipos de dados primitivos** são os tipos de dados não-definidos em termos de outros tipos.

- Praticamente todas as linguagens de programação oferecem um conjunto de tipos de dados primitivos.

- Usados com construções de tipo para fornecer os tipos estruturados. Os mais comuns são:
  - Tipos numéricos;
  - Tipos booleanos;
  - Tipos caracteres;

Tipos de Dados Primitivos: Inteiro

- O tipo de dados primitivo numérico mais comum é o inteiro.

- Atualmente, diferentes tamanhos para inteiros são suportados diretamente pelo hardware.
  - Exemplo – Java: byte, short, int, long

Tipo	Tamanho (Bits)	Descrição
byte	8 (1 byte)	Pode assumir valores entre $-2^7 = -128$ e $2^7 = +128$.
short	16 (2 bytes)	Pode assumir valores entre -2^{15} e 2^{15}
int	32 (4 bytes)	Pode assumir valores entre -2^{31} e 2^{31}
long	64 (8 bytes)	Pode assumir valores entre -2^{63} e 2^{63}

Tipos de Dados Primitivos: Ponto Flutuante

- Tipos de dados de ponto flutuante modelam os números reais.

- A maioria das linguagens de programação de fins científicos suportam pelo menos dois tipos de ponto flutuante.
  - Exemplo – Java: float e double

Tipo	Tamanho (Bits)	Descrição
float	32 (4 bytes)	O menor valor positivo representável por esse tipo é 1.40239846e-46 e o maior é 3.40282347e+38.
double	64 (8 bytes)	O menor valor positivo representável é 4.94065645841246544e-324 e o maior é 1.797693134862315 7e+308.

Tipos de Dados Primitivos: Ponto Flutuante

- Valores de ponto flutuante são representados como **frações expoentes** (máquinas mais antigas). Máquinas atuais usam o formato padrão **IEEE Floating-Point Standard 754**.

- Os valores que podem ser representados pelo tipo ponto flutuante é definido em termos de:
  - **Precisão:** exatidão da parte fracionária de um valor (medida pela quantidade de bits);
  - **Faixa:** combinação da faixa de frações e, o mais importante, de expoentes.

Tipos de Dados Primitivos: Booleano

- Tipo mais simples de dado primitivo;
- Faixa de valores: dois elementos, um para “true” e um para “false”;
- Um inteiro poderia ser utilizado para representá-lo, porem dificulta a legibilidade.
  - C não apresenta o tipo booleano: 0 = falso e 1 = verdadeiro

Tipos de Dados Primitivos: Caracteres

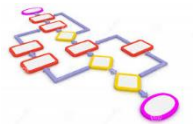
- Armazenados como codificações numéricas;
- Tradicionalmente, a codificação mais utilizada era o **ASCII** de 8 bits
  - Faixa de valores entre 0 e 127 para codificar 128 caracteres diferentes.
- Uma alternativa, codificação de 16 bits: **Unicode** (UCS-2)
  - Inclui caracteres da maioria das linguagens naturais
    - Usado em Java;
    - C# e JavaScript também suportam Unicode.

Tipo Cadeia de Caracteres (Strings)

- **String de Tamanho Estático:**
  - Tamanho especificado na declaração;
  - "Strings cheias", caso uma string mais curta for atribuída, os caracteres vazios são definidos como brancos (espaços).
- **String de Tamanho Dinâmico Limitado:**
  - Possuem tamanhos variáveis até um máximo declarado e fixo estabelecido pela definição da variável;
  - Tais variáveis podem armazenar qualquer número de caracteres entre zero e o máximo.
- **String de Tamanho Dinâmico:**
  - Possuem tamanhos variáveis sem limites;
  - Tal opção exige a alocação e desalocação dinâmica de armazenamento, mas proporciona máxima flexibilidade.

Tipo Cadeia de Caracteres (Strings)

- **Strings em C:**
  - Não são definidas como tipo primitivo
  - Usam vetores char e uma biblioteca de funções que oferecem operações (string.h)
  - Finalizadas por um caractere especial, nulo, representado com um caractere especial ('\0')
    - Operações são realizadas até encontrar este caractere.
    - Exemplo: char str[] = "teste";
      - "teste\0"
  - Biblioteca de funções:
    - strcpy, strcmp, strlen, strcat...



Leitura Complementar