MC-202
Operações em listas e variações

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

Universidade Estadual de Campinas

2º semestre/2018

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


Operações em lista ligada

Vamos ver três novas operações para listas ligadas

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

Operações em lista ligada

Vamos ver três novas operações para listas ligadas

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


Operações em lista ligada

Vamos ver três novas operações para listas ligadas

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


Operações em lista ligada

Vamos ver três novas operações para listas ligadas

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


Copiando

Versão recursiva:

Copiando

Versão recursiva:

```
1 p_no copiar_lista(p_no lista) {
```


Copiando

Versão recursiva:

```
1 p_no copiar_lista(p_no lista) {
```

3

Copiando

Versão recursiva:

```
1 p_no copiar_lista(p_no lista) {
2   p_no novo;
3   if (lista == NULL)
4     return NULL;
```

3

Copiando

Versão recursiva:

```
1 p_no copiar_lista(p_no lista) {
2   p_no novo;
3   if (lista == NULL)
4     return NULL;
5   novo = malloc(sizeof(No));
6   novo->dado = lista->dado;
7   novo->prox = copiar_lista(lista->prox);
```

3

Copiando

Versão recursiva:

```
1 p_no copiar_lista(p_no lista) {
2   p_no novo;
3   if (lista == NULL)
4     return NULL;
5   novo = malloc(sizeof(No));
6   novo->dado = lista->dado;
7   novo->prox = copiar_lista(lista->prox);
8   return novo;
9 }
```


Copiando

Versão recursiva:

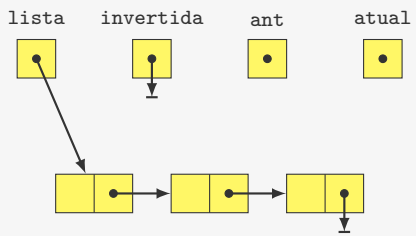
```
1 p_no copiar_lista(p_no lista) {
2   p_no novo;
3   if (lista == NULL)
4     return NULL;
5   novo = malloc(sizeof(No));
6   novo->dado = lista->dado;
7   novo->prox = copiar_lista(lista->prox);
8   return novo;
9 }
```

Exercício: implemente uma versão iterativa da função

3

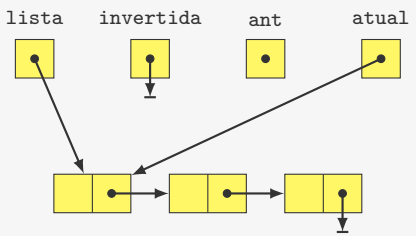
Invertendo

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



Invertendo

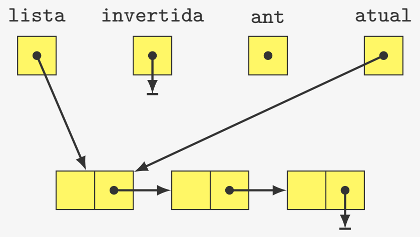
```
1 p_no inverter_lista(p_no lista) {
2   p_no atual, ant, invertida = NULL;
3   atual = lista;
4   while (atual != NULL) {
5     ant = atual;
6     atual = ant->prox;
7     ant->prox = invertida;
8     invertida = ant;
9   }
10  return invertida;
11 }
```



4

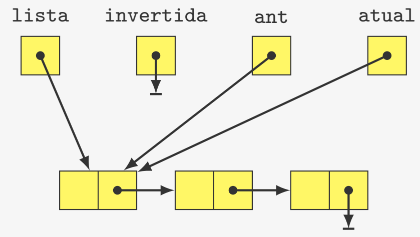
Invertendo

```c
p_no inverter_lista(p_no lista) {
  p_no atual, ant, invertida = NULL;
  atual = lista;
  while (atual != NULL) {
    ant = atual;
    atual = ant->prox;
    ant->prox = invertida;
    invertida = ant;
  }
  return invertida;
}
```



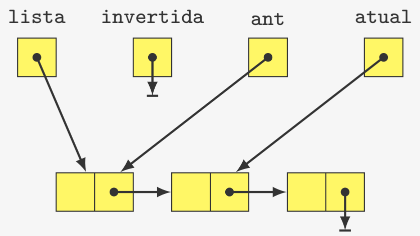
Invertendo

```c
p_no inverter_lista(p_no lista) {
  p_no atual, ant, invertida = NULL;
  atual = lista;
  while (atual != NULL) {
    ant = atual;
    atual = ant->prox;
    ant->prox = invertida;
    invertida = ant;
  }
  return invertida;
}
```



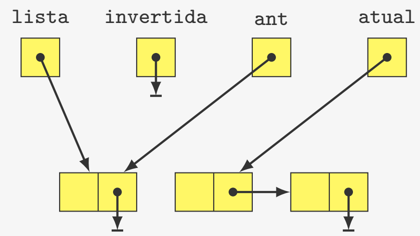
Invertendo

```c
p_no inverter_lista(p_no lista) {
  p_no atual, ant, invertida = NULL;
  atual = lista;
  while (atual != NULL) {
    ant = atual;
    atual = ant->prox;
    ant->prox = invertida;
    invertida = ant;
  }
  return invertida;
}
```



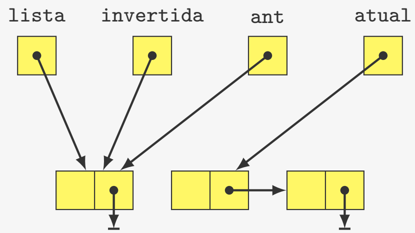
Invertendo

```c
p_no inverter_lista(p_no lista) {
  p_no atual, ant, invertida = NULL;
  atual = lista;
  while (atual != NULL) {
    ant = atual;
    atual = ant->prox;
    ant->prox = invertida;
    invertida = ant;
  }
  return invertida;
}
```



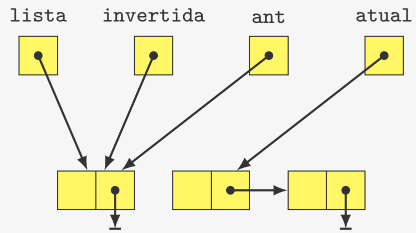
Invertendo

```c
p_no inverter_lista(p_no lista) {
  p_no atual, ant, invertida = NULL;
  atual = lista;
  while (atual != NULL) {
    ant = atual;
    atual = ant->prox;
    ant->prox = invertida;
    invertida = ant;
  }
  return invertida;
}
```



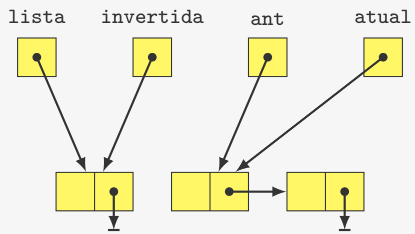
Invertendo

```c
p_no inverter_lista(p_no lista) {
  p_no atual, ant, invertida = NULL;
  atual = lista;
  while (atual != NULL) {
    ant = atual;
    atual = ant->prox;
    ant->prox = invertida;
    invertida = ant;
  }
  return invertida;
}
```



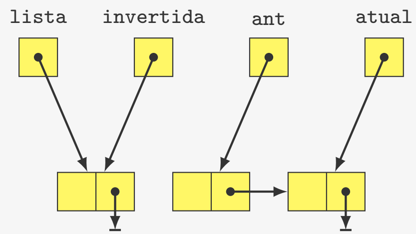
Invertendo

```c
p_no inverter_lista(p_no lista) {
  p_no atual, ant, invertida = NULL;
  atual = lista;
  while (atual != NULL) {
    ant = atual;
    atual = ant->prox;
    ant->prox = invertida;
    invertida = ant;
  }
  return invertida;
}
```



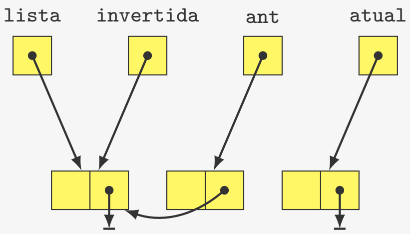
Invertendo

```c
p_no inverter_lista(p_no lista) {
  p_no atual, ant, invertida = NULL;
  atual = lista;
  while (atual != NULL) {
    ant = atual;
    atual = ant->prox;
    ant->prox = invertida;
    invertida = ant;
  }
  return invertida;
}
```



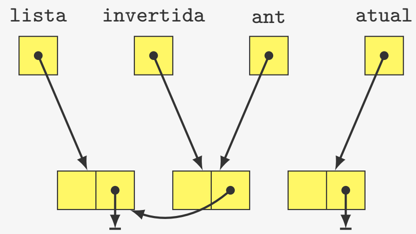
Invertendo

```
p_no inverter_lista(p_no lista) {
  p_no atual, ant, invertida = NULL;
  atual = lista;
  while (atual != NULL) {
    ant = atual;
    atual = ant->prox;
    ant->prox = invertida;
    invertida = ant;
  }
  return invertida;
}
```



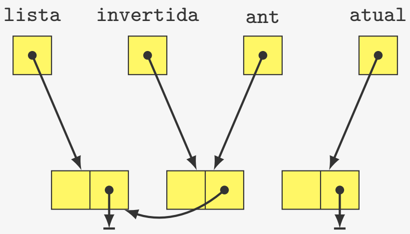
Invertendo

```
p_no inverter_lista(p_no lista) {
  p_no atual, ant, invertida = NULL;
  atual = lista;
  while (atual != NULL) {
    ant = atual;
    atual = ant->prox;
    ant->prox = invertida;
    invertida = ant;
  }
  return invertida;
}
```



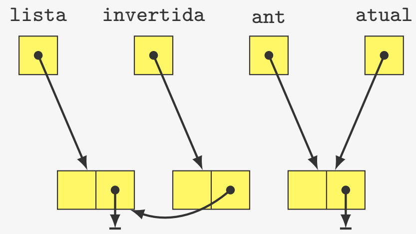
Invertendo

```
p_no inverter_lista(p_no lista) {
  p_no atual, ant, invertida = NULL;
  atual = lista;
  while (atual != NULL) {
    ant = atual;
    atual = ant->prox;
    ant->prox = invertida;
    invertida = ant;
  }
  return invertida;
}
```



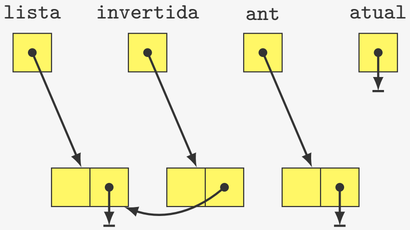
Invertendo

```
p_no inverter_lista(p_no lista) {
  p_no atual, ant, invertida = NULL;
  atual = lista;
  while (atual != NULL) {
    ant = atual;
    atual = ant->prox;
    ant->prox = invertida;
    invertida = ant;
  }
  return invertida;
}
```



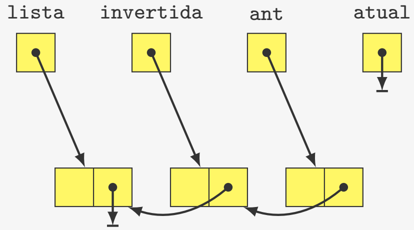
Invertendo

```c
p_no inverter_lista(p_no lista) {
  p_no atual, ant, invertida = NULL;
  atual = lista;
  while (atual != NULL) {
    ant = atual;
    atual = ant->prox;
    ant->prox = invertida;
    invertida = ant;
  }
  return invertida;
}
```



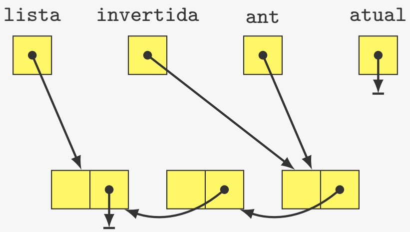
Invertendo

```c
p_no inverter_lista(p_no lista) {
  p_no atual, ant, invertida = NULL;
  atual = lista;
  while (atual != NULL) {
    ant = atual;
    atual = ant->prox;
    ant->prox = invertida;
    invertida = ant;
  }
  return invertida;
}
```



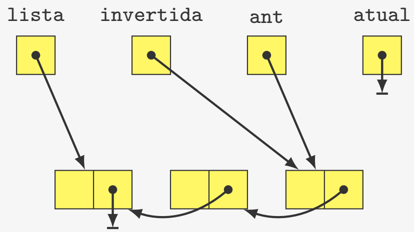
Invertendo

```c
p_no inverter_lista(p_no lista) {
  p_no atual, ant, invertida = NULL;
  atual = lista;
  while (atual != NULL) {
    ant = atual;
    atual = ant->prox;
    ant->prox = invertida;
    invertida = ant;
  }
  return invertida;
}
```



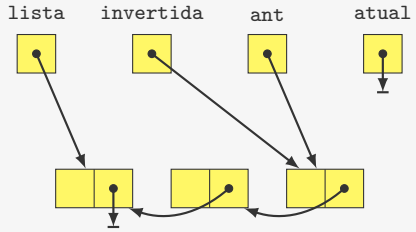
Invertendo

```c
p_no inverter_lista(p_no lista) {
  p_no atual, ant, invertida = NULL;
  atual = lista;
  while (atual != NULL) {
    ant = atual;
    atual = ant->prox;
    ant->prox = invertida;
    invertida = ant;
  }
  return invertida;
}
```



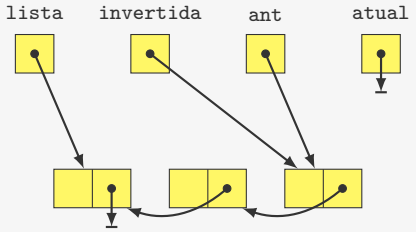
Invertendo

```
1  p_no inverter_lista(p_no lista) {
2    p_no atual, ant, invertida = NULL;
3    atual = lista;
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  p_no inverter_lista(p_no lista) {
2    p_no atual, ant, invertida = NULL;
3    atual = lista;
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  p_no concatenar_lista(p_no primeira, p_no segunda) {
```


Concatenando

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

Concatenando

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


Concatenando

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


Concatenando

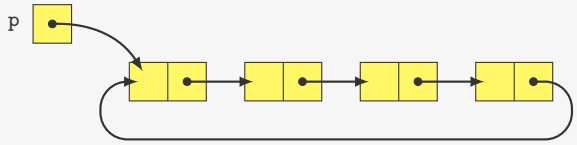
```
1 p_no concatenar_lista(p_no primeira, p_no segunda) {
2   if (primeira == NULL)
3     return segunda;
4   primeira->prox = concatenar_lista(primeira->prox, segunda);
```


Concatenando

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

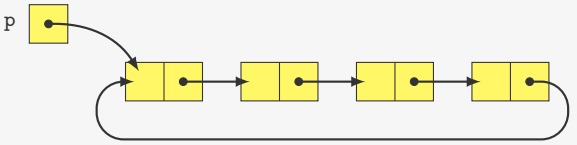
Variações - Listas circulares

Lista circular:



Variações - Listas circulares

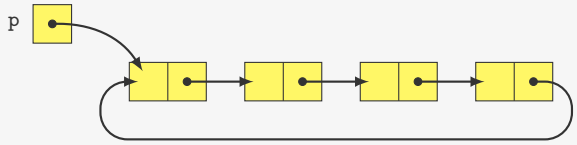
Lista circular:



Lista circular vazia:

Variações - Listas circulares

Lista circular:

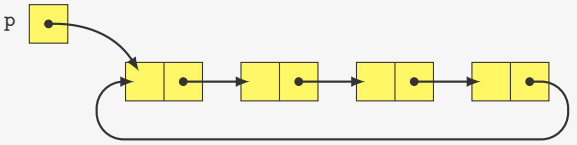


Lista circular vazia:

Exemplo de aplicações:

Variações - Listas circulares

Lista circular:



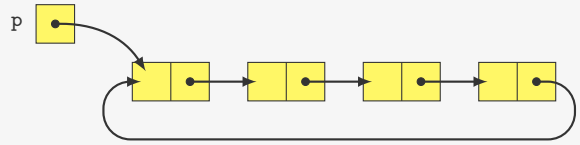
Lista circular vazia:

Exemplo de aplicações:

- Execução de processos no sistema operacional

Variações - Listas circulares

Lista circular:



Lista circular vazia:

Exemplo de aplicações:

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

Inserindo em lista circular

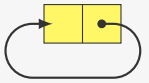
```
1 p_no inserir_circular(p_no lista, int x) {
```


Inserindo em lista circular

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

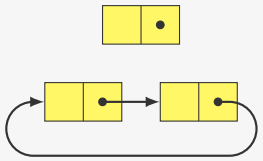

Inserindo em lista circular

```
1 p_no inserir_circular(p_no lista, int x) {
2   p_no novo;
3   novo = malloc(sizeof(No));
4   novo->dado = x;
5   if (lista == NULL)
6     novo->prox = novo;
```



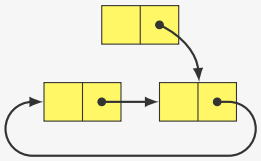
Inserindo em lista circular

```
1 p_no inserir_circular(p_no lista, int x) {
2   p_no novo;
3   novo = malloc(sizeof(No));
4   novo->dado = x;
5   if (lista == NULL)
6     novo->prox = novo;
7   else {
```



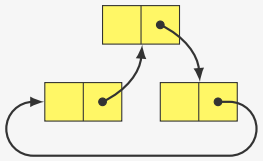
Inserindo em lista circular

```
1 p_no inserir_circular(p_no lista, int x) {
2   p_no novo;
3   novo = malloc(sizeof(No));
4   novo->dado = x;
5   if (lista == NULL)
6     novo->prox = novo;
7   else {
8     novo->prox = lista->prox;
```



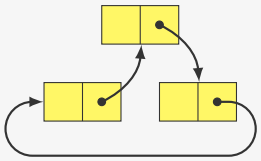
Inserindo em lista circular

```
1 p_no inserir_circular(p_no lista, int x) {
2   p_no novo;
3   novo = malloc(sizeof(No));
4   novo->dado = x;
5   if (lista == NULL)
6     novo->prox = novo;
7   else {
8     novo->prox = lista->prox;
9     lista->prox = novo;
10  }
```



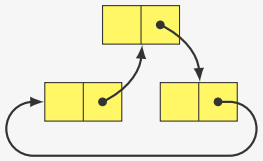
Inserindo em lista circular

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



Inserindo em lista circular

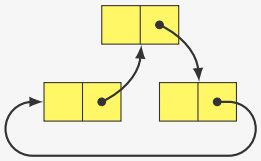
```
1  p_no inserir_circular(p_no lista, int x) {
2    p_no novo;
3    novo = malloc(sizeof(No));
4    novo->dado = x;
5    if (lista == NULL)
6      novo->prox = novo;
7    else {
8      novo->prox = lista->prox;
9      lista->prox = novo;
10   }
11   return novo;
12 }
```



Observações:

Inserindo em lista circular

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

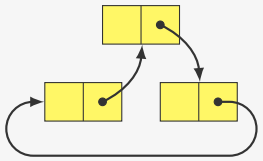


Observações:

- O elemento é inserido na segunda posição

Inserindo em lista circular

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

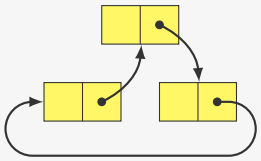


Observações:

- O elemento é inserido na segunda posição
  - Para inserir na primeira precisaria percorrer a lista... $\mathrm{O}(n)$

Inserindo em lista circular

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

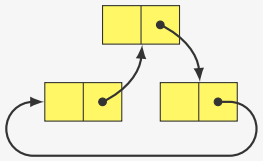


Observações:

- O elemento é inserido na segunda posição
  - Para inserir na primeira precisaria percorrer a lista... $\mathrm{O}(n)$
- É devolvido o ponteiro para o novo elemento

Inserindo em lista circular

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



Observações:

- O elemento é inserido na segunda posição
  - Para inserir na primeira precisaria percorrer a lista... $O(n)$
- É devolvido o ponteiro para o novo elemento
  - Ex: ao inserir 0 em $(3, 7, 2)$ ficamos com $(0, 7, 2, 3)$

Removendo de lista circular

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


Removendo de lista circular

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


Removendo de lista circular

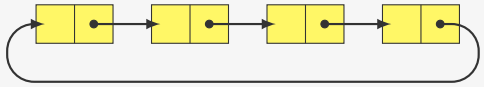
```
1  p_no remover_circular(p_no lista, p_no no) {
2    p_no ant;
3    if (no->prox == no) {
4      free(no);
5      return NULL;
6    }
```

Removendo de lista circular

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

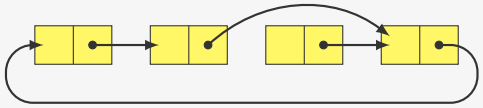

Removendo de lista circular

```
1 p_no remover_circular(p_no lista, p_no no) {
2   p_no 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;
```



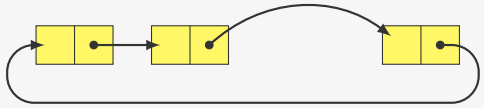
Removendo de lista circular

```
1 p_no remover_circular(p_no lista, p_no no) {
2   p_no 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 = lista->prox;
```



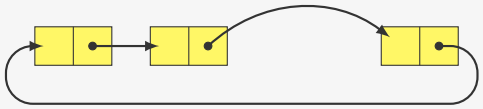
Removendo de lista circular

```
1 p_no remover_circular(p_no lista, p_no no) {
2   p_no 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 = lista->prox;
11  free(no);
```



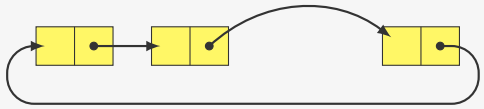
Removendo de lista circular

```
1  p_no remover_circular(p_no lista, p_no no) {
2    p_no 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 = lista->prox;
11   free(no);
12   return lista;
13 }
```



Removendo de lista circular

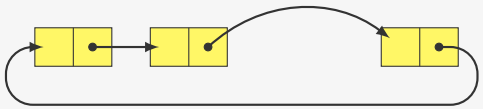
```
1  p_no remover_circular(p_no lista, p_no no) {
2    p_no ant;
3    if (no->prox == no) {
4      free(no);
5      return NULL;
6    }
7    for(ant = no->prox; ant->prox != no; ant = ant->prox);
8    ant->prox = no->prox;
9    if (lista == no)
10     lista = lista->prox;
11   free(no);
12   return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



8

Removendo de lista circular

```
1  p_no remover_circular(p_no lista, p_no no) {
2    p_no 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 = lista->prox;
11   free(no);
12   return lista;
13 }
```

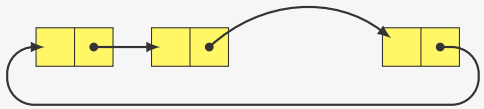


Tempo: $O(n)$

8

Removendo de lista circular

```
1  p_no remover_circular(p_no lista, p_no no) {
2    p_no 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 = lista->prox;
11   free(no);
12   return lista;
13 }
```



Tempo: $O(n)$

- Podemos melhorar se soubermos o nó anterior...

8

Percorrendo uma lista circular

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


Percorrendo uma lista circular

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

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

Percorrendo uma lista circular

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

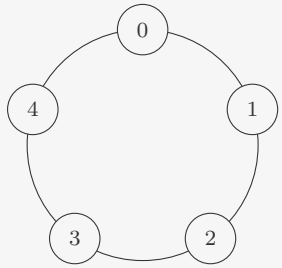
- Começamos a contar da primeira pessoa
- Contamos M pessoas
- Eliminamos $(M+1)$-ésima pessoa
- Continuamos da próxima pessoa
- Ciclamos ao chegar ao final

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 ao chegar ao final

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

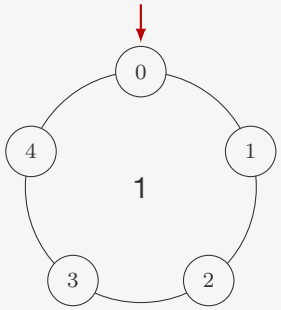


Exercício - Problema de Josephus

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 ao chegar ao final

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

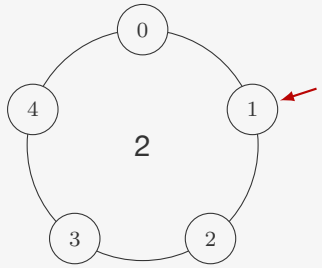


Exercício - Problema de Josephus

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 ao chegar ao final

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

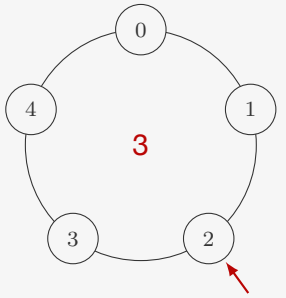


Exercício - Problema de Josephus

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 ao chegar ao final

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

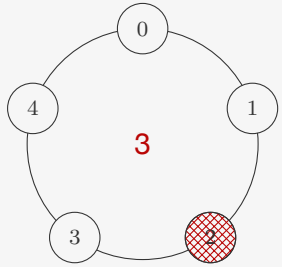


Exercício - Problema de Josephus

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 ao chegar ao final

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

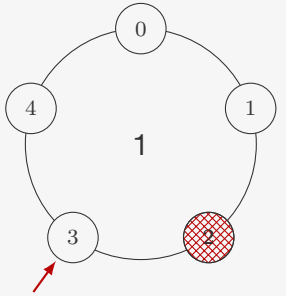


Exercício - Problema de Josephus

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 ao chegar ao final

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

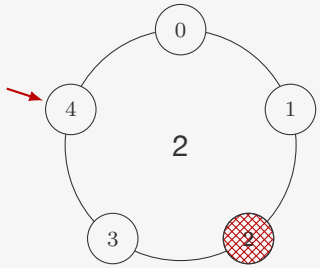


Exercício - Problema de Josephus

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 ao chegar ao final

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

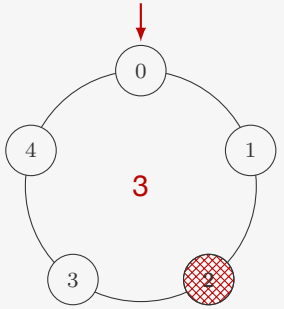


Exercício - Problema de Josephus

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 ao chegar ao final

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



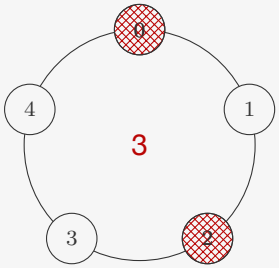
10

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 ao chegar ao final

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



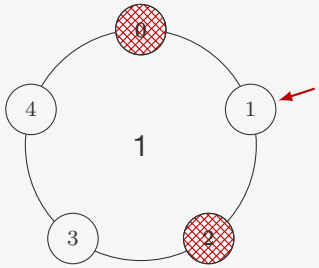
10

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 ao chegar ao final

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



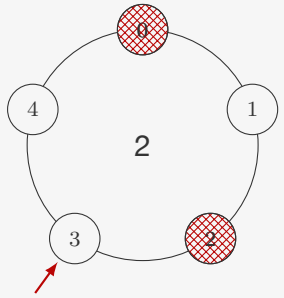
10

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 ao chegar ao final

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

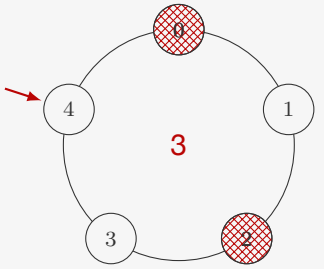


Exercício - Problema de Josephus

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 ao chegar ao final

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

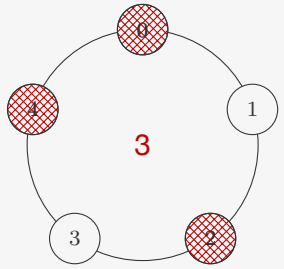


Exercício - Problema de Josephus

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 ao chegar ao final

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

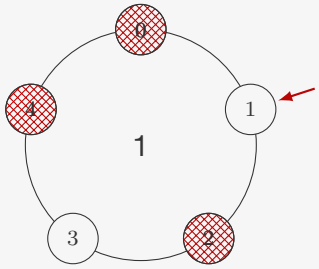


Exercício - Problema de Josephus

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 ao chegar ao final

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

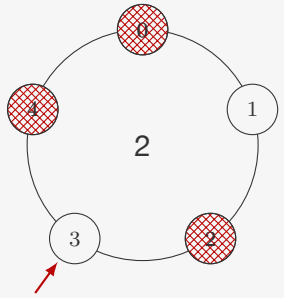


Exercício - Problema de Josephus

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 ao chegar ao final

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

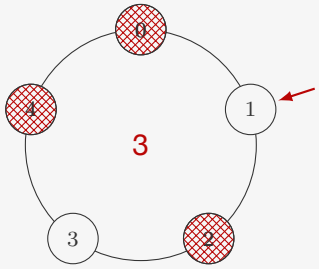


Exercício - Problema de Josephus

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 ao chegar ao final

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

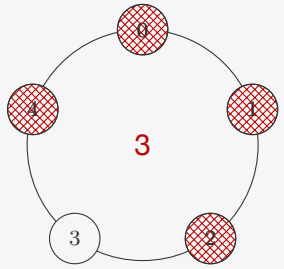


Exercício - Problema de Josephus

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 ao chegar ao final

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



Problema de Josephus

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

Problema de Josephus

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


Problema de Josephus

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


Problema de Josephus

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


Problema de Josephus

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

Problema de Josephus

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


Problema de Josephus

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


Problema de Josephus

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


Problema de Josephus

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

Problema de Josephus

```c
int main() {
  p_no lista, temp;
  int i, N = 5, M = 2;
  lista = criar_lista_circular();
  for (i = 0; i < N; i++)
    lista = inserir_circular(lista, i);
  while (lista != lista->prox) {
    for (i = 0; i < M; i++)
      lista = lista->prox;
    temp = lista->prox;
    lista->prox = lista->prox->prox;
    free(temp);
  }
  printf("%d\n", lista->dado);
  return 0;
}
```


Revistando a Inserção

O código para inserir em uma lista circular não está bom

Revistando a Inserção

O código para inserir em uma lista circular não está bom

```c
p_no inserir_circular(p_no lista, int x) {
  p_no novo;
  novo = malloc(sizeof(No));
  novo->dado = x;
  if (lista == NULL)
    novo->prox = novo;
  else {
    novo->prox = lista->prox;
    lista->prox = novo;
  }
  return novo;
}
```


Revistando a Inserção

O código para inserir em uma lista circular não está bom

```c
p_no inserir_circular(p_no lista, int x) {
  p_no novo;
  novo = malloc(sizeof(No));
  novo->dado = x;
  if (lista == NULL)
    novo->prox = novo;
  else {
    novo->prox = lista->prox;
    lista->prox = novo;
  }
  return novo;
}
```

Precisa lidar com dois casos

Revistando a Inserção

O código para inserir em uma lista circular não está bom

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

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

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

Precisa lidar com dois casos

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

Tem um comportamento estranho

Revistando a Inserção

O código para inserir em uma lista circular não está bom

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

Precisa lidar com dois casos

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

Tem um comportamento estranho

- Inserimos após o primeiro elemento

Revistando a Inserção

O código para inserir em uma lista circular não está bom

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

Precisa lidar com dois casos

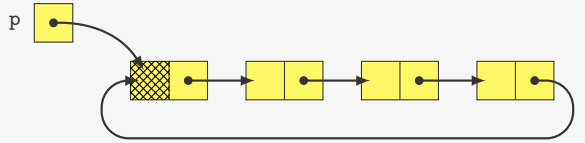
- lista vazia ou não vazia
- A remoção sofre com o mesmo problema

Tem um comportamento estranho

- Inserimos após o primeiro elemento
- Mudamos quem é o primeiro elemento da lista

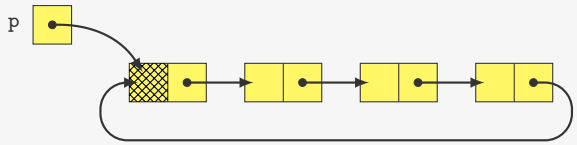
Listas circulares com cabeça

Lista circular com cabeça:

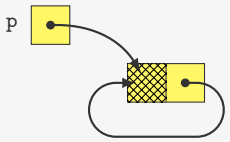


Listas circulares com cabeça

Lista circular com cabeça:

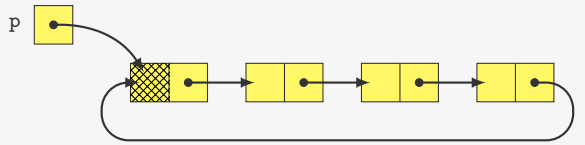


Lista circular vazia:

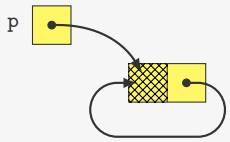


Listas circulares com cabeça

Lista circular com cabeça:



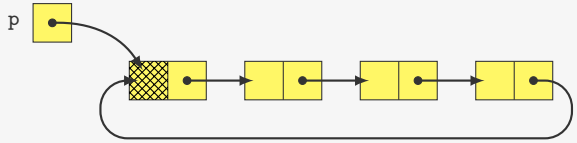
Lista circular vazia:



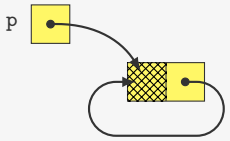
Diferenças para a versão sem cabeça:

Listas circulares com cabeça

Lista circular com cabeça:



Lista circular vazia:

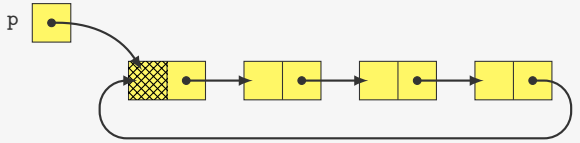


Diferenças para a versão sem cabeça:

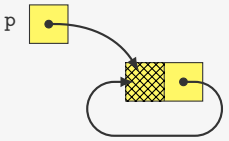
- lista sempre aponta para o nó *dummy*

Listas circulares com cabeça

Lista circular com cabeça:



Lista circular vazia:

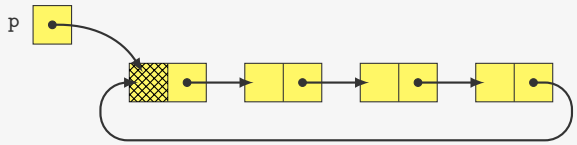


Diferenças para a versão sem cabeça:

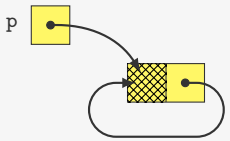
- lista sempre aponta para o nó *dummy*
- código de inserção e de remoção mais simples

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 tem que ignorar cabeça

Inserção e remoção simplificadas

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

Inserção e remoção simplificadas

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


Inserção e remoção simplificadas

```
1 p_no inserir_circular(p_no lista, int x) {
2   p_no novo;
3   novo = malloc(sizeof(No));
```


Inserção e remoção simplificadas

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


Inserção e remoção simplificadas

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

Inserção e remoção simplificadas

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


Inserção e remoção simplificadas

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


Inserção e remoção simplificadas

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

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


Inserção e remoção simplificadas

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

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

Inserção e remoção simplificadas

```c
p_no inserir_circular(p_no lista, int x) {
  p_no novo;
  novo = malloc(sizeof(No));
  novo->dado = x;
  novo->prox = lista->prox;
  lista->prox = novo;
  return lista;
}
```

```c
p_no remover_circular(p_no lista, p_no no) {
  p_no ant;
  for(ant = no->prox; ant->prox != no; ant = ant->prox);
```


Inserção e remoção simplificadas

```c
p_no inserir_circular(p_no lista, int x) {
  p_no novo;
  novo = malloc(sizeof(No));
  novo->dado = x;
  novo->prox = lista->prox;
  lista->prox = novo;
  return lista;
}
```

```c
p_no remover_circular(p_no lista, p_no no) {
  p_no ant;
  for(ant = no->prox; ant->prox != no; ant = ant->prox);
  ant->prox = no->prox;
```

14

Inserção e remoção simplificadas

```c
p_no inserir_circular(p_no lista, int x) {
  p_no novo;
  novo = malloc(sizeof(No));
  novo->dado = x;
  novo->prox = lista->prox;
  lista->prox = novo;
  return lista;
}
```

```c
p_no remover_circular(p_no lista, p_no no) {
  p_no ant;
  for(ant = no->prox; ant->prox != no; ant = ant->prox);
  ant->prox = no->prox;
```

```c
  free(no);
```

14

Inserção e remoção simplificadas

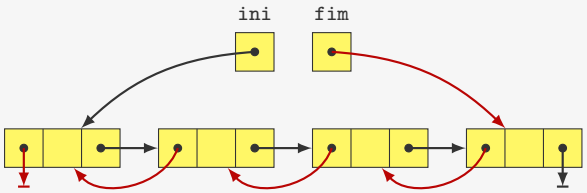
```c
p_no inserir_circular(p_no lista, int x) {
  p_no novo;
  novo = malloc(sizeof(No));
  novo->dado = x;
  novo->prox = lista->prox;
  lista->prox = novo;
  return lista;
}
```

```c
p_no remover_circular(p_no lista, p_no no) {
  p_no ant;
  for(ant = no->prox; ant->prox != no; ant = ant->prox);
  ant->prox = no->prox;
  free(no);
```

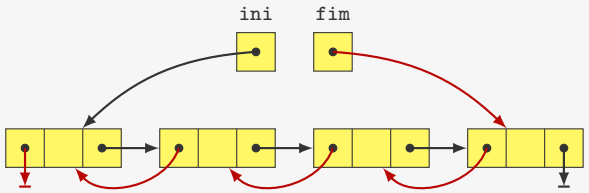
```c
  return lista;
}
```

14

Variações - Duplamente ligada



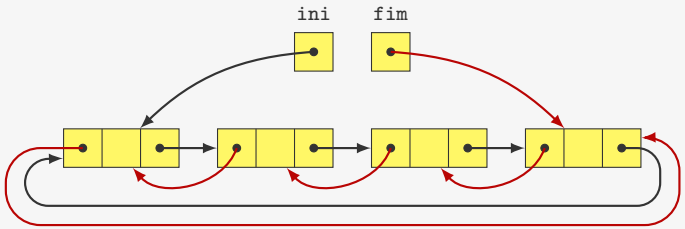
Variações - Duplamente ligada



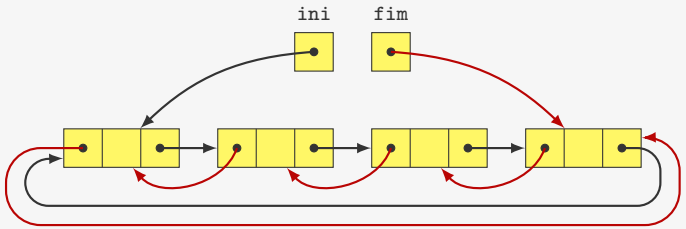
Exemplos:

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

Variações - Lista dupla circular

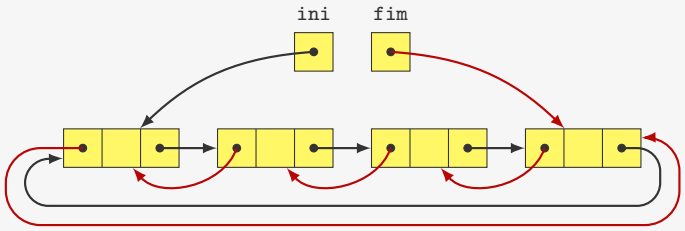


Variações - Lista dupla circular



Permite inserção e remoção em $O(1)$

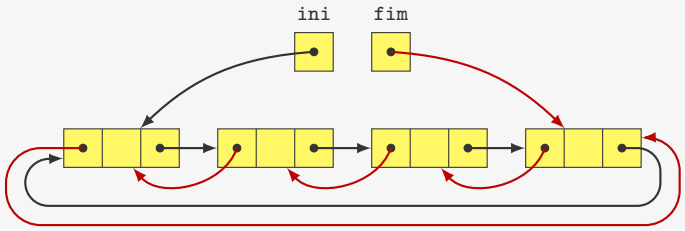
Variações - Lista dupla circular



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.

Exercício

Escreva uma função que dada uma lista duplamente ligada (sem cabeça) e dois de seus nós, troca os dois nós de lugar na lista.