Estrutura de Dados 1

Estruturas lineares - Lista

Prof. Lucas Boaventura
lucas.boaventura@unb.br

Listas Encadeadas Simples

- **O que é uma lista encadeada?**
  - Estrutura formada por uma sequência de células
  - Cada célula possui:
    - Um dado
    - Um ponteiro para a próxima célula

Listas Encadeadas Simples

- Todos os objetos armazenados são do mesmo tipo
- A ligação entre as células é feita por endereços de memória
- Inicialmente, vamos considerar que o tipo de dado armazenado será **int**

Listas Encadeadas Simples

- Cada célula é representada por uma **struct**
- **Exemplo em C:**

```
1 struct registro {
2       int conteudo;
3       struct registro * prox;
4 };
```

Listas Encadeadas Simples

- **conteudo =** É o dado armazenado
- **prox =** É o endereço da próxima célula

```
1 struct registro {
2       int conteudo;
3       struct registro * prox;
4 };
```

Listas Encadeadas Simples

- É conveniente tratar a célula como um novo tipo
- Uso de typedef para simplificar o código:

```
1 typedef struct registro celula;
```

Listas Encadeadas Simples

- Exemplos de declarações de uma célula:

```
1 celula c;
2 celula * p;
```

- **c = uma célula**
- **p = um ponteiro para uma célula**

Listas Encadeadas Simples

- Quando temos a **célula diretamente:**
  - c.conteudo
  - c.proximo

- Quando temos **um ponteiro para a célula:**
  - **p->conteudo**
  - **p->prox**

Listas Encadeadas Simples

- A última célula da lista:
  - Possui **prox == NULL**

- NULL indica:
  - Não existe próxima célula
  - Fim da lista encadeada

...bles

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

typedef struct registro {
    int dado;
    struct registro * prox;
} Celula;

int main() {
    Celula p1;
    Celula p2;

    p1.dado = 5;
    p1.prox = &p2;

    p2.dado = 10;
    p2.prox = NULL;

    Celula * p = &p1;
    while(p) {
        printf("Dado = %d\n", p->dado);
        p = p->prox;
    }

    return 0;
}
```

Listas Encadeadas Simples

- Como fica a Organização na Memória?
  - As células não ocupam posições consecutivas na memória
- Elas são alocadas:
  - De forma dinâmica
  - Em locais imprevisíveis da memória
- A ligação é feita exclusivamente pelos ponteiros

Listas Encadeadas Simples

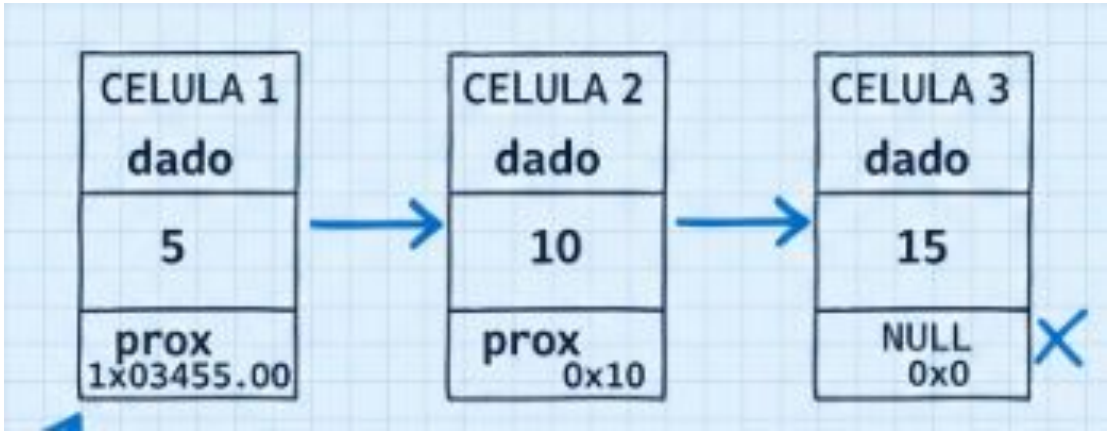
- O endereço de uma lista encadeada é:
  - **O endereço da primeira célula**

- Se **‘le’** é esse endereço:
  - Dizemos simplesmente que **‘le’ é a lista**

- Vamos ver a representação do que é o **‘le’**

Listas Encadeadas Simples

- **‘le’** é um ponteiro para célula
- Ele representa **Toda a lista encadeada**
- **‘le’ significa Lista Encadeada**



Listas Encadeadas Simples

- **Uma lista está vazia quando:**
  - **le == NULL**

- **Significa:**
  - Não existe nenhuma célula
  - A lista não possui elementos

Listas Encadeadas Simples

- Listas encadeadas são estruturas naturalmente **recursivas, mas podemos utilizar iteração também, sem problemas**

- **Observação fundamental:**
  - Se **le** é uma lista não vazia
  - Então **le->prox** também é uma lista

Listas Encadeadas Simples

- **Exemplo: Impressão Recursiva**
  - Imprime o primeiro elemento
  - Chama a função para o restante da lista

```
1 void imprime (Celula * le) {
2     if (le != NULL) {
3         printf("%d\n", le->dado);
4         imprime(le->prox);
5     }
6 }
```

Listas Encadeadas Simples

- **Exemplo: Impressão Iterativa**
  - Uso explícito de um ponteiro auxiliar
  - Mesma funcionalidade, mas usando laço

```
1 void imprime (Celula *le) {
2     for (Celula * p = le; p != NULL; p = p->prox)
3         printf("%d\n", p->dado);
4 }
```

Listas Encadeadas Simples

- **Recursão vs Iteração**

- **Recursiva:**
  - Código mais próximo da definição da lista
  - Mais elegante conceitualmente

- **Iterativa:**
  - Evita chamadas recursivas
  - Geralmente mais eficiente em memória

Listas Encadeadas Simples

- **Busca em Lista Encadeada**
  - Objetivo é verificar se um valor x pertence à lista

- **Ideia:**
  - Percorrer a lista célula por célula
  - Comparar x com o conteúdo de cada célula

Listas Encadeadas Simples

- **O que a Função de Busca Retorna?**
  - A função devolve o endereço da célula que contém x
- Caso x não exista **retorna NULL**
- A vantagem é não ser necessário usar variáveis booleanas

Listas Encadeadas Simples

- **Implementação da busca Iterativa:**

```
1 Celula * busca (int x, Celula *le) {
2     Celula *p = le;
3
4     while (p != NULL && p->dado != x)
5         p = p->prox;
6
7     return p;
8 }
```

Listas Encadeadas Simples

- **Condição do while:**
  - p != NULL → ainda há células
  - p->conteudo != x → valor ainda não encontrado

- **O laço termina quando:**
  - x é encontrado **OU**
  - O fim da lista é atingido

```
1 Celula * busca (int x, Celula *le) {
2     Celula *p = le;
3
4     while (p != NULL && p->dado != x)
5         p = p->prox;
6
7     return p;
8 }
```

Listas Encadeadas Simples

- **Se a lista estiver vazia:**
  - **le == NULL**
  - O laço não executa
  - A função retorna NULL
  - Comportamento correto e seguro

```
1 Celula * busca (int x, Celula *le) {
2     Celula *p = le;
3
4     while (p != NULL && p->dado != x)
5         p = p->prox;
6
7     return p;
8 }
```

Listas Encadeadas Simples

- **Busca Recursiva:**

```c
Celula * busca_rec(int x, Celula *le) {
    if (le == NULL) return NULL;
    if (le->dado == x) return le;
    return busca_rec(x, le->prox);
}
```

- **Casos base:**
  - Lista vazia → retorna NULL
  - Valor encontrado → retorna a célula

- **Passo recursivo:** Busca no restante da lista

Listas Encadeadas Simples

- **Em algumas situações a primeira célula não armazena dados úteis**
- **Essa célula funciona apenas como um marcador de início da lista**
- **Essa célula especial é chamada de cabeça da lista (head cell ou dummy cell)**

Listas Encadeadas Simples

- **Por que Usar Cabeça de Lista?**
  - Simplifica algoritmos de Inserção e Remoção
  - Evita casos especiais para Lista vazia e Inserção no início da lista
  - O ponteiro da lista nunca é NULL

- Uma lista com cabeça está vazia quando:
  - **le->prox == NULL**
  - Diferente da lista sem cabeça, aqui ‘le’ sempre aponta para uma célula válida

Listas Encadeadas Simples

- **Criação de uma lista encadeada vazia com cabeça:**
  - O campo conteudo/dado da cabeça é ignorado

```
1 Celula * cria_le() {
2     Celula * le;
3     le = malloc(sizeof(Celula));
4     le->prox = NULL;
5
6     return le;
7 }
```

Listas Encadeadas Simples

- **Impressão ignora a célula cabeça:**

```
1 void imprime (Celula *le) {
2     Celula *p;
3     for (p = le->prox; p != NULL; p = p->prox)
4         printf("%d\n", p->dado);
5 }
```

Listas Encadeadas Simples

- **Diferença: Lista com e sem Cabeça**

- **Sem cabeça:**
  - Lista vazia: **le == NULL**
  - Mais casos especiais

- **Com cabeça:**
  - Lista vazia: **le->prox == NULL**
  - Algoritmos mais simples e uniformes

Dúvidas?

- lucas.boaventura@unb.br