ICC043/IEC582 – Paradigmas de Linguagens de Programação

Métodos, Elementos e Critérios de Avaliação

Prof. Dr. Rafael Giusti
rgiusti@icomp.ufam.edu.br

Leitura recomendada

- **SEBESTA**. *Concepts of Programming Languages*
  - **Capítulo 1**: introdução
    - Em particular as seções 1.3, 1.5 e 1.6
  - **Capítulo 5**: "nomes, vínculos e escopo"

- **SCOTT, Michael L**. *Programming Language Pragmatics*
  - **Capítulo 3**: "*names, scopes, and bindings*"

Agenda

- Paradigmas de linguagens
- Métodos de implementação
- Tempos do programa
- Elementos de linguagens
- Critérios de avaliação

Agenda

- Paradigmas de linguagens
- Métodos de implementação
- Tempos do programa
- Elementos de linguagens
- Critérios de avaliação

Formas de olhar para as LP

- Por tempo e herança
  - Como as linguagens se desenvolveram e o que herdaram umas das outras
- Por domínio de aplicação
  - Científicas
  - Comerciais
  - Inteligência Artificial
  - Para ensino
  - Propósito geral?

Formas de olhar para as LP

- Por características/recursos dominantes
  - Baixo/alto nível
  - Linguagens "dinâmicas" vs. "estáticas"
- Por tipos de dados
  - Assembly: tudo é "byte" ou múltiplo de bytes
  - C, C++, Java: vários tipos, mas diferem muito em como eles podem ser manipulados
  - Lua: tabelas
  - MATLAB, R, Python (com NumPy): matrizes

Paradigmas de Linguagens

- **Paradigma**[1]
  - Algo que serve de exemplo ou modelo; padrão
    - Paradigma de linguagem
    - Paradigma de projeto
    - Paradigma de desenvolvimento

[1] michaelis.uol.com.br/busca?r=0&f=0&t=0&palavra=paradigma

Paradigmas de Linguagens

- Podemos observar as linguagens do ponto de vista dos seus paradigmas
  - Cada linguagem tem uma maneira particular de "modelar" algoritmos
  - A escolha de um paradigma influencia na forma como pensamos no problema e o traduzimos em algoritmos
  - Podemos agrupar e estudar as linguagens em paradigmas
    - Pode haver interseção entre paradigmas

Paradigmas de Linguagens

- Uma visão de paradigmas
  - Linguagens imperativas
    - Procedimental / procedural
    - Orientada a objetos
  - Linguagens declarativas
    - Linguagens funcionais
    - Lógicas
    - Linguagens de especificação de *hardware*
    - Linguagens de fluxo de dados

Paradigmas de Linguagens

- Visão alternativa (categorias não excludentes)
  - Vetoriais
  - Concorrentes / paralelas
  - Imperativas / declarativas
  - Metaprogramação
    - Reflexão / macros / gabaritos (*templates*)
  - Simbólicas
  - Baseadas em pilha
  - Baseadas em contratos

Paradigmas de Linguagens

- **Os "quatro grandes" paradigmas**
  - Procedimental / procedural
  - Orientado a objetos
  - Funcional
  - Lógico

Um problema

- Máximo divisor comum (mdc)
  - Dados a, b ∈ $\mathbb{N}^+$, qual é o maior divisor comum a ambos?
  - Algoritmo ingênuo: fatorização

980	2
490	2
245	5
49	7
7	7
1	

1925	5
385	5
77	7
11	11
1	

Algoritmo de Euclides

- Descrito por Euclides de Alexandria por volta de 300 AC
- Um dos algoritmos mais antigos da história
- Descreve uma sequência de operações algébricas simples para calcular o máximo divisor comum entre dois números
  - Podemos reescrever esse algoritmo em diferentes paradigmas?

Algoritmo de Euclides

- **Paradigma Imperativo**
  - Para encontrar o MDC entre dois números a e b, verifique se $a = b$. Caso sejam, então a é o MDC. Caso contrário, substitua o maior pela diferença entre eles e repita o processo.
- "Você, computador"
  - Siga as instruções como se fossem comandos (daí o nome), modificando a e b de acordo com o algoritmo até alcançar o resultado final

Algoritmo de Euclides

- **Paradigma Funcional**

$$\text{mdc}(a, b) = \begin{cases} a & , \text{se } a = b \\ \text{mdc}(a - b, b) & , \text{se } a > b \\ \text{mdc}(a, b - a) & , \text{se } a < b \end{cases}$$

- "Você, computador"
  - Expanda a relação recursiva mdc(a, b) até alcançar o caso base

Algoritmo de Euclides

- **Paradigma Lógico**
  - É verdade que os mdc entre a e b é um número inteiro g se
    - Todos os valores a, b e c são idênticos; ou
    - $a > b$ e existe um número g tal que g é o MDC entre b e $a - b$; ou
    - $b > a$ e existe um número g tal que g é o MDC entre a e $b - a$
- "Você, computador"
  - Encontre a, b e g que satisfaçam as proposições e as condições

A respeito desses paradigmas

- Existem **muitos** termos (que às vezes se sobrepõem) para classificar as linguagens
  - Concorrente, baseada em restrições, de *scripting* etc.
- Caracterizar uma linguagem pode ser subjetivo
  - Algumas parecem claramente "pertencer" a um paradigma
  - A maioria das linguagens modernas é considerada "multi-paradigma"

A respeito desses paradigmas

- É possível aplicar conceitos de um paradigma em linguagens que parecem pertencer a outros!
- Funcional em C

```c
int mdc(int a, int b)
{
    if (a == b)
        return a;
    if (a < b)
        return mdc(b, a);
    return mdc(a - b, b);
}
```

A respeito desses paradigmas

- É possível aplicar conceitos de um paradigma em linguagens que parecem pertencer a outros!
- Procedimental em Java

```
class Foo {
    public static void main(String[] args ) {
        // todo o programa aqui!
    }
}
```

A respeito desses paradigmas

- Essa terminologia é um pouco "*fuzzy*"
  - Pergunta incorreta
    - A linguagem Java/Python/X é uma linguagem orientada a objetos?
  - Mais adequado
    - Quais são os aspectos de Java/Python/X que favorecem um estilo de programação orientado a objetos?

Agenda

- Paradigmas de linguagens
- Métodos de implementação
- Tempos do programa
- Elementos de linguagens
- Critérios de avaliação

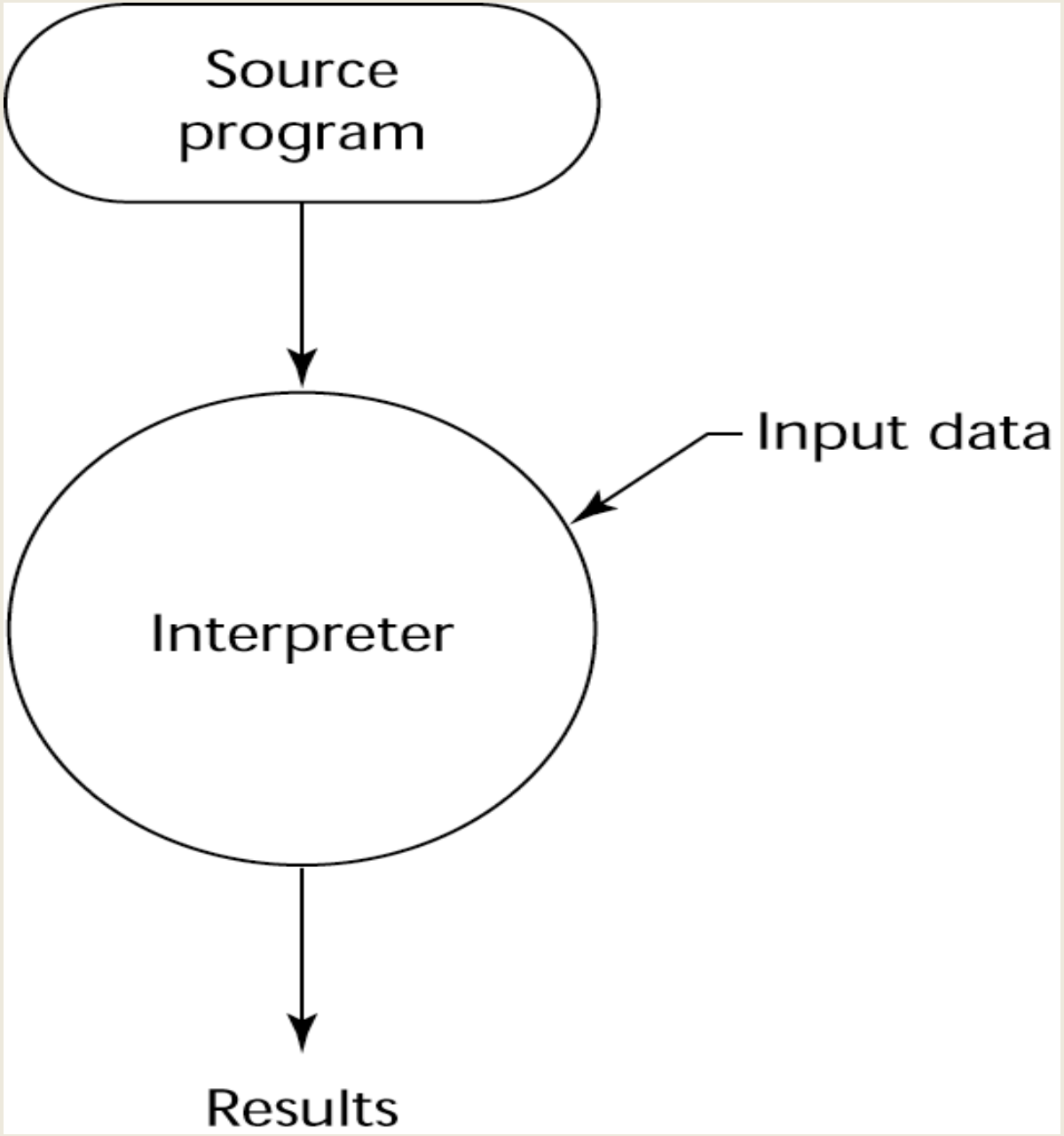
Métodos de implementação

- "Linguagens compiladas"
  - Os programas são traduzidos para linugagem de máquina e então executados nativamente pela unidade de processamento
- "Linguagems interpretadas"
  - A unidade de processamento é substituída por uma máquina virtual (interpretador)
- "Linguagens híbridas"
  - Os programas são pré-compilados para uma linguagem intermediária que é interpretada

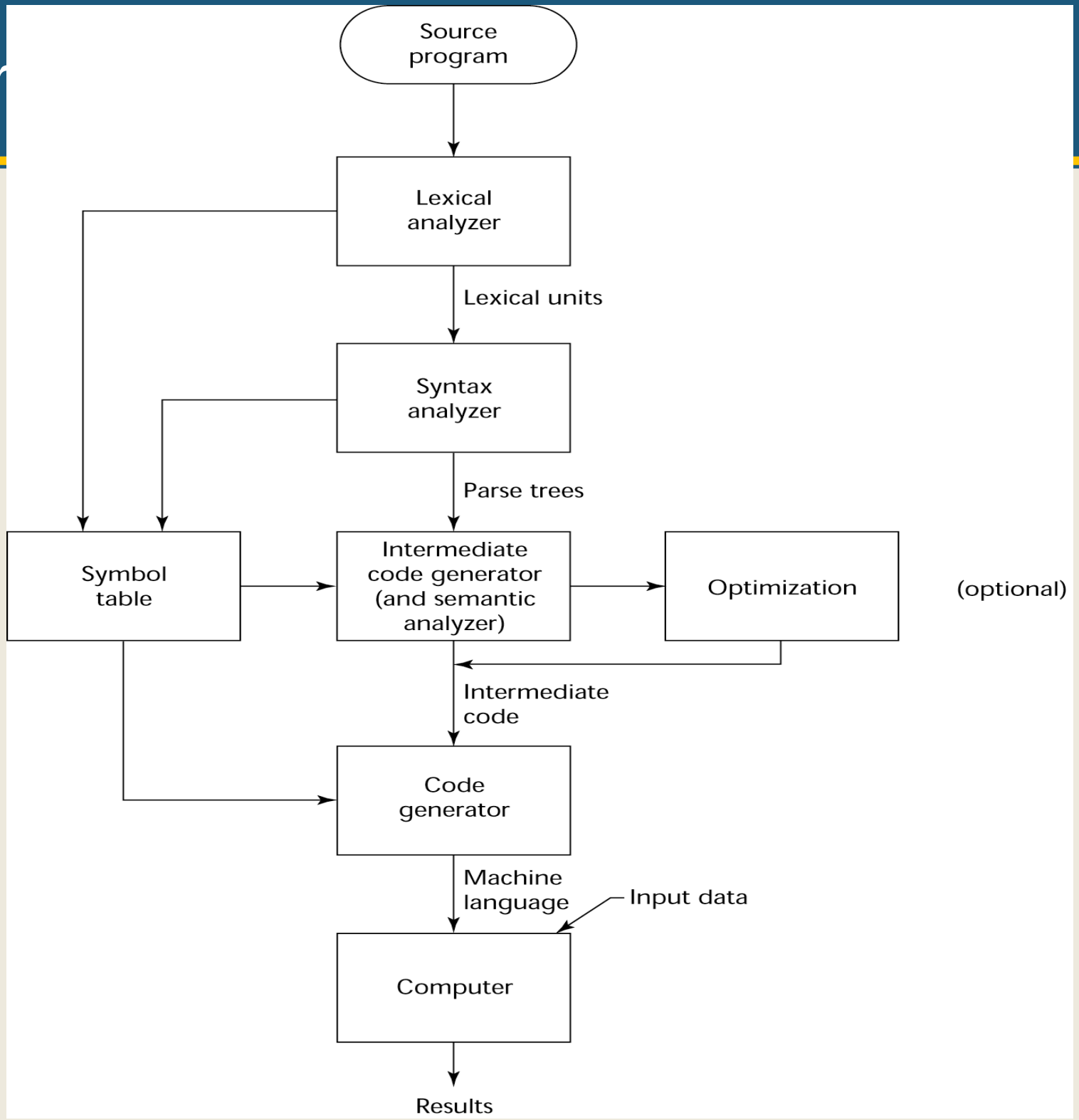
Métodos de implementação

- Esses métodos são tipicamente associados às linguagens
  - Mas nem sempre são características essenciais das linguagens
  - Programas em C++ são normalmente compilados, mas é possível interpretar
    - Jupyter Notebook possui uma máquina virtual para executar programas C++ usando o navegador como interface
  - Programas em Java são normalmente híbridos, mas podem ser totalmente compilados

Interpretação e compilação



Interpr


Source program
Lexical analyzer
Lexical units
Syntax analyzer
Parse trees
Symbol table
Intermediate code generator (and semantic analyzer)
Optimization
(optional)
Intermediate code
Code generator
Machine language
Input data
Computer
Results

Criação

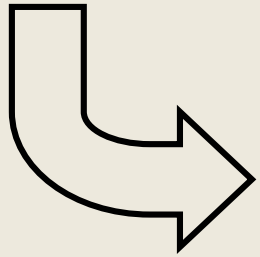
- Usando um editor de textos ou um ambiente de desenvolvimento, criamos o programa
  - Linguagem relativamente alto nível
    - Relativamente independente de arquitetura
    - O programa pode ser construído e executado sem modificações ou com poucas modificações em qualquer plataforma

Compilação

- O **compilador** traduz o programa
  - Assembly ou outra linguagem intermediária
    - Baixo-nível
    - Dependente de arquitetura
  - Cada linha do programa traduzido representa um dado ou uma instrução da máquina

Compilação

```
int i;
void main() {
  for (i = 1; i <= 100; i++)
    fred(i);
}
```



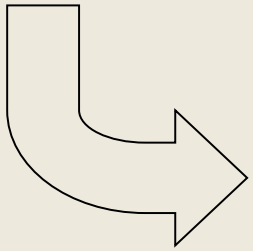
```
i:      data word 0
main:   move 1 to i
t1:     compare i with 100
        jump to t2 if greater
        push i
        call fred
        add 1 to i
        go to t1
t2:     return
```

Montagem

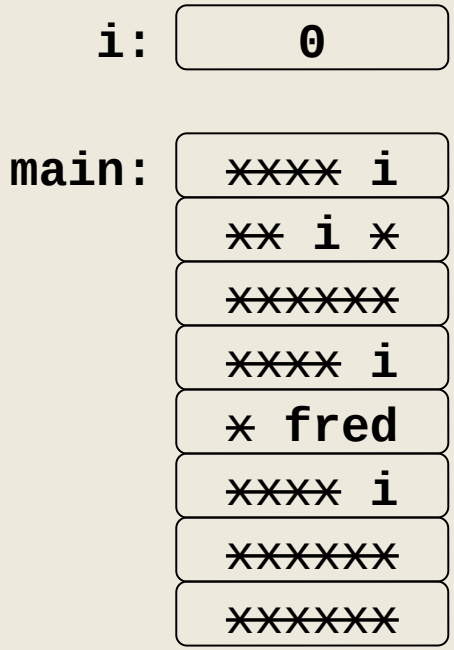
- Assembly (ou outra linguagem intermediária) ainda não é executável
  - Ainda é texto em linguagem quase natural
  - Não é baixo-nível o suficiente para o processador
- O ***assembler*** (montador) converte o programa anteriormente compilado para linguagem de máquina
  - O resultado é um **arquivo objeto** que não é legível por pessoas

Montagem

```
i:      data word 0
main:   move 1 to i
t1:     compare i with 100
        jump to t2 if greater
        push i
        call fred
        add 1 to i
        go to t1
t2:     return
```



assembler



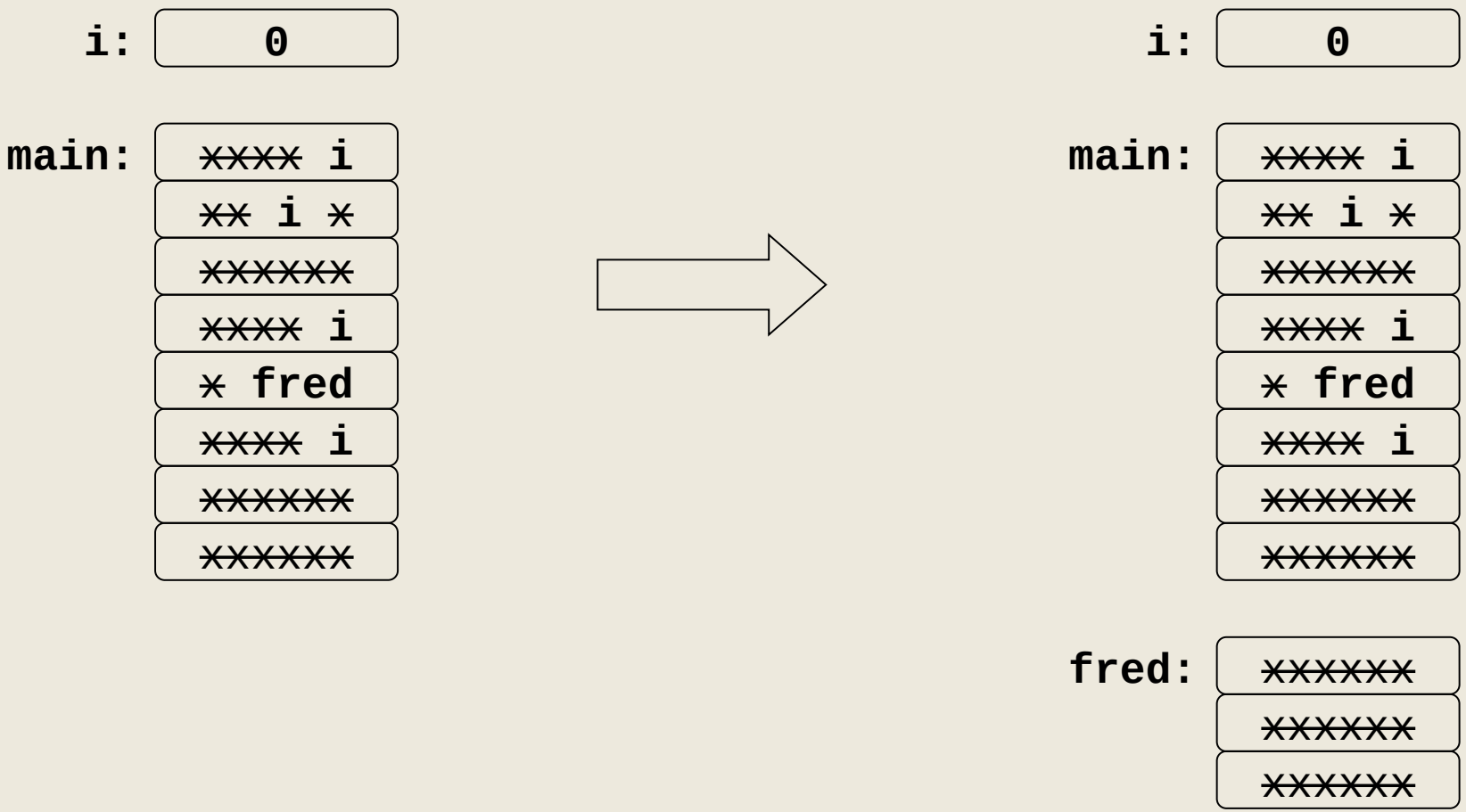
Linking (ligação)

- Os arquivos objeto ainda não são diretamente executáveis pelo processador
  - Podem ter algumas partes ausentes
    - No nosso exemplo, a função `fred()` está em outro lugar
  - Ainda contêm alguns símbolos (nomes)
  - Quase totalmente código de máquina, mas ainda não 100%

Linking (ligação)

- O ***linker*** (ligador) coleta todos os arquivos objeto e resolve as referências
  - Permite compilação separada
  - Permite a implementação de bibliotecas dinâmicas
    - Referências que só serão resolvidas depois
    - O *linker* apenas verifica essas referências
- O resultado do trabalho do *linker* é um arquivo binário (executável)

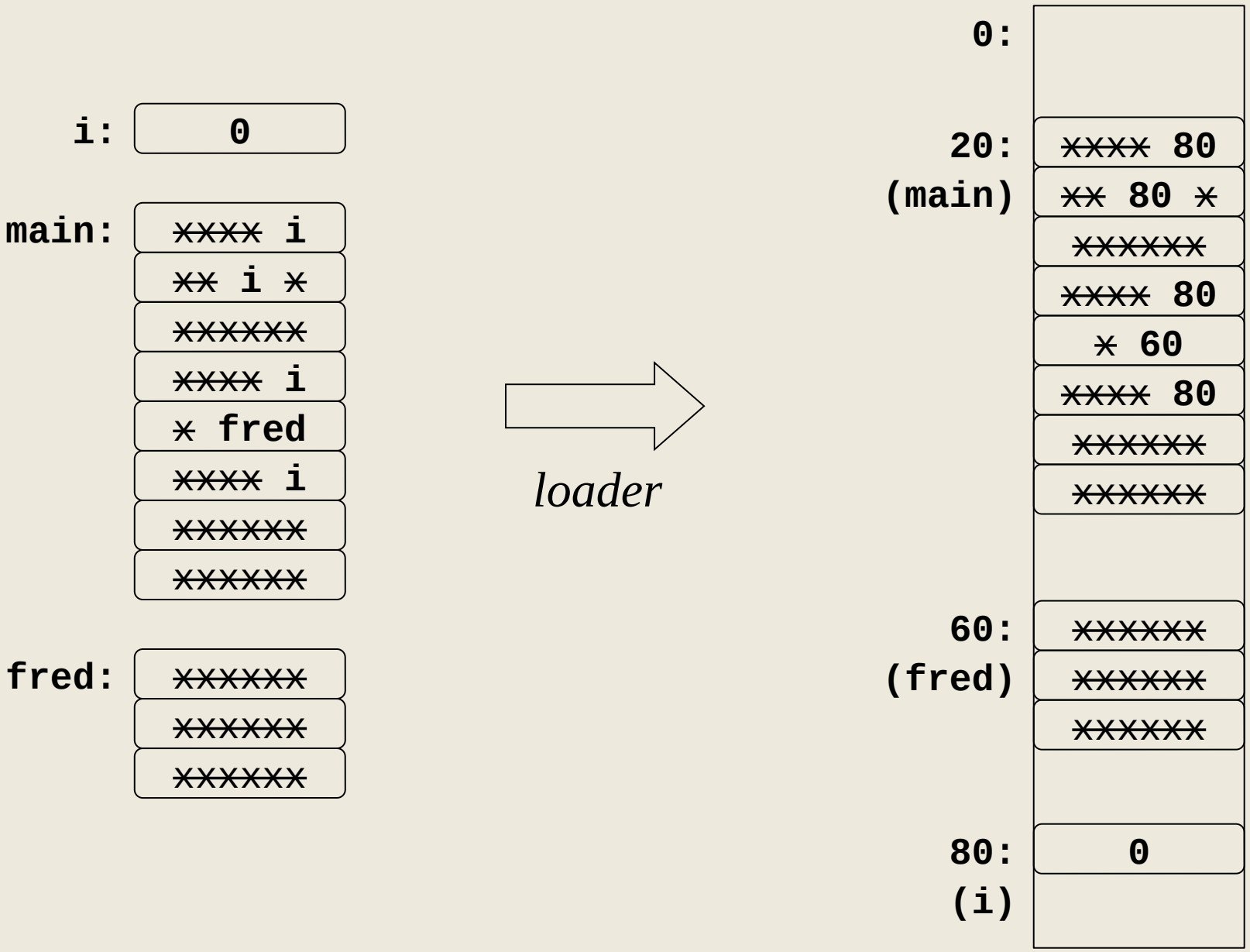
Linking (ligação)



Carga

- O executável ainda não é totalmente "executável"
  - **Ainda** contém alguns símbolos
  - Quase completamente linguagem de máquina, mas **ainda** não 100%
- Passo final
  - **Execução**
  - Quando o programa é carregado em memória, o carregador (***loader***) carrega o programa e transforma os nomes restantes em endereços

Carga

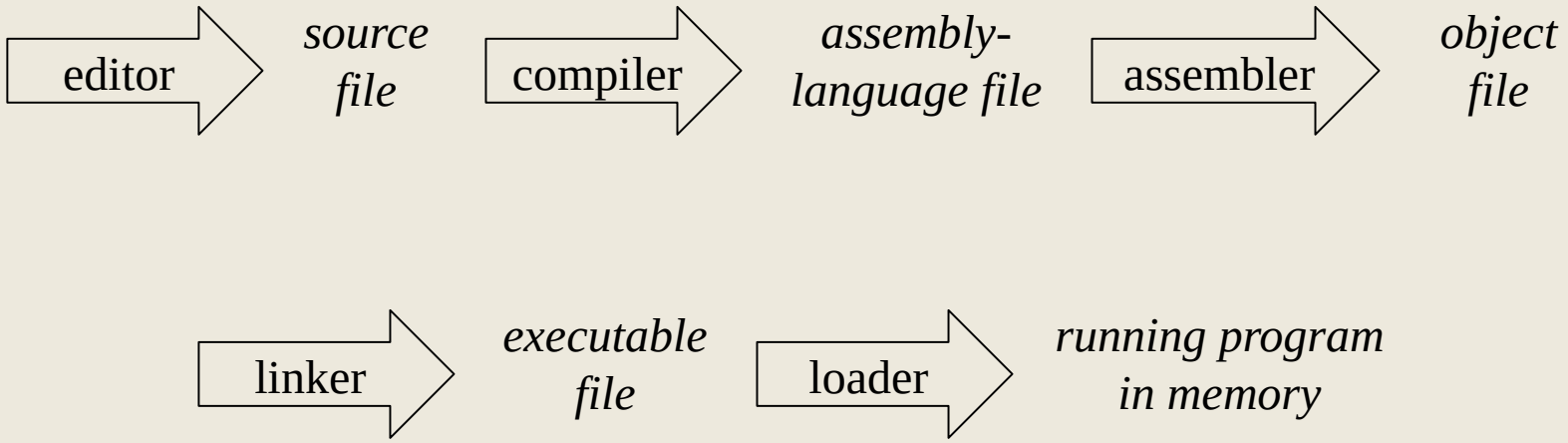

i:
0
main:
xxxx i
xx i x
xxxxxx
xxxx i
x fred
xxxx i
xxxxxx
xxxxxx
fred:
xxxxxx
xxxxxx
xxxxxx
loader
0:
20:
xxxx 80
(main)
xx 80 x
xxxxxx
xxxx 80
x 60
xxxx 80
xxxxxx
xxxxxx
60:
xxxxxx
(fred)
xxxxxx
xxxxxx
80:
0
(i)

Abrindo um parêntese

- Sobre memória
  - No nosso exemplo, estamos assumindo um modelo bastante simples de memória
  - Um "vetor de bytes"
  - Cada índice é o endereço
  - Antes da carga, o programa não sabe quais são os endereços de memória
  - O carregador encontra os endereços de memória e substitui os nomes

Execução

- Finalmente, o programa está totalmente convertido em linguagem de máquina



Agenda

- Paradigmas de linguagens
- Métodos de implementação
- Tempos do programa
- Elementos de linguagens
- Critérios de avaliação
- Critérios de avaliação

Vínculos

- **Vínculo**, **amarração** ou **binding** significa associar duas coisas
  - Especificamente, associar uma propriedade a um identificador de um programa
- No nosso programa de exemplo
  - Quais são os conjuntos de valores associados com o tipo de dados `int`?
  - Qual é o tipo de dados de `fred()`?
  - Qual é o endereço da função `main()`?
  - Qual é o valor da variável `i`?

Tempos de amarração

- Diferentes amarrações acontecem em "tempos" diferentes
- Os "tempos" padrões são
  - Tempo de definição da linguagem
  - Tempo de implantação da linguagem
  - Tempo de compilação
  - Tempo de ligação
  - Tempo de carga
  - Tempo de execução

Tempo de definição da linguagem

- Algumas propriedades são definidas durante a definição da linguagem
  - Significado das palavras reservadas
  - Em C:
    - Qual é o intervalo **mínimo** que o tipo de dados **int** deve suportar?

```
int i;
void main() {
  for (i=1; i<=100; i++)
    fred(i);
}
```

Tempo de implantação da linguagem

- Algumas propriedades são estabelecidas quando o sistema da linguagem é construído
  - Em C
    - Qual é o intervalo **exato** que o tipo de dados `int` deve ser capaz de representar?
    - Quantos bits possui um byte?
      - Na definição da linguagem é estabelecido que o tipo char possui sempre 1 byte
      - O byte é a quantidade mínima que pode ser endereçada em um computador (normalmente 8 bits)[1]

[1] https://www.iso.org/obp/ui/#iso:std:iso-iec:2382:ed-1:v1:en

Tempo de implantação da linguagem

- Para identificar os tamanhos dos tipos em C

```c
#include <stdio.h>
#include <limits.h>

int main(void)
{
  printf("Bits por byte: %d\n", CHAR_BIT);
  printf("sizeof (char): 1 byte (%d bits)\n", CHAR_BIT);
  printf("sizeof (int) : %d bytes (%d bits)\n",
      sizeof (int), sizeof (int) * CHAR_BIT);
  printf("Intervalo do (char): [%d, %d]\n",
      CHAR_MIN, CHAR_MAX);
  printf("Intervalo do (int) : [%d, %d]\n",
      INT_MIN, INT_MAX);
  return 0;
}
```

Tempo de compilação

- Algumas propriedades são vinculadas quando o programa está sendo preparado para compilação ou interpretação
  - Tipos das variáveis em linguagens que utilizam tipagem estática
  - Valores iniciais de variáveis estáticas
  - Valores de constantes
  - Declarações de variáveis em escopo limitado

Tempo de ligação

- Alguns vínculos ocorrem em tempo de ligação
  - Código de funções declaradas externamente (compilação separada)

```
int i;
void main() {
  for (i=1; i<=100; i++)
    fred(i);
}
```

Tempo de carga

- Algumas propriedades são vinculadas quando o programa é carregado em memória
  - Endereço de memória de variáveis estáticas e de funções

```
int variavel_estatica = 42;

int main(void)
{
  return variavel_estatica;
}
```

Tempo de execução

- Algumas propriedades são definidas apenas quando o programa está em execução
  - Valores das variáveis
  - Tipos de variáveis em linguagens com tipagem de dados dinâmica (*e.g.*, Python)
  - Declarações de variáveis em linguagens que utilizam escopo dinâmico
- Esses vínculos são denominados vínculos **tardios** ou **dinâmicos**
  - Todos os anteriores são denominados vínculos **antecipados** ou **estáticos**

Vínculo antecipado ou tardio?

- A pergunta mais importante sobre um vínculo
  - Tardio
    - Mais flexível em tempo de execução
    - Mais genérico
  - Antecipado
    - Geralmente mais rápido e seguro
    - Menos flexível
- **Podemos dizer muito sobre uma linguagem observando os tempos dos seus vínculos**

Agenda

- Paradigmas de linguagens
- Métodos de implementação
- Tempos do programa
- Elementos de linguagens
- Critérios de avaliação
- Critérios de avaliação

Variáveis

- Variáveis são abstrações de uma célula de memória de um computador
- Caracterizadas pelos atributos:
  - Nome
  - Endereço
  - Valor
  - Tipo
  - Tempo de Vida
  - Escopo

Nomes

- Identifica uma entidade (**identificadores**)
  - `int var1;`
- Quando ocorre a vinculação de uma variável ao identificador?
  - No caso geral, em tempo de compilação e desenvolvimento
  - Quando existem apelidos, isso pode ocorrer em tempo de execução

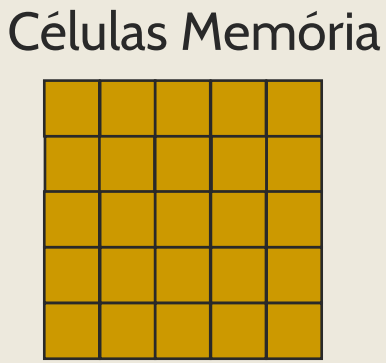
Nomes

- Alguns nomes são reservados
  - Palavras reservadas facilitam a implementação do compilador

```
int main() {
    crashAndBurn(
    return 0;
}
```

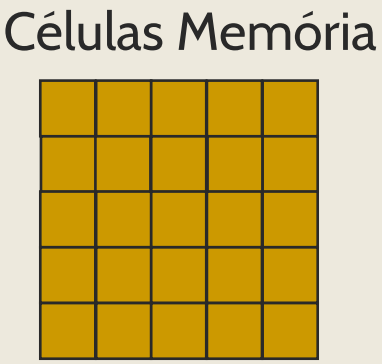
*A palavra reservada **return** não pode aparecer como argumento de chamada de função, então o compilador sabe que aqui tem uma chamada de função incompleta*

Endereços



- Endereço de memória ao qual uma variável está associada
  - Uma variável pode ter mais de uma célula de memória
    - O endereço de uma variável é o endereço de sua primeira célula de memória
  - Em C e C++, o operador & retorna o endereço de uma variável
  - Em Python, a memória é gerenciada automaticamente pela máquina virtual e normalmente os endereços não são conhecidos

Endereços

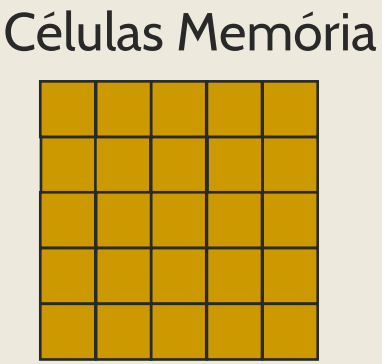


- Em linguagens dinâmicas, é comum que a variável seja vinculada ao endereço em tempo de execução
- Em linguagens estáticas, algumas variáveis são vinculadas ao endereço em tempo de carga, outras em tempo de execução

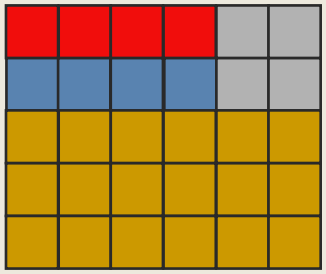
```
int vglobal = 40;
int main()
{
    int vlocal = 2;
    return vglobal + vlocal;
}
```

Apelidos (*aliases*)

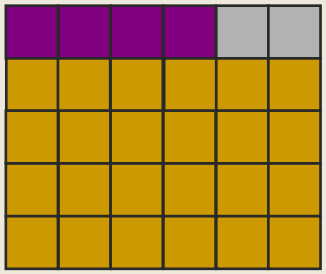
- Os Apelidos existem quando duas ou mais variáveis compartilham um endereço de memória
  - Em C, os campos de uma união compartilham o mesmo endereço




```c
struct s {
    int umInteiro;
    float umFloat;
};
```

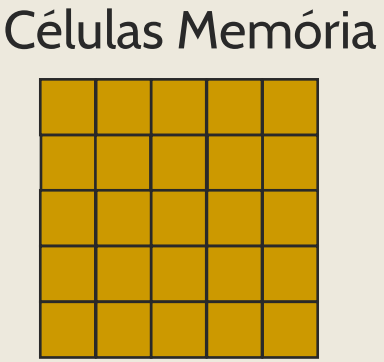


```c
union s {
    int umInteiro;
    float umFloat;
};
```



Apelidos (*aliases*)

- Os Apelidos existem quando duas ou mais variáveis compartilham um endereço de memória
  - Em Perl, um laço **`foreach`** utiliza apelidos para percorrer uma coleção



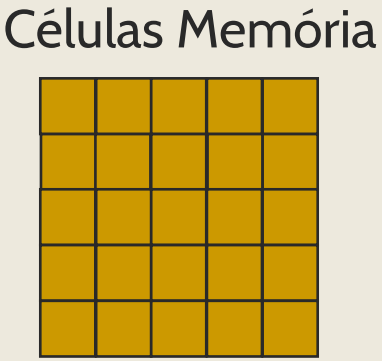

```perl
my @lista = (1, 2, 3);

foreach my $it (@lista) {
  # A variável $it é um apelido para cada elemento da lista,
  # portanto o incremento abaixo altera a lista
  $it++;
}

print "@lista \n";
```

Apelidos (*aliases*)

- Em Python, atribuições normalmente criam apelidos




```
>>> lista = [1, 2, 3]
>>> apelido = lista
>>> apelido.append(4);
>>> print(lista)
[1, 2, 3, 4]
```

- Apelidos podem ser muito úteis, principalmente para parâmetros de funções
  - Porém, podem dificultar o entendimento do programa

Tipos de dados

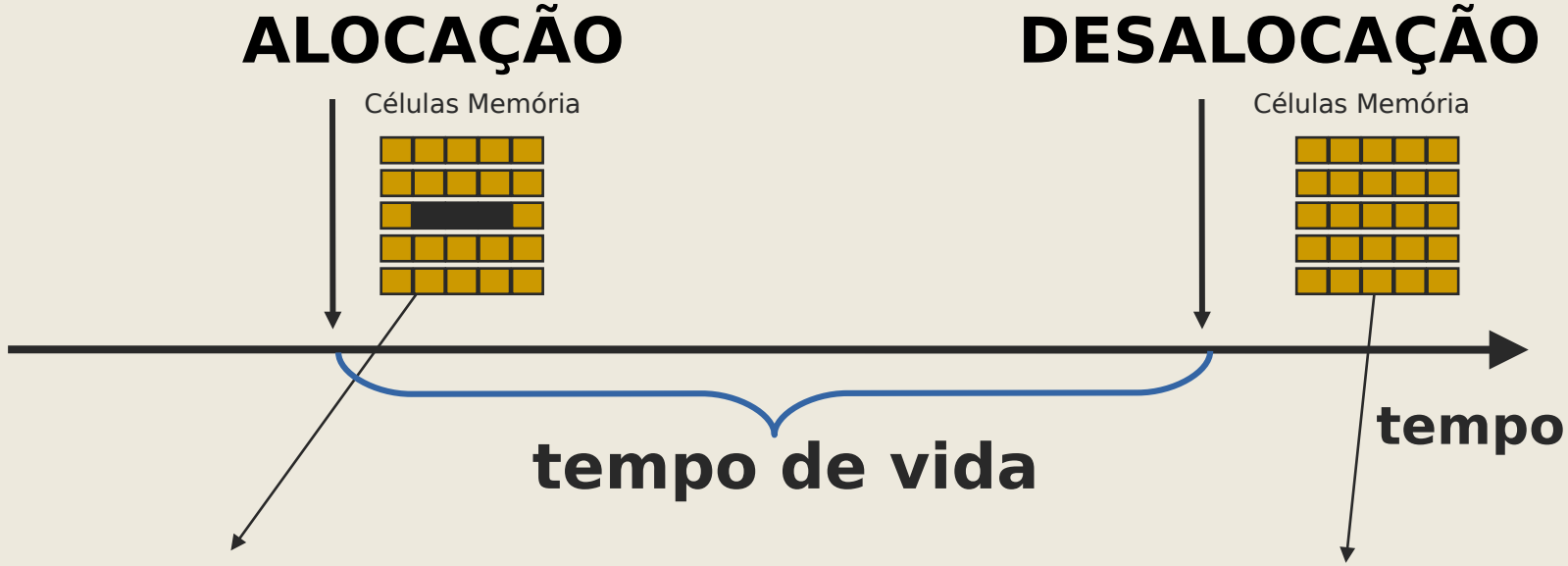
- Determina os valores permitidos para uma variável, bem como o conjunto de operações definidas para elas
  - **Tipagem estática**
    - O vínculo entre a variável e o seu tipo ocorre em tempo de compilação
  - **Tipagem dinâmica**
    - O vínculo entre a variável e o seu tipo ocorre em tempo de execução

Valor

- O **valor** de uma variável é o conteúdo da(s) célula(s) de memória associada(s)
  - Normalmente o vínculo entre a variável e o valor ocorre em tempo de execução

Tempo de vida

- O tempo de vida é o intervalo de tempo durante o qual um conjunto de células de memória está vinculado à uma variável



Conjunto de células alocadas para o armazenamento dos dados de uma variável

Células novamente livres, após a Desalocação

Escopo

- O escopo de uma variável representa a área do programa onde esta é visível
  - ```
    int main() {
       int y;
       /* ... */
       int x;
       /* ... */
     }
    ```
- Escopo e tempo de vida são conceitos bastante próximos
  - Frequentemente o tempo de vida de uma variável coincide com seu escopo
    ```

Blocos

- Um bloco é um novo escopo em um programa
  - Em Java, blocos são estabelecidos entre chaves
  - Em Pascal e Object Pascal, um novo bloco pode ser iniciado com a palavra reservada **begin** e deve ser finalizado com **end**
  - Variáveis declaradas dentro de um bloco normalmente são limitadas a esse escopo

Blocos

- Teste o seguinte programa:

```
#include <iostream>
using std::cout;
int main()
{
    int x = 0;
    {
        int x = 10;
        cout << x << "\n";
    }
    cout << x << "\n";
}
```

Blocos e escopo

- Algumas regras de escopo:
  - Procurar variáveis localmente
  - Procurar em ordem crescente do escopo até encontrar uma declaração para o nome da variável
- **Escopos aninhados**
  - Definição de escopos dentro de outros, formando uma sequência de escopos com sucessores e antecessores

Blocos e escopo

- A redefinição de uma variável com o mesmo nome (num escopo interior) de uma já existente num escopo exterior, permite "esconder" a definição exterior
  - C++, Pascal, ADA, etc. permitem o acesso a estas variáveis escondidas em escopos exteriores
  - Operador de resolução de escopo `::`

Espaço de nomes

- Um espaço de nomes é uma estrutura de escopo que permite agrupar nomes
  - Aumenta a verbosidade da linguagem
  - Aumenta a segurança e melhora a organização
- Algumas linguagens permitem manipulação do escopo
  - Em Python, usa-se **`import`** e **`import from`** para trazer um nome de um escopo para o escopo atual
  - Em C++, **`using`** `<escopo>::<nome>` tem a mesma finalidade

Agenda

- Paradigmas de linguagens
- Métodos de implementação
- Tempos do programa
- Elementos de linguagens
- Critérios de avaliação
- Critérios de avaliação

Como julgar paradigmas?

- Linguagens e paradigmas possuem pontos fortes e fracos
  - Diferentes paradigmas são mais adequados para diferentes problemas
    - Ao final do semestre, vocês devem poder decidir quais são mais adequadas
  - Não vamos julgar os paradigmas com base apenas no exemplo do algoritmo de Elucides!
    - É um tipo de função ideal para o paradigma funcional, um pouco menos óbvio para o imperativo e "forçado" para o paradigma lógico

Critérios de Avaliação

- Consideraremos quatro critérios
  - Legibilidade
  - Facilidade de escrita
  - Confiabilidade
  - Custo
- E várias características
  - Como observar a linguagem sob a ótica desses critérios?
    - Simplicidade, tipos de dados, expressividade, ortogonalidade etc.

Critérios de Avaliação

	Critérios		
Características	Legibilidade	Facilidade de escrita	Confiabilidade
Simplicidade	O	O	O
Ortogonalidade	O	O	O
Tipos de dados	O	O	O
Sintaxe	O	O	O
Abstração		O	O
Expressividade		O	O
Checagem de tipos			O
Exceções			O
Restricted Aliasing			O

O quarto critério, custo, não aparece na tabela porque não tem muita relação com os outros critérios e as características que os influenciam