MC-202
Operações em listas e variações

Rafael C. S. Schouery
rafael@ic.unicamp.br

Universidade Estadual de Campinas

2º semestre/2023

Operações em lista ligada

Vamos ver três novas operações para listas ligadas

Operações em lista ligada

Vamos ver três novas operações para listas ligadas

```
1  typedef struct no *Lista;
2
3  struct no {
4    int dado;
5    Lista prox;
6  };
7
8  Lista criar_lista();
9  void destruir_lista(Lista l);
10 Lista adicionar_elemento(Lista l, int x);
11 void imprime(Lista lista);
```

Operações em lista ligada

Vamos ver três novas operações para listas ligadas

```
1  typedef struct no *Lista;
2
3  struct no {
4    int dado;
5    Lista prox;
6  };
7
8  Lista criar_lista();
9  void destruir_lista(Lista l);
10 Lista adicionar_elemento(Lista l, int x);
11 void imprime(Lista lista);
```

Operações em lista ligada

Vamos ver três novas operações para listas ligadas

```
1  typedef struct no *Lista;
2
3  struct no {
4    int dado;
5    Lista prox;
6  };
7
8  Lista criar_lista();
9  void destruir_lista(Lista l);
10 Lista adicionar_elemento(Lista l, int x);
11 void imprime(Lista lista);
12
13 Lista copiar_lista(Lista l);
```

Operações em lista ligada

Vamos ver três novas operações para listas ligadas

```
1  typedef struct no *Lista;
2
3  struct no {
4    int dado;
5    Lista prox;
6  };
7
8  Lista criar_lista();
9  void destruir_lista(Lista l);
10 Lista adicionar_elemento(Lista l, int x);
11 void imprime(Lista lista);
12
13 Lista copiar_lista(Lista l);
14 Lista inverter_lista(Lista l);
```

Operações em lista ligada

Vamos ver três novas operações para listas ligadas

```
1  typedef struct no *Lista;
2
3  struct no {
4    int dado;
5    Lista prox;
6  };
7
8  Lista criar_lista();
9  void destruir_lista(Lista l);
10 Lista adicionar_elemento(Lista l, int x);
11 void imprime(Lista lista);
12
13 Lista copiar_lista(Lista l);
14 Lista inverter_lista(Lista l);
15 Lista concatenar_lista(Lista l1, Lista l2);
```

Copiando

Versão recursiva:

Copiando

Versão recursiva:

```
1 Lista copiar_lista(Lista l) {
```

Copiando

Versão recursiva:

```
1 Lista copiar_lista(Lista l) {
```

Copiando

Versão recursiva:

```
1 Lista copiar_lista(Lista l) {
2   Lista novo;
3   if (l == NULL)
4     return NULL;
```

Copiando

Versão recursiva:

```
1 Lista copiar_lista(Lista l) {
2   Lista novo;
3   if (l == NULL)
4     return NULL;
5   novo = malloc(sizeof(struct no));
6   novo->dado = l->dado;
7   novo->prox = copiar_lista(l->prox);
```

Copiando

Versão recursiva:

```
1 Lista copiar_lista(Lista l) {
2   Lista novo;
3   if (l == NULL)
4     return NULL;
5   novo = malloc(sizeof(struct no));
6   novo->dado = l->dado;
7   novo->prox = copiar_lista(l->prox);
8   return novo;
9 }
```

Copiando

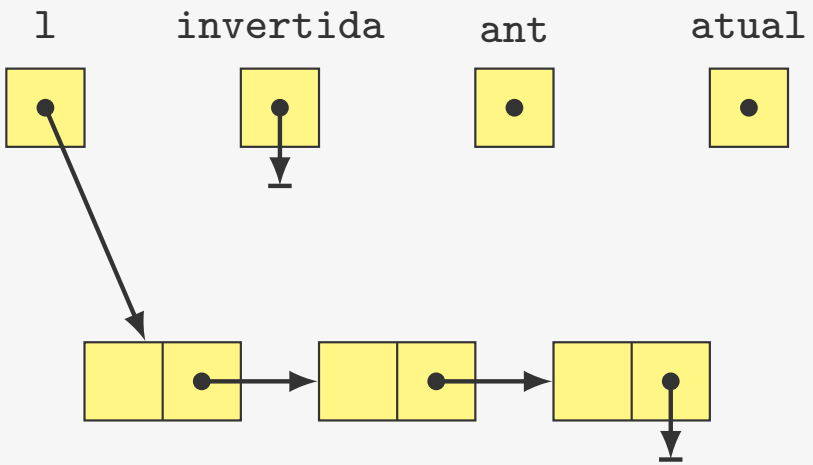
Versão recursiva:

```
1 Lista copiar_lista(Lista l) {
2   Lista novo;
3   if (l == NULL)
4     return NULL;
5   novo = malloc(sizeof(struct no));
6   novo->dado = l->dado;
7   novo->prox = copiar_lista(l->prox);
8   return novo;
9 }
```

Exercício: implemente uma versão iterativa da função

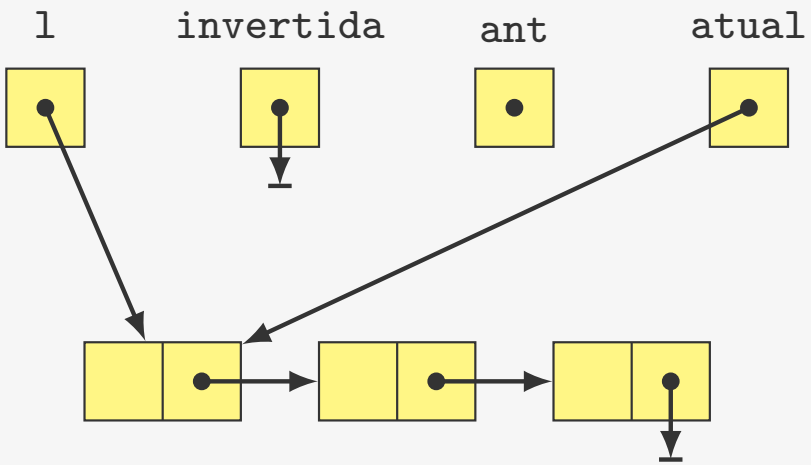
Invertendo

```
1  Lista inverter_lista(Lista l) {
2    Lista atual, ant, invertida = NULL;
3    atual = l;
4    while (atual != NULL) {
5      ant = atual;
6      atual = ant->prox;
7      ant->prox = invertida;
8      invertida = ant;
9    }
10   return invertida;
11 }
```



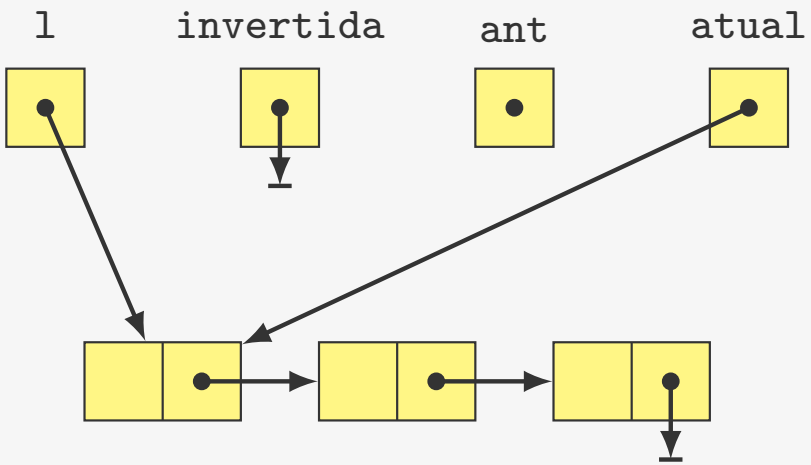
Invertendo

```
1  Lista inverter_lista(Lista l) {
2    Lista atual, ant, invertida = NULL;
3    atual = l;
4    while (atual != NULL) {
5      ant = atual;
6      atual = ant->prox;
7      ant->prox = invertida;
8      invertida = ant;
9    }
10   return invertida;
11 }
```



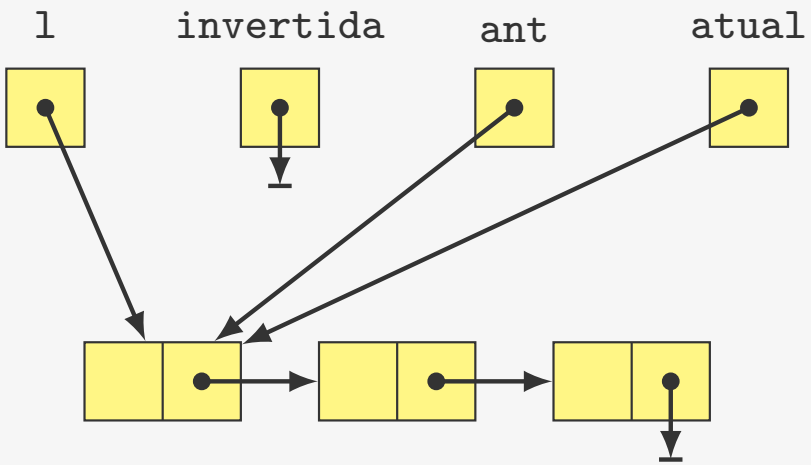
Invertendo

```
1  Lista inverter_lista(Lista l) {
2    Lista atual, ant, invertida = NULL;
3    atual = l;
4    while (atual != NULL) {
5      ant = atual;
6      atual = ant->prox;
7      ant->prox = invertida;
8      invertida = ant;
9    }
10   return invertida;
11 }
```



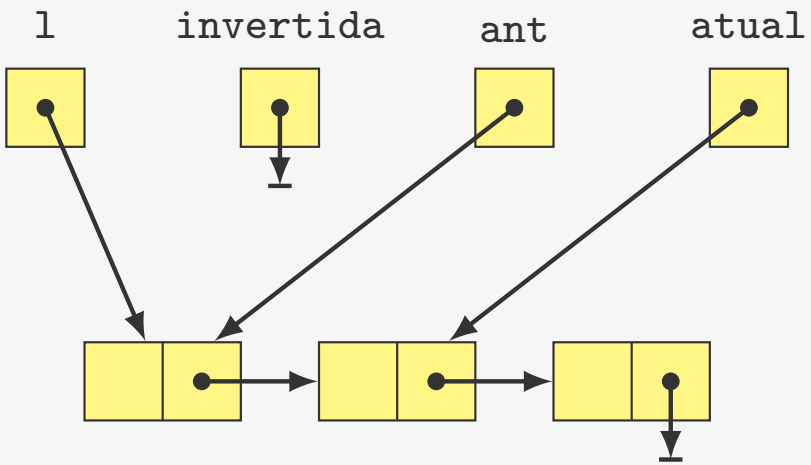
Invertendo

```
1  Lista inverter_lista(Lista l) {
2    Lista atual, ant, invertida = NULL;
3    atual = l;
4    while (atual != NULL) {
5      ant = atual;
6      atual = ant->prox;
7      ant->prox = invertida;
8      invertida = ant;
9    }
10   return invertida;
11 }
```



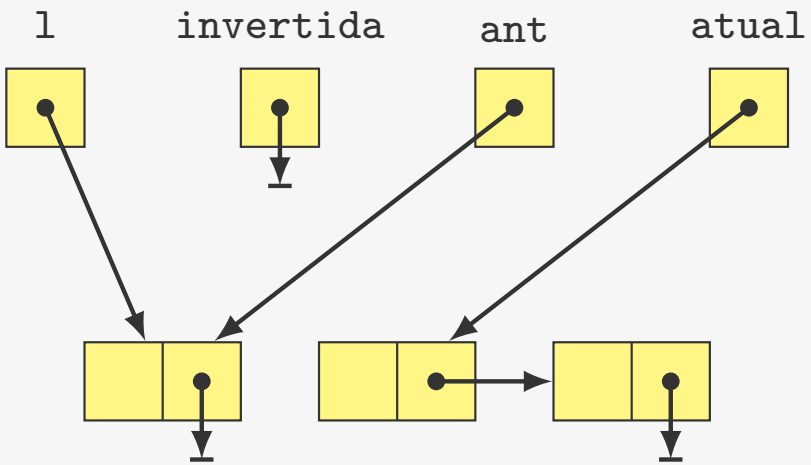
Invertendo

```
1  Lista inverter_lista(Lista l) {
2    Lista atual, ant, invertida = NULL;
3    atual = l;
4    while (atual != NULL) {
5      ant = atual;
6      atual = ant->prox;
7      ant->prox = invertida;
8      invertida = ant;
9    }
10   return invertida;
11 }
```



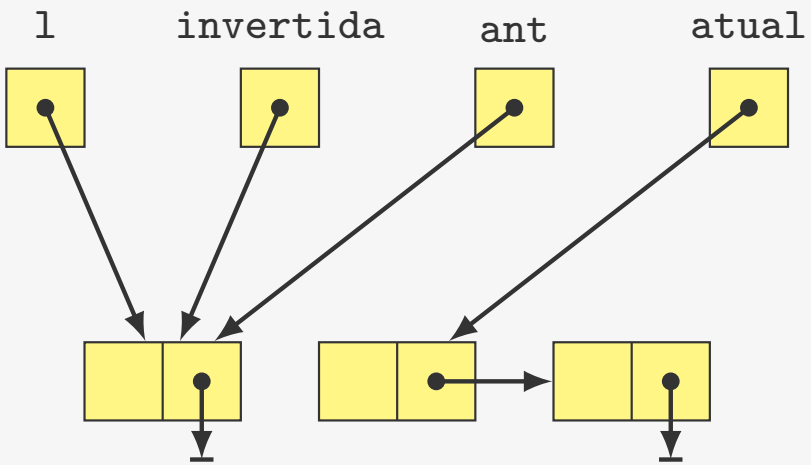
Invertendo

```
1  Lista inverter_lista(Lista l) {
2    Lista atual, ant, invertida = NULL;
3    atual = l;
4    while (atual != NULL) {
5      ant = atual;
6      atual = ant->prox;
7      ant->prox = invertida;
8      invertida = ant;
9    }
10   return invertida;
11 }
```



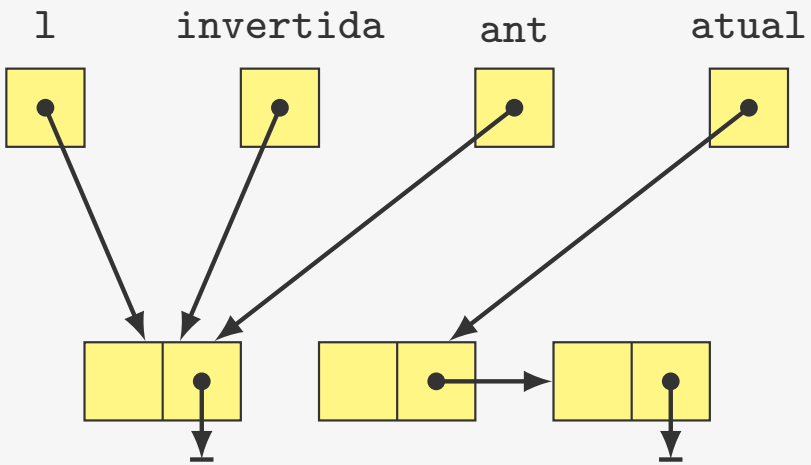
Invertendo

```
1  Lista inverter_lista(Lista l) {
2    Lista atual, ant, invertida = NULL;
3    atual = l;
4    while (atual != NULL) {
5      ant = atual;
6      atual = ant->prox;
7      ant->prox = invertida;
8      invertida = ant;
9    }
10   return invertida;
11 }
```



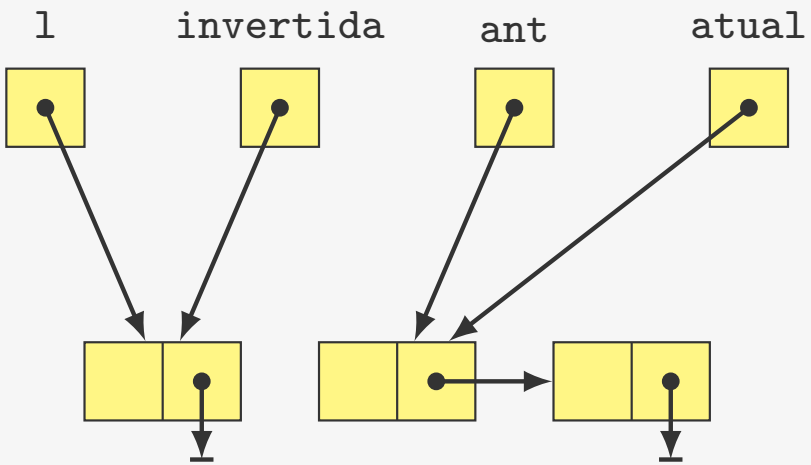
Invertendo

```
1  Lista inverter_lista(Lista l) {
2    Lista atual, ant, invertida = NULL;
3    atual = l;
4    while (atual != NULL) {
5      ant = atual;
6      atual = ant->prox;
7      ant->prox = invertida;
8      invertida = ant;
9    }
10   return invertida;
11 }
```



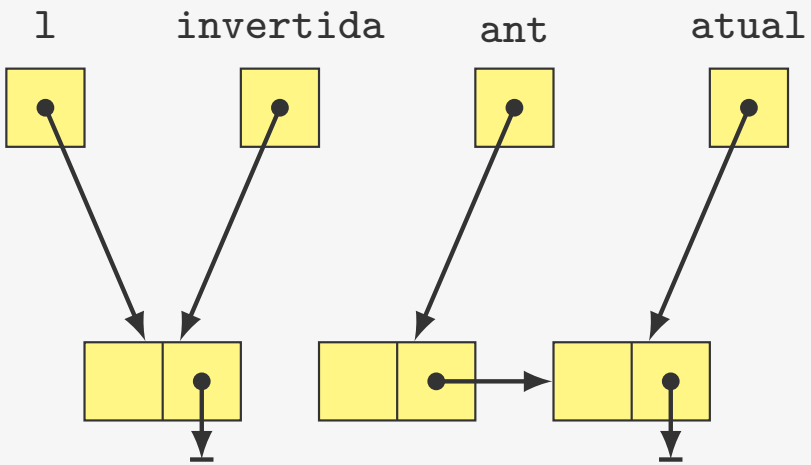
Invertendo

```
1  Lista inverter_lista(Lista l) {
2    Lista atual, ant, invertida = NULL;
3    atual = l;
4    while (atual != NULL) {
5      ant = atual;
6      atual = ant->prox;
7      ant->prox = invertida;
8      invertida = ant;
9    }
10   return invertida;
11 }
```



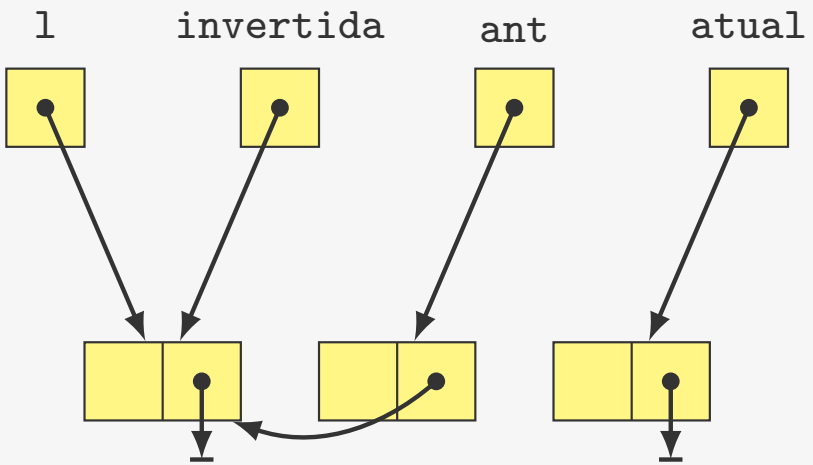
Invertendo

```
1  Lista inverter_lista(Lista l) {
2    Lista atual, ant, invertida = NULL;
3    atual = l;
4    while (atual != NULL) {
5      ant = atual;
6      atual = ant->prox;
7      ant->prox = invertida;
8      invertida = ant;
9    }
10   return invertida;
11 }
```



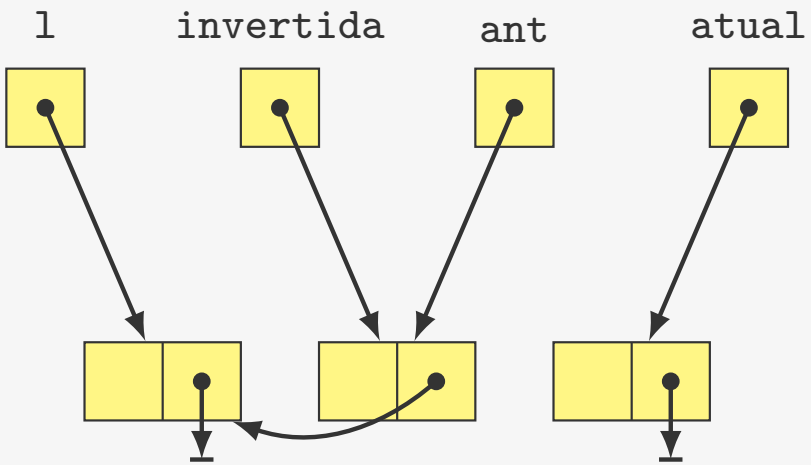
Invertendo

```
1  Lista inverter_lista(Lista l) {
2    Lista atual, ant, invertida = NULL;
3    atual = l;
4    while (atual != NULL) {
5      ant = atual;
6      atual = ant->prox;
7      ant->prox = invertida;
8      invertida = ant;
9    }
10   return invertida;
11 }
```



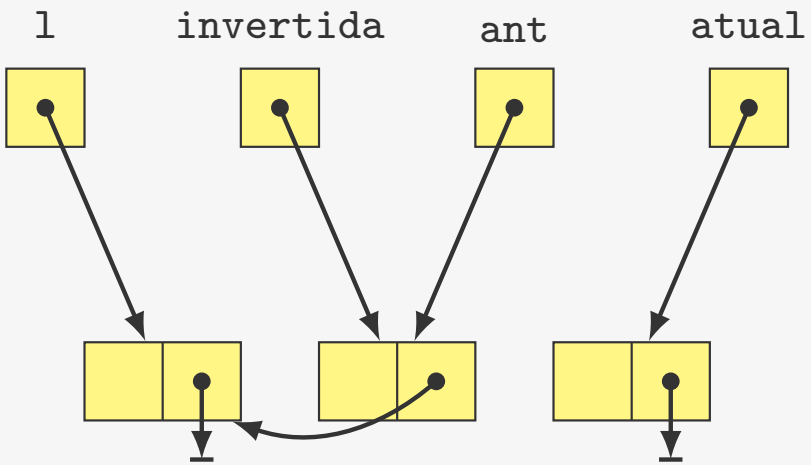
Invertendo

```
1  Lista inverter_lista(Lista l) {
2    Lista atual, ant, invertida = NULL;
3    atual = l;
4    while (atual != NULL) {
5      ant = atual;
6      atual = ant->prox;
7      ant->prox = invertida;
8      invertida = ant;
9    }
10   return invertida;
11 }
```



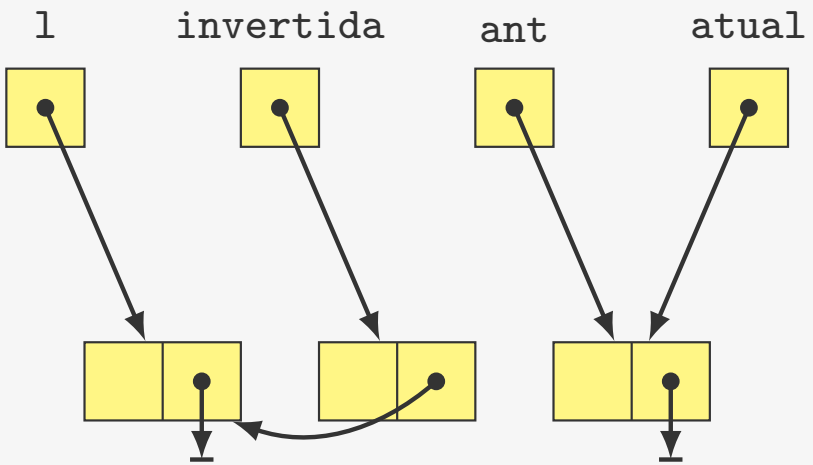
Invertendo

```
1  Lista inverter_lista(Lista l) {
2    Lista atual, ant, invertida = NULL;
3    atual = l;
4    while (atual != NULL) {
5      ant = atual;
6      atual = ant->prox;
7      ant->prox = invertida;
8      invertida = ant;
9    }
10   return invertida;
11 }
```



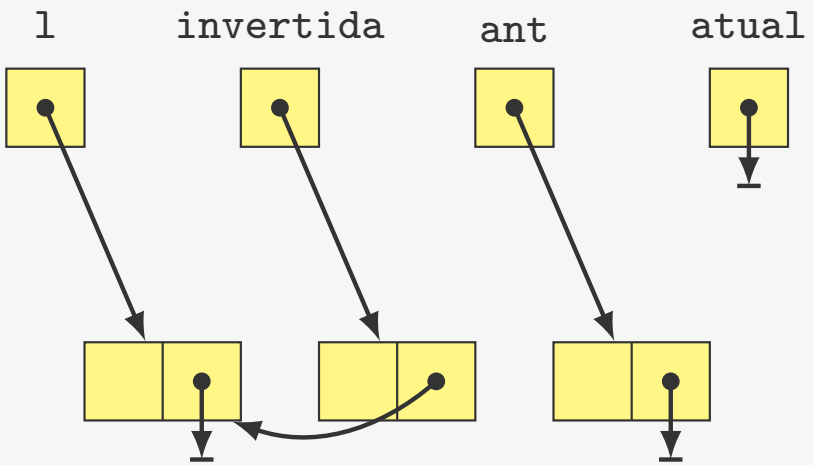
Invertendo

```
1  Lista inverter_lista(Lista l) {
2    Lista atual, ant, invertida = NULL;
3    atual = l;
4    while (atual != NULL) {
5      ant = atual;
6      atual = ant->prox;
7      ant->prox = invertida;
8      invertida = ant;
9    }
10   return invertida;
11 }
```



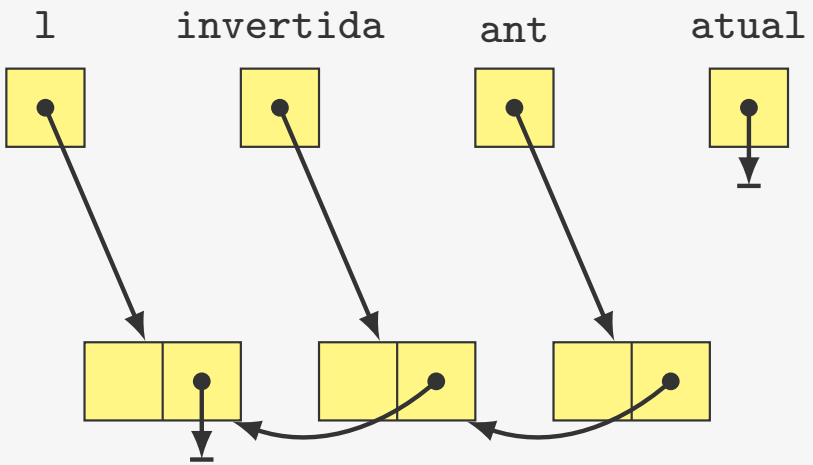
Invertendo

```
1  Lista inverter_lista(Lista l) {
2    Lista atual, ant, invertida = NULL;
3    atual = l;
4    while (atual != NULL) {
5      ant = atual;
6      atual = ant->prox;
7      ant->prox = invertida;
8      invertida = ant;
9    }
10   return invertida;
11 }
```



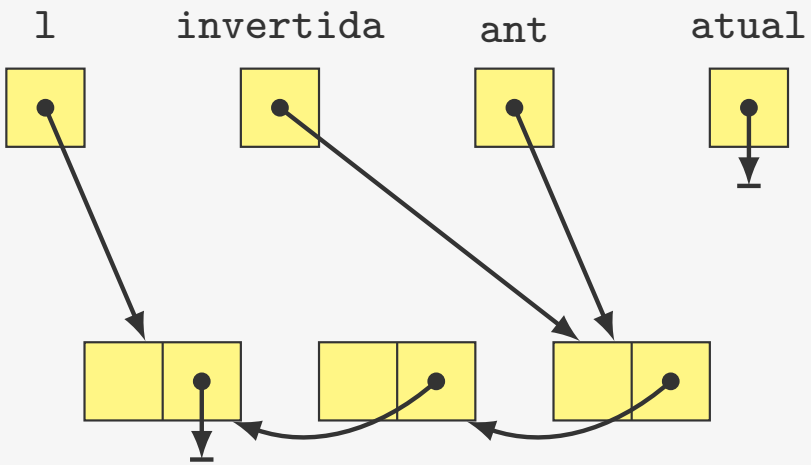
Invertendo

```
1  Lista inverter_lista(Lista l) {
2    Lista atual, ant, invertida = NULL;
3    atual = l;
4    while (atual != NULL) {
5      ant = atual;
6      atual = ant->prox;
7      ant->prox = invertida;
8      invertida = ant;
9    }
10   return invertida;
11 }
```



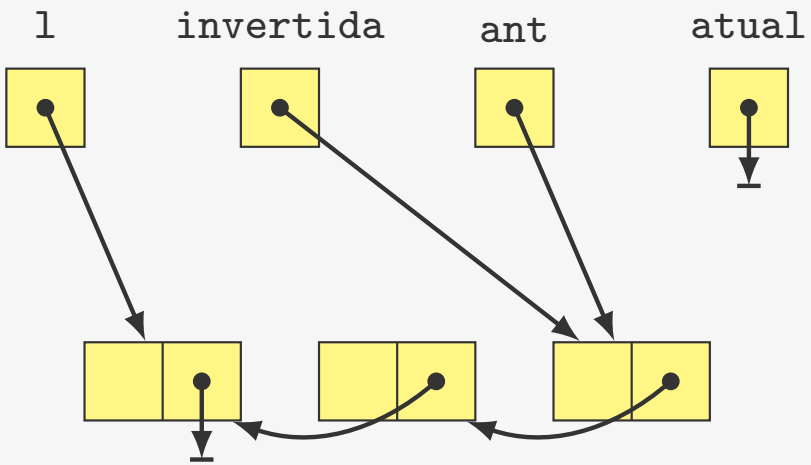
Invertendo

```
1  Lista inverter_lista(Lista l) {
2    Lista atual, ant, invertida = NULL;
3    atual = l;
4    while (atual != NULL) {
5      ant = atual;
6      atual = ant->prox;
7      ant->prox = invertida;
8      invertida = ant;
9    }
10   return invertida;
11 }
```



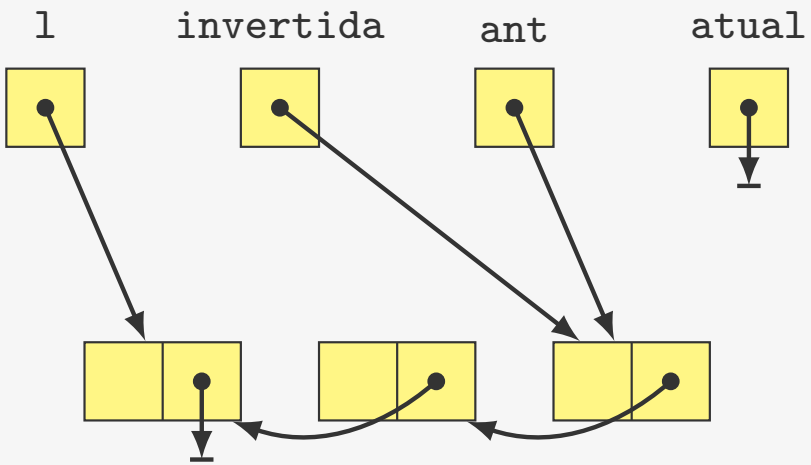
Invertendo

```
1  Lista inverter_lista(Lista l) {
2    Lista atual, ant, invertida = NULL;
3    atual = l;
4    while (atual != NULL) {
5      ant = atual;
6      atual = ant->prox;
7      ant->prox = invertida;
8      invertida = ant;
9    }
10   return invertida;
11 }
```



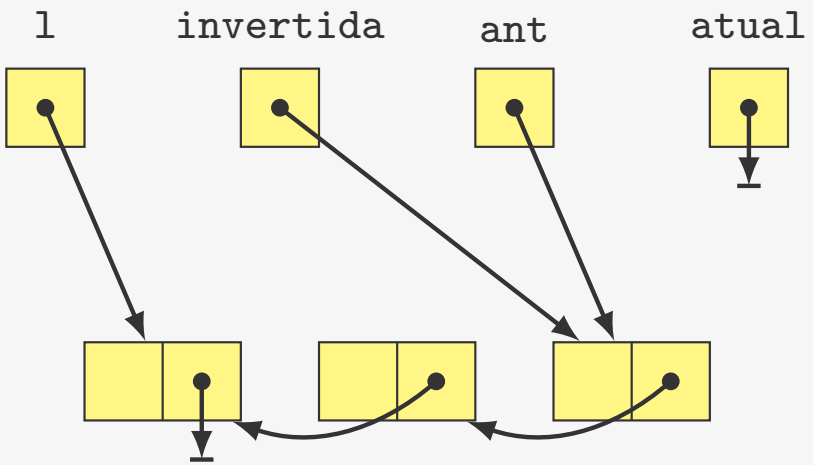
Invertendo

```
1  Lista inverter_lista(Lista l) {
2    Lista atual, ant, invertida = NULL;
3    atual = l;
4    while (atual != NULL) {
5      ant = atual;
6      atual = ant->prox;
7      ant->prox = invertida;
8      invertida = ant;
9    }
10   return invertida;
11 }
```



Invertendo

```
1  Lista inverter_lista(Lista l) {
2    Lista atual, ant, invertida = NULL;
3    atual = l;
4    while (atual != NULL) {
5      ant = atual;
6      atual = ant->prox;
7      ant->prox = invertida;
8      invertida = ant;
9    }
10   return invertida;
11 }
```



Exercício: implemente uma versão recursiva da função

Concatenando

```
1 Lista concatenar_lista(Lista primeira, Lista segunda) {
```

Concatenando

```
1 Lista concatenar_lista(Lista primeira, Lista segunda) {
```

Concatenando

```
1 Lista concatenar_lista(Lista primeira, Lista segunda) {
2   if (primeira == NULL)
```

Concatenando

```
1 Lista concatenar_lista(Lista primeira, Lista segunda) {
2   if (primeira == NULL)
3     return segunda;
```

Concatenando

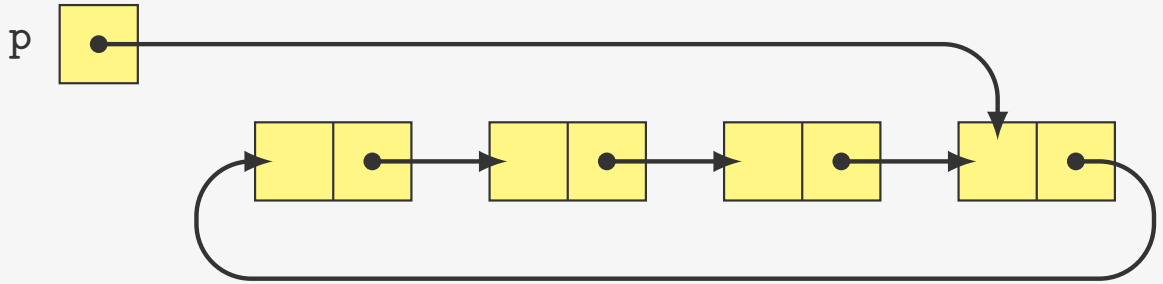
```
1 Lista concatenar_lista(Lista primeira, Lista segunda) {
2   if (primeira == NULL)
3     return segunda;
4   primeira->prox = concatenar_lista(primeira->prox, segunda);
```

Concatenando

```
1 Lista concatenar_lista(Lista primeira, Lista segunda) {
2   if (primeira == NULL)
3     return segunda;
4   primeira->prox = concatenar_lista(primeira->prox, segunda);
5   return primeira;
6 }
```

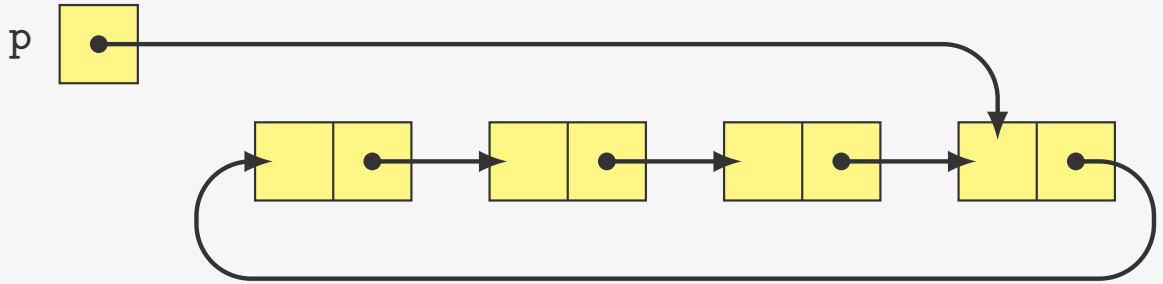
Variações — Listas circulares

Lista circular:

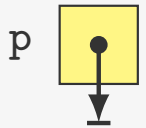


Variações — Listas circulares

Lista circular:

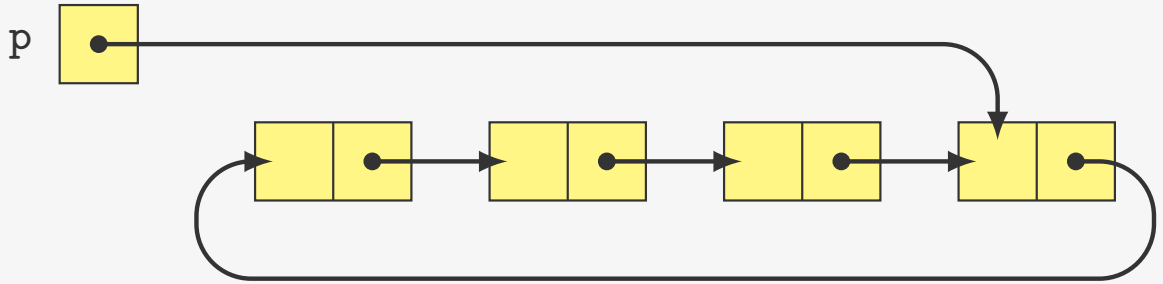


Lista circular vazia:

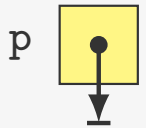


Variações — Listas circulares

Lista circular:



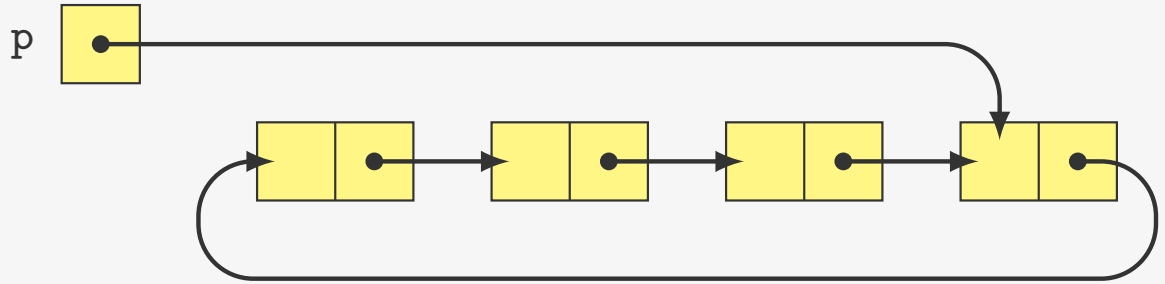
Lista circular vazia:



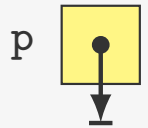
Exemplos de aplicações:

Variações — Listas circulares

Lista circular:



Lista circular vazia:

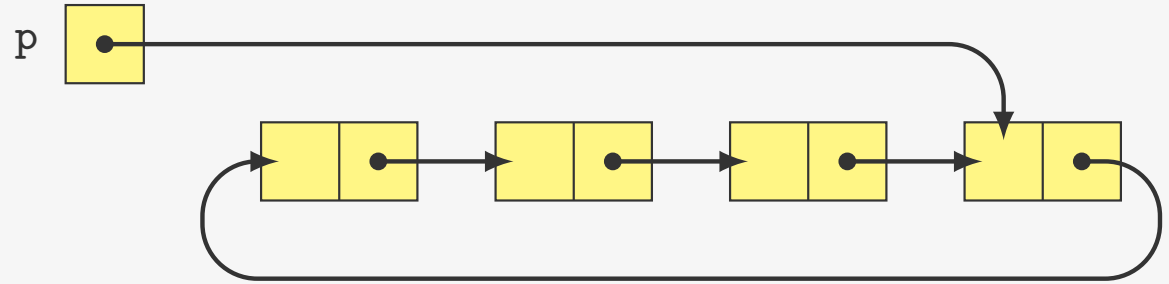


Exemplos de aplicações:

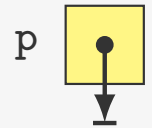
- Execução de processos no sistema operacional

Variações — Listas circulares

Lista circular:



Lista circular vazia:



Exemplos de aplicações:

- Execução de processos no sistema operacional
- Controlar de quem é a vez em um jogo de tabuleiro

Inserindo em lista circular

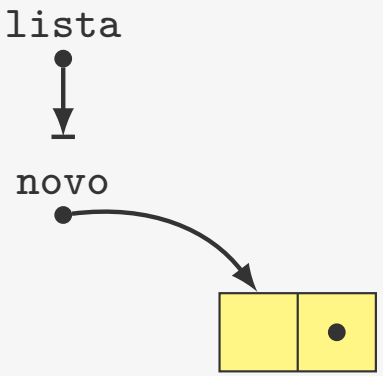
```
1  Lista inserir_circular(Lista lista, int x) {
2    Lista novo;
3    novo = malloc(sizeof(struct no));
4    novo->dado = x;
5    if (lista == NULL) {
6      novo->prox = novo;
7      lista = novo;
8    } else {
9      novo->prox = lista->prox;
10     lista->prox = novo;
11   }
12   return lista;
13 }
```

Inserindo em lista circular

```
1  Lista inserir_circular(Lista lista, int x) {
2    Lista novo;
3    novo = malloc(sizeof(struct no));
4    novo->dado = x;
5    if (lista == NULL) {
6      novo->prox = novo;
7      lista = novo;
8    } else {
9      novo->prox = lista->prox;
10     lista->prox = novo;
11   }
12   return lista;
13 }
```

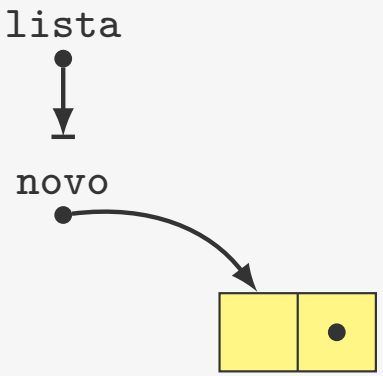
Inserindo em lista circular

```
1  Lista inserir_circular(Lista lista, int x) {
2    Lista novo;
3    novo = malloc(sizeof(struct no));
4    novo->dado = x;
5    if (lista == NULL) {
6      novo->prox = novo;
7      lista = novo;
8    } else {
9      novo->prox = lista->prox;
10     lista->prox = novo;
11   }
12   return lista;
13 }
```



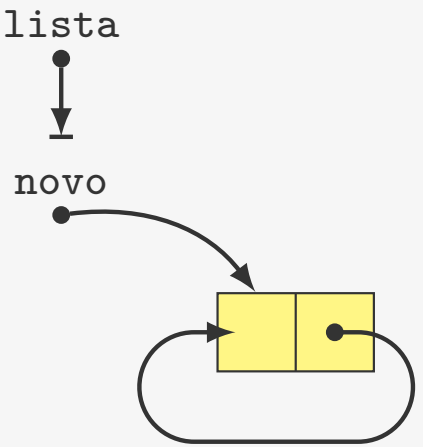
Inserindo em lista circular

```
1  Lista inserir_circular(Lista lista, int x) {
2    Lista novo;
3    novo = malloc(sizeof(struct no));
4    novo->dado = x;
5    if (lista == NULL) {
6      novo->prox = novo;
7      lista = novo;
8    } else {
9      novo->prox = lista->prox;
10     lista->prox = novo;
11   }
12   return lista;
13 }
```



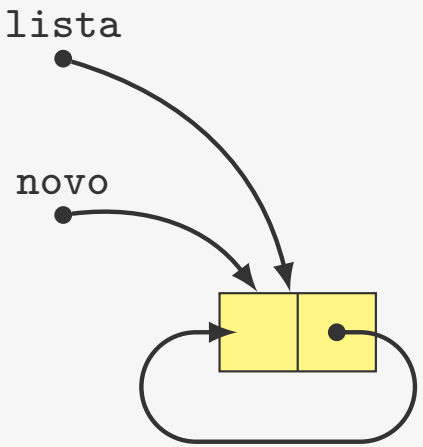
Inserindo em lista circular

```
1  Lista inserir_circular(Lista lista, int x) {
2    Lista novo;
3    novo = malloc(sizeof(struct no));
4    novo->dado = x;
5    if (lista == NULL) {
6      novo->prox = novo;
7      lista = novo;
8    } else {
9      novo->prox = lista->prox;
10     lista->prox = novo;
11   }
12   return lista;
13 }
```



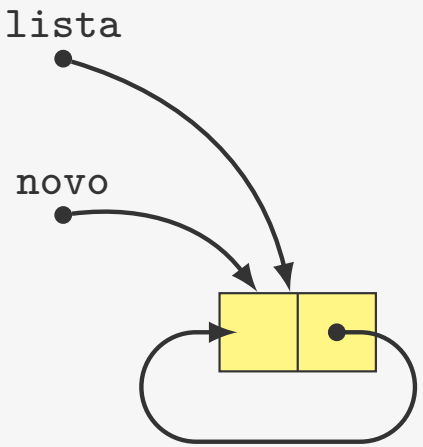
Inserindo em lista circular

```
1  Lista inserir_circular(Lista lista, int x) {
2    Lista novo;
3    novo = malloc(sizeof(struct no));
4    novo->dado = x;
5    if (lista == NULL) {
6      novo->prox = novo;
7      lista = novo;
8    } else {
9      novo->prox = lista->prox;
10     lista->prox = novo;
11   }
12   return lista;
13 }
```



Inserindo em lista circular

```
1  Lista inserir_circular(Lista lista, int x) {
2    Lista novo;
3    novo = malloc(sizeof(struct no));
4    novo->dado = x;
5    if (lista == NULL) {
6      novo->prox = novo;
7      lista = novo;
8    } else {
9      novo->prox = lista->prox;
10     lista->prox = novo;
11   }
12   return lista;
13 }
```

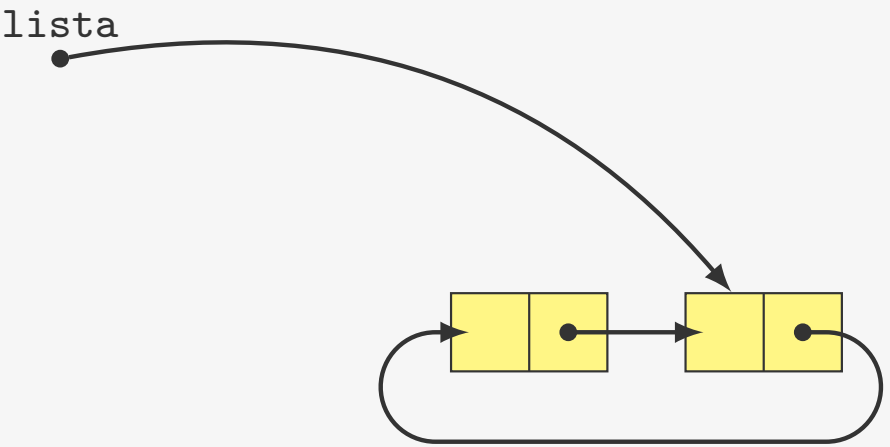


Inserindo em lista circular

```
1  Lista inserir_circular(Lista lista, int x) {
2    Lista novo;
3    novo = malloc(sizeof(struct no));
4    novo->dado = x;
5    if (lista == NULL) {
6      novo->prox = novo;
7      lista = novo;
8    } else {
9      novo->prox = lista->prox;
10     lista->prox = novo;
11   }
12   return lista;
13 }
```

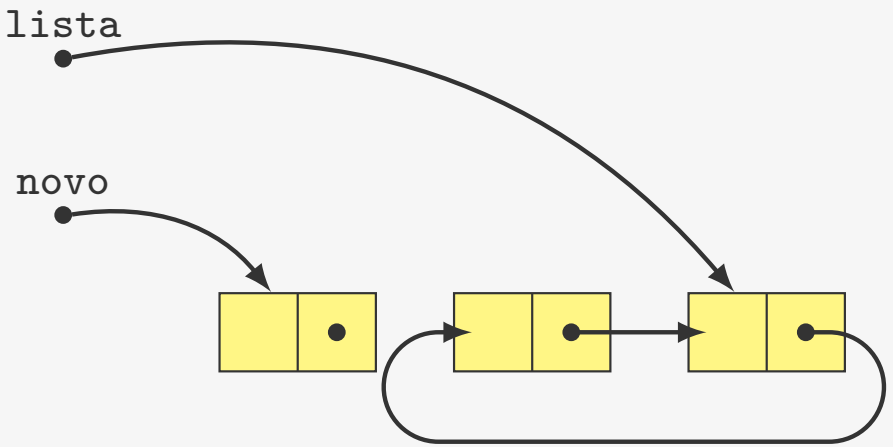
Inserindo em lista circular

```
1  Lista inserir_circular(Lista lista, int x) {
2    Lista novo;
3    novo = malloc(sizeof(struct no));
4    novo->dado = x;
5    if (lista == NULL) {
6      novo->prox = novo;
7      lista = novo;
8    } else {
9      novo->prox = lista->prox;
10     lista->prox = novo;
11   }
12   return lista;
13 }
```



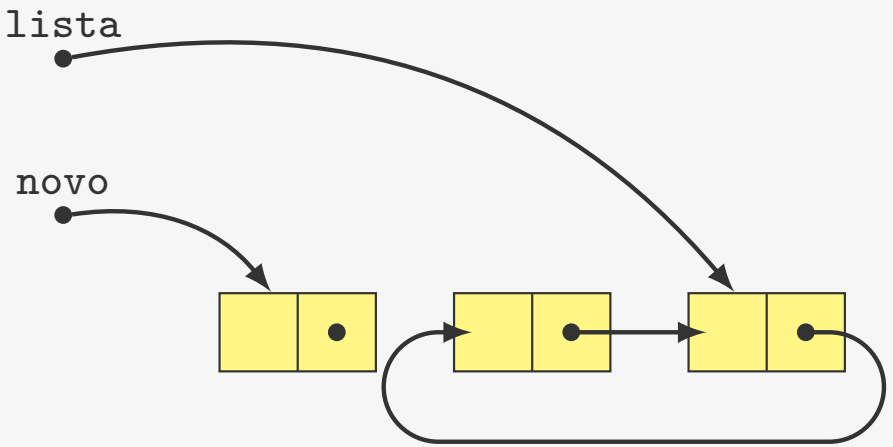
Inserindo em lista circular

```
1  Lista inserir_circular(Lista lista, int x) {
2    Lista novo;
3    novo = malloc(sizeof(struct no));
4    novo->dado = x;
5    if (lista == NULL) {
6      novo->prox = novo;
7      lista = novo;
8    } else {
9      novo->prox = lista->prox;
10     lista->prox = novo;
11   }
12   return lista;
13 }
```



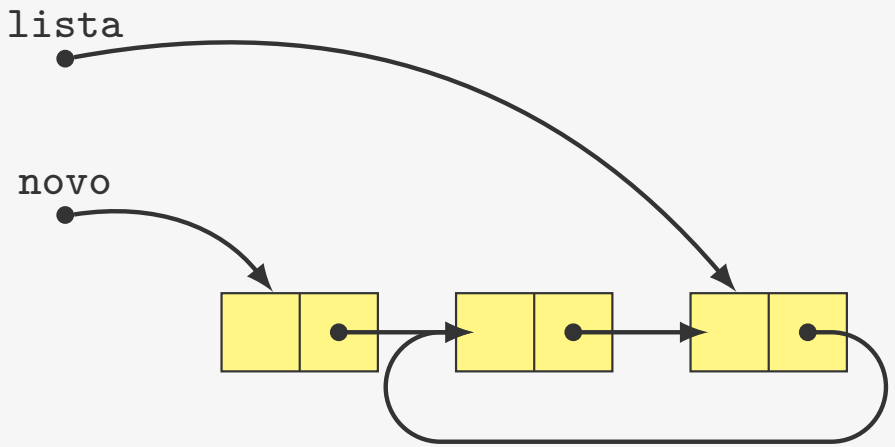
Inserindo em lista circular

```
1  Lista inserir_circular(Lista lista, int x) {
2    Lista novo;
3    novo = malloc(sizeof(struct no));
4    novo->dado = x;
5    if (lista == NULL) {
6      novo->prox = novo;
7      lista = novo;
8    } else {
9      novo->prox = lista->prox;
10     lista->prox = novo;
11   }
12   return lista;
13 }
```



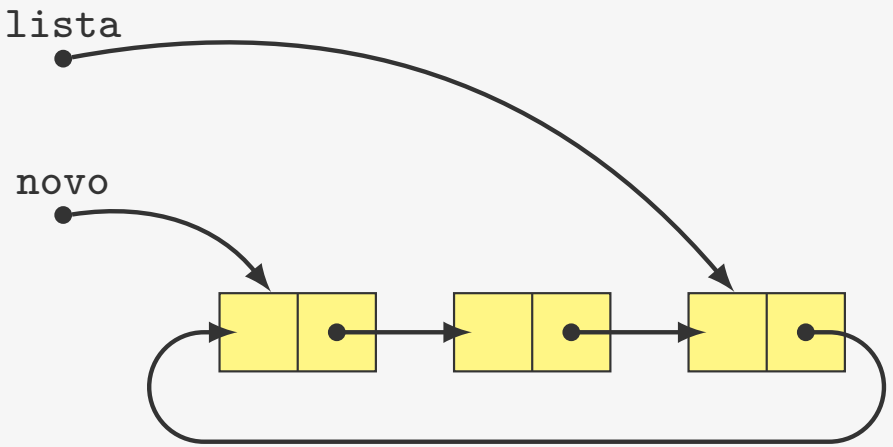
Inserindo em lista circular

```
1  Lista inserir_circular(Lista lista, int x) {
2    Lista novo;
3    novo = malloc(sizeof(struct no));
4    novo->dado = x;
5    if (lista == NULL) {
6      novo->prox = novo;
7      lista = novo;
8    } else {
9      novo->prox = lista->prox;
10     lista->prox = novo;
11   }
12   return lista;
13 }
```



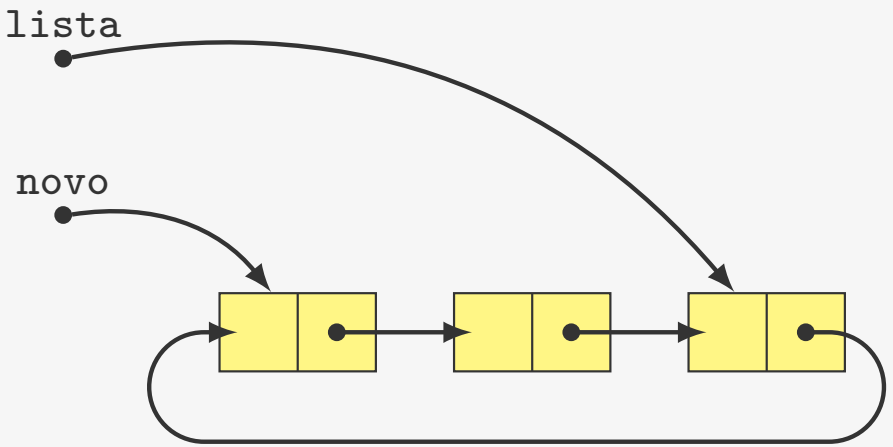
Inserindo em lista circular

```
1  Lista inserir_circular(Lista lista, int x) {
2    Lista novo;
3    novo = malloc(sizeof(struct no));
4    novo->dado = x;
5    if (lista == NULL) {
6      novo->prox = novo;
7      lista = novo;
8    } else {
9      novo->prox = lista->prox;
10     lista->prox = novo;
11   }
12   return lista;
13 }
```



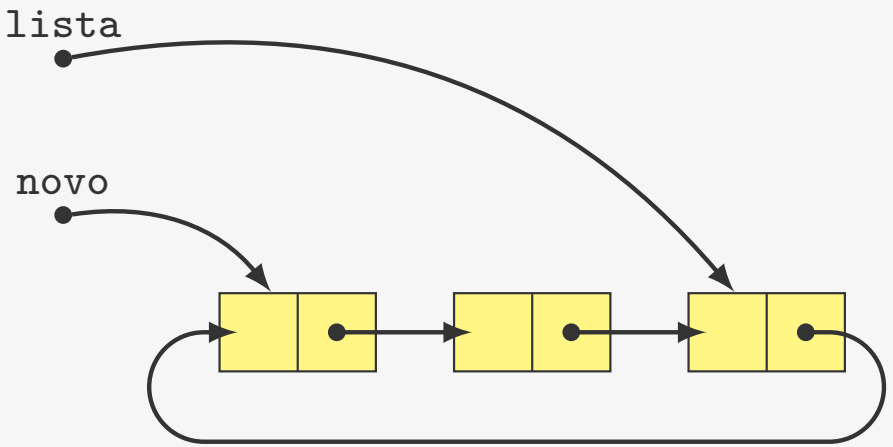
Inserindo em lista circular

```
1  Lista inserir_circular(Lista lista, int x) {
2    Lista novo;
3    novo = malloc(sizeof(struct no));
4    novo->dado = x;
5    if (lista == NULL) {
6      novo->prox = novo;
7      lista = novo;
8    } else {
9      novo->prox = lista->prox;
10     lista->prox = novo;
11   }
12   return lista;
13 }
```



Inserindo em lista circular

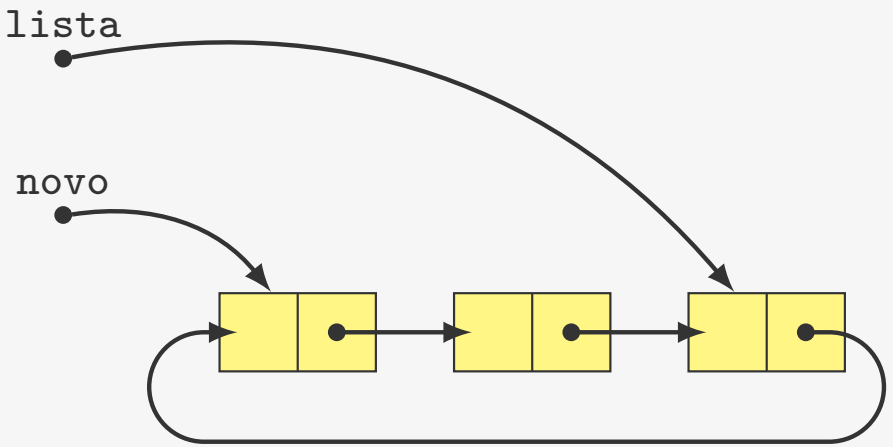
```
1  Lista inserir_circular(Lista lista, int x) {
2    Lista novo;
3    novo = malloc(sizeof(struct no));
4    novo->dado = x;
5    if (lista == NULL) {
6      novo->prox = novo;
7      lista = novo;
8    } else {
9      novo->prox = lista->prox;
10     lista->prox = novo;
11   }
12   return lista;
13 }
```



Observações:

Inserindo em lista circular

```
1  Lista inserir_circular(Lista lista, int x) {
2    Lista novo;
3    novo = malloc(sizeof(struct no));
4    novo->dado = x;
5    if (lista == NULL) {
6      novo->prox = novo;
7      lista = novo;
8    } else {
9      novo->prox = lista->prox;
10     lista->prox = novo;
11   }
12   return lista;
13 }
```

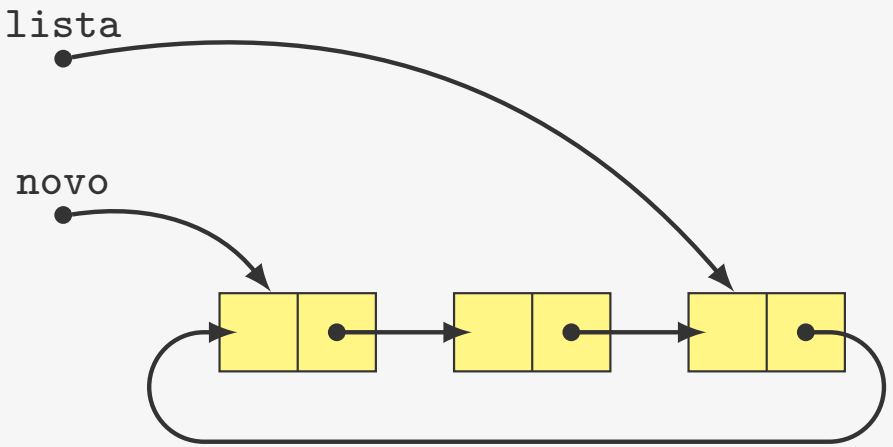


Observações:

- A lista sempre aponta para o último elemento

Inserindo em lista circular

```
1  Lista inserir_circular(Lista lista, int x) {
2    Lista novo;
3    novo = malloc(sizeof(struct no));
4    novo->dado = x;
5    if (lista == NULL) {
6      novo->prox = novo;
7      lista = novo;
8    } else {
9      novo->prox = lista->prox;
10     lista->prox = novo;
11   }
12   return lista;
13 }
```

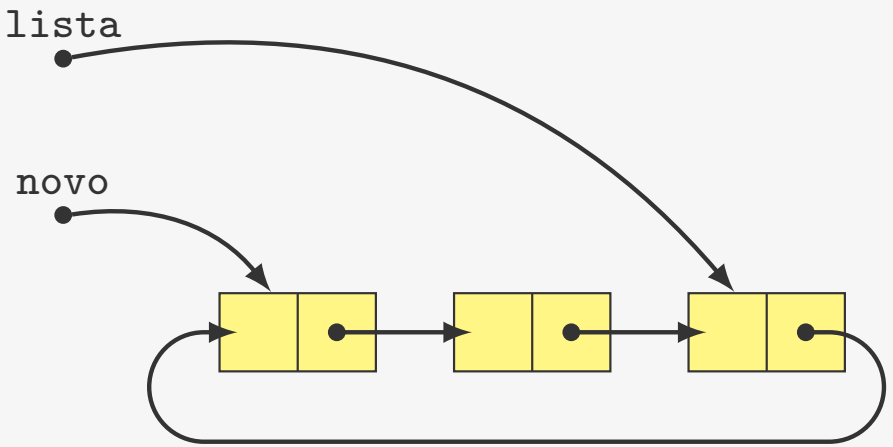


Observações:

- A lista sempre aponta para o último elemento
  - O dado do primeiro nó elemento é `lista->prox->dado`

Inserindo em lista circular

```
1  Lista inserir_circular(Lista lista, int x) {
2    Lista novo;
3    novo = malloc(sizeof(struct no));
4    novo->dado = x;
5    if (lista == NULL) {
6      novo->prox = novo;
7      lista = novo;
8    } else {
9      novo->prox = lista->prox;
10     lista->prox = novo;
11   }
12   return lista;
13 }
```

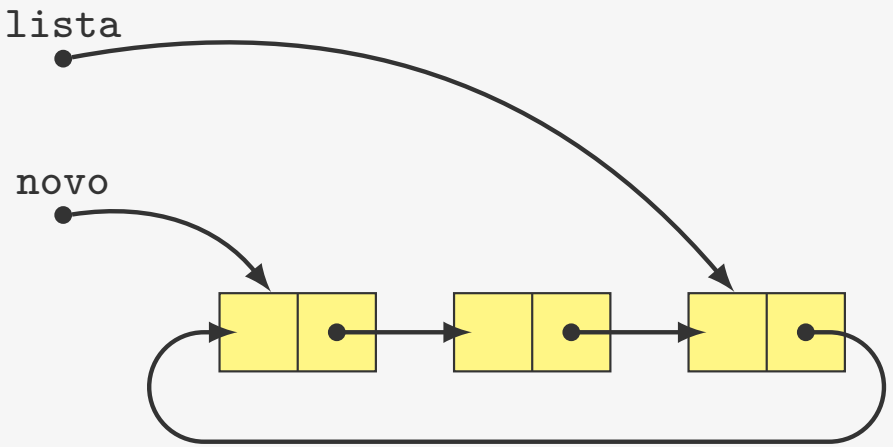


Observações:

- A lista sempre aponta para o último elemento
  - O dado do primeiro nó elemento é `lista->prox->dado`
  - O dado do último nó elemento é `lista->dado`

Inserindo em lista circular

```
1  Lista inserir_circular(Lista lista, int x) {
2    Lista novo;
3    novo = malloc(sizeof(struct no));
4    novo->dado = x;
5    if (lista == NULL) {
6      novo->prox = novo;
7      lista = novo;
8    } else {
9      novo->prox = lista->prox;
10     lista->prox = novo;
11   }
12   return lista;
13 }
```

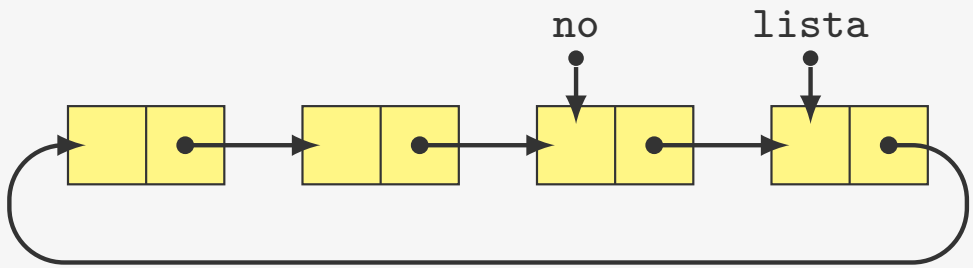


Observações:

- A lista sempre aponta para o último elemento
  - O dado do primeiro nó elemento é `lista->prox->dado`
  - O dado do último nó elemento é `lista->dado`
  - Para inserir no final, basta devolver `novo` ao invés de `lista`

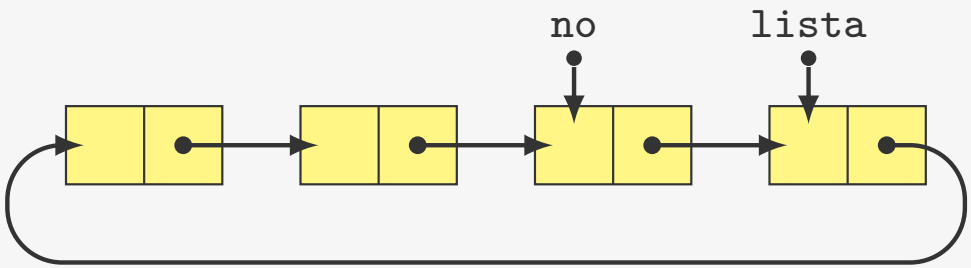
Removendo de lista circular

```
1 Lista remover_circular(Lista lista, Lista no) {
```



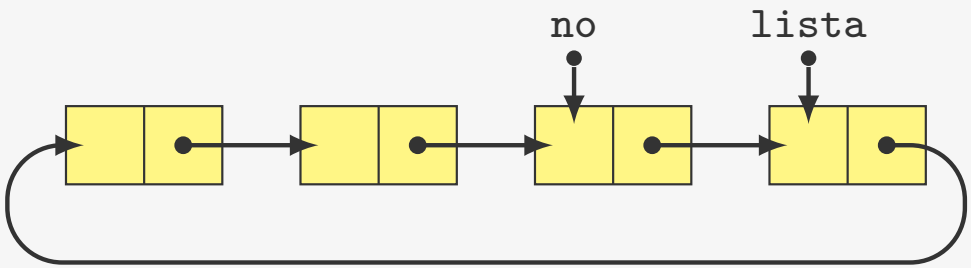
Removendo de lista circular

```
1 Lista remover_circular(Lista lista, Lista no) {
2   Lista ant;
3   if (no->prox == no) {
```



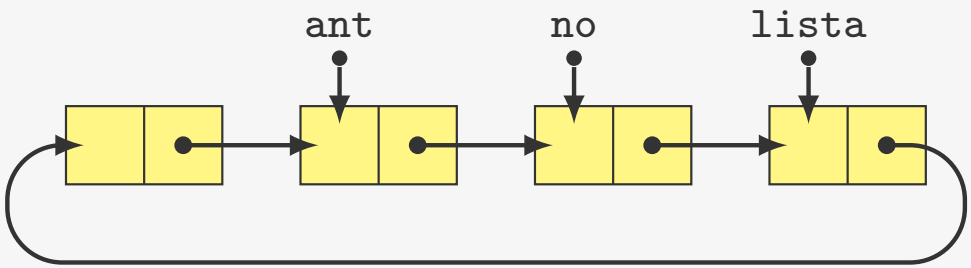
Removendo de lista circular

```
1 Lista remover_circular(Lista lista, Lista no) {
2   Lista ant;
3   if (no->prox == no) {
4     free(no);
5     return NULL;
6   }
```



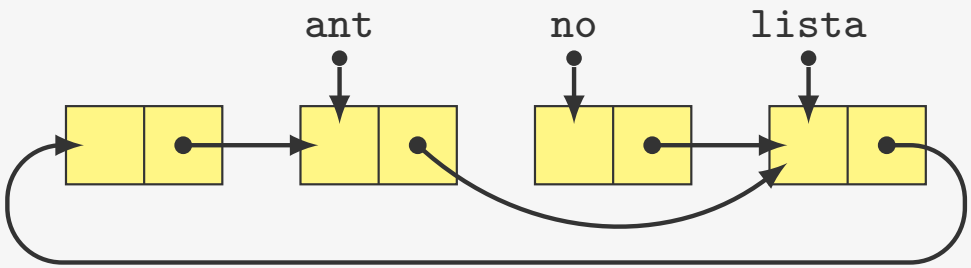
Removendo de lista circular

```
1 Lista remover_circular(Lista lista, Lista no) {
2   Lista ant;
3   if (no->prox == no) {
4     free(no);
5     return NULL;
6   }
7   for(ant = no->prox; ant->prox != no; ant = ant->prox);
```



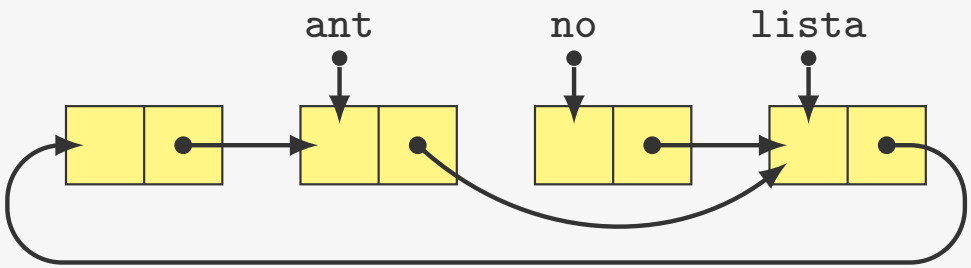
Removendo de lista circular

```
1 Lista remover_circular(Lista lista, Lista no) {
2   Lista ant;
3   if (no->prox == no) {
4     free(no);
5     return NULL;
6   }
7   for(ant = no->prox; ant->prox != no; ant = ant->prox);
8   ant->prox = no->prox;
```



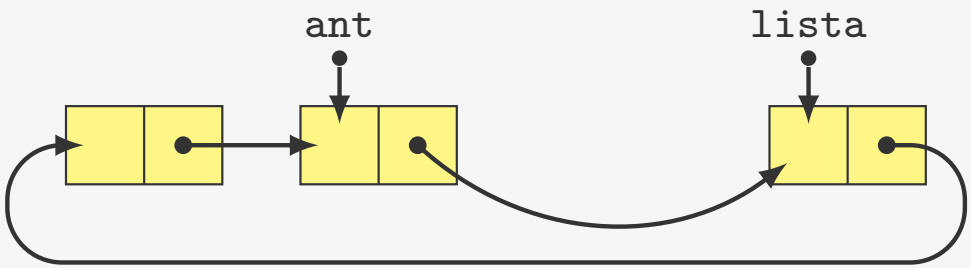
Removendo de lista circular

```
1  Lista remover_circular(Lista lista, Lista no) {
2    Lista ant;
3    if (no->prox == no) {
4      free(no);
5      return NULL;
6    }
7    for(ant = no->prox; ant->prox != no; ant = ant->prox);
8    ant->prox = no->prox;
9    if (lista == no)
10     lista = ant;
```



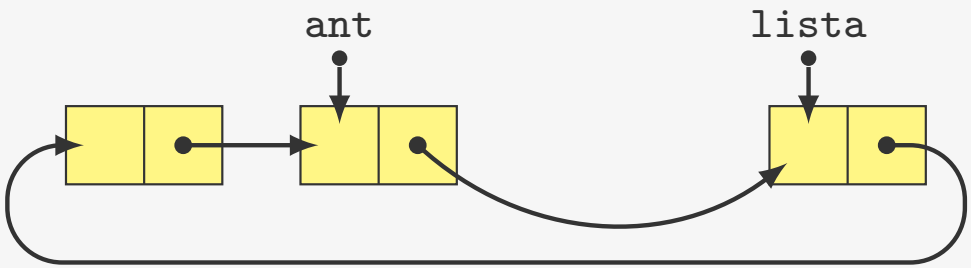
Removendo de lista circular

```
1  Lista remover_circular(Lista lista, Lista no) {
2    Lista ant;
3    if (no->prox == no) {
4      free(no);
5      return NULL;
6    }
7    for(ant = no->prox; ant->prox != no; ant = ant->prox);
8    ant->prox = no->prox;
9    if (lista == no)
10     lista = ant;
11   free(no);
```



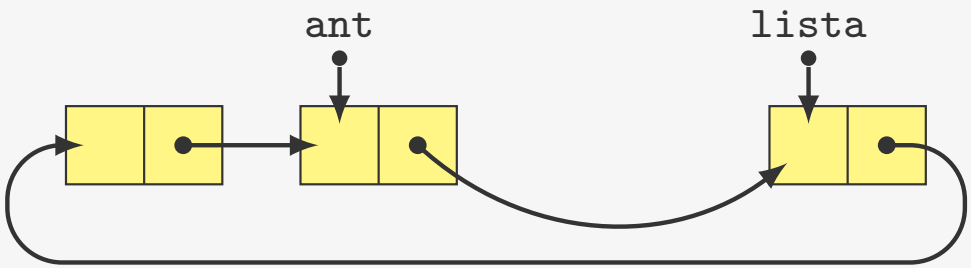
Removendo de lista circular

```
1  Lista remover_circular(Lista lista, Lista no) {
2    Lista ant;
3    if (no->prox == no) {
4      free(no);
5      return NULL;
6    }
7    for(ant = no->prox; ant->prox != no; ant = ant->prox);
8    ant->prox = no->prox;
9    if (lista == no)
10     lista = ant;
11   free(no);
12   return lista;
13 }
```



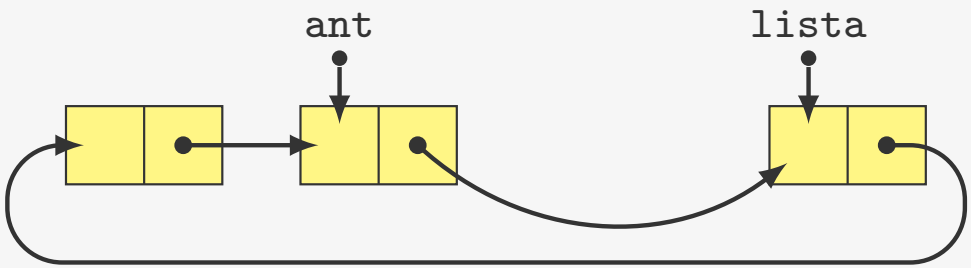
Removendo de lista circular

```
1  Lista remover_circular(Lista lista, Lista no) {
2    Lista ant;
3    if (no->prox == no) {
4      free(no);
5      return NULL;
6    }
7    for(ant = no->prox; ant->prox != no; ant = ant->prox);
8    ant->prox = no->prox;
9    if (lista == no)
10     lista = ant;
11   free(no);
12   return lista;
13 }
```



Removendo de lista circular

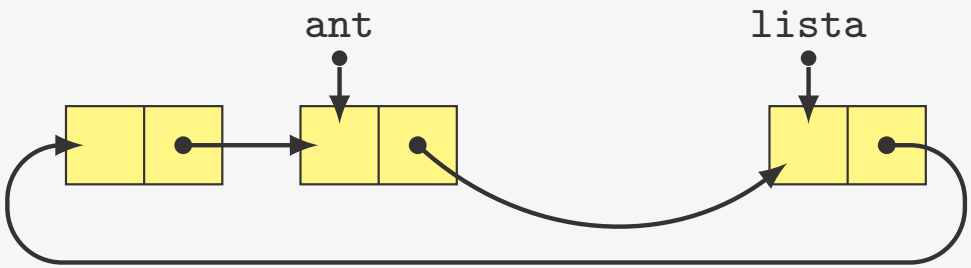
```
1  Lista remover_circular(Lista lista, Lista no) {
2    Lista ant;
3    if (no->prox == no) {
4      free(no);
5      return NULL;
6    }
7    for(ant = no->prox; ant->prox != no; ant = ant->prox);
8    ant->prox = no->prox;
9    if (lista == no)
10     lista = ant;
11   free(no);
12   return lista;
13 }
```



Tempo: $\mathrm{O}(n)$

Removendo de lista circular

```
1  Lista remover_circular(Lista lista, Lista no) {
2    Lista ant;
3    if (no->prox == no) {
4      free(no);
5      return NULL;
6    }
7    for(ant = no->prox; ant->prox != no; ant = ant->prox);
8    ant->prox = no->prox;
9    if (lista == no)
10     lista = ant;
11   free(no);
12   return lista;
13 }
```



Tempo: $O(n)$

- tempo contante se soubermos o nó anterior
- e.g., para remover o primeiro da lista

Percorrendo uma lista circular

```
1 void imprimir_lista_circular(Lista lista) {
2   Lista p;
3   p = lista->prox;
4   do {
5     printf("%d\n", p->dado);
6     p = p->prox;
7   } while (p != lista->prox);
8 }
```

Percorrendo uma lista circular

```
1 void imprimir_lista_circular(Lista lista) {
2   Lista p;
3   p = lista->prox;
4   do {
5     printf("%d\n", p->dado);
6     p = p->prox;
7   } while (p != lista->prox);
8 }
```

- E se tivéssemos usado `while` ao invés de `do ... while`?

Percorrendo uma lista circular

```
1 void imprimir_lista_circular(Lista lista) {
2   Lista p;
3   p = lista->prox;
4   do {
5     printf("%d\n", p->dado);
6     p = p->prox;
7   } while (p != lista->prox);
8 }
```

- E se tivéssemos usado `while` ao invés de `do ... while`?
- Essa função pode ser usada com lista vazia?

Percorrendo uma lista circular

```
1 void imprimir_lista_circular(Lista lista) {
2   Lista p;
3   p = lista->prox;
4   do {
5     printf("%d\n", p->dado);
6     p = p->prox;
7   } while (p != lista->prox);
8 }
```

- E se tivéssemos usado `while` ao invés de `do ... while`?
- Essa função pode ser usada com lista vazia?
  - Como corrigir isso?

Exercício — Problema de Josephus

Vamos eleger um líder entre N pessoas

Exercício — Problema de Josephus

Vamos eleger um líder entre N pessoas

- Começamos a contar da primeira pessoa

Exercício — Problema de Josephus

Vamos eleger um líder entre N pessoas

- Começamos a contar da primeira pessoa
- Contamos M pessoas

Exercício — Problema de Josephus

Vamos eleger um líder entre N pessoas

- Começamos a contar da primeira pessoa
- Contamos M pessoas
- Eliminamos $(M+1)$-ésima pessoa

Exercício — Problema de Josephus

Vamos eleger um líder entre N pessoas

- Começamos a contar da primeira pessoa
- Contamos M pessoas
- Eliminamos $(M+1)$-ésima pessoa
- Continuamos da próxima pessoa

Exercício — Problema de Josephus

Vamos eleger um líder entre N pessoas

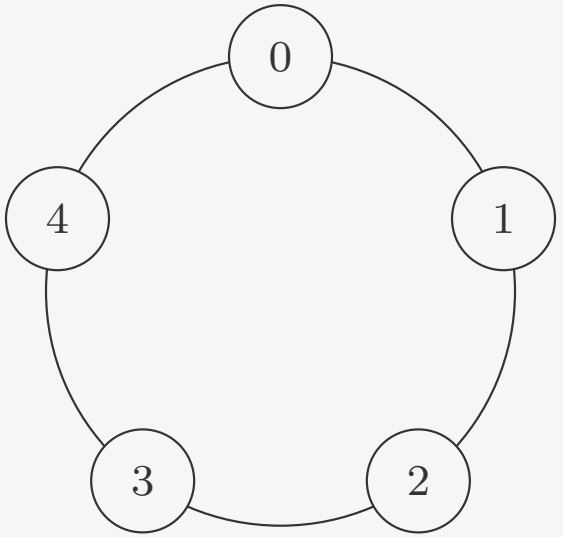
- Começamos a contar da primeira pessoa
- Contamos M pessoas
- Eliminamos $(M+1)$-ésima pessoa
- Continuamos da próxima pessoa
- Ciclamos quando chegamos ao final

Exercício — Problema de Josephus

Vamos eleger um líder entre N pessoas

- Começamos a contar da primeira pessoa
- Contamos M pessoas
- Eliminamos $(M+1)$-ésima pessoa
- Continuamos da próxima pessoa
- Ciclamos quando chegamos ao final

Exemplo: $N = 5$ e $M = 2$

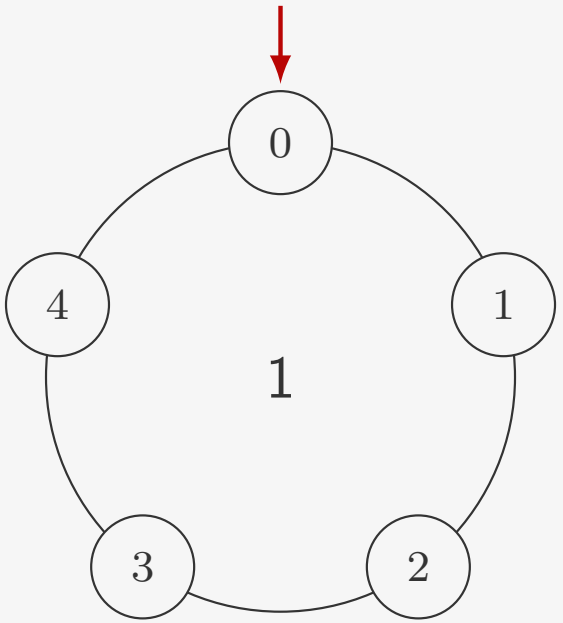


Exercício — Problema de Josephus

Vamos eleger um líder entre N pessoas

- Começamos a contar da primeira pessoa
- Contamos M pessoas
- Eliminamos $(M+1)$-ésima pessoa
- Continuamos da próxima pessoa
- Ciclamos quando chegamos ao final

Exemplo: $N = 5$ e $M = 2$

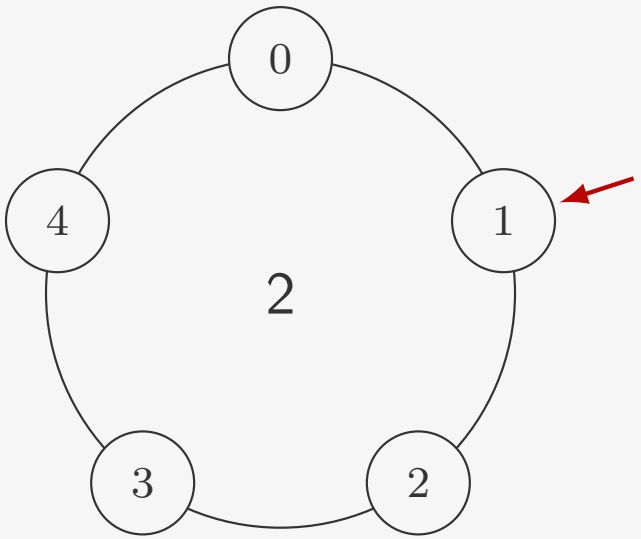


Exercício — Problema de Josephus

Vamos eleger um líder entre N pessoas

- Começamos a contar da primeira pessoa
- Contamos M pessoas
- Eliminamos $(M+1)$-ésima pessoa
- Continuamos da próxima pessoa
- Ciclamos quando chegamos ao final

Exemplo: $N = 5$ e $M = 2$

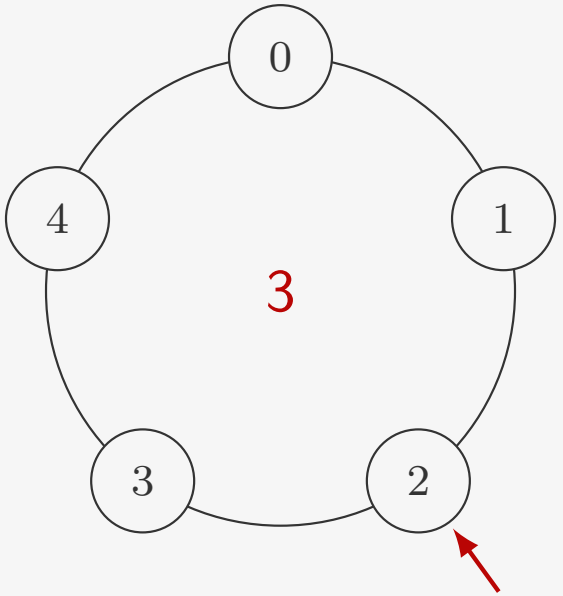


Exercício — Problema de Josephus

Vamos eleger um líder entre N pessoas

- Começamos a contar da primeira pessoa
- Contamos M pessoas
- Eliminamos $(M+1)$-ésima pessoa
- Continuamos da próxima pessoa
- Ciclamos quando chegamos ao final

Exemplo: $N = 5$ e $M = 2$

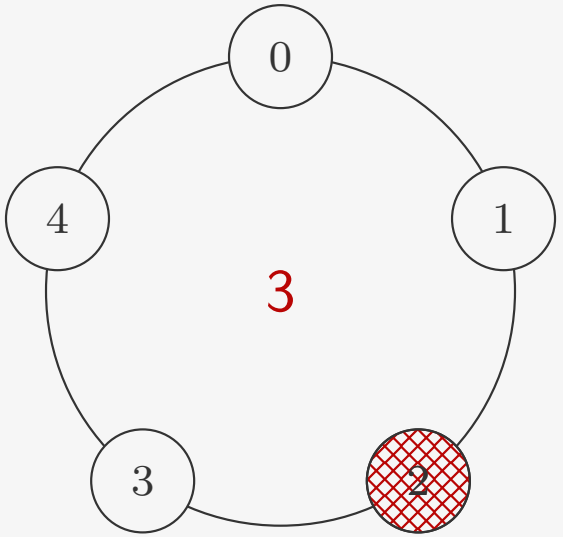


Exercício — Problema de Josephus

Vamos eleger um líder entre N pessoas

- Começamos a contar da primeira pessoa
- Contamos M pessoas
- Eliminamos $(M+1)$-ésima pessoa
- Continuamos da próxima pessoa
- Ciclamos quando chegamos ao final

Exemplo: $N = 5$ e $M = 2$

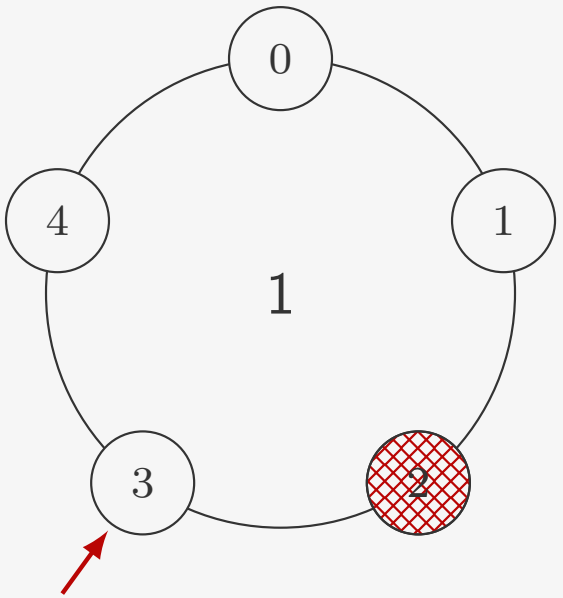


Exercício — Problema de Josephus

Vamos eleger um líder entre N pessoas

- Começamos a contar da primeira pessoa
- Contamos M pessoas
- Eliminamos $(M+1)$-ésima pessoa
- Continuamos da próxima pessoa
- Ciclamos quando chegamos ao final

Exemplo: $N = 5$ e $M = 2$

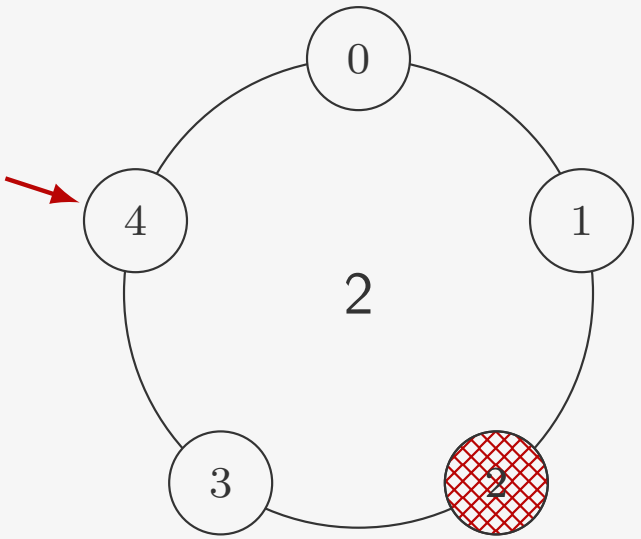


Exercício — Problema de Josephus

Vamos eleger um líder entre N pessoas

- Começamos a contar da primeira pessoa
- Contamos M pessoas
- Eliminamos $(M+1)$-ésima pessoa
- Continuamos da próxima pessoa
- Ciclamos quando chegamos ao final

Exemplo: $N = 5$ e $M = 2$

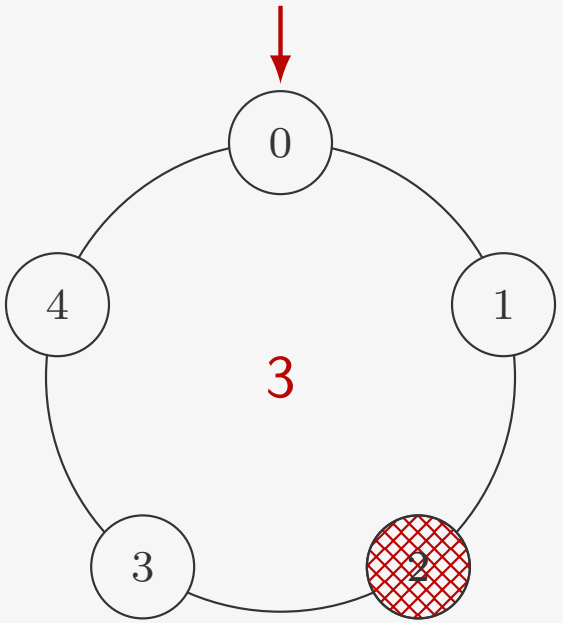


Exercício — Problema de Josephus

Vamos eleger um líder entre N pessoas

- Começamos a contar da primeira pessoa
- Contamos M pessoas
- Eliminamos $(M+1)$-ésima pessoa
- Continuamos da próxima pessoa
- Ciclamos quando chegamos ao final

Exemplo: $N = 5$ e $M = 2$

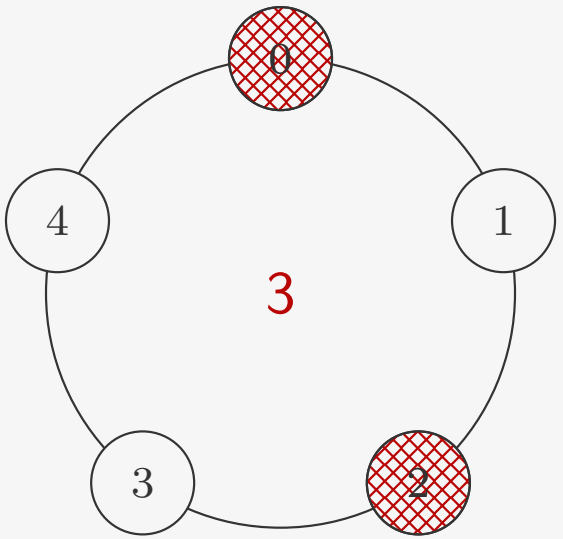


Exercício — Problema de Josephus

Vamos eleger um líder entre N pessoas

- Começamos a contar da primeira pessoa
- Contamos M pessoas
- Eliminamos $(M+1)$-ésima pessoa
- Continuamos da próxima pessoa
- Ciclamos quando chegamos ao final

Exemplo: $N = 5$ e $M = 2$

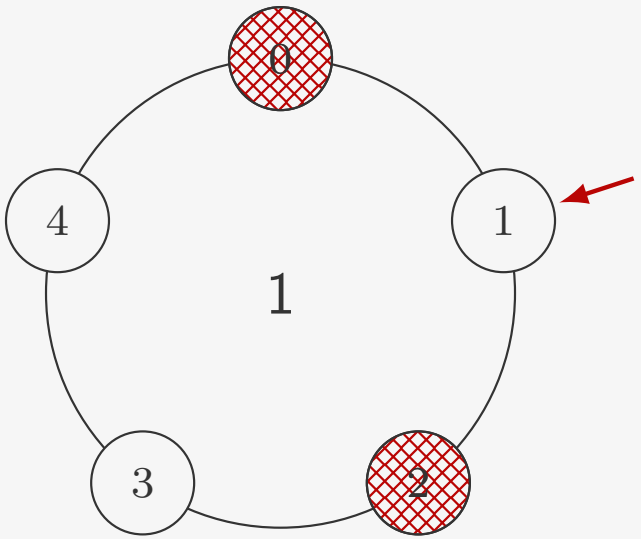


Exercício — Problema de Josephus

Vamos eleger um líder entre N pessoas

- Começamos a contar da primeira pessoa
- Contamos M pessoas
- Eliminamos $(M+1)$-ésima pessoa
- Continuamos da próxima pessoa
- Ciclamos quando chegamos ao final

Exemplo: $N = 5$ e $M = 2$

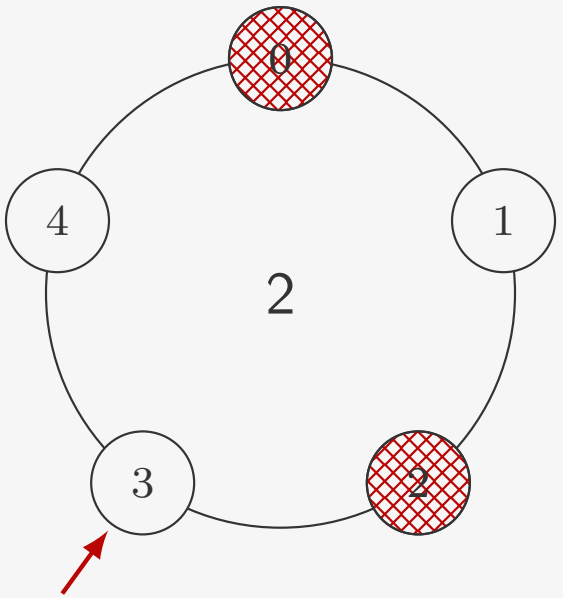


Exercício — Problema de Josephus

Vamos eleger um líder entre N pessoas

- Começamos a contar da primeira pessoa
- Contamos M pessoas
- Eliminamos $(M+1)$-ésima pessoa
- Continuamos da próxima pessoa
- Ciclamos quando chegamos ao final

Exemplo: $N = 5$ e $M = 2$

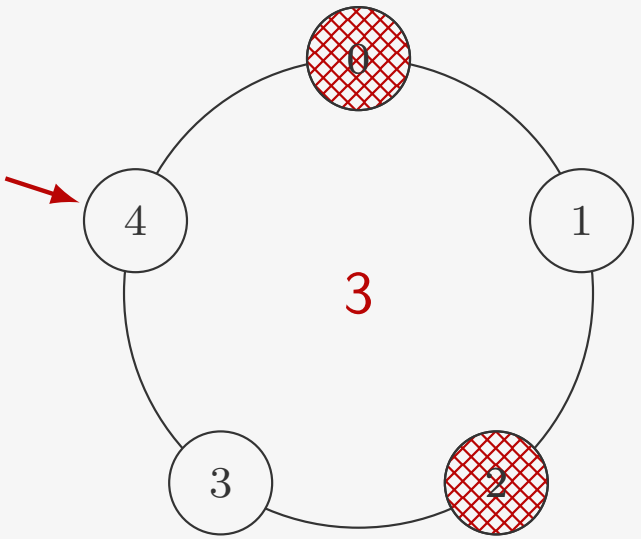


Exercício — Problema de Josephus

Vamos eleger um líder entre N pessoas

- Começamos a contar da primeira pessoa
- Contamos M pessoas
- Eliminamos $(M+1)$-ésima pessoa
- Continuamos da próxima pessoa
- Ciclamos quando chegamos ao final

Exemplo: $N = 5$ e $M = 2$

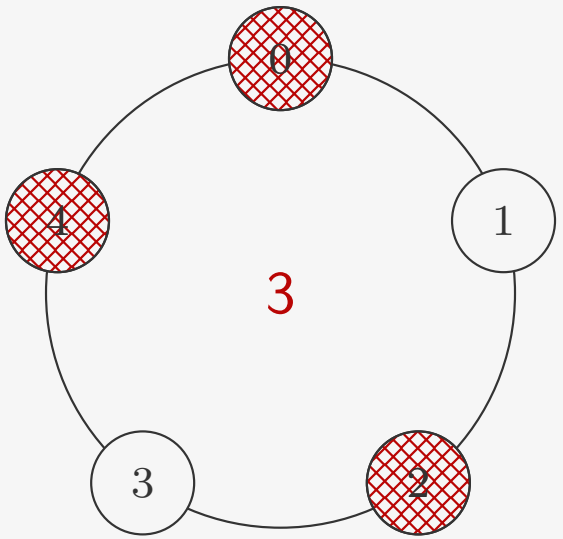


Exercício — Problema de Josephus

Vamos eleger um líder entre N pessoas

- Começamos a contar da primeira pessoa
- Contamos M pessoas
- Eliminamos $(M+1)$-ésima pessoa
- Continuamos da próxima pessoa
- Ciclamos quando chegamos ao final

Exemplo: $N = 5$ e $M = 2$

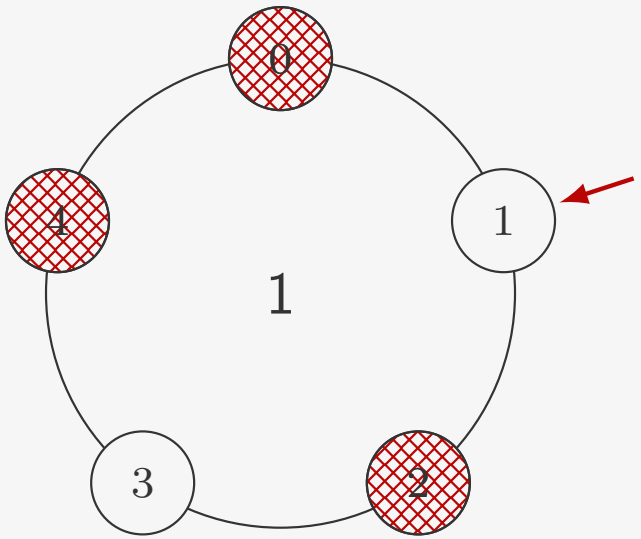


Exercício — Problema de Josephus

Vamos eleger um líder entre N pessoas

- Começamos a contar da primeira pessoa
- Contamos M pessoas
- Eliminamos $(M+1)$-ésima pessoa
- Continuamos da próxima pessoa
- Ciclamos quando chegamos ao final

Exemplo: $N = 5$ e $M = 2$

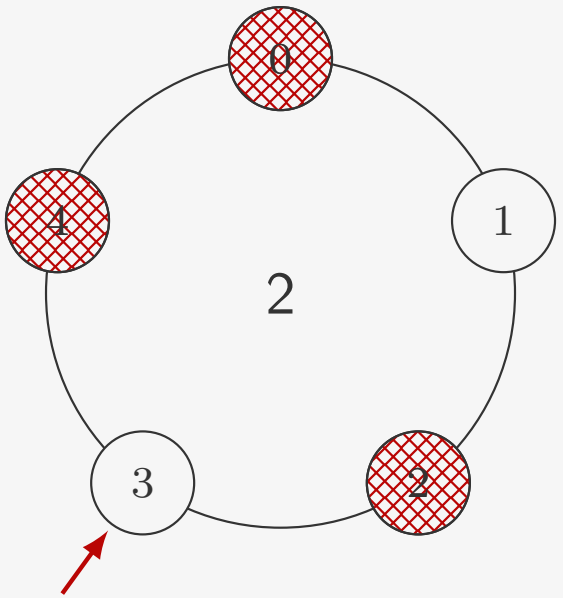


Exercício — Problema de Josephus

Vamos eleger um líder entre N pessoas

- Começamos a contar da primeira pessoa
- Contamos M pessoas
- Eliminamos $(M+1)$-ésima pessoa
- Continuamos da próxima pessoa
- Ciclamos quando chegamos ao final

Exemplo: $N = 5$ e $M = 2$

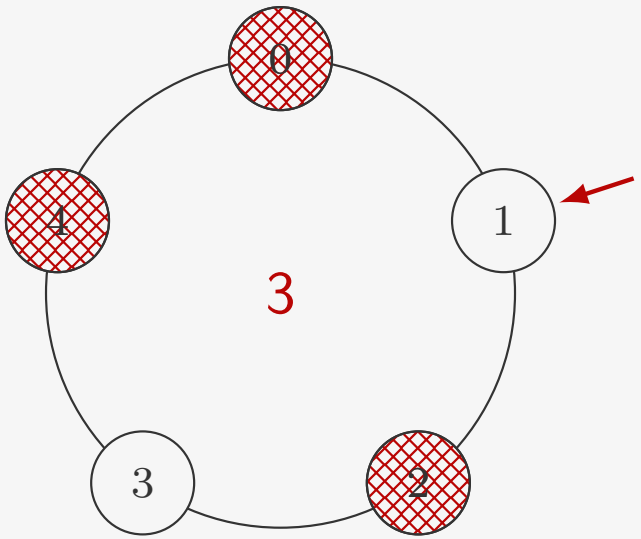


Exercício — Problema de Josephus

Vamos eleger um líder entre N pessoas

- Começamos a contar da primeira pessoa
- Contamos M pessoas
- Eliminamos $(M+1)$-ésima pessoa
- Continuamos da próxima pessoa
- Ciclamos quando chegamos ao final

Exemplo: $N = 5$ e $M = 2$

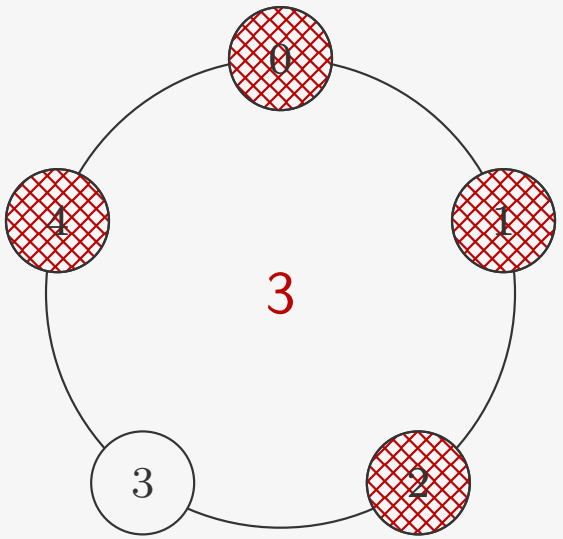


Exercício — Problema de Josephus

Vamos eleger um líder entre N pessoas

- Começamos a contar da primeira pessoa
- Contamos M pessoas
- Eliminamos $(M+1)$-ésima pessoa
- Continuamos da próxima pessoa
- Ciclamos quando chegamos ao final

Exemplo: $N = 5$ e $M = 2$



Problema de Josephus

```
1 int main() {
2   Lista lista, temp;
```

Problema de Josephus

```
1 int main() {
2   Lista lista, temp;
3   int i, N = 5, M = 2;
```

Problema de Josephus

```
1 int main() {
2   Lista lista, temp;
3   int i, N = 5, M = 2;
4   lista = criar_lista_circular();
```

Problema de Josephus

```
1 int main() {
2   Lista lista, temp;
3   int i, N = 5, M = 2;
4   lista = criar_lista_circular();
5   for (i = 0; i < N; i++)
6     lista = inserir_fim_circular(lista, i);
```

Problema de Josephus

```
1 int main() {
2   Lista lista, temp;
3   int i, N = 5, M = 2;
4   lista = criar_lista_circular();
5   for (i = 0; i < N; i++)
6     lista = inserir_fim_circular(lista, i);
7   while (lista != lista->prox) {
```

Problema de Josephus

```
1 int main() {
2   Lista lista, temp;
3   int i, N = 5, M = 2;
4   lista = criar_lista_circular();
5   for (i = 0; i < N; i++)
6     lista = inserir_fim_circular(lista, i);
7   while (lista != lista->prox) {
8     for (i = 1; i <= M; i++)
9       lista = lista->prox;
```

Problema de Josephus

```
1  int main() {
2    Lista lista, temp;
3    int i, N = 5, M = 2;
4    lista = criar_lista_circular();
5    for (i = 0; i < N; i++)
6      lista = inserir_fim_circular(lista, i);
7    while (lista != lista->prox) {
8      for (i = 1; i <= M; i++)
9        lista = lista->prox;
10     temp = lista->prox;
```

Problema de Josephus

```
1  int main() {
2    Lista lista, temp;
3    int i, N = 5, M = 2;
4    lista = criar_lista_circular();
5    for (i = 0; i < N; i++)
6      lista = inserir_fim_circular(lista, i);
7    while (lista != lista->prox) {
8      for (i = 1; i <= M; i++)
9        lista = lista->prox;
10     temp = lista->prox;
11     lista->prox = lista->prox->prox;
```

Problema de Josephus

```
1  int main() {
2    Lista lista, temp;
3    int i, N = 5, M = 2;
4    lista = criar_lista_circular();
5    for (i = 0; i < N; i++)
6      lista = inserir_fim_circular(lista, i);
7    while (lista != lista->prox) {
8      for (i = 1; i <= M; i++)
9        lista = lista->prox;
10     temp = lista->prox;
11     lista->prox = lista->prox->prox;
12     free(temp);
```

Problema de Josephus

```
1  int main() {
2    Lista lista, temp;
3    int i, N = 5, M = 2;
4    lista = criar_lista_circular();
5    for (i = 0; i < N; i++)
6      lista = inserir_fim_circular(lista, i);
7    while (lista != lista->prox) {
8      for (i = 1; i <= M; i++)
9        lista = lista->prox;
10     temp = lista->prox;
11     lista->prox = lista->prox->prox;
12     free(temp);
13   }
14   printf("%d\n", lista->dado);
15   return 0;
16 }
```

Revistando a Inserção

O código para inserir em uma lista circular não está bom

```
1  Lista inserir_circular(Lista lista, int x) {
2    Lista novo;
3    novo = malloc(sizeof(struct no));
4    novo->dado = x;
5    if (lista == NULL) {
6      novo->prox = novo;
7      lista = novo;
8    } else {
9      novo->prox = lista->prox;
10     lista->prox = novo;
11   }
12   return lista;
13 }
```

Revistando a Inserção

O código para inserir em uma lista circular não está bom

```
1  Lista inserir_circular(Lista lista, int x) {
2    Lista novo;
3    novo = malloc(sizeof(struct no));
4    novo->dado = x;
5    if (lista == NULL) {
6      novo->prox = novo;
7      lista = novo;
8    } else {
9      novo->prox = lista->prox;
10     lista->prox = novo;
11   }
12   return lista;
13 }
```

Precisa lidar com dois casos

Revistando a Inserção

O código para inserir em uma lista circular não está bom

```
1  Lista inserir_circular(Lista lista, int x) {
2    Lista novo;
3    novo = malloc(sizeof(struct no));
4    novo->dado = x;
5    if (lista == NULL) {
6      novo->prox = novo;
7      lista = novo;
8    } else {
9      novo->prox = lista->prox;
10     lista->prox = novo;
11   }
12   return lista;
13 }
```

Precisa lidar com dois casos

- Lista vazia ou não vazia

Revistando a Inserção

O código para inserir em uma lista circular não está bom

```
1  Lista inserir_circular(Lista lista, int x) {
2    Lista novo;
3    novo = malloc(sizeof(struct no));
4    novo->dado = x;
5    if (lista == NULL) {
6      novo->prox = novo;
7      lista = novo;
8    } else {
9      novo->prox = lista->prox;
10     lista->prox = novo;
11   }
12   return lista;
13 }
```

Precisa lidar com dois casos

- Lista vazia ou não vazia
- A remoção sofre com o mesmo problema

Revistando a Inserção

O código para inserir em uma lista circular não está bom

```
1  Lista inserir_circular(Lista lista, int x) {
2    Lista novo;
3    novo = malloc(sizeof(struct no));
4    novo->dado = x;
5    if (lista == NULL) {
6      novo->prox = novo;
7      lista = novo;
8    } else {
9      novo->prox = lista->prox;
10     lista->prox = novo;
11   }
12   return lista;
13 }
```

Precisa lidar com dois casos

- Lista vazia ou não vazia
- A remoção sofre com o mesmo problema

O ponteiro de acesso da lista muda

Revistando a Inserção

O código para inserir em uma lista circular não está bom

```
1  Lista inserir_circular(Lista lista, int x) {
2    Lista novo;
3    novo = malloc(sizeof(struct no));
4    novo->dado = x;
5    if (lista == NULL) {
6      novo->prox = novo;
7      lista = novo;
8    } else {
9      novo->prox = lista->prox;
10     lista->prox = novo;
11   }
12   return lista;
13 }
```

Precisa lidar com dois casos

- Lista vazia ou não vazia
- A remoção sofre com o mesmo problema

O ponteiro de acesso da lista muda

- Quando removemos o nó na última posição

Revistando a Inserção

O código para inserir em uma lista circular não está bom

```
1  Lista inserir_circular(Lista lista, int x) {
2    Lista novo;
3    novo = malloc(sizeof(struct no));
4    novo->dado = x;
5    if (lista == NULL) {
6      novo->prox = novo;
7      lista = novo;
8    } else {
9      novo->prox = lista->prox;
10     lista->prox = novo;
11   }
12   return lista;
13 }
```

Precisa lidar com dois casos

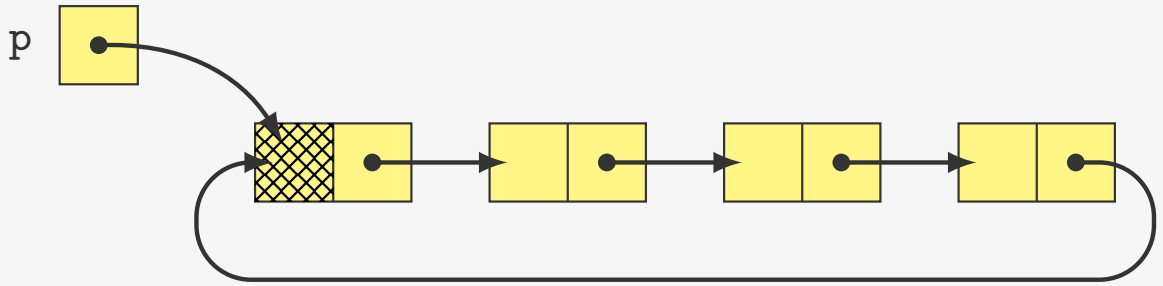
- Lista vazia ou não vazia
- A remoção sofre com o mesmo problema

O ponteiro de acesso da lista muda

- Quando removemos o nó na última posição
- Quando removemos todos os nós

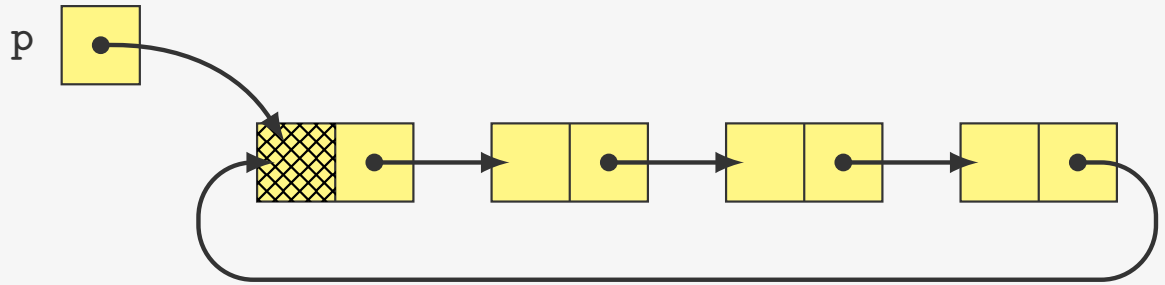
Listas circulares com cabeça

Lista circular com cabeça:

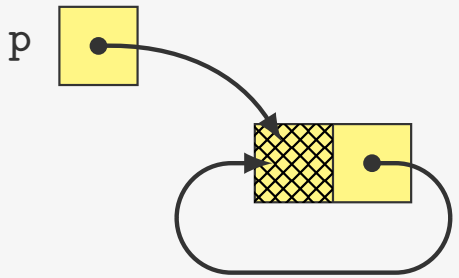


Listas circulares com cabeça

Lista circular com cabeça:

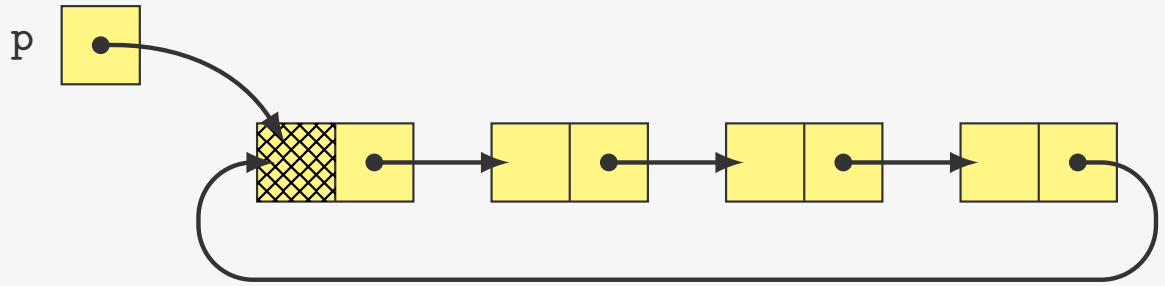


Lista circular vazia:

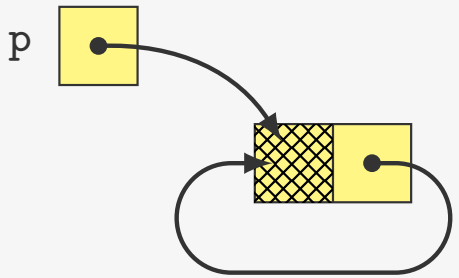


Listas circulares com cabeça

Lista circular com cabeça:



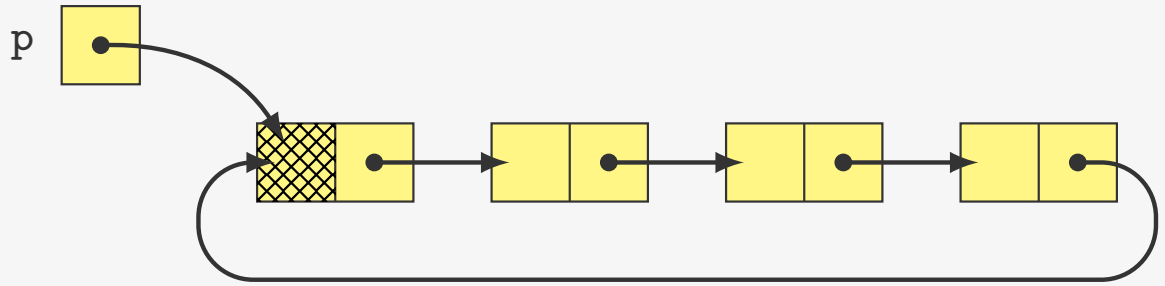
Lista circular vazia:



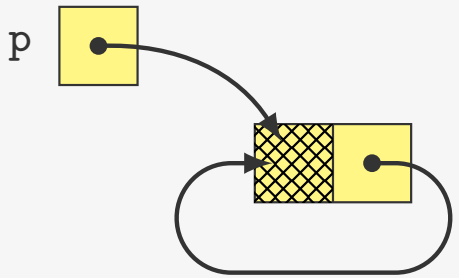
Diferenças para a versão sem cabeça:

Listas circulares com cabeça

Lista circular com cabeça:



Lista circular vazia:

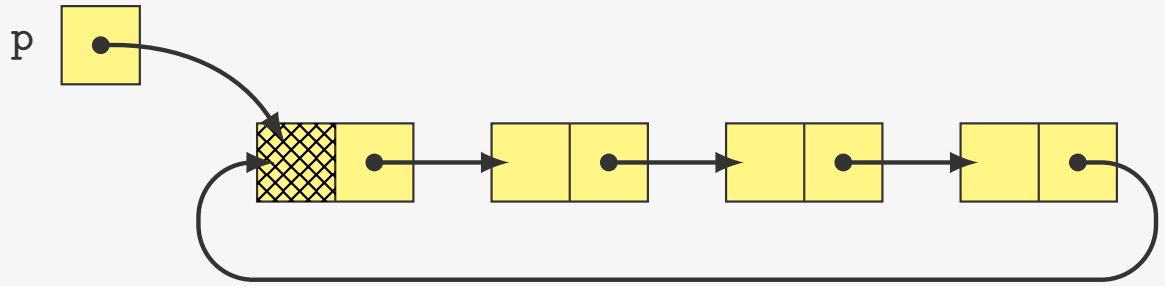


Diferenças para a versão sem cabeça:

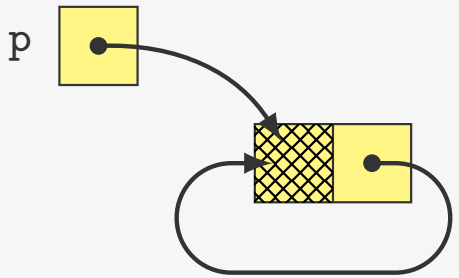
- lista sempre aponta para o nó *dummy*

Listas circulares com cabeça

Lista circular com cabeça:



Lista circular vazia:

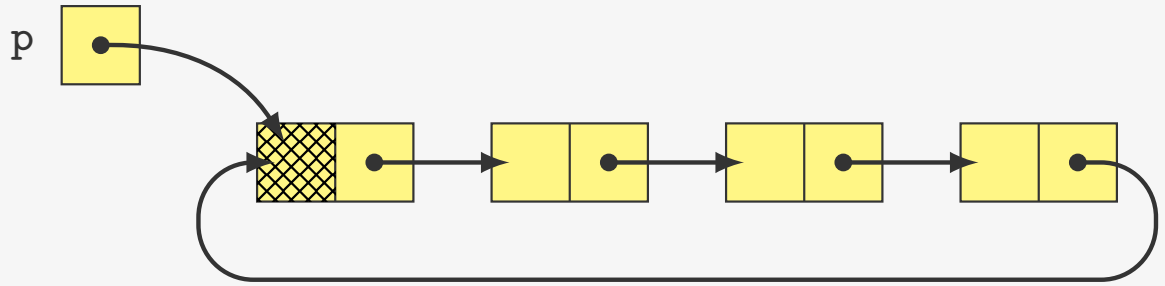


Diferenças para a versão sem cabeça:

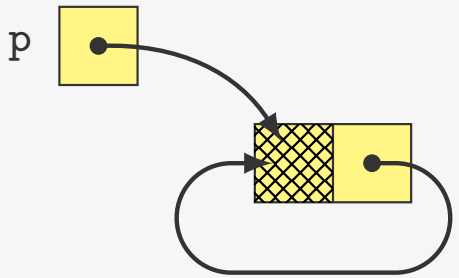
- lista sempre aponta para o nó *dummy*
- código de inserção e de remoção mais simples

Listas circulares com cabeça

Lista circular com cabeça:



Lista circular vazia:



Diferenças para a versão sem cabeça:

- lista sempre aponta para o nó *dummy*
- código de inserção e de remoção mais simples
- ao percorrer, temos que ignorar a cabeça

Inserção e remoção simplificadas

```
1 Lista inserir_circular(Lista lista, int x) {
```

Inserção e remoção simplificadas

```
1 Lista inserir_circular(Lista lista, int x) {
2   Lista novo;
```

Inserção e remoção simplificadas

```
1 Lista inserir_circular(Lista lista, int x) {
2   Lista novo;
3   novo = malloc(sizeof(struct no));
```

Inserção e remoção simplificadas

```
1 Lista inserir_circular(Lista lista, int x) {
2   Lista novo;
3   novo = malloc(sizeof(struct no));
4   novo->dado = x;
```

Inserção e remoção simplificadas

```
1 Lista inserir_circular(Lista lista, int x) {
2   Lista novo;
3   novo = malloc(sizeof(struct no));
4   novo->dado = x;
5   novo->prox = lista->prox;
```

Inserção e remoção simplificadas

```
1 Lista inserir_circular(Lista lista, int x) {
2   Lista novo;
3   novo = malloc(sizeof(struct no));
4   novo->dado = x;
5   novo->prox = lista->prox;
6   lista->prox = novo;
```

Inserção e remoção simplificadas

```
1 Lista inserir_circular(Lista lista, int x) {
2   Lista novo;
3   novo = malloc(sizeof(struct no));
4   novo->dado = x;
5   novo->prox = lista->prox;
6   lista->prox = novo;
7   return lista;
8 }
```

Inserção e remoção simplificadas

```
1 Lista inserir_circular(Lista lista, int x) {
2   Lista novo;
3   novo = malloc(sizeof(struct no));
4   novo->dado = x;
5   novo->prox = lista->prox;
6   lista->prox = novo;
7   return lista;
8 }
```

```
1 Lista remover_circular(Lista lista, Lista no) {
```

Inserção e remoção simplificadas

```
1 Lista inserir_circular(Lista lista, int x) {
2   Lista novo;
3   novo = malloc(sizeof(struct no));
4   novo->dado = x;
5   novo->prox = lista->prox;
6   lista->prox = novo;
7   return lista;
8 }
```

```
1 Lista remover_circular(Lista lista, Lista no) {
2   Lista ant;
```

Inserção e remoção simplificadas

```
1 Lista inserir_circular(Lista lista, int x) {
2   Lista novo;
3   novo = malloc(sizeof(struct no));
4   novo->dado = x;
5   novo->prox = lista->prox;
6   lista->prox = novo;
7   return lista;
8 }
```

```
1 Lista remover_circular(Lista lista, Lista no) {
2   Lista ant;
3   for(ant = no->prox; ant->prox != no; ant = ant->prox);
```

Inserção e remoção simplificadas

```
1 Lista inserir_circular(Lista lista, int x) {
2   Lista novo;
3   novo = malloc(sizeof(struct no));
4   novo->dado = x;
5   novo->prox = lista->prox;
6   lista->prox = novo;
7   return lista;
8 }
```

```
1 Lista remover_circular(Lista lista, Lista no) {
2   Lista ant;
3   for(ant = no->prox; ant->prox != no; ant = ant->prox);
4   ant->prox = no->prox;
```

Inserção e remoção simplificadas

```
1 Lista inserir_circular(Lista lista, int x) {
2   Lista novo;
3   novo = malloc(sizeof(struct no));
4   novo->dado = x;
5   novo->prox = lista->prox;
6   lista->prox = novo;
7   return lista;
8 }
```

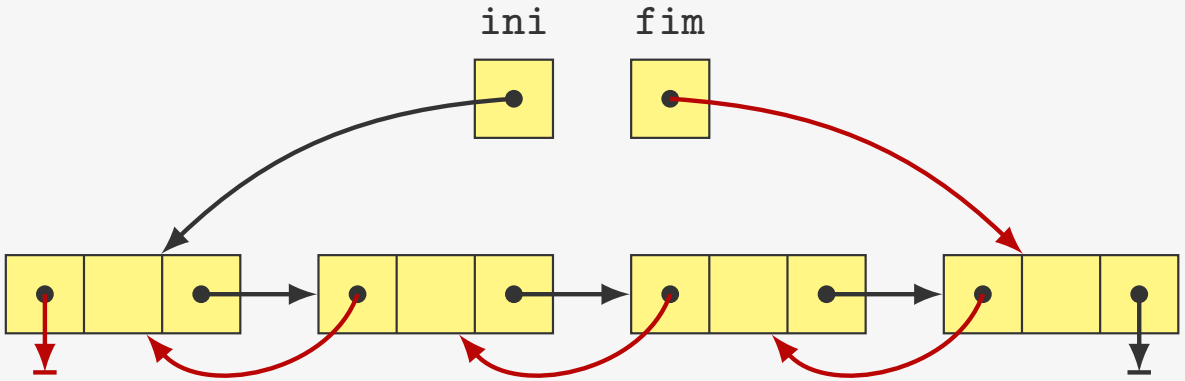
```
1 Lista remover_circular(Lista lista, Lista no) {
2   Lista ant;
3   for(ant = no->prox; ant->prox != no; ant = ant->prox);
4   ant->prox = no->prox;
5   free(no);
```

Inserção e remoção simplificadas

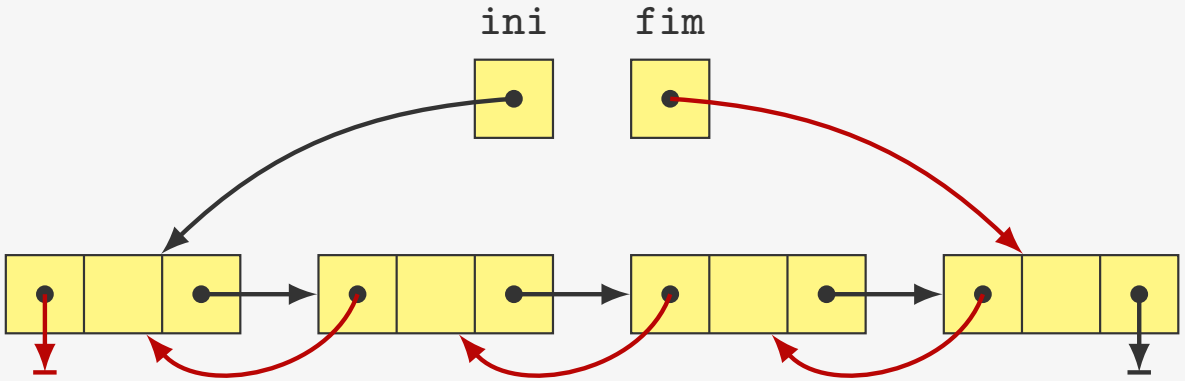
```
1 Lista inserir_circular(Lista lista, int x) {
2   Lista novo;
3   novo = malloc(sizeof(struct no));
4   novo->dado = x;
5   novo->prox = lista->prox;
6   lista->prox = novo;
7   return lista;
8 }
```

```
1 Lista remover_circular(Lista lista, Lista no) {
2   Lista ant;
3   for(ant = no->prox; ant->prox != no; ant = ant->prox);
4   ant->prox = no->prox;
5   free(no);
6   return lista;
7 }
```

Variações — Duplamente ligada



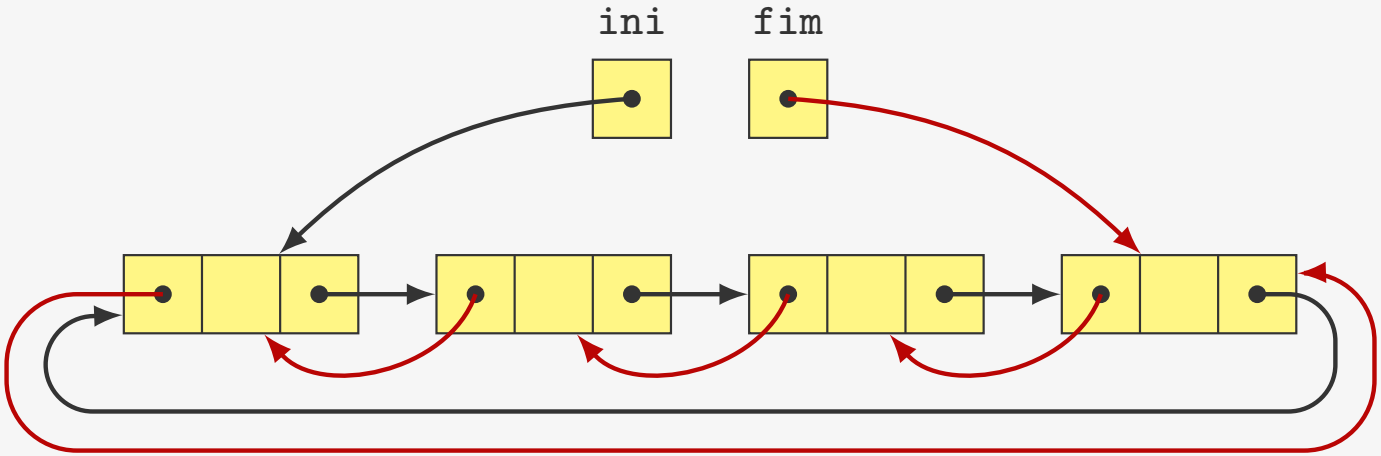
Variações — Duplamente ligada



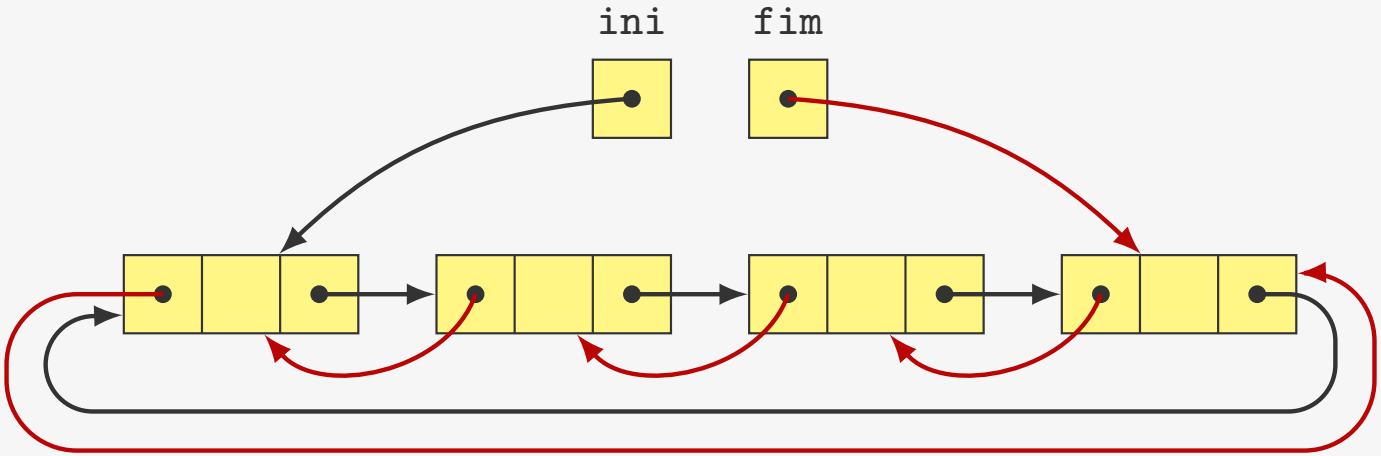
Exemplos:

- Operações desfazer/refazer em software
- Player de música (música anterior e próxima música)

Variações — Lista dupla circular

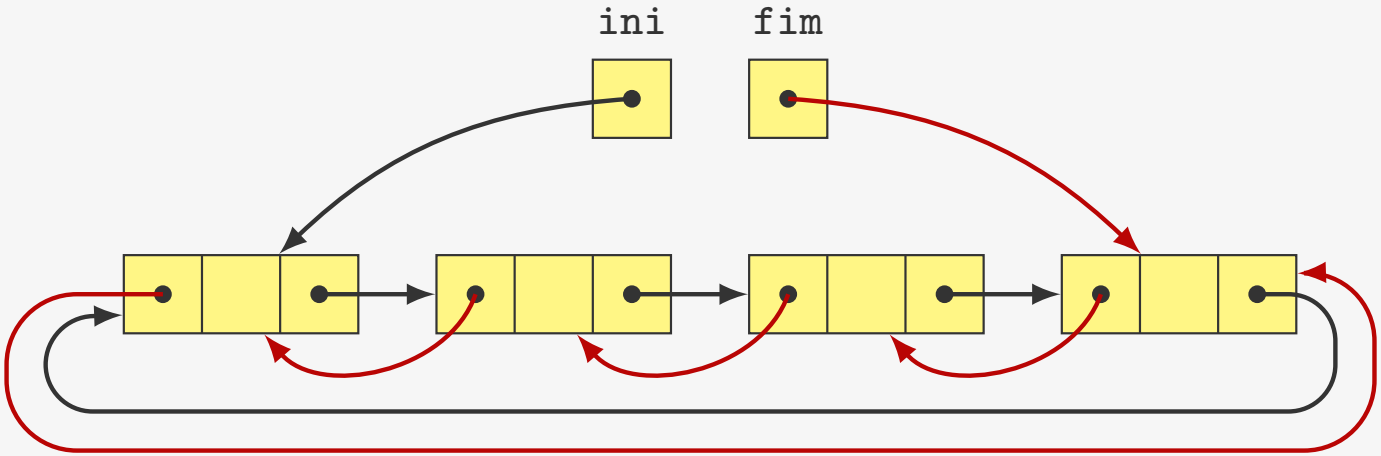


Variações — Lista dupla circular



Permite inserção e remoção em $O(1)$

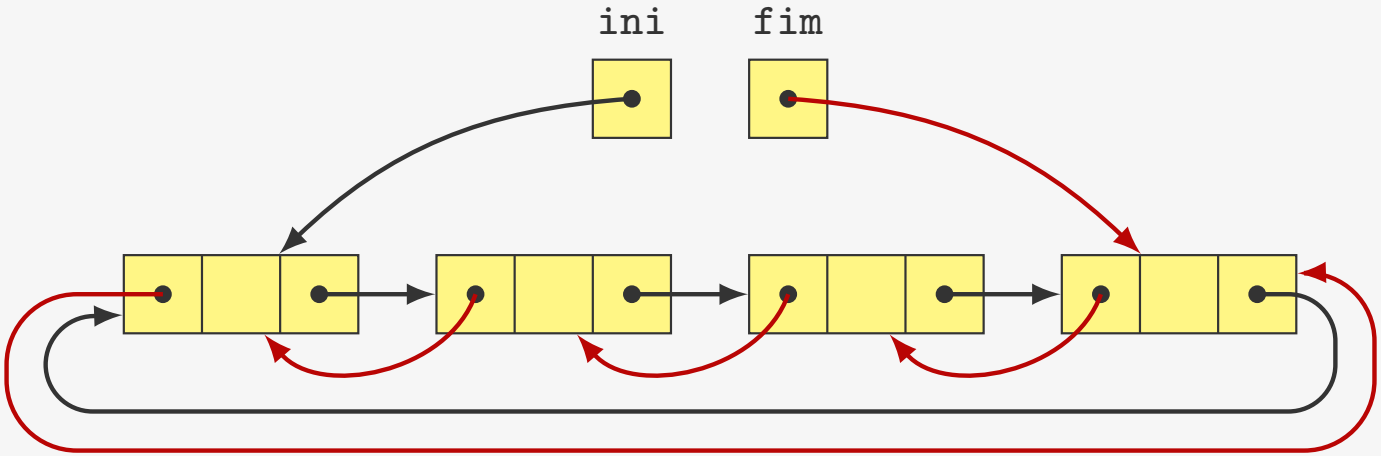
Variações — Lista dupla circular



Permite inserção e remoção em $O(1)$

- Variável `fim` é opcional (`fim == ini->ant`)

Variações — Lista dupla circular



Permite inserção e remoção em $O(1)$

- Variável `fim` é opcional (`fim == ini->ant`)

Podemos ter uma lista dupla circular com cabeça também...

Exercício

Represente polinômios utilizando listas ligadas e apresente uma função que soma dois polinômios.

Exercício

Implemente a operação *inserir elemento* de uma lista duplamente ligada.

Exercício

Escreva uma função que devolve a concatenação de duas listas circulares dadas. Sua função pode destruir a estrutura das listas dadas.