

# Linguagens de Programação e Compiladores

*Paradigmas de Construção de Software*

José Osvano da Silva, PMP

# Sumário

- › Paradigma de Construção de Software;
- › Programação Estruturada;
- › Ideias básicas da programação estruturada;
- › Estruturas de Controle;
- › Parâmetros;
- › Exemplos;
- › Escolha de Mecanismos;
- › Amarração, Entidades e Atributos;
- › Variável;
- › Exercício.

# Paradigma de Construção de Software

- › Paradigma é a filosofia adotada na construção de software:
  - Imperativo ou Procedural (C, Fortran, Pascal, etc...);
  - Lógico (Prolog, etc...);
  - Funcional (Lisp, OCAML, etc...);
  - Orientado a Objetos (Java, C++, SmallTalk, etc... );
  - Orientado a Aspectos (AspectJ, AspectC++, etc...).

# Programação Estruturada

- › As linguagens desse paradigma são muitas vezes chamadas de linguagens **convencionais, procedurais ou imperativas**.
- › São linguagens que “reconstroem” uma máquina para torná-la mais conveniente para programação.
- › Máquinas que influenciaram fortemente a estrutura das linguagens de programação: arquitetura de von Neumann.

# Programação Estruturada

- › **Características da Arquitetura de Máquina:**
  1. unidade de processamento
  2. memória
  3. registradores que estabelecem comunicação entre 1º e 2º
- › **Conceitos introduzidos por essa máquina:**
  - variável, valor e atribuição
  - processamento seqüencial
  - programa armazenado
- › As linguagens imperativas são projetadas de acordo com o princípio do **Modelo de Máquina**: uma linguagem deve permitir usar diretamente uma máquina orientada por atribuições.

# Ideias básicas da programação estruturada

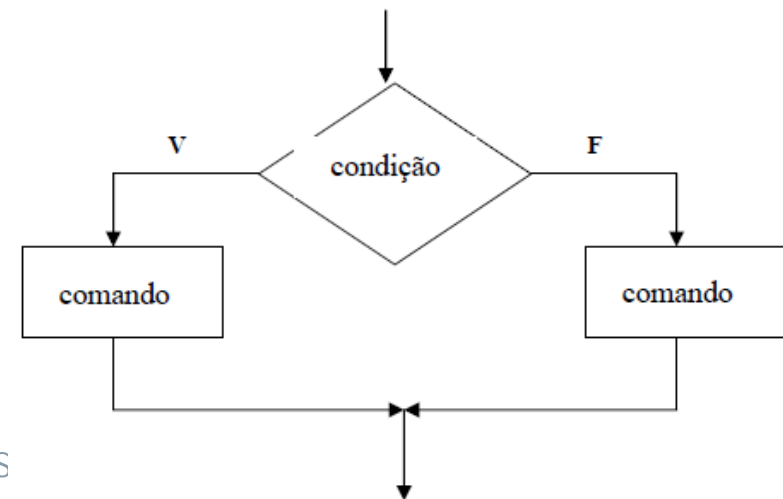
- › **Controle de Fluxo Estruturado** – um programa é dito estruturado quando o controle de fluxo é evidente da estrutura sintática do texto do programa.
- › **Invariantes** – é uma afirmação (condição) em um ponto p do programa que é válida toda vez que o controle alcança o ponto p.

# Estruturas de Controle

- › Estruturas que determinam o fluxo de execução (que comando é executado depois do outro).
- › Podem ser em nível de comando ou em nível de unidades.

# Estruturas de Controle em Nível de Comando

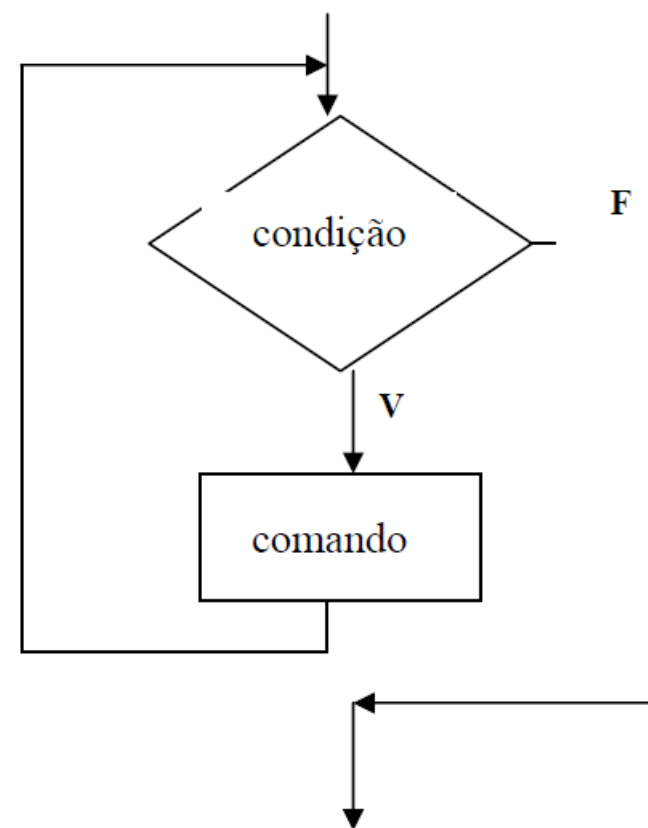
- › **Controle Sequencial** – o mais simples.
  - Os comandos são escritos na seqüência, e assim são executados:
  - A;B;C...
- › **Ramificação ou Seleção** – permitem especificar uma escolha entre comandos alternativos.
  - Exemplo: if-then-else
  - Representação básica:





# Estruturas de Controle em Nível de Comando

- › **Iteração ou Repetição** – permitem especificar repetição de certos comandos
  - Exemplo: while
  - Outros exemplos: repeat, for



# Estruturas de Controle em Nível de Unidades

- › Mecanismos que permitem fazer chamadas de unidades.
- › **Chamadas de unidades explícitas:**
  - funções, procedimentos.
- › **Chamadas de unidades implícitas:**
  - tratadores de exceção, corrotinas, unidades concorrentes.

# Unidades Subordinadas Chamadas Explicitamente

- › Inclui os subprogramas: subrotinas, funções, procedimentos
- › Função – abstrai uma expressão a ser avaliada
  - Funções retornam valores.
  - Exemplo: cálculo de fatorial de um dado número:
  - **fatorial(n)** deve retornar  $n!$
  - Efeito secundário: quando os parâmetros da função retornam valores.
- › Procedimento – abstrai um comando a ser executado
  - Modifica variáveis
  - Exemplo: ordenação de um vetor de números.
  - **ordena(v)** deve ordena o vetor  $v$ .

# Unidades Subordinadas Chamadas Explicitamente

› PASCAL:

```
function <nome da função> ( <lista de argumentos> ) : <tipo>;
```

```
Begin
```

```
    < corpo da função>
```

```
End
```

```
procedure <nome> (<parâmetros>);
```

```
Begin
```

```
    <corpo do procedimento>
```

```
End
```

# Unidades Subordinadas Chamadas Explicitamente

C:

<tipo do resultado> <nome> ( <declaração de parâmetros formais. >

{

    <lista de declarações>

    <lista de comandos>

}

# Parâmetros

- › Permitem a aplicação de subprogramas a dados diferentes.
- › Melhora o reuso de código
- › Sem parâmetros, a utilidade dos subprogramas se restringiria a segmentação do código.
- › Parâmetro formal – identificadores usados no cabeçalho do subprograma (definição do subprograma)
- › Parâmetro real – identificadores, expressões ou valores usados na chamada do subprograma.
- › Argumento – usado como sinônimo de parâmetro real ou para referir o valor passado do parâmetro real para o formal.

## Correspondência Parâmetros Formais e Reais

- › A maioria das linguagens usa um **critério posicional** para amarração de argumentos e parâmetros:
- › Definição do procedimento –  $p_i$  são parâmetros:  
    procedure S(  $p_1$ ;  $p_2$ ; ....;  $p_n$ );  
    ..  
    end;
- › Chamada do procedimento –  $a_i$  são argumentos:  
    S(  $a_1$ ;  $a_2$ ; .....;  $a_n$ );
- ›  $p_i$  corresponde a  $a_i$  para  $i=1,...,n$ .

# Convenções para passagem de parâmetros

- › Passagem por Referência
  - (Call by Reference) ou
  - Compartilhamento (Sharing)
- › Unidade chamadora passa para a unidade chamada o **endereço** do argumento.
- › A variável usada como argumento é **compartilhada** e pode ser modificada.



# Convenções para passagem de parâmetros

- › Passagem por Cópia - Os parâmetros se comportam como variáveis locais. Pode ter 3 tipos:
  - Passagem de Valor – argumentos são usados para inicializar parâmetros, que funcionam como variáveis locais. Valores não podem retornar por esses parâmetros.
  - Passagem de Resultado – parâmetros não recebem valores na chamada, funcionam como variáveis locais mas retornam valores na saída
  - Passagem de Valor-Resultado – engloba os dois anteriores

## Convenções para passagem de parâmetros

- › Passagem de nome – a amarração do parâmetro à posição não é feita na hora da chamada, mas a cada vez que ele é usado na unidade chamada.
- › Portanto, atribuições ao mesmo parâmetro podem ser feitas a posições diferentes a cada ocorrência.

# Exemplo: Passagem por Referência x passagem por valor

› PASCAL:

```
var a, b: integer;  
procedure P (x: integer; var y: integer);  
Begin  
    x := x + 1 ; y := y + 1 ;  
    writeln (x , y)  
  
end;  
  
begin  
    a := 0 ; b := 0 ;  
    P (a, b);  
    writeln (a, b)  
  
end.
```

Saída: 1 1  
0 1

# Exemplo: Passagem por Referência x passagem por valor

› C:

```
void troca1(int x, int y) {
```

```
int z;
```

```
    z = x;
```

```
    x = y;
```

```
    y = z;
```

```
}
```

```
a = 0; b = 1;
```

```
troca1(a, b);
```

› os valores de a e b não são trocados

# Exemplo: Passagem por Referência x passagem por valor

› C:

```
void troca(int *px, int *py) {
```

```
int z;
```

```
    z = *px;
```

```
    *px = *py;
```

```
    *py = z;
```

```
}
```

```
a = 0; b = 1;
```

```
troca(&a,&b);
```

› os valores de a e b são trocados

# Passagem por valor-resultado

- › Deve ter o mesmo efeito da passagem por referência.
- › Problemas que devem ser evitados:

program

.....

procedure exemplo(x, y);

begin

    i := y

end;

begin

    i := 2; A[i] := 99;

    exemplo( i, A[i]);

end.

- › i pode ser alterado diretamente pela atribuição, ou indiretamente pela cópia do resultado
- › Na chamada de *exemplo (i, A[i])*:
  - endereços de i e A[i] são armazenados
  - valores de i e A[i] são copiados em x e y
  - valor de i é modificado em i := y
  - valores de x e y são copiados de volta em i e A[i], logo **valor antigo de i é restaurado**

## Escolha de Mecanismo

- › Parâmetros que devem retornar valores, devem ser passados por referência;
- › Parâmetros que não retornam valores podem ser passados por valor, por segurança;
- › Eficiência da implementação:
  - passagem por referência é cara em termos de processamento pois faz acesso indireto;
  - passagem por cópia pode custar em termos de memória se os objetos forem grandes.

## Passagem por nome

- › Parâmetro real substitui textualmente o parâmetro formal correspondente em todas as suas ocorrências.
- › O parâmetro formal é amarrado ao **método de acesso** no momento da chamada mas a amarração a valor ou endereço é feita só na hora que o parâmetro é atribuído ou referenciado.
- › Objetivo: flexibilidade



## Passagem por nome

- › A forma do parâmetro real determina o método de implementação. Isto distingue a passagem por nome dos outros métodos
- › Se parâmetro real é variável escalar:
  - Equivale a passagem por referência
- › Se parâmetro real é uma constante:
  - Equivale a passagem por valor

# Amarração, Entidades e Atributos

- › Programas envolvem *entidades*: variáveis, subprogramas, comandos, etc.
- › Entidades tem *atributos*.
  - Variável - nome, tipo, valor
  - Subprograma - nome, parâmetros
- › *Amarração*: especificação da natureza exata dos atributos de uma entidade
- › *Tempo de amarração*: momento em que a amarração ocorre – importante na diferenciação de linguagens.

## Tempo de amarração

- › Amarração pode ser:
- › **estática** – ocorre antes da execução do programa e não pode ser mudada.
- › **dinâmica** – ocorre em tempo de execução e pode ser mudada, respeitando as regras da linguagem.

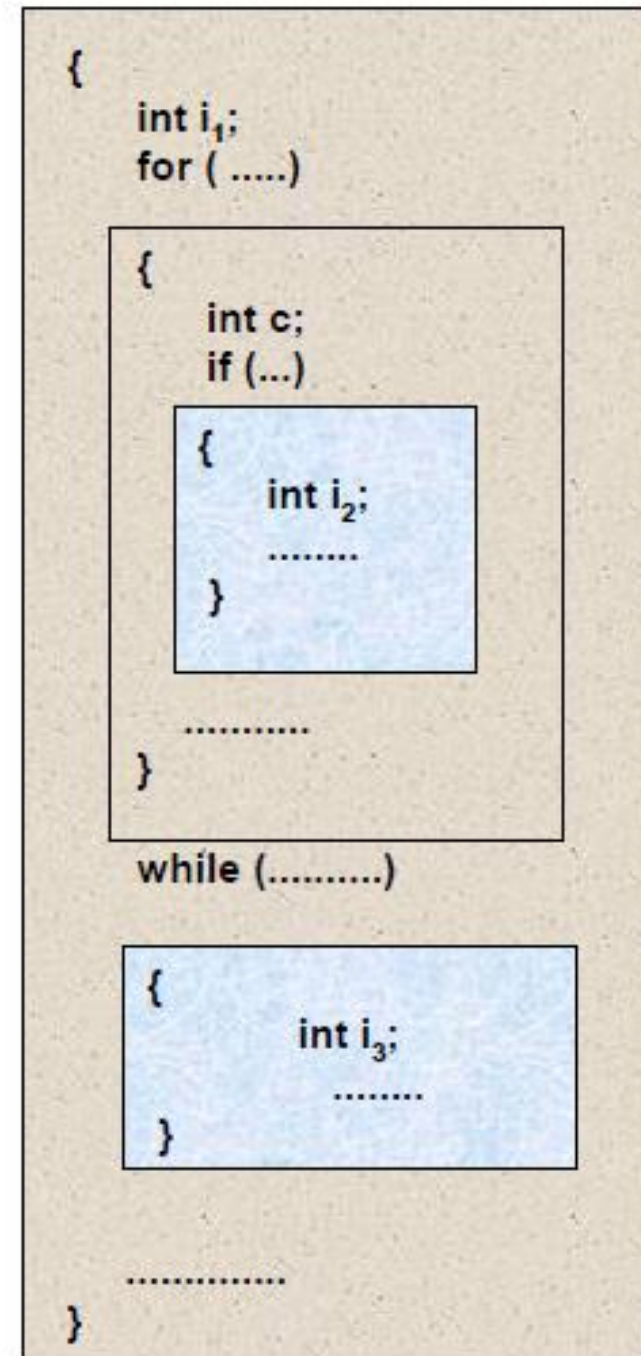
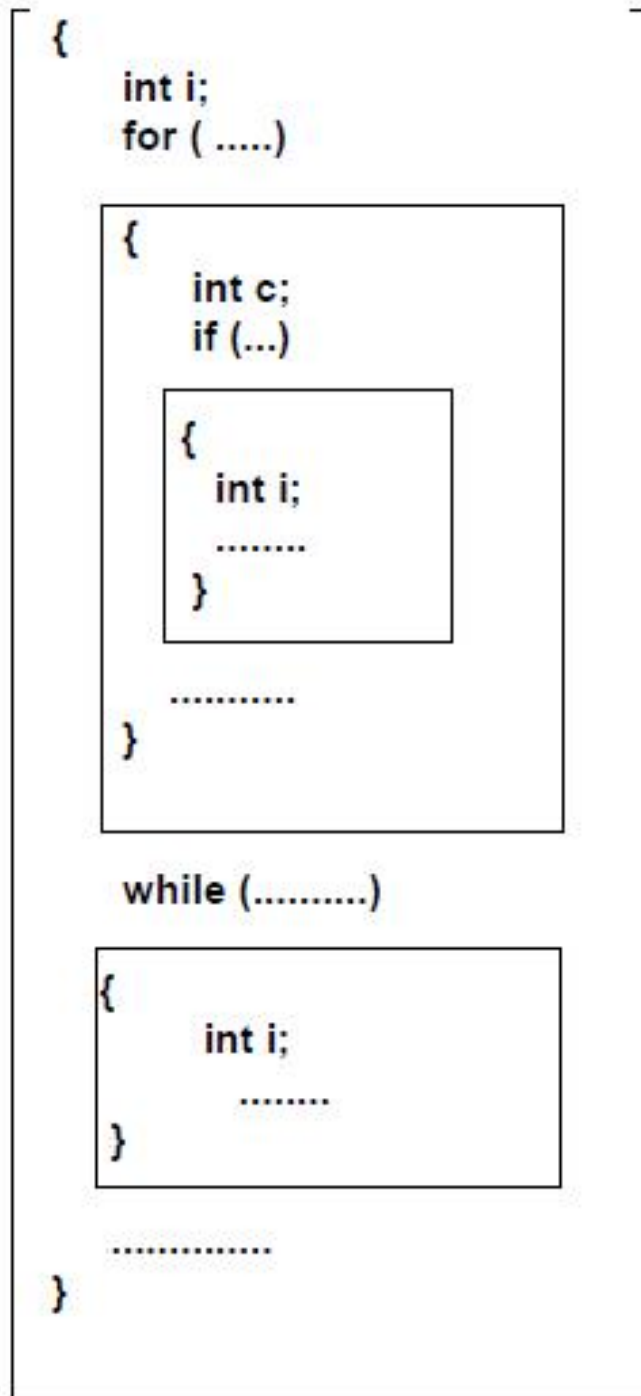
# Variáveis

- › Variáveis são conceitos abstratos de células de memória. São caracterizadas por um **nome** e quatro atributos básicos:
  - escopo,
  - tempo de vida,
  - valor e
  - tipo.

# Escopo

- › Trecho do programa onde uma variável é conhecida.
- › Uma variável é *visível* dentro do seu escopo e *invisível* fora dele.
- › Amarração estática a escopo: o escopo é definido pela estrutura léxica do programa.

# Escopo



# Amarração Dinâmica a Escopo

- › o escopo é definido em função da execução do programa. O efeito de uma declaração se estende até que uma nova declaração com o mesmo nome seja encontrada.

```
procedure big;  
var x : integer;  
    procedure sub1;  
    begin { sub1 }  
        ..... X .....  
    end; { sub1 }  
  
    procedure sub2;  
    var x : integer;  
    begin { sub2 }  
        .....  
    end; { sub2 }  
begin { big }  
.....  
end; { big }
```

## Amarração Dinâmica a Escopo

- › As referências a um identificador não podem ser identificadas na compilação
- › Para a sequência de chamadas: big - sub2 - sub1
  - a referência a x em sub1 é ao x declarado em sub2
- › Para a sequência de chamadas: big - sub1
  - a referência a x em sub1 é ao x declarado em big



## Valor de Variável

- › Representado de forma codificada na área de memória amarrada a variável. Pode ser uma referência (ponteiro).
- › Amarração dinâmica (mais comum)
  - através do comando de atribuição.
- › Amarração estática
  - definição de constantes
  - PASCAL: `const n = 50`

## Tipo de Variável

- › Especificação da classe de valores que podem ser associados à variável, bem como das operações que podem ser usadas sobre eles, através da declaração de variável.
- › Amarração de tipo estática: definida através da declaração de variáveis, que pode ser:
  - Declaração Explícita – comando que lista as variáveis e seus tipos;
  - Declaração Implícita – feita normalmente por convenções definidas para os nomes das variáveis.
  - Exemplo: no FORTRAN, variáveis que começam com I, J, K, L, M ou N são inteiras.

## Tipo de Variável

- › Amarração de tipo dinâmica: variáveis não são declaradas, e a mesma variável pode conter valores de tipos diferentes durante a execução do programa.
  - O tipo da variável é definido quando um valor é atribuído a essa variável.
  - Vantagem:
    - › flexibilidade de programação
  - Desvantagens:
    - › Diminui a capacidade de detecção de erros pelo compilador
    - › Custo maior, verificação de tipo em tempo de execução
    - › Memória de tamanho variável

## Tempo de Vida de Variável

- › Intervalo de tempo durante o qual uma área de memória está amarrada a uma variável
- › *Alocação*: ação que adquire áreas de memória para variáveis de um *pool* de memória disponível.

# Alocação Estática

- › Ocorre antes da execução do programa, nos casos de:
- › variáveis globais – devem ser acessíveis a todo o programa
- › variáveis locais estáticas – são declaradas dentro de um subprograma mas devem ser reter valores entre execuções separadas do subprograma (sensíveis à história).
- › Vantagem:
  - eficiência – endereçamento direto, não exige custo adicional para alocação
- › Desvantagem:
  - pouca flexibilidade
  - linguagens que usam apenas alocação estática não dão suporte à subprogramas recursivos nem a compartilhamento de memória.

## Exemplo

- › FORTRAN I, II, IV – todas as variáveis eram estáticas
- › C, C++, JAVA – permitem definir uma variável local como estática
- › Pascal – não possui variáveis estáticas

# Alocação Dinâmica

- › Ocorre durante a execução do programa, nos casos de:
- › Variáveis Dinâmicas de Pilha: são alocadas na pilha de execução
- › Variáveis Dinâmicas de Heap – são alocadas na heap

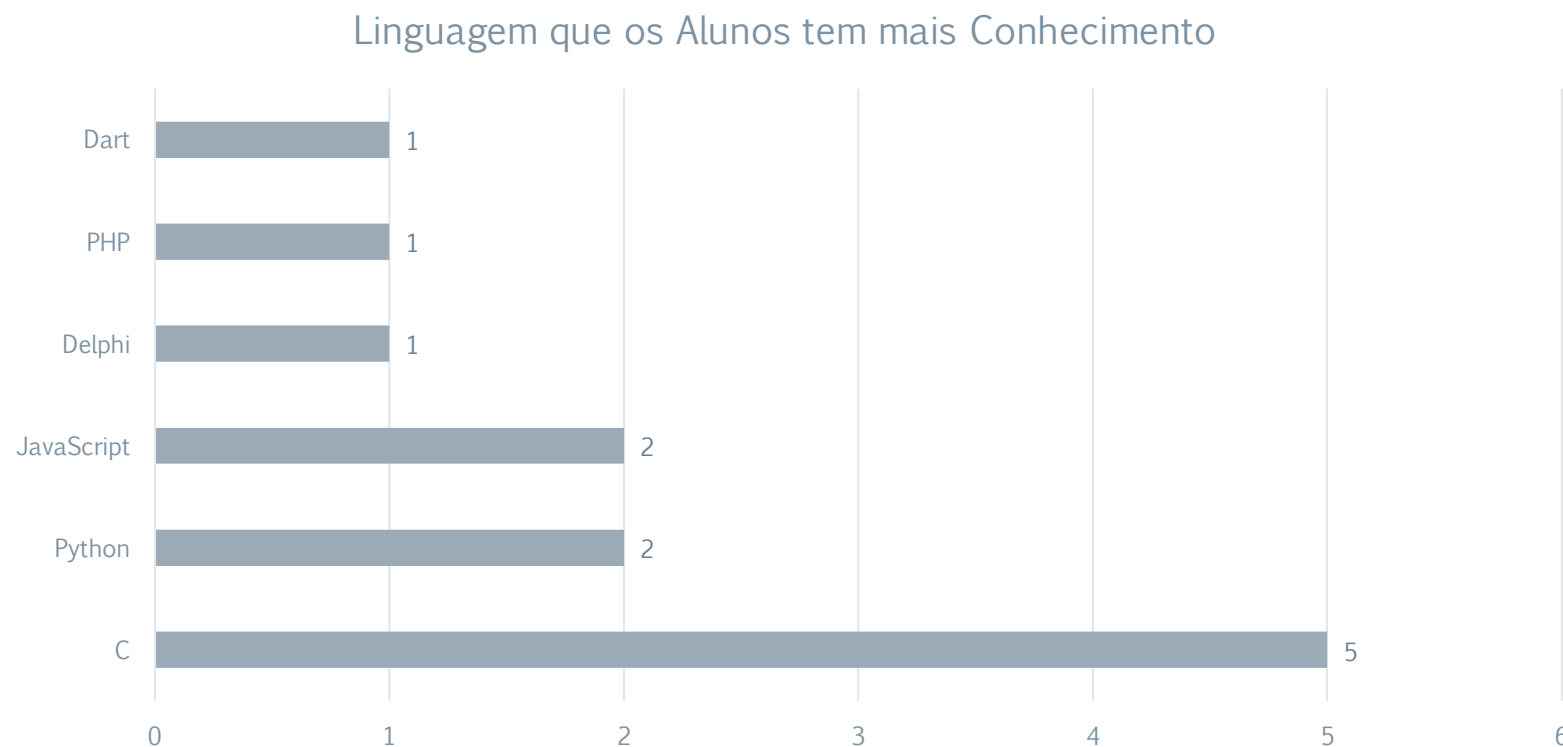
# Verificação de Tipo

- › Atividade que certifica que os operandos de um operador são de tipos compatíveis.
- › São considerados operadores e operandos:
  - operadores usuais (aritméticos, relacionais, etc)
  - subprogramas (operadores) e parâmetros (operandos)
  - atribuição (operador) e variável / expressão (operandos)
- › Os tipos de operandos são compatíveis com um operador se:
  - são aceitos pelo operador ou
  - podem ser convertidos implicitamente pelo compilador (coerção)
- › Erro de tipo: aplicação de um operador a um operando de tipo não apropriado.



# Exercício de Fixação 01 – Resultado Parcial

Qual a linguagem que você considera que tem mais conhecimento?



## Exercício de Fixação 02

### 1) Avalie a necessidade a seguir:

Você está participando de um processo seletivo para uma vaga de emprego e uma questão da prova prática é escrever um algoritmo onde o usuário vai informar dois números e o sistema fará o cálculo da média dos mesmos e mostrará o resultado, você deve realizar esse desenvolvimento em 3 linguagens de programação diferentes à sua escolha.

### 2) Escolha três linguagens de programação que desejar, pesquise quais IDEs (Front End) poderá usar para desenvolver.

### 3) Obs.:

Pode ser feito em dupla;

É preciso entregar o código fonte;

É preciso informar quais linguagens foram usadas e descrever nos comentários as principais diferenças que você identificou entre elas;

Pode ser usado o portal: <https://repl.it/>

Data de Entrega: 27/02/2023 23:59.

# Dúvidas



**José Osvano da Silva**  
[joseosvano@unipac.br](mailto:joseosvano@unipac.br)