Estruturas de Dados Fundamentais

Prof. Marcos Caetano

(Material Base – Prof. Eduardo Alchieri)

Estruturas de Dados Fundamentais

- Todos os tipos abstratos de dados (pilhas, filas, deques, etc.) podem ser implementados usando um vetor (array) ou um tipo de estrutura encadeada (lista encadeada);
- Por isso, o vetor e a lista encadeada são chamados de estruturas de dados fundamentais;
- Estudaremos as seguintes estruturas fundamentais:
  - Vetores
  - Vetores multidimencionais (matrizes)
  - Listas Encadeadas

Vetores

- Provavelmente o vetor é a estrutura mais comum (simples) usada para agregar dados;
- **Estrutura homogênea:** conjunto de dados do mesmo tipo;
- Cada elemento pode ser acessado pela sua posição (índice);
- Alocação de memória é estática e sequencial;
  - Uma vez alocado, o tamanho de um vetor está fixado
  - Exemplo: int* a = (int *) malloc(sizeof(int)*5);

Vetores

- Exemplo: int* a = (int *) malloc(sizeof(int)*5);
- a[0] = 1

1	0	0	0	0

- a[1] = 2

1	2	0	0	0

- a[2] = 3

1	2	3	0	0

- a[3] = 4

1	2	3	4	0

- a[4] = 5

1	2	3	4	5

Vetores

- **Vantagens:**
  - Simplicidade;
  - Acesso direto;
- **Desvantagens:**
  - Tamanho fixo;

Vetores Multidimencionais

- Um vetor multidimencional de dimensão *n* é uma coleção de ítens que são acessados através de *n* expressões de subscritos;
  - Exemplo: (i,j)-ésimo elemento de um vetor x bidimencional é acessado pela expressão x[i,j]
    - x[i] seleciona o i-ésimo vetor unidimencional
    - Enquanto, x[i][j] seleciona o j-ésimo elemento deste vetor

- Como implementar em C uma Matriz 3x5?

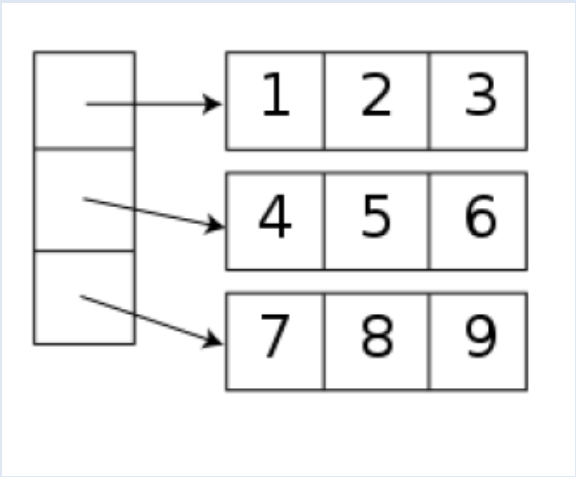
Vetores Multidimencionais

- Como implementar, utilizando alocação dinâmica, uma Matriz 3x5?

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

int main(){
    int i, j;
    int ** matriz = (int **) malloc(3*sizeof(int *));

    for(i=0; i < 3; i++)
        matriz[i] = (int *) malloc(5 * sizeof(int));

    matriz[1][2] = 7;

    for(i = 0; i < 3; i++)
        for(j = 0; j < 5; j++)
            printf("%d,%d=%d \n", i, j, matriz[i][j]);

    for(i=0; i < 3; i++)
        free(matriz[i]);

    free(matriz);

    return 0;
}
```

Matrizes

- Vetores bidimencionais
- Conjunto de dados do mesmo tipo (estrutura homogênea)
- Cada elemento pode ser acessado pela sua posição (índices)
- Alocação estática e sequencial
  - Uma vez alocado, o tamanho de uma matriz está fixado
- Representação (simplificada) na memória



Matrizes

- Vantagens:
  - Simplicidade
  - Acesso direto
- Desvantagens:
  - Tamanho fixo

Listas Encadeadas

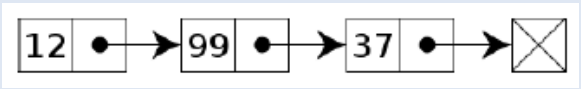
- É possível criar "vetores" de tamanhos variáveis usando alocação dinâmica ?
  - Elementos são criados somente quando necessário
    - Estão espaçados na memória
  - **Como saber onde está o próximo elemento do vetor ?**
    - **Ponteiros:** cada elemento guarda o endereço do próximo
      - Além disso, um ponteiro para o primeiro elemento e um ponteiro para o último elemento
    - Cada elemento possui uma ligação com o próximo elemento
  - Esta estrutura recebe o nome de lista encadeada ou lista ligada

Listas Encadeadas

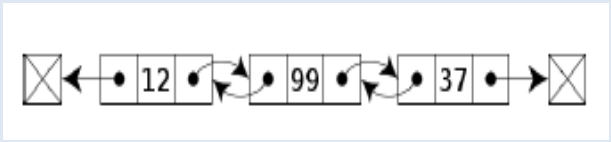
- Considere um vetor e uma lista encadeada, qual a complexidade de inserção de um item novo no início da estrutura ?
- **Vetor**
  - Criar um novo vetor de tamanho uma únidade maior
  - Inserir o novo item na posição inicial
  - Copiar os items do vetor velho para o novo vetor
  - Complexidade: O(n)
- **Lista encadeada**
  - Definir o primeiro item como o próximo do novo item
  - Definir o primeiro item como o novo item
  - Complexidade: O(1)

Listas Encadeadas

- **Uma definição para lista encadeada**
  - Conjunto ordenado de EDs (elementos) com acesso sequencial onde inclusão, exclusão e consulta de seus elementos podem acontecer de forma aleatoria;
  - **Aberta:** último elemento (nó) aponta para nada (null)
  - **Fechada (circular):** último nó aponta para o primeiro
  - Simplesmente encadeada: referência para o próximo nó



  - Duplamente encadeada: referência para o próximo nó e referência para o nó anterior



Listas Encadeadas

- Exemplos de utilização:

	Aberta	*Fechada*
Simples	lista telefônica	escalonamento Round-Robin
Dupla	histórico do navegador	cantos de um polígono

Listas Encadeadas

- Vantagem
  - Tamanho variável
- Desvantagem
  - Acesso sequencial

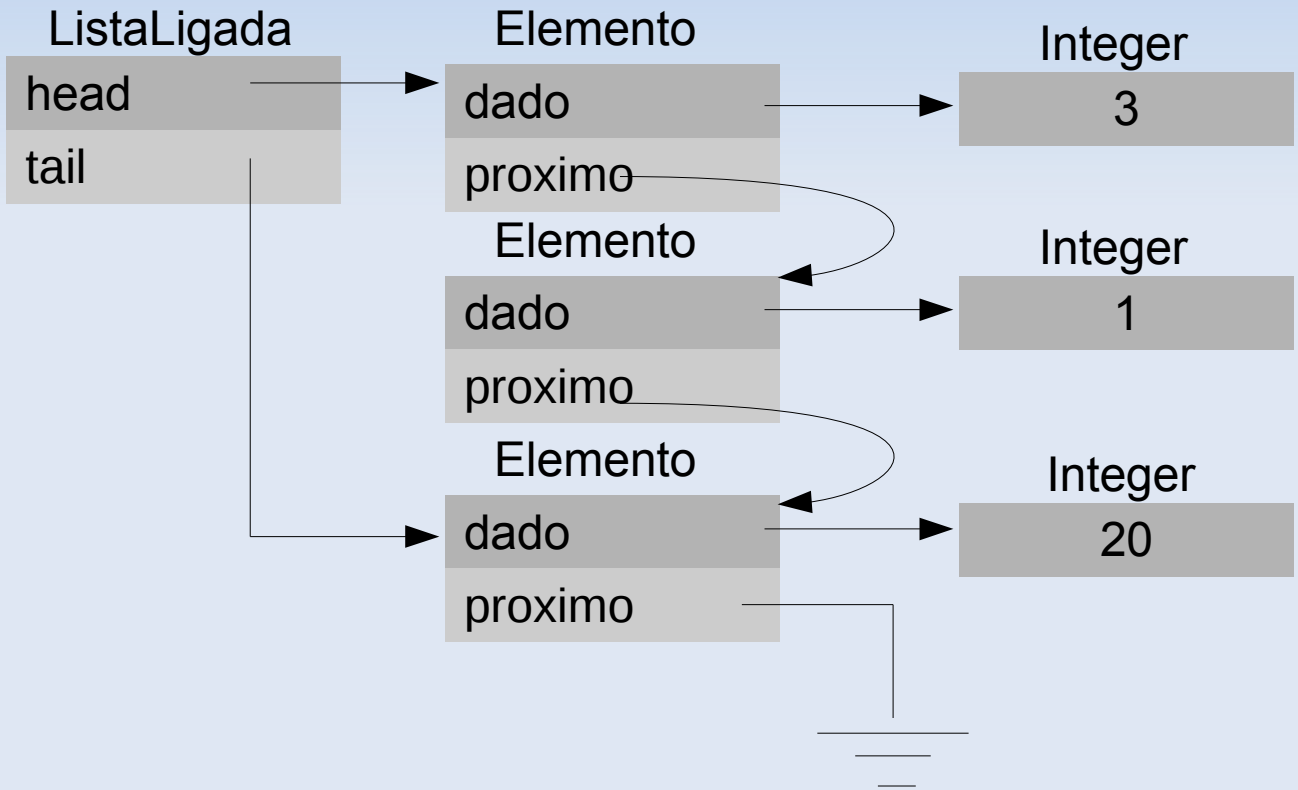
- Existem algumas variações de listas encadeadas
  - Adotaremos a seguinte estrutura:
    - Cada elemento aponta apenas para o próximo elemento (ou null para o último elemento);
    - Uma referência para o primeiro elemento
    - Uma referência para o último elemento

Listas Encadeadas

- Operações sobre as listas
  - Criar uma lista vazia
  - Inserir um item novo (em qualquer posição)
  - Remover um item
  - Localizar um item
  - Copiar a lista
  - Combinar duas ou mais listas
  - Dividir uma lista em duas ou mais
  - Etc.

Listas Encadeadas

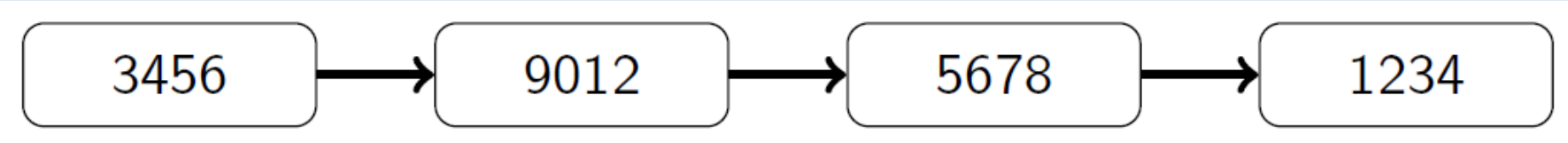
- Representação de uma lista encadeada na memória



- Vamos implementar uma lista encadeada!!!

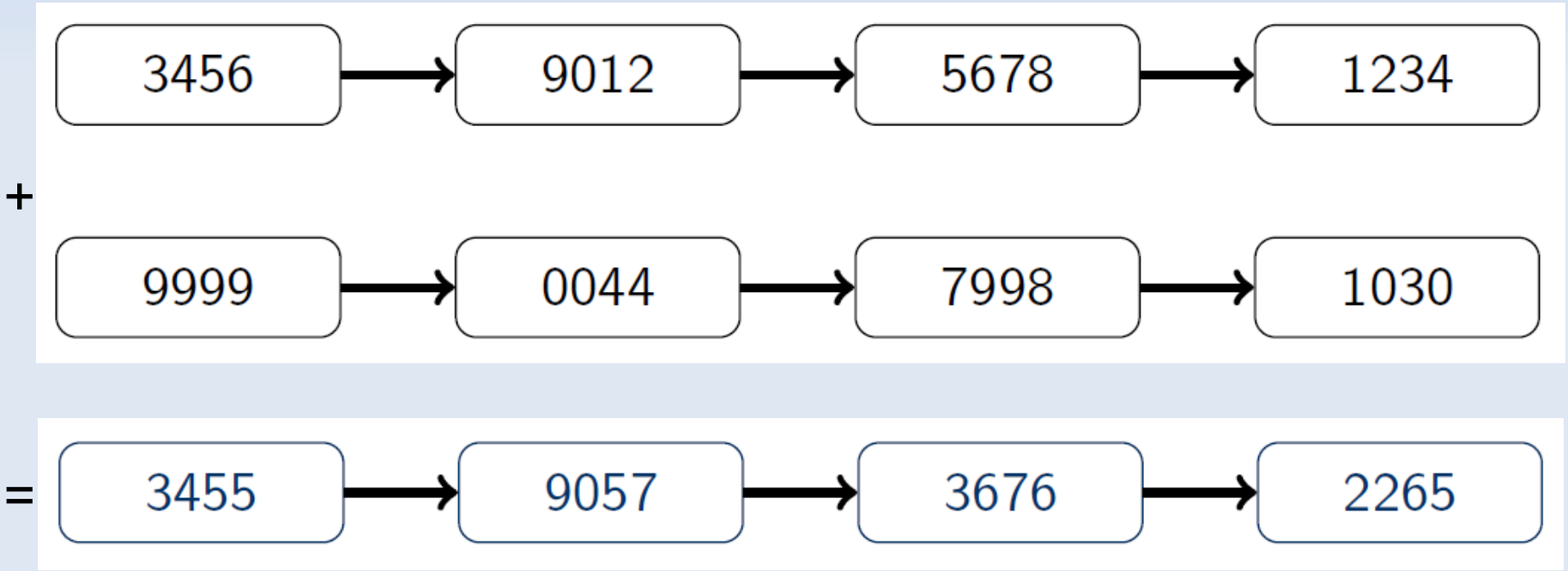
Listas Encadeadas

- Aplicação para listas simplesmente encadeada:
  - Soma de inteiros (super) longos
  - O hardware da maioria dos computadores só permite inteiros de um tamanho máximo específico. Como representar inteiros positivos de tamanhos arbitrariamente grandes ?
  - Usando uma lista para armazenar pedaços do valor: cada elemento armazena alguns dígitos do número.
  - Por exemplo, o número 1234 5678 9012 3456, seria armazenado como:



Listas Encadeadas

- Soma de inteiros (super) longos
  - Como escrever uma função que retorne a soma de dois números inteiros deste tipo ?



Listas Encadeadas

- Soma de inteiros (super) longos
  - Como recuperar o número somado para colocar na nova lista ?
    - Soma % 10000 (por que 4 zeros ?)
  - Como recuperar o "sobe 1" ?
    - Soma / 10000

Listas Encadeadas

- Algoritmo para soma de inteiros de duas listas l1 e l2
  - Considerando que que existe uma função getDado(i,l) que retorna o dado da posição i ou 0 (zero) caso o tamanho da lista l é menor do que i; e a função adicionaFim(d,l) para adicionar um dado d no final da lista l

```
sobe1 = 0
max = tamanho (número de dados) da maior lista entre l1 e l2
for(i=1; i <= max, i++){
    e1 = getDado(i,l1);
    e2 = getDado(i,l2);
    soma = e1+e2+sobe1;
    resultado = soma % 10000;
    sobe1 = soma / 10000;
    adicionaFim(resultado,l);
}
if(sobe1 > 0){
     adicionaFim(sobe1,l);
}
retorna l;
```

Matrizes Esparças

- Uma matriz é dita esparsa quando possui uma grande quantidade de elementos que valem zero
- Matrizes esparsas têm aplicações em problemas de engenharia, física (por exemplo, o método das malhas para resolução de circuitos elétricos ou sistemas de equações lineares)
- A matriz esparsa é implementada através de um conjunto de listas ligadas que apontam para elementos diferentes de zero. De forma que os elementos que possuem valor zero não são armazenados
  - Um espaço significativo de memória é economizado armazenando apenas os termos diferentes de zero
  - As operações usais sobre essas matrizes (somar, multiplicar, etc) também podem ser feitas em tempo muito menor se não armazenarmos as posições que contêm zeros