

Estrutura de Dados

Conceitos Fundamentais

Sumário

1 Tipos Primitivos

2 Vetor

3 Structs

4 Alocação Dinâmica

5 Lista

6 Fila

7 Pilha

8 Lista Duplamente Encadeada

9 Estruturas Avançadas

Tipos Primitivos

- › São os tipos básicos para armazenar dados simples.
- › Exemplos comuns: `int`, `char`, `float`, `double`.

00000000 00000000 00000000 00110110

23 bits são a mantissa (parte fracionária).

52 bits são a mantissa (parte fracionária).

01111000

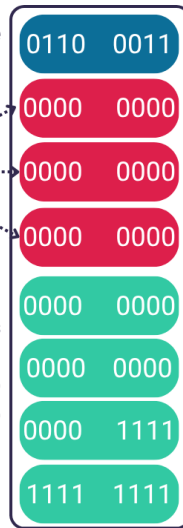
Tipos Primitivos em C III

char c = 'c'; → 1 byte
ASCII 'c' → 01100011

padding

int n = 270; → 4 bytes

0000 0000
0000 0000
0000 1111
1111 1111



Ponteiro

- › Variável que armazena o endereço de memória de outra variável.
- › Permite acessar e modificar dados indiretamente.
- › Ao passar um ponteiro para uma função, esta pode modificar a variável original.
- › Permite evitar cópias desnecessárias e maior eficiência.



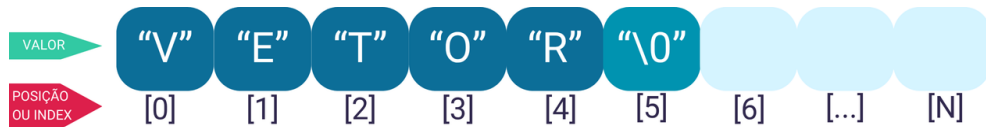
0x1024

Um ponteiro para qualquer tipo (**void**)
ocupa 8 bytes, ou seja,
64 bits em um arquitetura de 64 bits.

- › Em resumo, dados são armazenados em memória como sequências de bits (0s e 1s).
- › Cada tipo tem um tamanho fixo e maneira específica de se guardar.

Vetor

- › Vetores (*arrays*) são elementos do mesmo tipo (homogêneos) armazenados em posições contíguas.
- › Permitem acesso rápido a qualquer elemento por índice, podemos chamar de acesso aleatório.





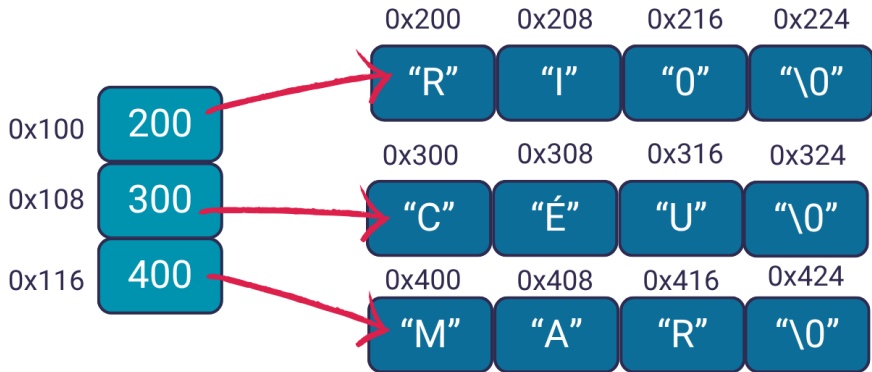


Representação

A representação física dos pode ser diferente da forma lógica, isto é, como se pensa nos dados.

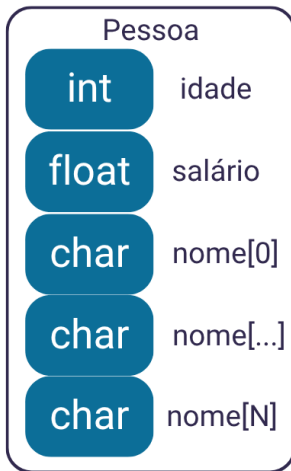
Quanto mais complexa a estrutura de dados, mais a forma física se difere da forma lógica visualizada pelo programador!

- › Uma matriz é um vetor de vetores ou um vetor bidimensional.
- › É possível criar estruturas multidimensionais.
- › Sua estrutura física é completamente diferente da sua representação lógica!
- › Matrizes 2D podem ser vistas como ponteiros para ponteiros (`int **`).
- › Em C, matrizes estáticas são blocos contínuos de memória.
- › No entanto, linha é um vetor separado, que pode estar alocado dinamicamente.

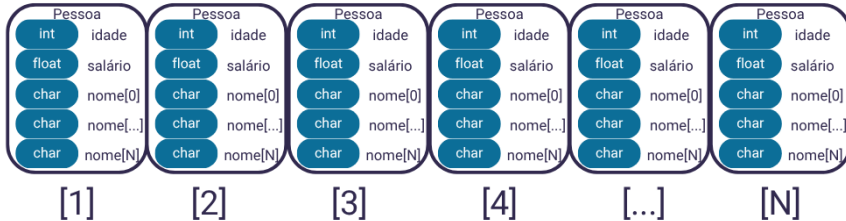


Structs

- › Structs ou registros agrupam diferentes tipos de dados para formar entidades complexas.
- › Permitem agrupar diferentes tipos de dados em um único bloco.
- › Úteis para representar entidades complexas, como uma pessoa ou produto.
- › Em conjunto com as operações referentes ao grupo de dados, formam Tipos Abstratos de Dados (TAD).
- › É como criar seu próprio tipos de dados, um tipo abstrato e heterogêneo!

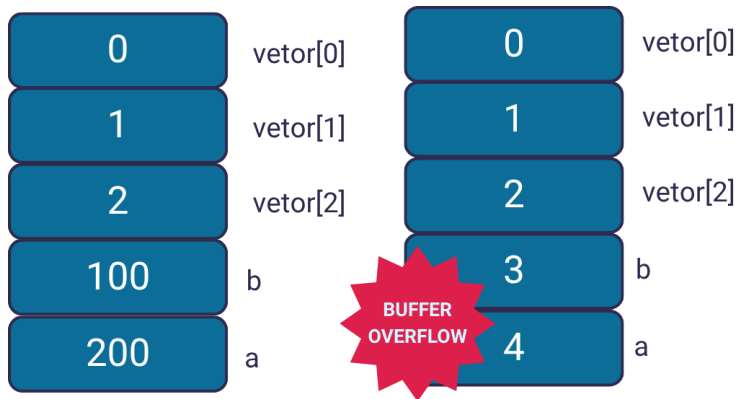


Vetor de *struct* do tipo Pessoa:
`struct Pessoa pessoas[n]`



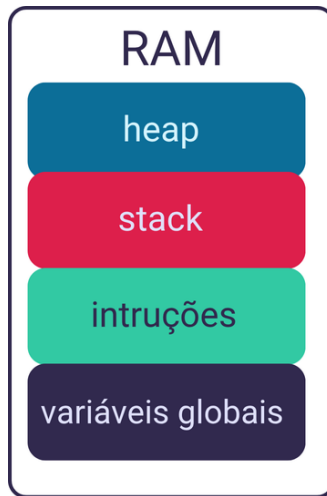
Alocação Dinâmica

Memória Dinamicamente Alocada I



- › Stack: memória para variáveis locais, rápida e automática.
- › Heap: memória dinâmica para alocação manual de dados que precisam durar.
- › Stack é limitada em tamanho; heap é flexível, mas precisa de cuidado (malloc, free).

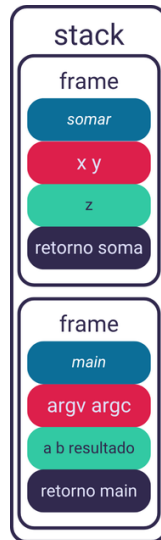
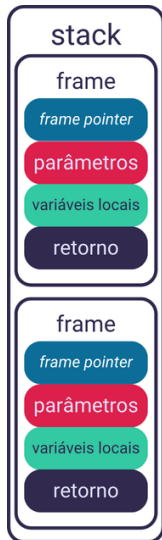
Memória Dinamicamente Alocada III



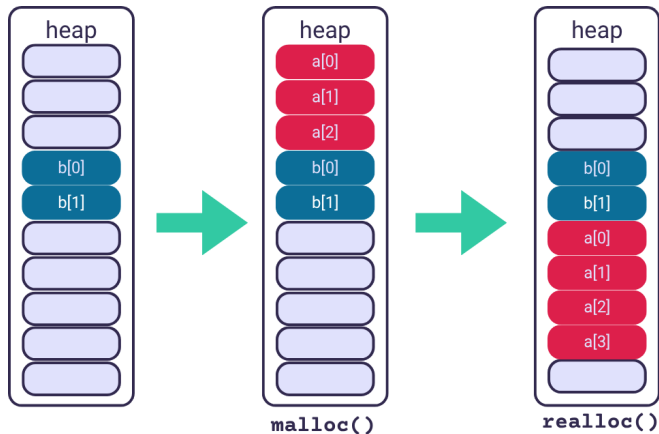
Memória Dinamicamente Alocada IV



Memória Dinamicamente Alocada V



Memória Dinamicamente Alocada VI



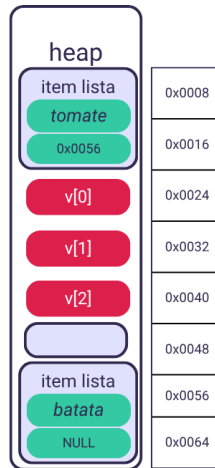
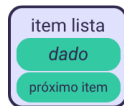
- Vetores têm acesso imediato (tempo constante).
- Estruturas com ponteiros são mais flexíveis, porém exigem mais operações para navegar.
- Alocação dinâmica é mais lenta que stack, impactando no desempenho.

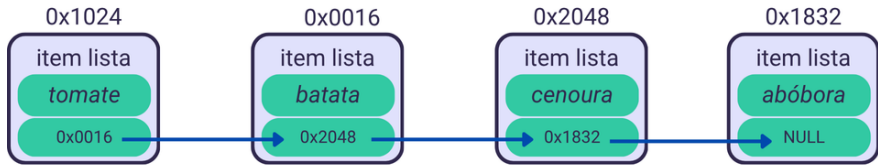
Lista

- › Elementos (nós) ligados por ponteiros.
- › Facilitam inserções e remoções sem mover muitos dados.
- › Exemplos: listas, filas, pilhas dinâmicas.

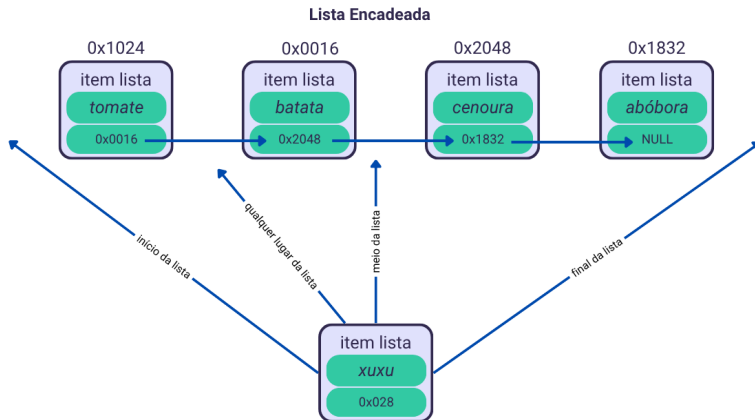
Lista Genérica

- › Lista capaz de armazenar qualquer tipo usando ponteiros.
- › Flexível, mas requer cuidado no uso e no tratamento dos tipos.



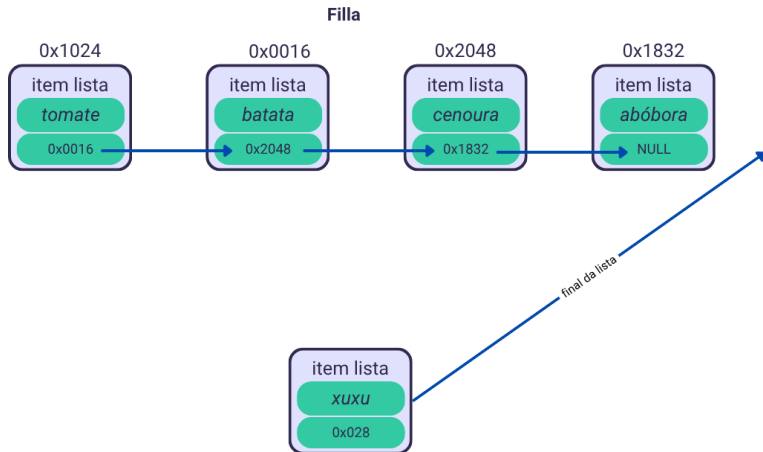


Lista Encadeada

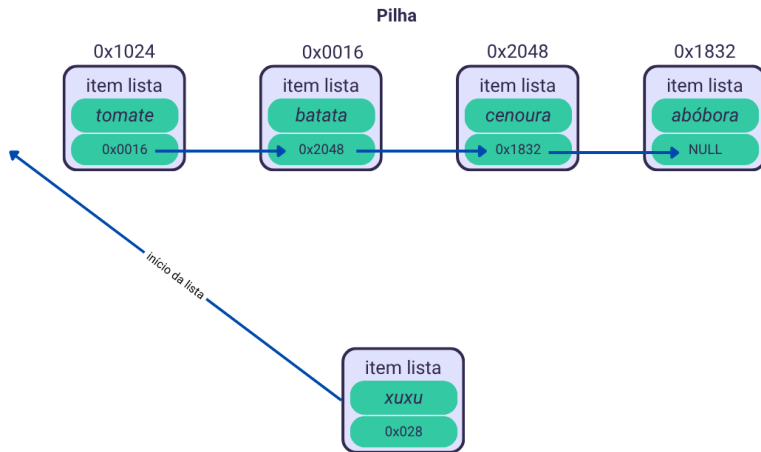


Fila

- › Estrutura onde o primeiro elemento a entrar é o primeiro a sair.
- › Usada para organizar tarefas e processos em ordem.



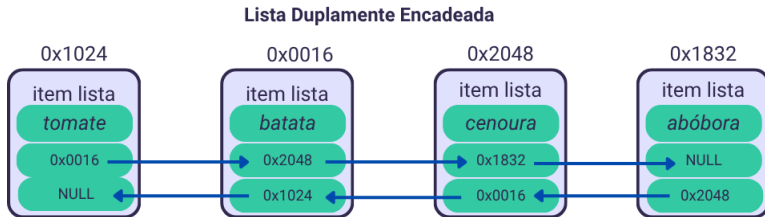
Pilha



Lista Duplamente Encadeada

- › Cada nó aponta para o próximo e o anterior.
- › Permite navegação para ambos os lados e facilita operações variadas.

Lista Duplamente Encadeada II



Estruturas Avançadas

- › Estruturas de dados avançadas permitem representar e manipular relações complexas entre informações, como hierarquias e conexões.
- › Em especial, as **árvores** organizam dados em forma hierárquica, onde cada nó pode ter múltiplos filhos.
- › A **recursão** é o método mais natural para percorrer e manipular árvores, pois reflete sua estrutura hierárquica.
- › Alternativamente, pode-se utilizar **pilhas** para realizar percursos iterativos, simulando o comportamento recursivo.

- › Operações de **busca** são fundamentais:
 - ›› **Busca linear**: percorre os elementos um a um ($O(n)$).
 - ›› **Busca binária**: aplicável apenas a vetores ordenados, dividindo o problema ao meio a cada passo ($O(\log n)$).
- › Algoritmos de **ordenação**, como *mergesort* e *quicksort*, utilizam recursão para dividir o conjunto em partes menores e ordená-las de forma eficiente ($O(n \log n)$).

Nota:

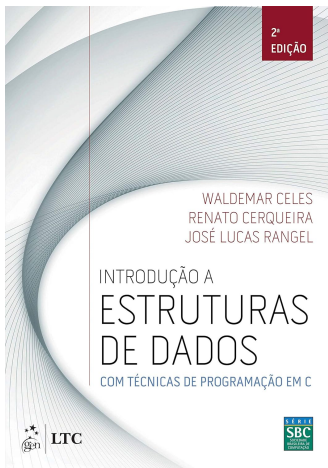
- › Estruturas de dados abstraem entidades do mundo real, permitindo modelar informações de forma lógica.
- › A **organização lógica** dos dados nem sempre corresponde à **organização física** na memória.
- › Quanto mais complexa for a estrutura, maior tende a ser essa diferença.

- **Grafos** representam conjuntos de vértices conectados por arestas, modelando relações entre elementos.
- As representações mais comuns são:
 - » **Matriz de adjacência**: consome mais espaço, porém de acesso direto.
 - » **Lista de adjacência**: mais eficiente para grafos esparsos.

- › Estruturas voltadas para desempenho incluem:
 - » **Tabelas hash**: permitem acesso rápido a valores com base em chaves.
 - » **Árvores balanceadas**: mantêm os dados ordenados, otimizando as operações de busca, inserção e remoção.

- › As **estruturas de mapeamento** armazenam pares **chave-valor**.
- › Permitem buscas rápidas do valor associado a uma chave específica.
- › Em C, são frequentemente implementadas por meio de **tabelas hash**.

(Deitel; Deitel, 2011) - Capítulo 14



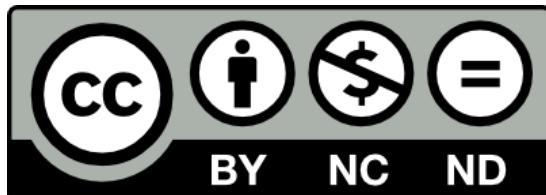
CELES, Waldemar; CERQUEIRA, Renato; RANGEL, José Lucas. **Introdução a estruturas de dados: com técnicas de programação em C**. Rio de Janeiro: Elsevier, 2004.

DEITEL, Paul; DEITEL, Harvey. **C: Como Programar**. 6. ed. São Paulo: Pearson Universidades, 2011.

PUGA, Sandra; RISSETTI, Gerson. **Lógica de programação e estruturas de dados, com aplicações em Java**. [S. l.]: Pearson, 2016. Livro eletrônico. ISBN 9788543019147.

TENENBAUM, Aaron M.; AUGENSTEIN, Moshe J.; LANGSAM, Yedidyah. **Estruturas de Dados Usando C**. 1. ed. São Paulo: Pearson Universidades, jun. 1995. ISBN 9788521201947.

Estes slides estão protegidos por uma licença Creative Commons



Este modelo foi adaptado de Maxime Chupin.