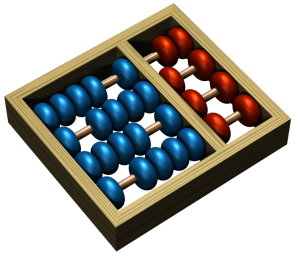
MC202 - Estruturas de Dados

Emilio Francesquini
francesquini@ic.unicamp.br

Instituto de Computação - UNICAMP

Aula 02 - 7 de março de 2017



UNICAMP

Disclaimer

- Esses slides foram preparados para um curso de Estrutura de Dados ministrado na UNICAMP
- Este material pode ser usado livremente desde que sejam mantidos os créditos dos autores e da instituição.
- Muitos dos exemplos apresentados aqui foram retirados de materiais preparados pelos Profs. Tomasz Kowaltowski e Orlando Lee da UNICAMP assim como do Prof. Paulo Feofiloff do IME-USP

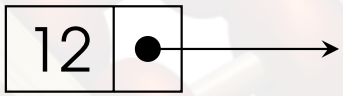
Lista Ligada

- Uma lista ligada (=lista encadeada=*linked list*) é uma estrutura de dados para armazenar uma sequência de elementos
  - Neste sentido é **parecida** com o bom e velho **vetor**
- Cada elemento é armazenado em uma **célula** (=nó=nodo)
- Cada célula também **armazena o endereço do próximo elemento da lista**

UNICAMP

Lista Ligada - Estrutura

```
struct reg {
  int          conteudo;
  struct reg *prox;
};
```



- Normalmente criamos um `typedef` para facilitar o uso

```
typedef struct reg celula;
```

- E então podemos declarar uma célula e um ponteiro para uma célula assim:

```
celula  c;
celula *p;
```

Listas Ligadas - Estrutura

- Se `c` é uma célula então `c.conteudo` é o conteúdo da célula e `c.prox` é o endereço da próxima célula
- Se `p` é o endereço de uma célula, então `p->conteudo` é o conteúdo da célula e `p->prox` é o endereço da próxima célula
- Se `p` é o endereço da última célula da lista então `p->prox` vale `NULL`

Listas Ligadas - Representando a lista

- Repare que a definição das células que compõem uma lista ligada é uma **definição recursiva**

```
struct reg {
  int          conteudo;
  struct reg *prox;
};
```

- Não por acaso, algoritmos que lidam com listas ligadas podem ser expressos recursivamente de maneira natural

UNICAMP

Listas Ligadas - Representando a lista

- Podemos representar a lista através da primeira célula. Assim, o endereço da lista ligada é, efetivamente, o endereço da sua primeira célula

```
celula *lista;
```

- Pergunta: Como verificar se uma lista está vazia?

```
int lista_vazia (celula *lista) {
    ...
};
```

UNICAMP

Listas Ligadas - Representando a lista

- Podemos representar a lista através da primeira célula. Assim, o endereço da lista ligada é, efetivamente, o endereço da sua primeira célula

```
celula *lista;
```

- Pergunta: Como verificar se uma lista está vazia?

```
int lista_vazia (celula *lista) {
    //equivalente a if (!lista), por que?
    if (lista == NULL)
        return 1;
    else
        return 0;
};
```

Listas Ligadas - Representando a lista

- Podemos representar a lista através da primeira célula. Assim, o endereço da lista ligada é, efetivamente, o endereço da sua primeira célula

```
celula *lista;
```

- Pergunta: Como verificar se uma lista está vazia?

```
int lista_vazia (celula *lista) {
    //equivalente a if (!lista), por que?
    if (lista == NULL)
        return 1;
    else
        return 0;
};
```

Listas Ligadas - Representando a lista

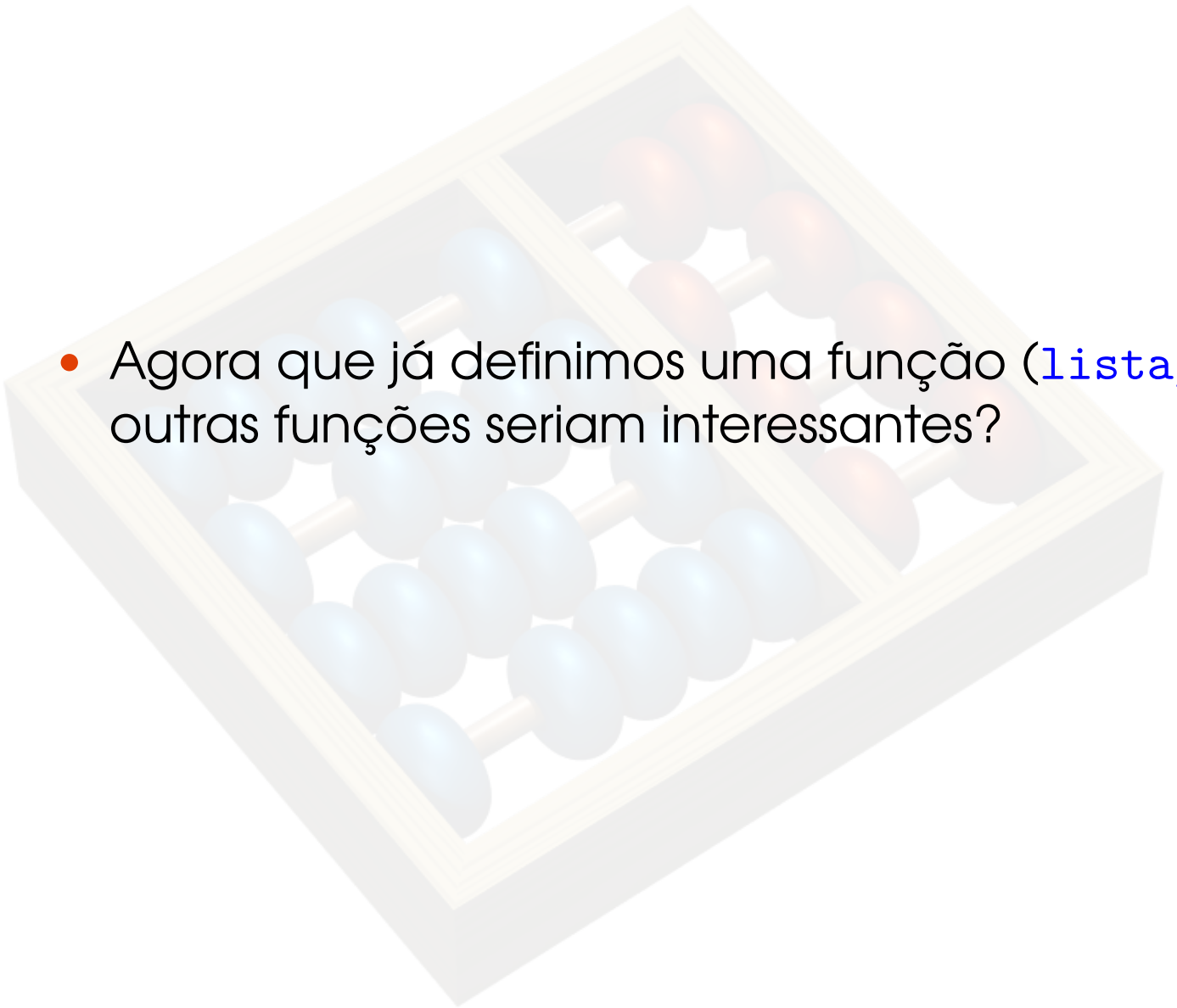
- Podemos representar a lista através da primeira célula. Assim, o endereço da lista ligada é, efetivamente, o endereço da sua primeira célula

```
celula *lista;
```

- Pergunta: Como verificar se uma lista está vazia?

```
//Versao super resumida
int lista_vazia (celula *lista) {
    return !lista;
};
```

Listas Ligadas - API



- Agora que já definimos uma função (`lista_vazia`), quais outras funções seriam interessantes?

UNICAMP

Listas Ligadas - API

- Agora que já definimos uma função (`lista_vazia`), quais outras funções seriam interessantes?

```
struct reg {
    int          conteudo;
    struct reg *prox;
};

typedef struct reg celula;

int     lista_vazia (celula* lista);
void    imprime_elementos (celula* lista);
celula* busca_elemento (int elem, celula* lista);
void    insere_elemento (int elem, celula* lista);
void    remove_elemento (celula* lista);
int     busca_e_remove (int elem, celula* lista);
```

Listas Ligadas - Varrendo elementos

```c
struct reg {
    int          conteudo;
    struct reg *prox;
};
typedef struct reg celula;

...

void imprime_elementos (celula* ll) {
  celula* atual = ll;
  while (atual) {
    printf("%d\n", atual->conteudo);
    atual = atual->prox;
  }
};
```

```c
void imprime_rec (celula* ll) {
  if (ll) {
    printf("%d\n", ll->conteudo);
    imprime_rec(ll->prox);
  }
};
```

Exercício: faça uma versão de `imprime_elementos` que imprime os elementos da lista na ordem inversa.

Listas Ligadas - Varrendo elementos

```
struct reg {
    int          conteudo;
    struct reg *prox;
};

typedef struct reg celula;

...

celula* busca_elemento (int elem, celula* lista) {
  celula* atual = lista;
  while (atual) {
    if (atual->conteudo = elem) return atual;
    atual = atual->prox;
  }
  return NULL;
};
```

Listas Ligadas - Varrendo elementos

```
struct reg {
    int          conteudo;
    struct reg *prox;
};

typedef struct reg celula;

...

celula* busca_elemento_rec (int elem, celula* lista) {
  if (!lista) return NULL;
  if (lista->conteudo == elem) {
    return lista;
  } else {
    return busca_elemento_rec(elem, lista->prox);
  }
};
```

Listas Ligadas - Inserindo elementos

```
void insere_elemento (int elem, celula* lista) {
  celula *nova;
  nova = calloc (1, sizeof (celula));
  nova->conteudo = elem;
  nova->prox = lista->prox;
  lista->prox = nova;
}

//por que a versao abaixo nao funciona?
void insere_elemento (int elem, celula* lista) {
  celula nova;
  nova.conteudo = elem;
  nova.prox = lista->prox;
  lista->prox = &nova;
}
```

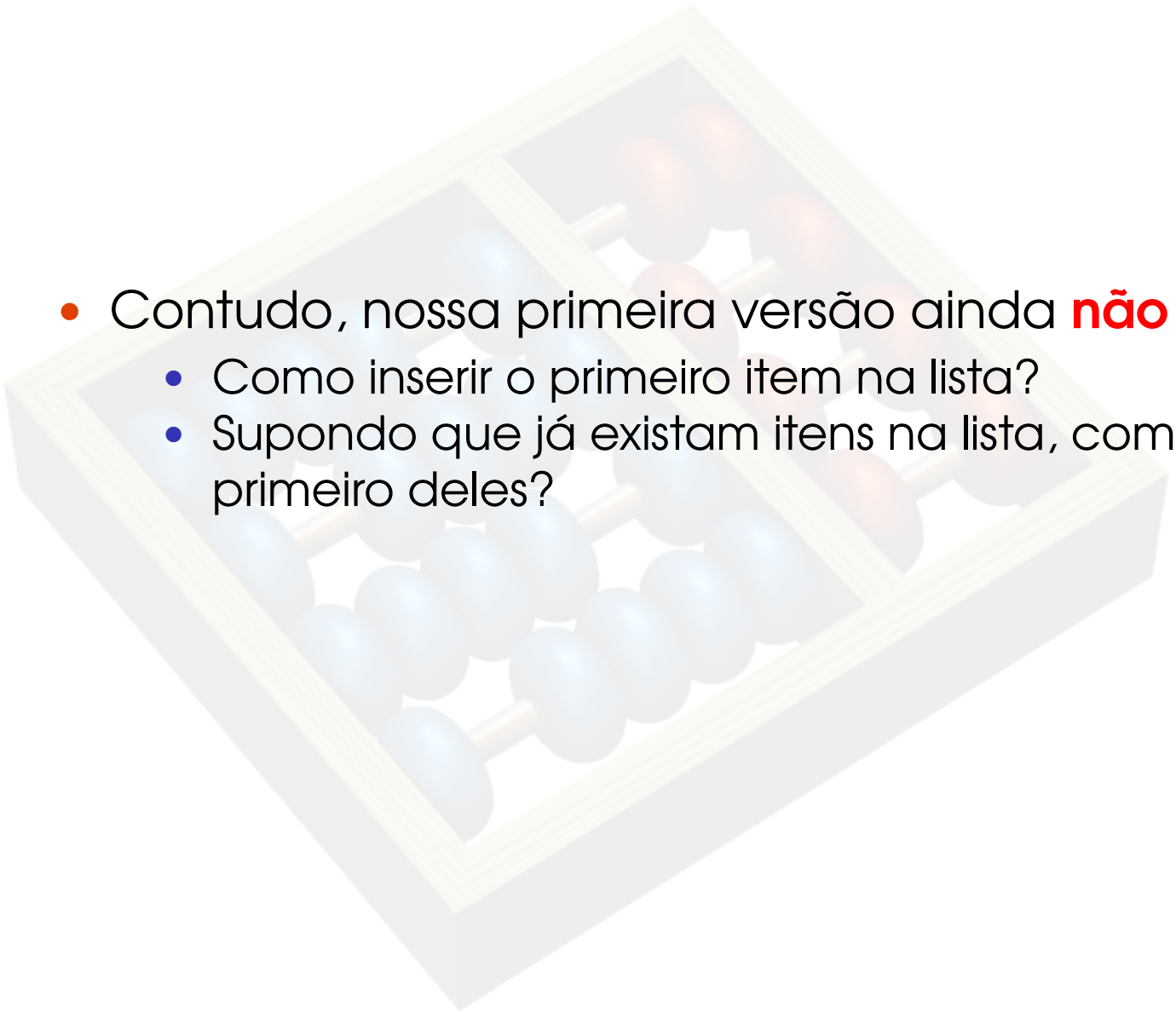
Listas Ligadas - Removendo elementos

```
//Remove a celula seguinte a celula apontada por p
void remove_elemento (celula *p) {
  celula *removida;
  removida = p->prox;
  p->prox = removida->prox;
  free (removida);
}
```

- Note que a operação remove a célula seguinte daquela apontada por `p`

Listas Ligadas - Manipulando elementos

- Contudo, nossa primeira versão ainda **não** funciona!
  - Como inserir o primeiro item na lista?
  - Supondo que já existam itens na lista, como remover o primeiro deles?



UNICAMP

Inserção - Tentativa de conserto 1

```
void insere_elemento_inicio (int elem, celula* inicial) {
  celula *nova;
  nova = calloc (1, sizeof (celula));
  nova->conteudo = elem;
  nova->prox = inicial;
  inicial = nova;
}
```

Inserção - Tentativa de conserto 1

```
void insere_elemento_inicio (int elem, celula* inicial) {
  celula *nova;
  nova = calloc (1, sizeof (celula));
  nova->conteudo = elem;
  nova->prox = inicial;
  inicial = nova;
}
```

```
No *primeiro;
int k = 7;
...
insere_elemento_inicio(k, primeiro); /* cria cópia de primeiro */
```

- Ao voltar da chamada, o valor da variável primeiro **não** foi alterado.

UNICAMP

Inserção - Tentativa de conserto 2

```
celula* insere_inicio (int elem, celula* inicial) {
  celula *nova;
  nova = calloc (1, sizeof (celula));
  nova->conteudo = elem;
  nova->prox = inicial;
  return nova; /* devolve o início da lista */
}

...
primeiro = insere_inicio(k, primeiro);
```

- Solução um tanto artificial, exige uma chamada especial se a lista está vazia.

UNICAMP

Inserção - Tentativa de conserto 3

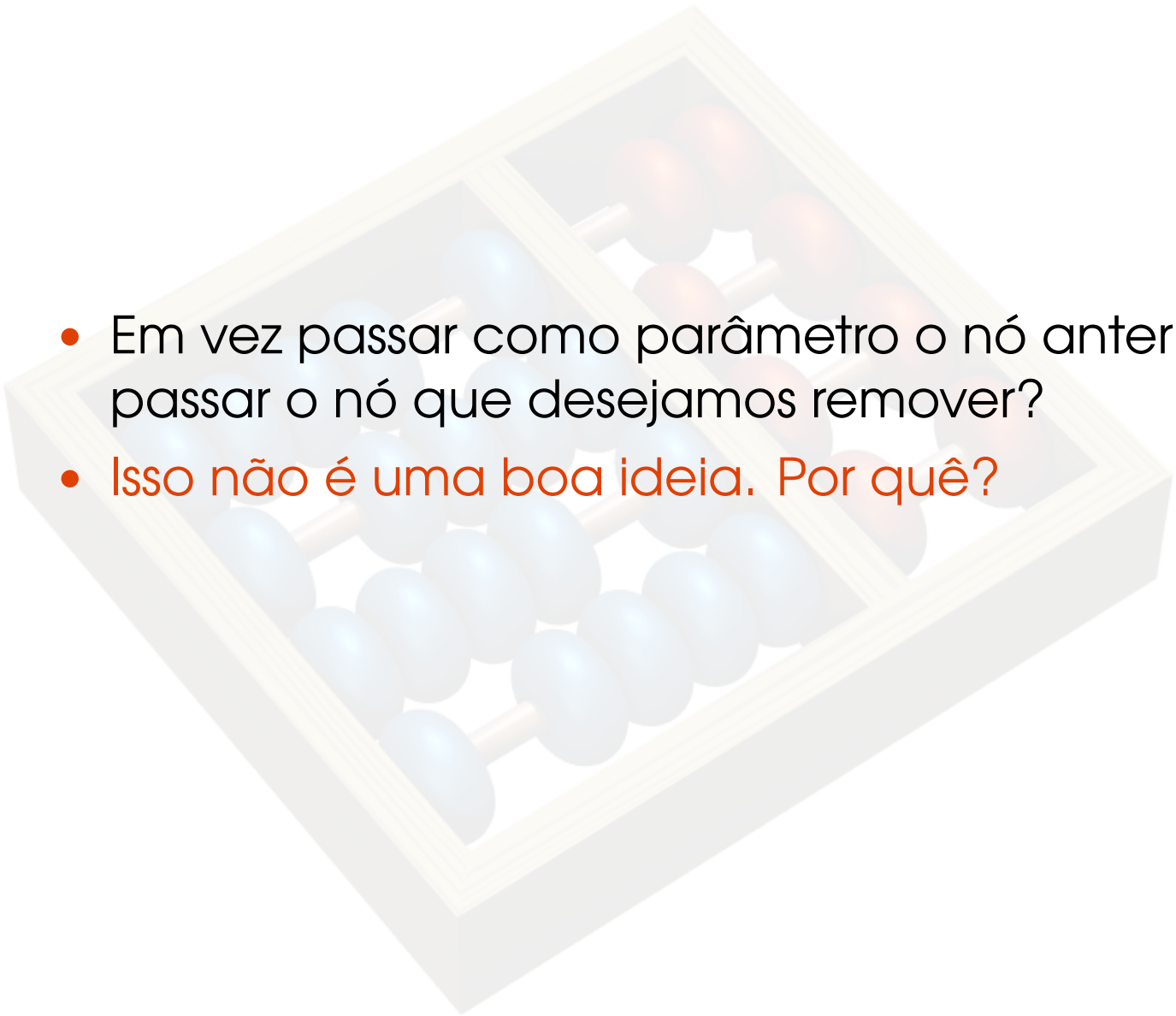
```
void insere_inicio (int elem, celula** pini) {
  celula *nova;
  nova = calloc (1, sizeof (celula));
  nova->conteudo = elem;
  nova->prox = *pini;
  *pini = nova;
}

...
insere_inicio(k, &primeiro);
```

- Um pouco difícil de ler por causa da indireção.
- Nenhuma das soluções é satisfatória porque elas diferem do caso geral.

UNICAMP

Remoção - Tentativa de conserto

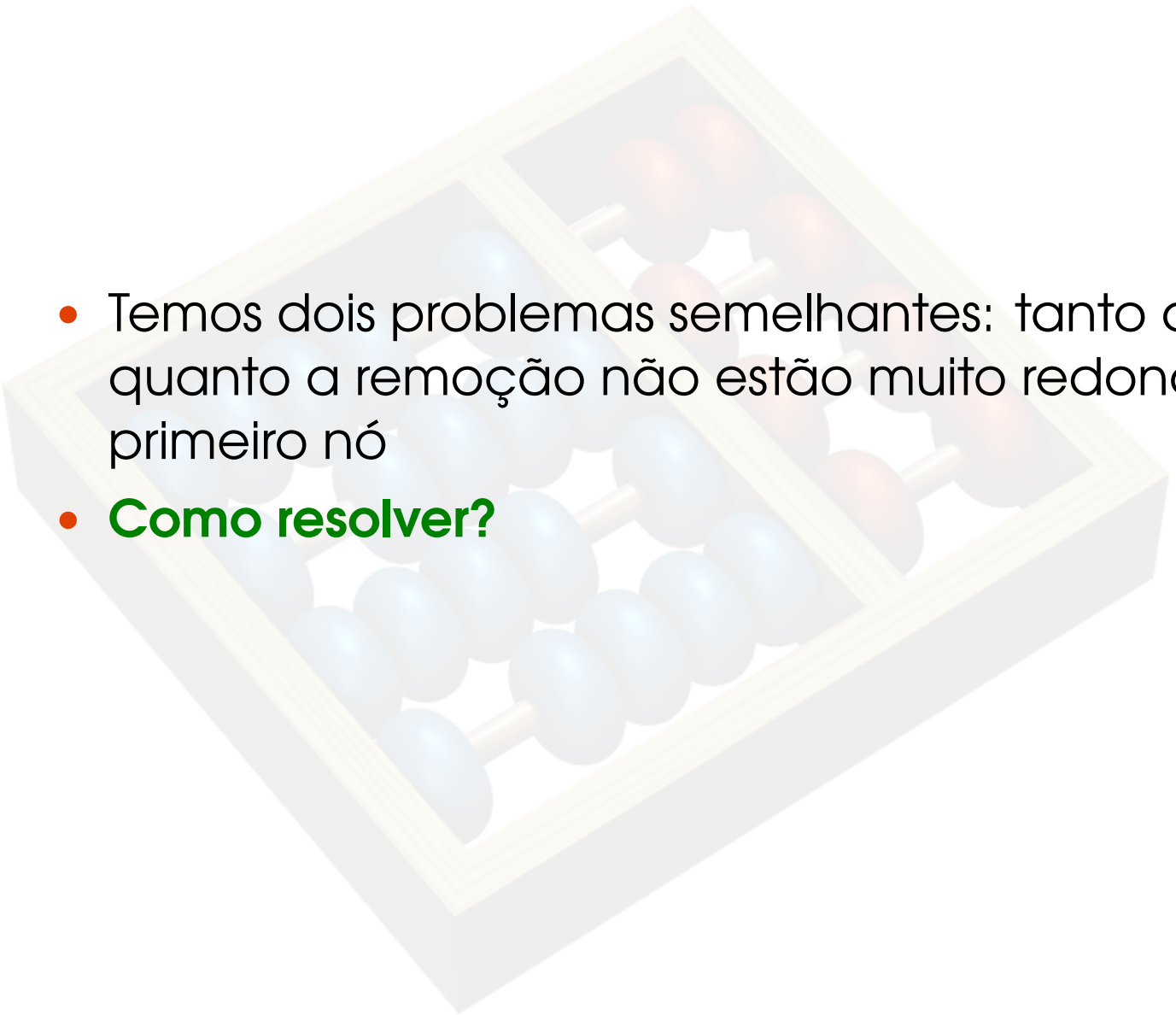


- Em vez passar como parâmetro o nó anterior, por que não passar o nó que desejamos remover?
- Isso não é uma boa ideia. Por quê?

UNICAMP

Consertando a inserção e a remoção

- Temos dois problemas semelhantes: tanto a inserção quanto a remoção não estão muito redondas no caso do primeiro nó
- **Como resolver?**



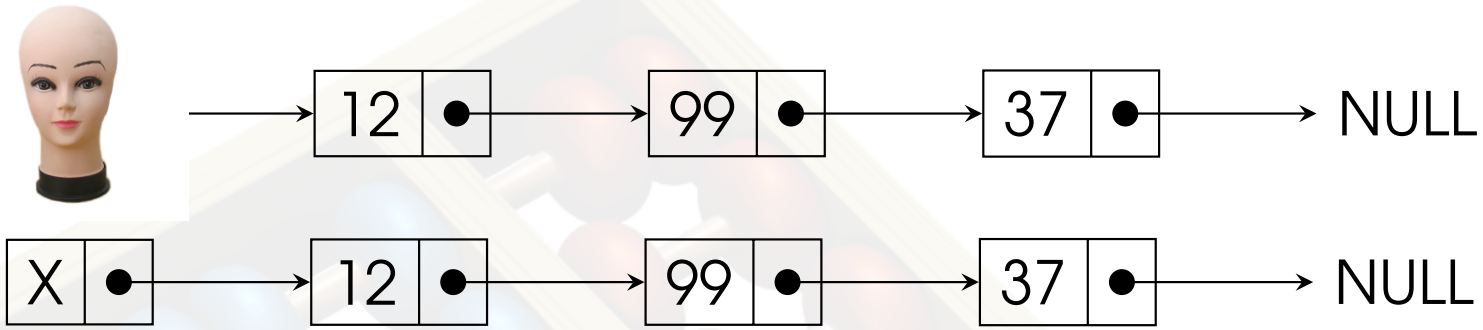
Consertando a inserção e a remoção

- Temos dois problemas semelhantes: tanto a inserção quanto a remoção não estão muito redondas no caso do primeiro nó
- **Como resolver?**

Solução: Listas com cabeças!

UNICAMP

Listas com cabeça



- A cabeça (=*head*) da lista ligada serve apenas como um marco de início. (Como tal, **ignoramos** o seu conteúdo.)
- Uma lista encadeada `lista` com cabeça está vazia se e somente se `lista->prox == NULL`.

```
//versao sem cabeca
int lista_vazia (celula *lista) {
    return !lista;
};
```

```
//versao com cabeca
int lista_vazia (celula *lista) {
  return !lista->prox;
};
```

UNICAMP

Listas com cabeça

- Para criar uma lista encadeada vazia com cabeça

```
celula *lista;
lista = malloc (sizeof (celula));
lista->prox = NULL;
```

- Para imprimir os elementos

```
void imprima (celula *lista) {
  celula *p;
  for (p = lista->prox; p != NULL; p = p->prox)
    printf ("%d\n", p->conteudo);
}
```

- Como adaptar as demais funções da API?

Manipulação em listas com cabeça

- Busca

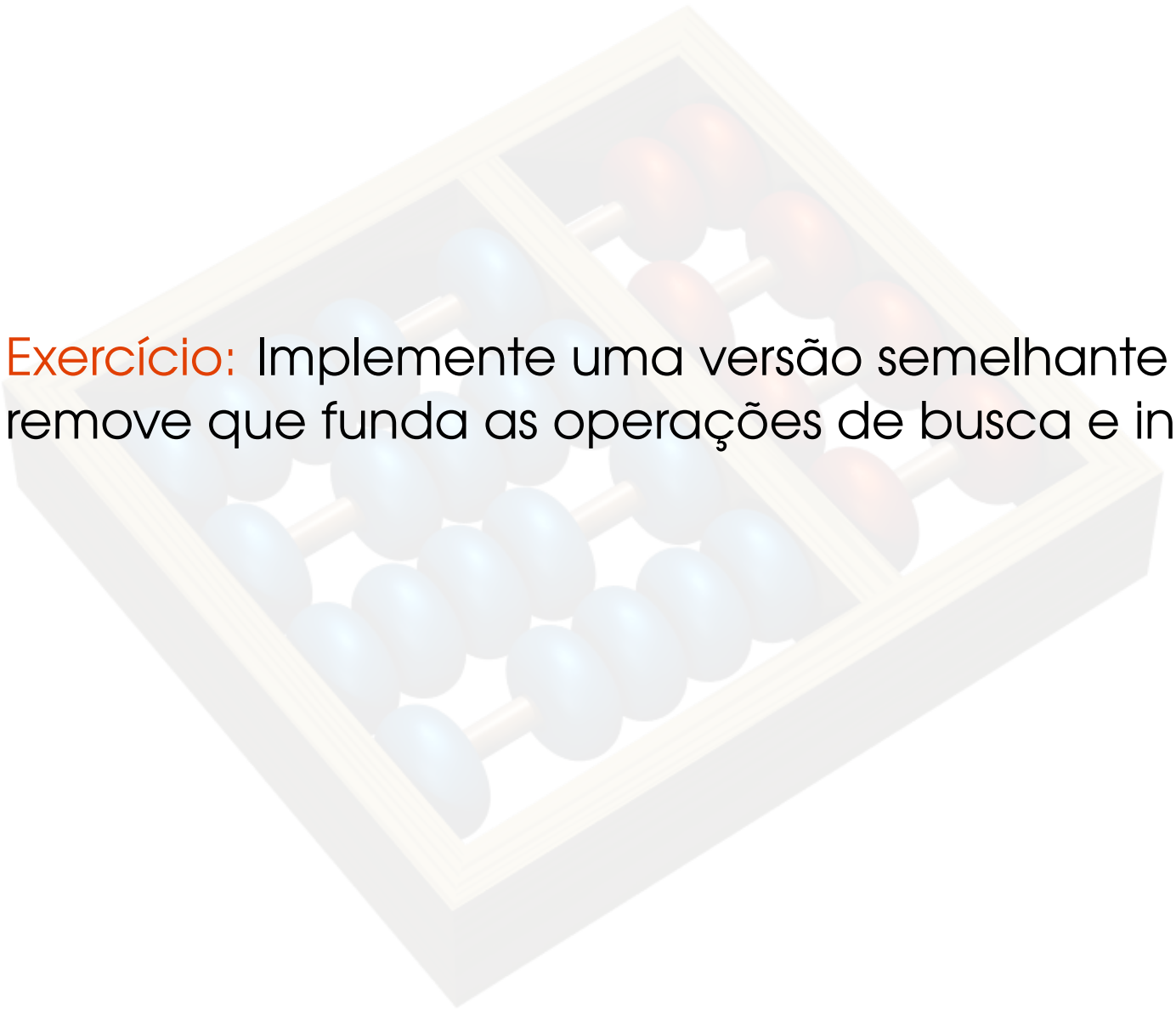
```
celula* busca_elemento (int elem, celula* lista) {
  celula* atual = lista->prox; //lista aponta para a cabeca
  while (atual) {
    if (atual->conteudo = elem) return atual;
    atual = atual->prox;
  }
  return NULL;
};
```

- Inserção e remoção
  - Não muda nada

Busca e remoção

```c
//remove da lista (com cabeça) a primeira célula que contém elem
void busca_e_remove (int elem, celula* lista) {
  celula *p, *q;
  p = lista;
  q = lista->prox;
  while (q != NULL && q->conteudo != elem) {
    p = q;
    q = q->prox;
  }
  if (q != NULL) {
    p->prox = q->prox;
    free (q);
  }
}
```

Busca e inserção



Exercício: Implemente uma versão semelhante ao busca e remove que funda as operações de busca e inserção.

UNICAMP