

Universidade Federal da Paraíba

Centro de Informática

Departamento de Informática

Estrutura de Dados

Revisão – Linguagem C

▶ Tiago Maritan

▶ tiago@ci.ufpb.br

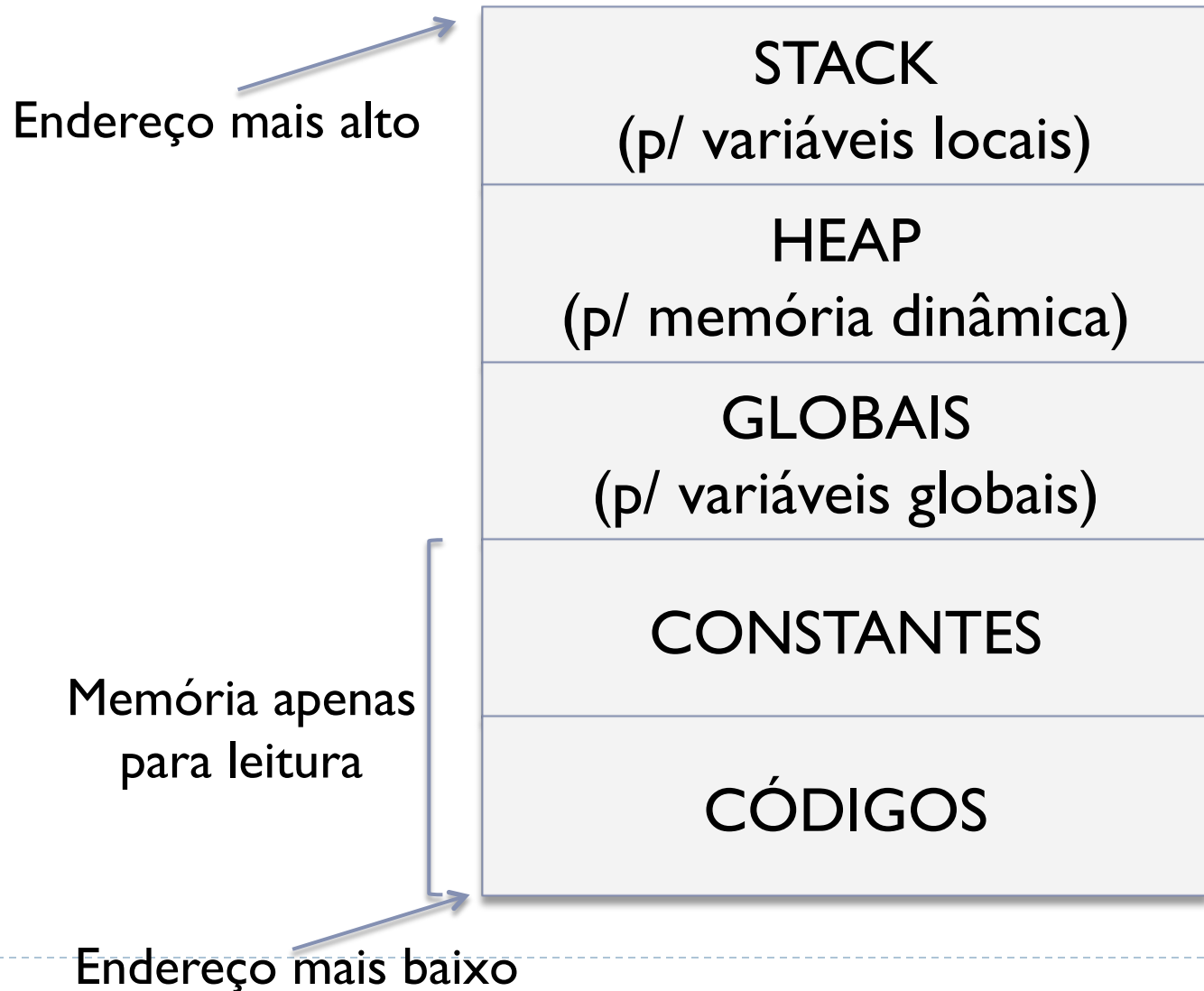
Assuntos Abordados

- ▶ Ponteiros
- ▶ Estruturas
- ▶ Módulos

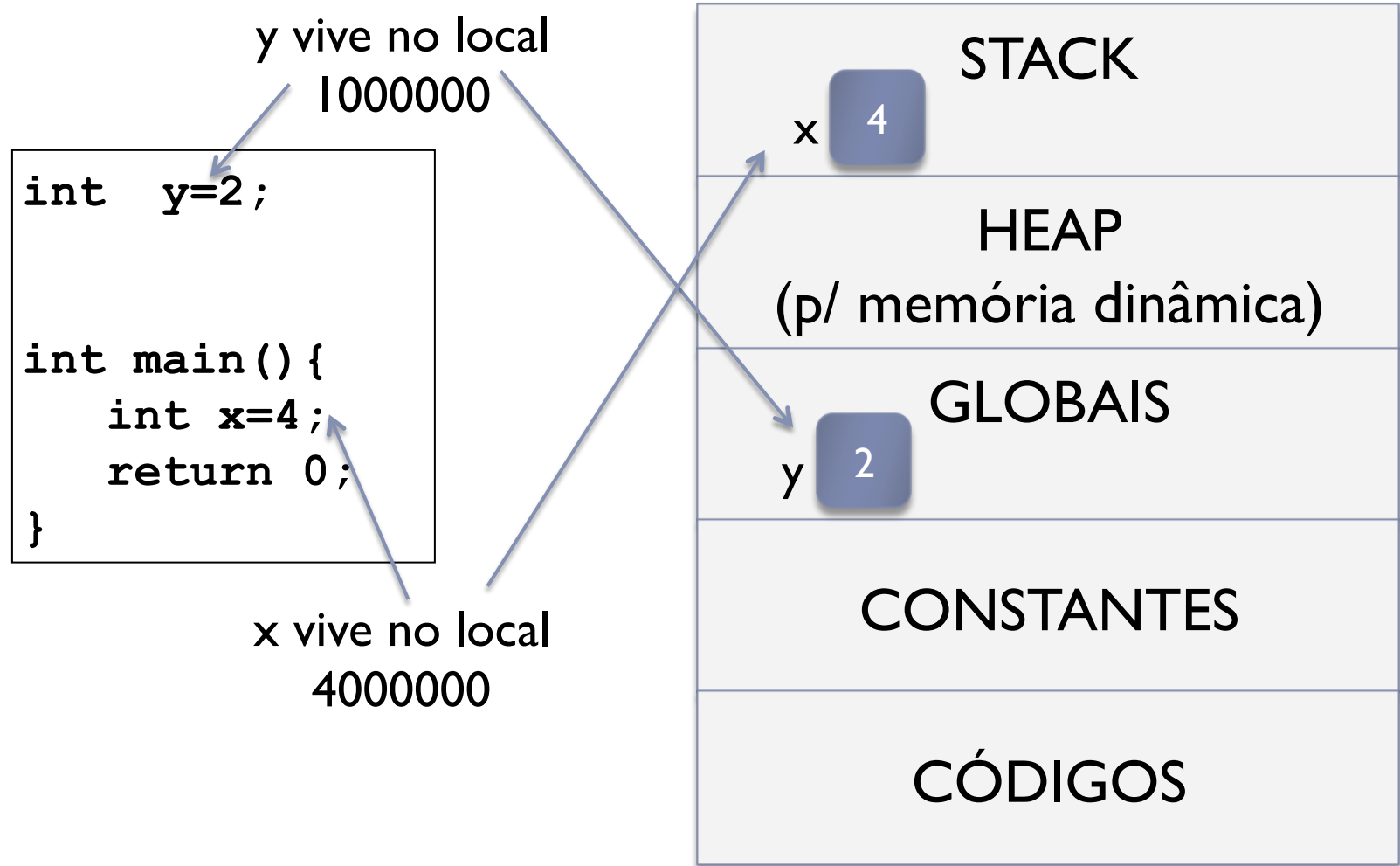
Ponteiros



Como a memória do computador está estruturada?



Como a memória do computador está estruturada?



Endereços e Ponteiros

► Endereço de uma variável

- Indica o **local (posição) em memória** onde ela é armazenada
- Determinado por meio do uso do operador de endereço **&**
- Exemplo:

```
int    x =4;  
...  
&x ←
```

Acessaria o endereço (localização) de x na memória.

Nesse exemplo, o valor 4000000.

Endereços e Ponteiros

- ▶ Não é permitido modificar o endereço de uma variável
 - ▶ Exemplo:

```
&x = 5000;      /* ILEGAL */
```

Ponteiros

- ▶ **Tipo especial de variável** que contém **um endereço**
- ▶ Funciona como um **apontador para outras variáveis**
 - ▶ **Pode apontar para um valor de qualquer tipo armazenado em memória**
- ▶ **Definição:**

```
<tipo apontado> *variavel-ponteiro;
```


Ponteiros

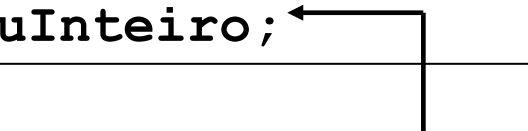
► Exemplo: Definição de Ponteiro

```
int meuInteiro;  
int *ponteiroParaInteiro;  
  
ponteiroParaInteiro = &meuInteiro;
```

Define `ponteiroParaInteiro` como um ponteiro capaz de apontar para uma variável do tipo `int`



`ponteiroParaInteiro` é um ponteiro para o tipo `int` iniciado com o endereço da variável `meuInteiro`



Indireção de um ponteiro

- ▶ Pode-se atribuir um valor ao conteúdo da posição de memória apontada por um ponteiro usando o operador de indireção

- ▶ **Exemplo:**

```
float  meuFloat = 3.14;  
float  *ponteiroParaFloat = &meuFloat;  
  
*ponteiroParaFloat = 1.6;
```

Atribui 1.6 ao conteúdo da posição de memória apontada por ponteiroParaFloat



Atribuições equivalentes

meuFloat = 1.6;

Indireção de um ponteiro

- ▶ O fato de C utilizar “*” para declaração e indireção de ponteiros pode causar alguma confusão em iniciantes.
- ▶ Exemplo:

```
float  meuFloat = 3.14;  
float  *ponteiroParaFloat = &meuFloat;  
...  
*ponteiroParaFloat = 1.6;
```

**O asterisco NÃO
representa o
operador de
indireção**

**O operador de indireção é
utilizado aqui**

Ponteiros

▶ Ponteiro nulo

- ▶ Não aponta para nenhum endereço válido
- ▶ Reconhecido pelo sistema como inválido
- ▶ Recebe o valor inteiro 0
 - ▶ Ou macro `NULL` (arquivo de cabeçalho `stdlib.h`)
- ▶ O programa é abortado quando se tenta acessar o conteúdo apontado por ele

▶ Exemplo:

<pre>char *p; ... p = 0;</pre>	Torna p um ponteiro nulo
---	---------------------------------

Estruturas



Estruturas

- ▶ **Estruturas são tipos estruturados** similares a arrays
- ▶ No entanto, seus **elementos (campos)** podem ser de **tipos diferentes**
 - ▶ Variáveis estruturadas heterogêneas
- ▶ Podem ser vistos como **tipos de dados definidos pelo programador**
 - ▶ Representam **dados de tipos diferentes relacionados** entre si

Definições de Estruturas

► Sintaxe:

```
typedef struct <rotulo_opcional> {  
    <tipo1> campo1;  
    <tipo2> campo2;  
    <tipoN> campoN;  
} nome-do-tipo;
```

- Define um tipo que é sinônimo de tudo que precede seu nome
- Rótulo só é necessário quando a estrutura tem auto-referência

Definições de Estruturas

► Exemplo:

```
typedef struct {  
    char    nome[30];  
    short   dia, mes, ano;  
} tRegistro;  
  
...  
  
tRegistro  registroDaPessoa, *ptrParaRegistro;
```


Iniciações de Estruturas

- ▶ Lista de valores deve ser colocada entre chaves
- ▶ N° de valores não deve exceder o n° de campos
- ▶ Cada valor deve ser compatível com o respectivo campo
- ▶ Exemplo:

```
typedef struct {  
    char    nome[30];  
    short   dia, mes, ano;  
} tRegistro;  
...  
tRegistro  registroDaPessoa = {"Jose da Silva", 12, 10, 1960};
```

Acesso a Campos de Estruturas

- ▶ Estrutura
 - ▶ **Utiliza-se o operador . (ponto)**
- ▶ Ponteiro para estrutura
 - ▶ **Utiliza-se o operador ->**
- ▶ O operador é colocado entre o nome da variável que representa a estrutura e o nome do campo

Acesso a Campos de Estruturas

► Exemplo:

```
typedef struct {  
    char    nome[30];  
    short   dia, mes, ano;  
} tRegistro, *tPtrParaRegistro;  
...  
tRegistro      registroDaPessoa  = { "Jose da Silva",  
                                     12, 10, 1960 };  
tPtrParaRegistro ptrParaRegistro = &registroDaPessoa;
```

Campo	Acesso com registroDaPessoa	Acesso com ptrParaRegistro
nome	registroDaPessoa.nome	ptrParaRegistro->nome
dia	registroDaPessoa.dia	ptrParaRegistro->dia
mes	registroDaPessoa.mes	ptrParaRegistro->mes
ano	registroDaPessoa.ano	ptrParaRegistro->ano

Acesso a Campos de Estruturas

- ▶ Um campo acessado de uma estrutura comporta-se como uma variável comum

- ▶ **Exemplos:**

```
registroDaPessoa.ano = 1976;
```

```
ptrParaRegistro->ano = 1976;
```

Módulos



Módulos

- ▶ Unidades que constituem um programa de certo tamanho
- ▶ Benefícios:
 - ▶ Torna o programa mais fácil de entender, manter e depurar
 - ▶ Permite dividi-lo entre uma equipe de programadores
 - ▶ Projeto de programação é mais fácil de ser gerenciado

Módulos

- ▶ Um módulo é dividido em duas partes:
 - ▶ **Arquivo de cabeçalho (extensão .h)**
 - ▶ **Arquivo de programa (extensão .c)**
- ▶ Módulo que contém **main()**
 - ▶ Possui apenas arquivo de programa contendo esta função
 - ▶ **Tipicamente denominado main.c**

Módulos

- ▶ Conteúdo típico de um arquivo de cabeçalho:
 - ▶ **Protótipos (alusões) de funções**
 - ▶ **Protótipos (alusões) de variáveis globais**
 - ▶ **Declarações de tipos**
 - ▶ **Definições de macros**

Alusão de Variáveis Globais

- ▶ Informa o compilador que uma variável é definida em outro ponto do programa.
- ▶ Similar a uma definição, mas não aloca memória
- ▶ Pode aparecer em vários arquivos que fazem parte do programa
- ▶ **Formato:**

```
extern <tipo> nome-variavel;
```

- ▶ **Exemplo:**

```
extern int gMinhaGlobal;
```

Alusões de Variáveis Globais

- ▶ **extern** é opcional
 - ▶ Sua ausência, no entanto, conduz ambigüidade
- ▶ Exemplo:

```
long gMinhaGlobal;
```

**Pode representar uma alusão
ou uma definição de variável**

Definições e Alusões de Variáveis Globais

► Recomendações:

- **Definição:** Omita **extern** e inclua uma iniciação
- **Alusão:** Omita qualquer iniciação e inclua **extern**

► Exemplos:

```
long gMinhaGlobal = 0L;
```

Definição

```
extern long MinhaGlobal;
```

Alusão

Alusão de Funções

- ▶ Permitem ao compilador reconhecer como legal uma chamada da função
 - ▶ Função aludida pode ser definida num arquivo que não aquele onde é feita a alusão

- ▶ Formato:

```
extern <tipo-ret> nome-funcao(tipos-param) ;
```

- ▶ Exemplo:

```
extern void Troca(int *, int *) ;
```

Módulos

▶ Arquivo de cabeçalho

- ▶ Nenhum componente deve gerar código
- ▶ Torna seus componentes disponíveis para outros arquivos que compõem o programa
- ▶ O arquivo que utiliza seus componentes inclui o arquivo de cabeçalho usando `#include`

▶ Arquivo de programa

- ▶ Seus componentes geram código
- ▶ Deve conter definições de variáveis e funções
- ▶ Pode conter qualquer componente típico de arquivo de cabeçalho
- ▶ Não deve jamais ser incluído em outro arquivo

Módulos

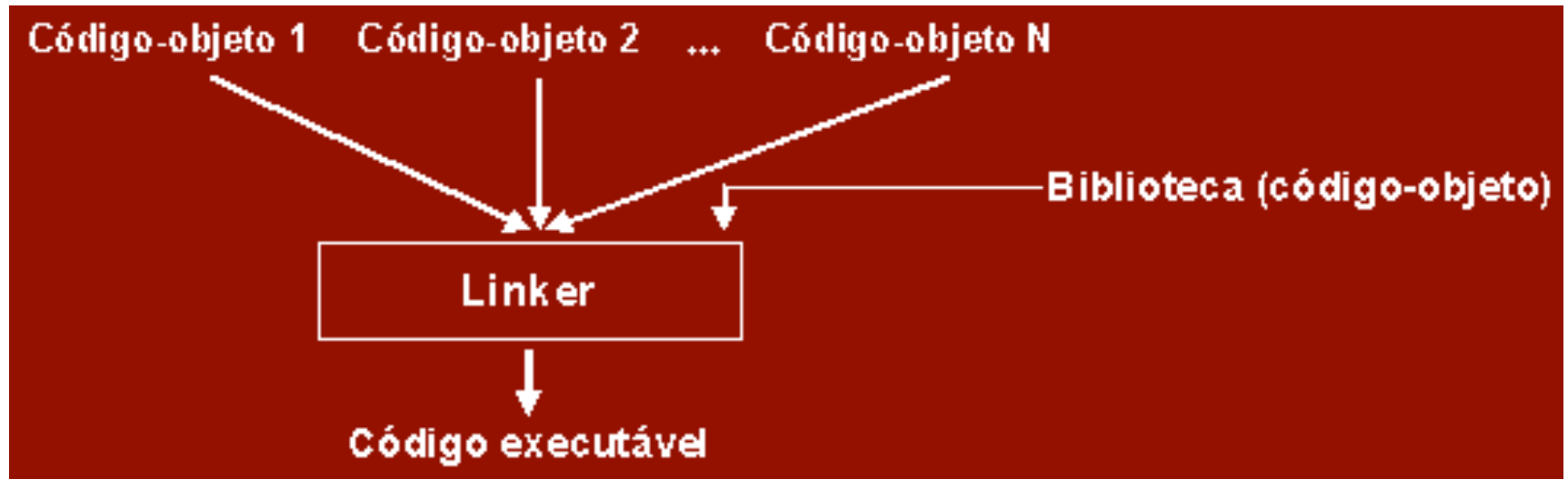
- ▶ Nomes dos arquivos que compõem um módulo
 - ▶ **Mesmo nome principal e extensões diferentes**
 - ▶ **.c** para o arquivo de programa
 - ▶ **.h** para o arquivo de cabeçalho
 - ▶ **Exemplo: Interface.h e Interface.c**

Diferenças entre Arquivo de Cabeçalho e Arquivo de Programa

ARQUIVO DE CABEÇALHO	ARQUIVO DE PROGRAMA
Não gera código	Gera código
Extensão usual: .h	Extensão usual: .c
Deve ser incluído por outros arquivos	Nunca deve ser incluído por outros arquivos

Como Construir Programas Multi-módulos

- ▶ **Construção de programas multi-módulos**
 - ▶ Cada arquivo que constitui o programa é compilado separadamente produzindo seu próprio código objeto
 - ▶ O editor de ligações combina arquivos objetos e cria um programa executável



Modelo de arquivo de cabeçalho de um módulo

```
#ifndef _Molde_H_  /* Substitua 'Molde' pelo nome do arquivo */
#define _Molde_H_  /* Idem */

/* Inclua aqui os arquivos de cabeçalhos necessários NESTE arquivo */
/* Declarações de macros */
/* Declarações de tipos */
/* Alusões das variáveis GLOBAIS definidas neste módulo */
/* Alusões das funções GLOBAIS definidas neste módulo */

#endif
```

Modelo de arquivo de programa de um módulo

```
/*  
 * Implementação do módulo: [Coloque aqui o nome do módulo]  
 * Autor: [Seu nome]  
 * Data de Criação: [Quando você começou a desenvolver este arquivo]  
 * Última alteração: [Data da última alteração feita neste arquivo]  
 * Descrição Geral: [Propósito do módulo]  
***/  
  
#include "Molde.h" /* Substitua 'Molde' pelo nome do arquivo */  
  
/* Inclua aqui outros arquivos de cabeçalho NESTE arquivo */  
  
/* Definições de variáveis locais e globais do módulo */  
  
/* Definições de funções locais e globais do módulo */
```

Módulo em C (Arquivo de cabeçalho .h)

lista.h

```
/* Definição da Estrutura de Dados */
# ifndef LISTA_H
# define LISTA_H

# define MAX 100 /* tamanho Máximo da Lista */

typedef struct {
    int dados[MAX]; /* vetor que contém a Lista */
    int n;          /* posição após o último elemento da Lista */
} tlista;          /* tipo Lista */

extern void cria (tlista *lista);
extern int vazia (tlista lista);
extern int cheia (tlista lista);
extern int tamanho (tlista lista);
extern int insere (tlista *lista, int pos, int dado );
extern int retira (tlista *lista, int pos, int *dado );

# endif
```

Módulo em C (Arquivo de programa .c)

```
# include "lista.h"
```

lista.c

```
void cria (tlista *L){  
    L->n = 0;  
}
```

```
int vazia (tlista L){  
    return (L.n == 0);  
}
```

```
•  
•  
•
```

Universidade Federal da Paraíba

Centro de Informática

Departamento de Informática

Estrutura de Dados

Revisão – Linguagem C

▶ Tiago Maritan

▶ tiago@ci.ufpb.br

