Processamento Cossequencial

Organização e Recuperação de Dados

Profa. Valéria

UEM – CTC – DIN

Slides preparados com base no Cap. 7 do livro FOLK, M.J. & ZOELLICK, B. *File Structures*. 2nd Edition, Addison-Wesley Publishing Company, 1992, e nos slides disponibilizados pelo Prof. Pedro de Azevedo Berger (DCC/UnB)

Processamento cossequencial

- Objetivo
  - Processar de forma coordenada duas ou mais listas sequenciais para produzir uma lista única como saída
    - Por ex., duas ou mais listas de chaves de primárias
- As listas de entrada:
  - Devem estar ordenadas por chave
  - Não devem possuir chaves duplicadas
- A lista de saída:
  - Será ordenada por chave
  - Não possuirá chaves duplicadas

Processamento cossequencial

- Tipos de listas resultantes:
  - Interseção (*Match*)
    - A lista de saída é formada por chaves que ocorrem em todas as listas de entrada
  - União/Intercalação (*Merge*)
    - A lista de saída é formada por todas as chaves das listas de entrada, sem duplicação
    - Os itens são intercalados pela ordem de chave

Processamento cossequencial

- Os algoritmos devem contemplar:
  - Inicialização
    - Abertura dos arquivos de entrada e saída
    - Inicialização da variável de controle do laço
    - Inicialização de variáveis usadas na checagem de sequência
  - Leitura dos arquivos
    - Teste da condição de parada
    - Reconhecimento de erro para os arquivos de entrada
      - Verificar se existem chaves duplicadas
      - Verificar se a sequência de chaves está desordenada
  - Sincronização das leituras e geração da saída

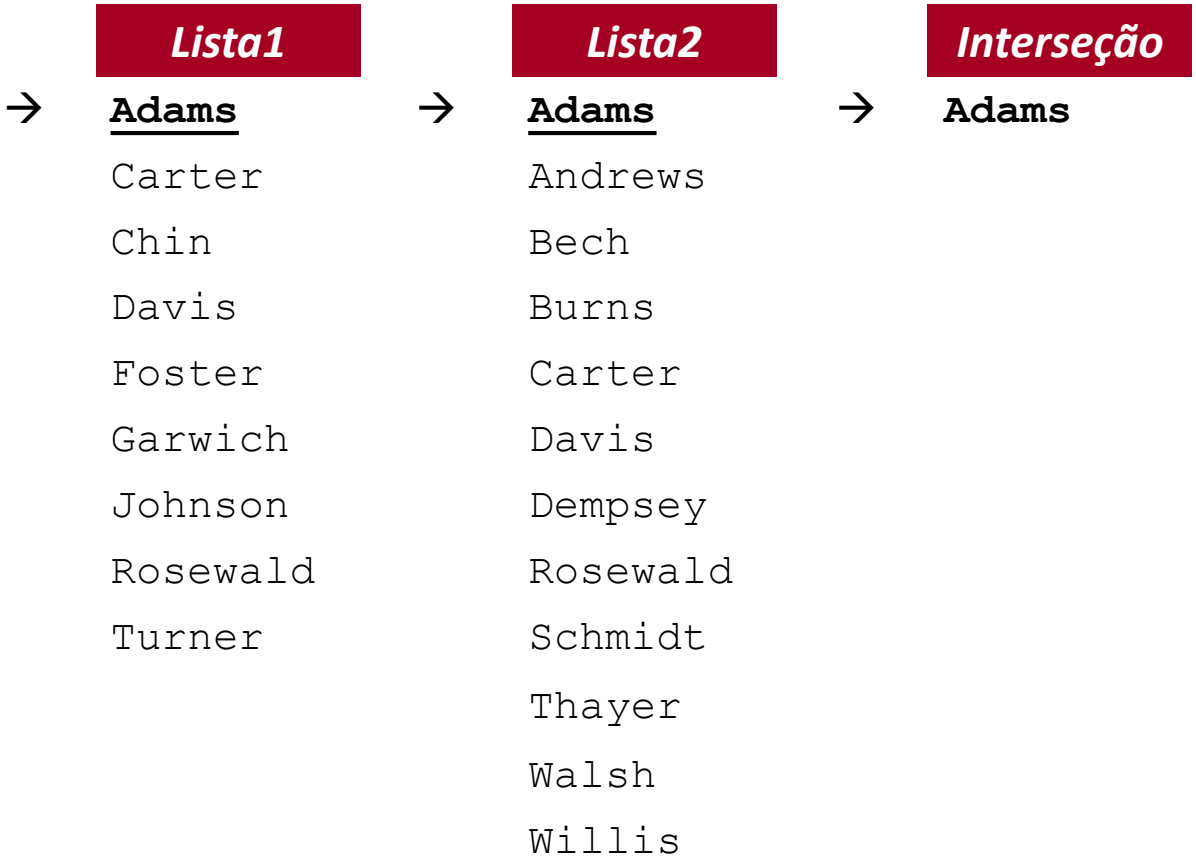
Processamento cossequencial

- Interseção (*Matching*)

Lista1	*Lista2*	*Interseção*
Adams	Adams	
Carter	Andrews	
Chin	Bech	
Davis	Burns	
Foster	Carter	
Garwich	Davis	
Johnson	Dempsey	
Rosewald	Rosewald	
Turner	Schmidt	
	Thayer	
	Walsh	
	Willis	

Processamento cossequencial

- Interseção (*Matching*)



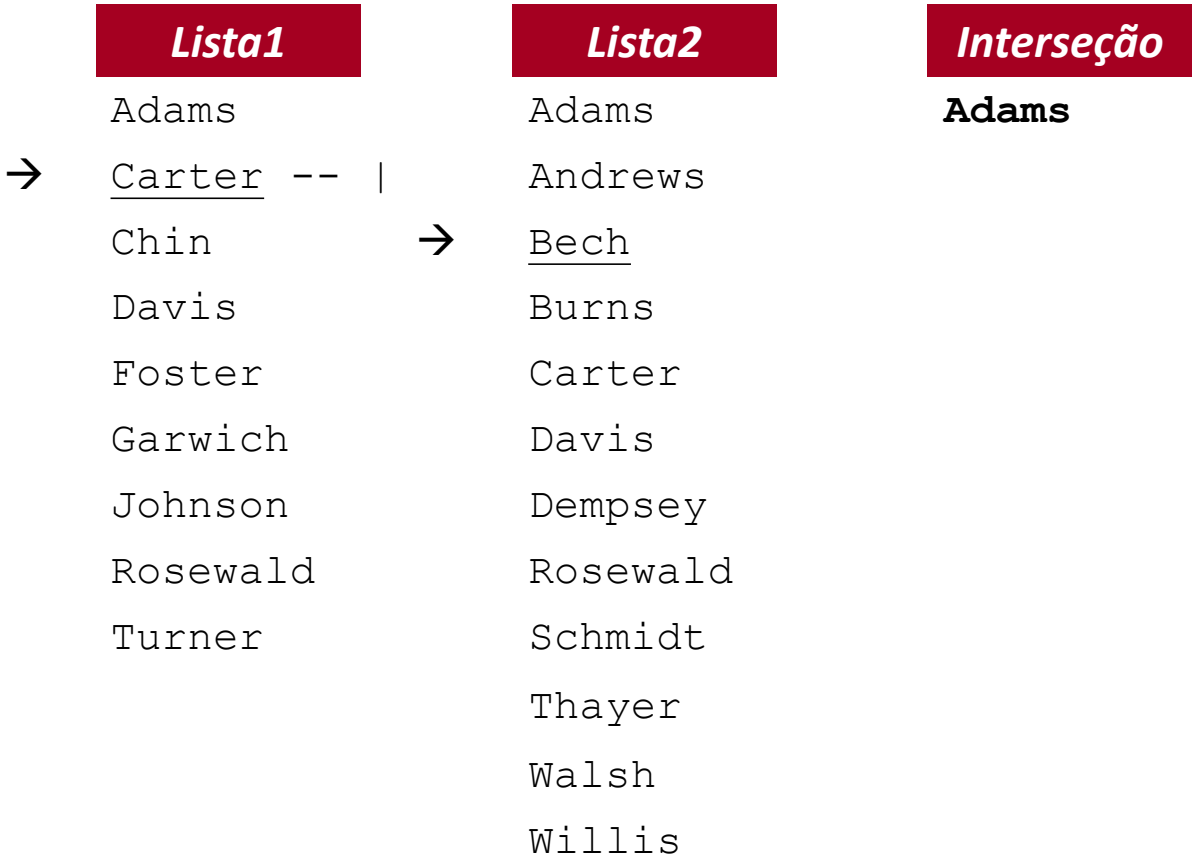
Processamento cossequencial

- Interseção (*Matching*)

	Lista1		*Lista2*	*Interseção*
	Adams		Adams	**Adams**
→	Carter	→	Andrews	
	Chin		Bech	
	Davis		Burns	
	Foster		Carter	
	Garwich		Davis	
	Johnson		Dempsey	
	Rosewald		Rosewald	
	Turner		Schmidt	
			Thayer	
			Walsh	
			Willis	

Processamento cossequencial

- Interseção (*Matching*)



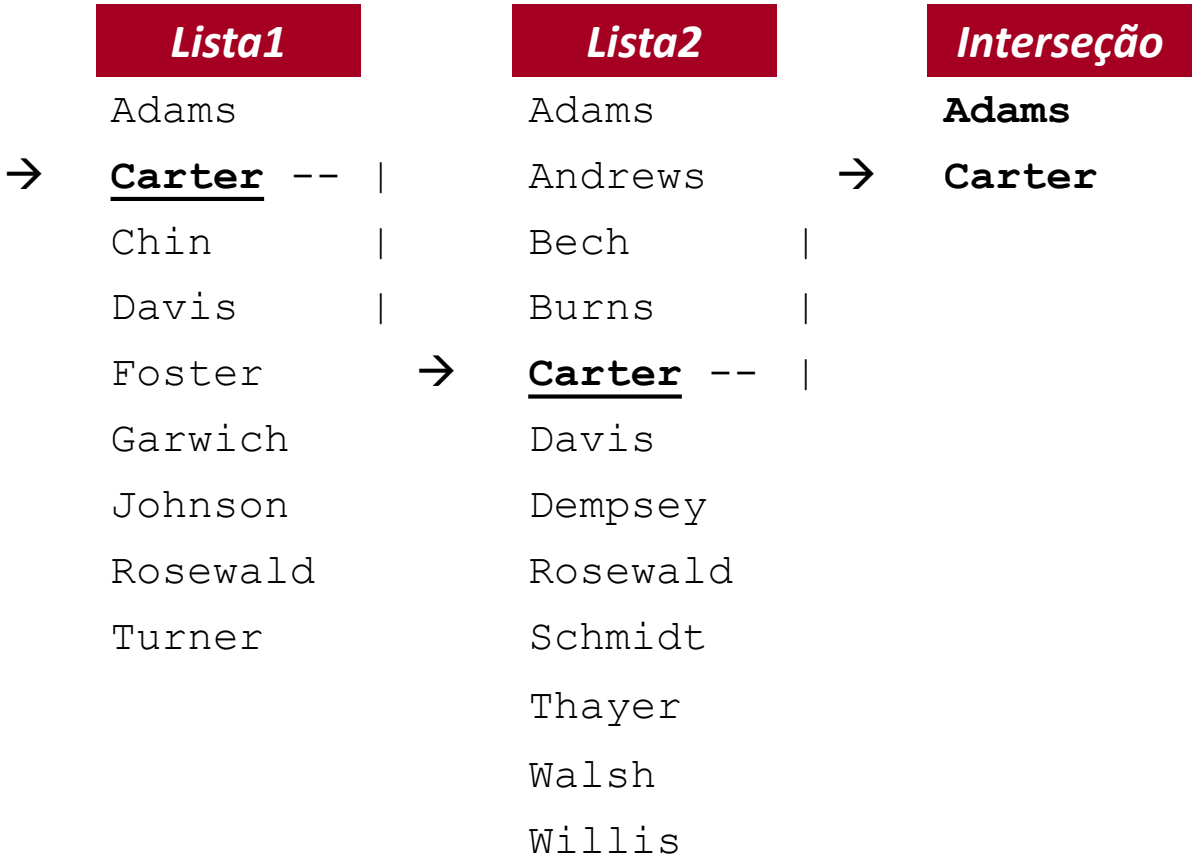
Processamento cossequencial

- Interseção (*Matching*)

	Lista1		Lista2	Interseção
	Adams		Adams	**Adams**
→	Carter -- |		Andrews	
	Chin |		Bech	
	Davis	→	Burns	
	Foster		Carter	
	Garwich		Davis	
	Johnson		Dempsey	
	Rosewald		Rosewald	
	Turner		Schmidt	
			Thayer	
			Walsh	
			Willis	

Processamento cossequencial

- Interseção (*Matching*)



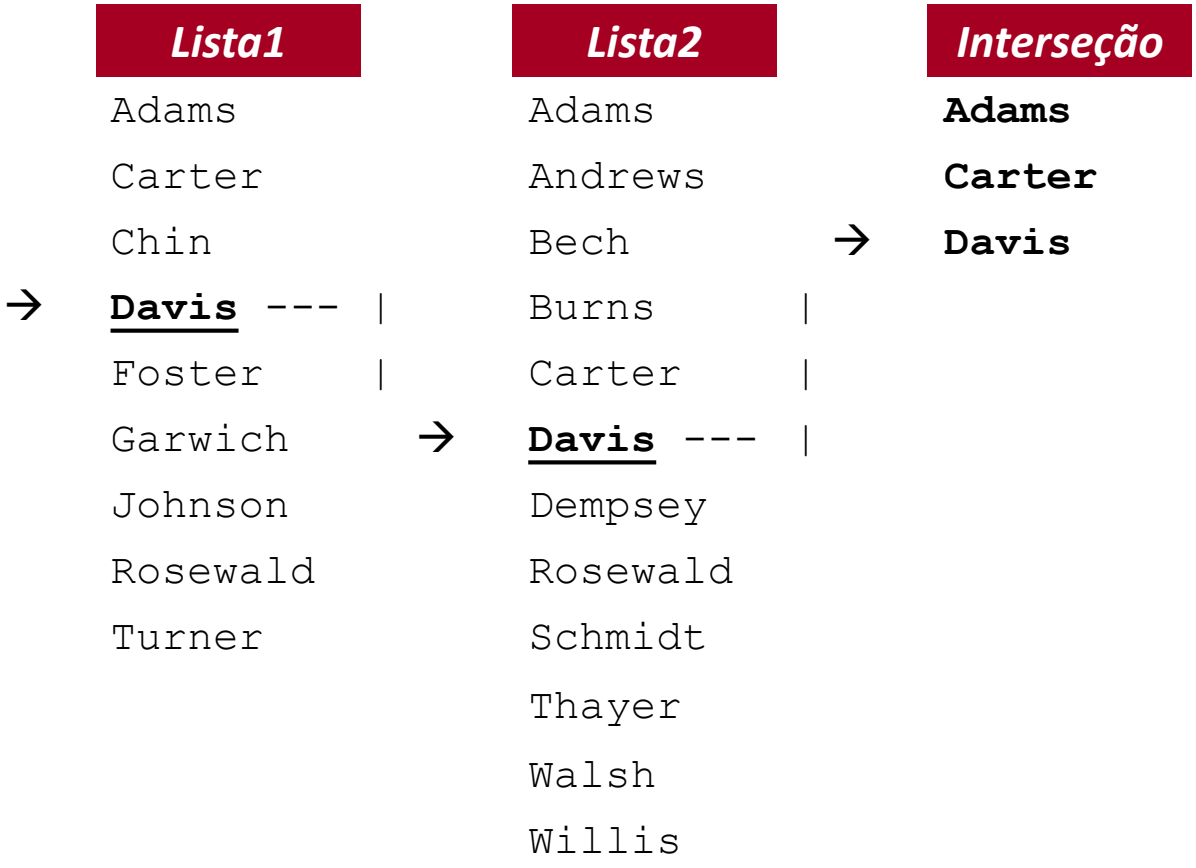
Processamento cossequencial

- Interseção (*Matching*)

Lista1	*Lista2*	*Interseção*
Adams	Adams	**Adams**
Carter	Andrews	**Carter**
→ Chin ---- |	Bech	
Davis |	Burns	
Foster |	Carter	
Garwich	→ Davis	
Johnson	Dempsey	
Rosewald	Rosewald	
Turner	Schmidt	
	Thayer	
	Walsh	
	Willis	

Processamento cossequencial

- Interseção (*Matching*)



Processamento cossequencial

- Interseção (*Matching*)

	Lista1		*Lista2*	*Interseção*
	Adams		Adams	**Adams**
	Carter		Andrews	**Carter**
	Chin		Bech	**Davis**
	Davis		Burns	
→	Foster -- |		Carter	
	Garwich |		Davis	
	Johnson	→	Dempsey	
	Rosewald		Rosewald	
	Turner		Schmidt	
			Thayer	
			Walsh	
			Willis	

Processamento cossequencial

- Interseção (*Matching*)

Lista1	*Lista2*	*Interseção*
Adams	Adams	**Adams**
Carter	Andrews	**Carter**
Chin	Bech	**Davis**
Davis	Burns	
→ Foster -- |	Carter	
Garwich |	Davis	
Johnson |	Dempsey	
Rosewald	→ Rosewald	
Turner	Schmidt	
	Thayer	
	Walsh	
	Willis	

Processamento cossequencial

- Interseção (*Matching*)

Lista1	*Lista2*	*Interseção*
Adams	Adams	**Adams**
Carter	Andrews	**Carter**
Chin	Bech	**Davis**
Davis	Burns	
Foster	Carter	
→ Garwich - |	Davis	
Johnson |	Dempsey	
Rosewald →	Rosewald	
Turner	Schmidt	
	Thayer	
	Walsh	
	Willis	

Processamento cossequencial

- Interseção (*Matching*)

Lista1	*Lista2*	*Interseção*
Adams	Adams	**Adams**
Carter	Andrews	**Carter**
Chin	Bech	**Davis**
Davis	Burns	
Foster	Carter	
Garwich	Davis	
→ Johnson - |	Dempsey	
Rosewald	→ Rosewald	
Turner	Schmidt	
	Thayer	
	Walsh	
	Willis	

Processamento cossequencial

- Interseção (*Matching*)

	Lista1		*Lista2*		*Interseção*
	Adams		Adams		**Adams**
	Carter		Andrews		**Carter**
	Chin		Bech		**Davis**
	Davis		Burns	→	**Rosewald**
	Foster		Carter	|	
	Garwich		Davis	|	
	Johnson		Dempsey	|	
→	**Rosewald**	→	**Rosewald**	|	
	Turner		Schmidt		
			Thayer		
			Walsh		
			Willis		

Processamento cossequencial

- Interseção (*Matching*)

	Lista1		*Lista2*	*Interseção*
	Adams		Adams	**Adams**
	Carter		Andrews	**Carter**
	Chin		Bech	**Davis**
	Davis		Burns	**Rosewald**
	Foster		Carter	
	Garwich		Davis	
	Johnson		Dempsey	
	Rosewald		Rosewald	
→	Turner	→	Schmidt	
			Thayer	
			Walsh	
			Willis	

Processamento cossequencial

- Interseção (*Matching*)

Lista1	*Lista2*	*Interseção*
Adams	Adams	**Adams**
Carter	Andrews	**Carter**
Chin	Bech	**Davis**
Davis	Burns	**Rosewald**
Foster	Carter	
Garwich	Davis	
Johnson	Dempsey	
Rosewald	Rosewald	
→ Turner -- |	Schmidt	
	→ Thayer	
	Walsh	
	Willis	

Processamento cossequencial

- Interseção (*Matching*)

Lista1	*Lista2*	*Interseção*
Adams	Adams	**Adams**
Carter	Andrews	**Carter**
Chin	Bech	**Davis**
Davis	Burns	**Rosewald**
Foster	Carter	
Garwich	Davis	
Johnson	Dempsey	
Rosewald	Rosewald	
→ Turner -- |	Schmidt	
|	Thayer	
→	Walsh	
	Willis	

Processamento cossequencial

- Interseção (*Matching*)

Lista1	*Lista2*	*Interseção*
Adams	Adams	**Adams**
Carter	Andrews	**Carter**
Chin	Bech	**Davis**
Davis	Burns	**Rosewald**
Foster	Carter	
Garwich	Davis	
Johnson	Dempsey	
Rosewald	Rosewald	
Turner	Schmidt	
EOF	Thayer	
	→ Walsh	
	Willis	

Interseção (*Matching*)

- Sincronização dos arquivos
  - Se *Nome1* < *Nome2*, lemos o próximo da Lista1
  - Se *Nome1* > *Nome2,* lemos o próximo da Lista2
  - Se *Nome1* = *Nome2*, escrevemos o nome na saída e lemos os próximos nomes das listas 1 e 2

Interseção (*Matching*)

```
FUNÇÃO inicialize()
    # VALOR_BAIXO é um valor constante que sempre será
    # menor que qualquer valor das listas de entrada
     ANTERIOR1 := VALOR_BAIXO
     ANTERIOR2 := VALOR_BAIXO
     abra o arquivo com a lista 1 para leitura como LISTA1
     abra o arquivo com a lista 2 para leitura como LISTA2
     abra o arquivo de saída para escrita como SAIDA
     EXISTEM_MAIS_NOMES := TRUE
     retorne ANTERIOR1, ANTERIOR2, LISTA1, LISTA2, SAIDA, EXISTEM_MAIS_NOMES
fim FUNÇÃO
```

Interseção (*Matching*)

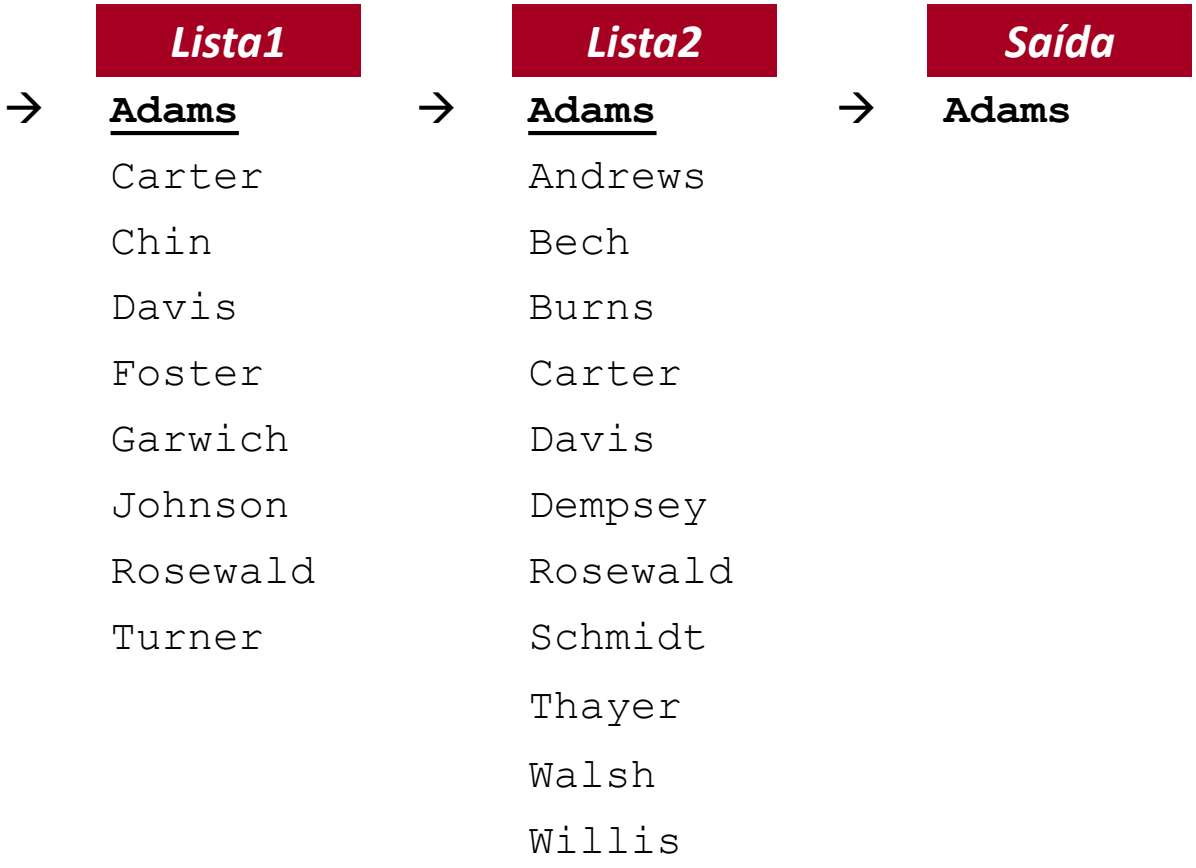
```
FUNÇÃO leia_nome(LISTA, NOME_ANT, EXISTEM_MAIS_NOMES)
    leia NOME do arquivo LISTA
    se EOF então
            EXISTEM_MAIS_NOMES := FALSE
    senão
            se NOME <= NOME_ANT então
                    gere um erro: 'Erro na checagem de sequência'
    fim se
    NOME_ANT := NOME
    retorne NOME, NOME_ANT, EXISTEM_MAIS_NOMES
fim FUNÇÃO
```

Interseção (*Matching*)

```
FUNÇÃO match()
    chame inicialize()
    chame leia_nome() para ler NOME1 da LISTA1
    chame leia_nome() para ler NOME2 da LISTA2
    enquanto EXISTEM_MAIS_NOMES faça
            se NOME1 < NOME2 então
                    chame leia_nome() para ler NOME1 da LISTA1
            senão se NOME1 > NOME2 então
                    chame leia_nome() para ler NOME2 da LISTA2
            senão   /* chaves iguais */
                    escreva NOME1 em SAIDA
                    chame leia_nome() para ler NOME1 da LISTA1
                    chame leia_nome() para ler NOME2 da LISTA2
    feche LISTA1, LISTA2 e SAIDA
fim FUNÇÃO
```

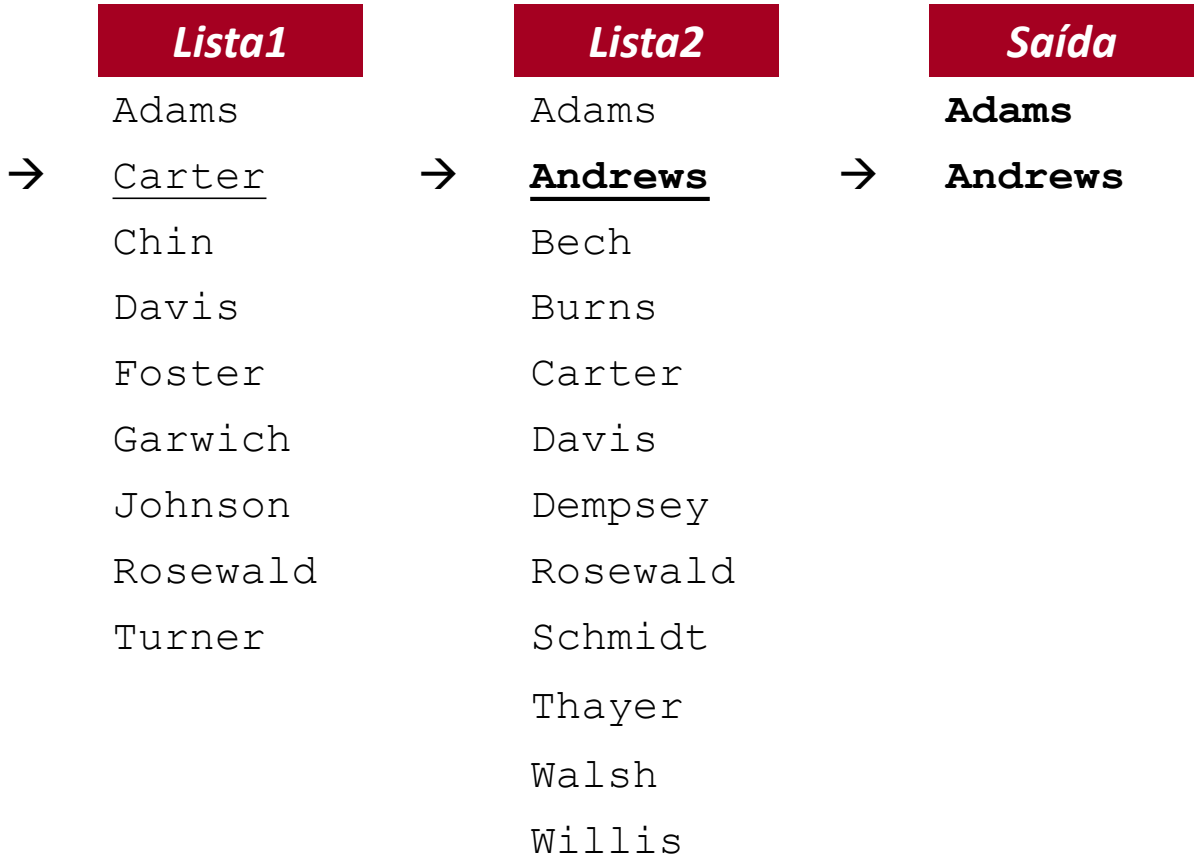
Exemplo

- Intercalação (*Merge*)



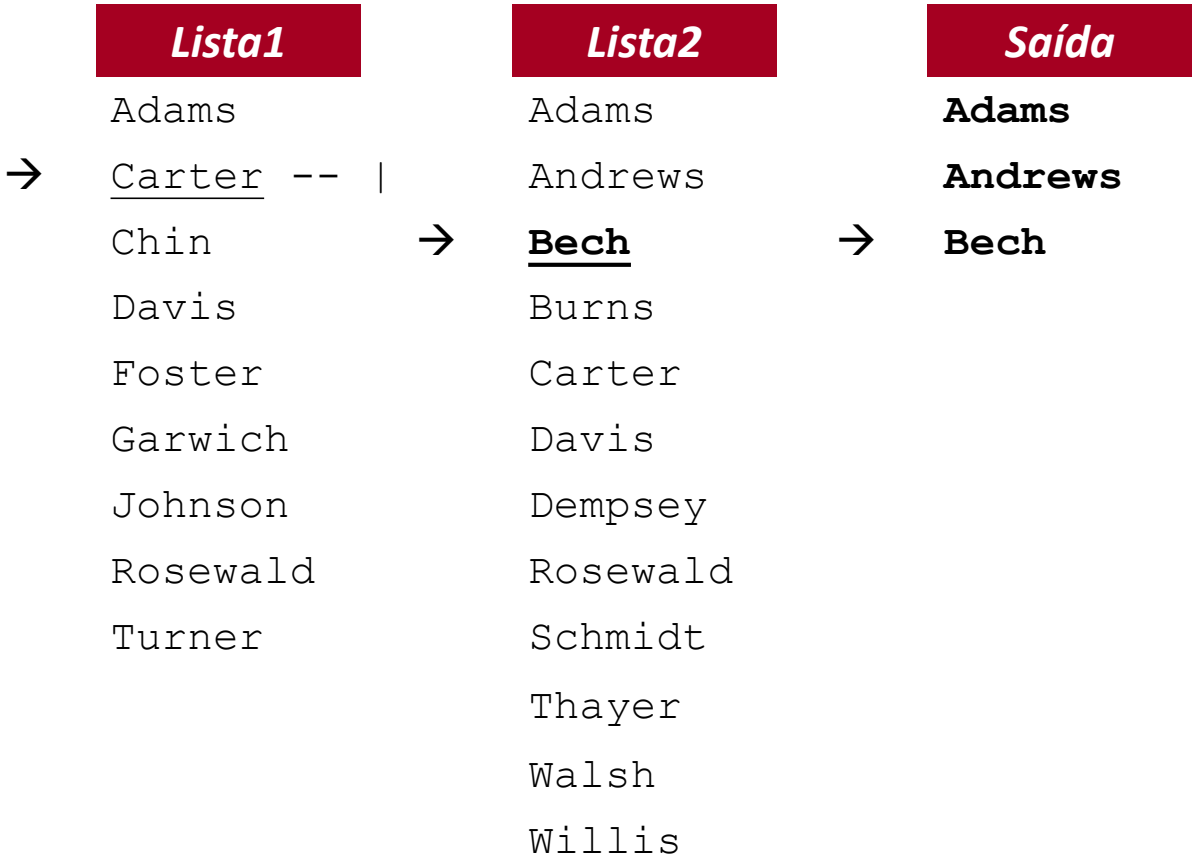
Exemplo

- Intercalação (*Merge*)



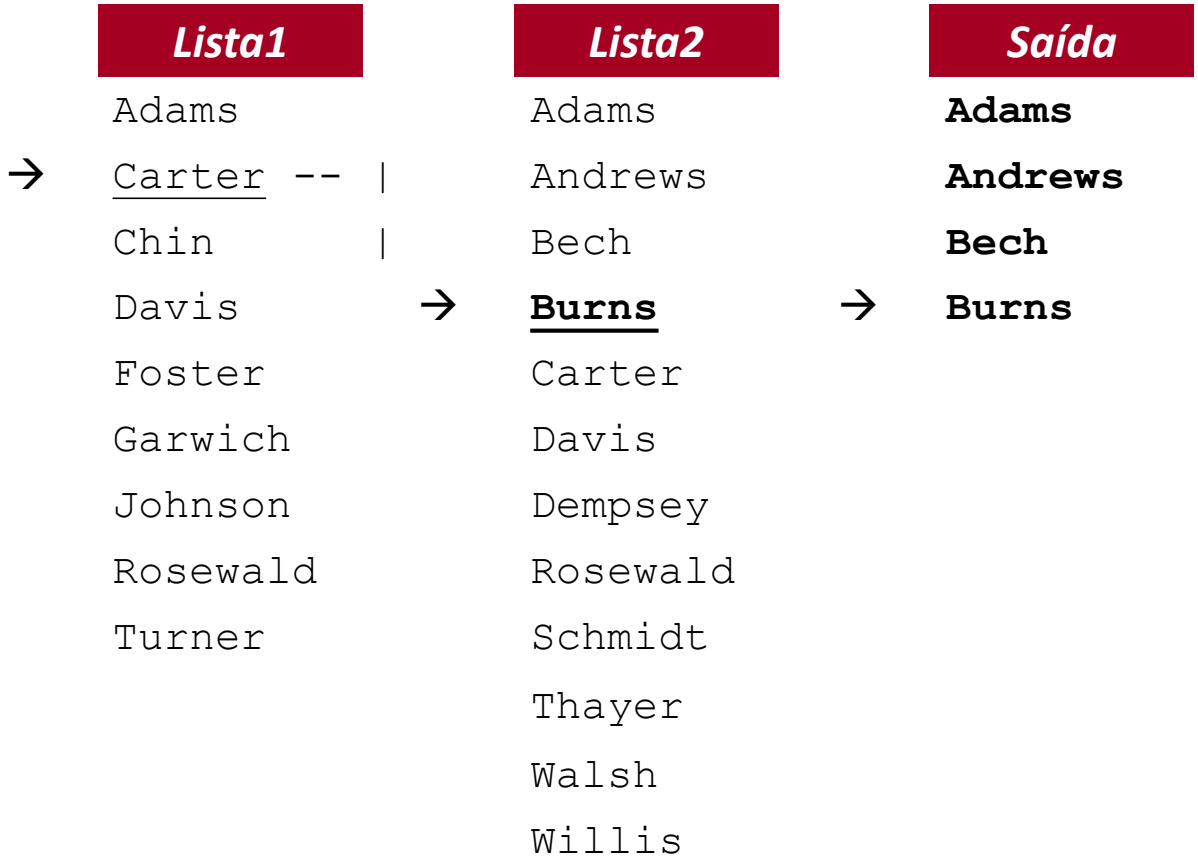
Exemplo

- Intercalação (*Merge*)



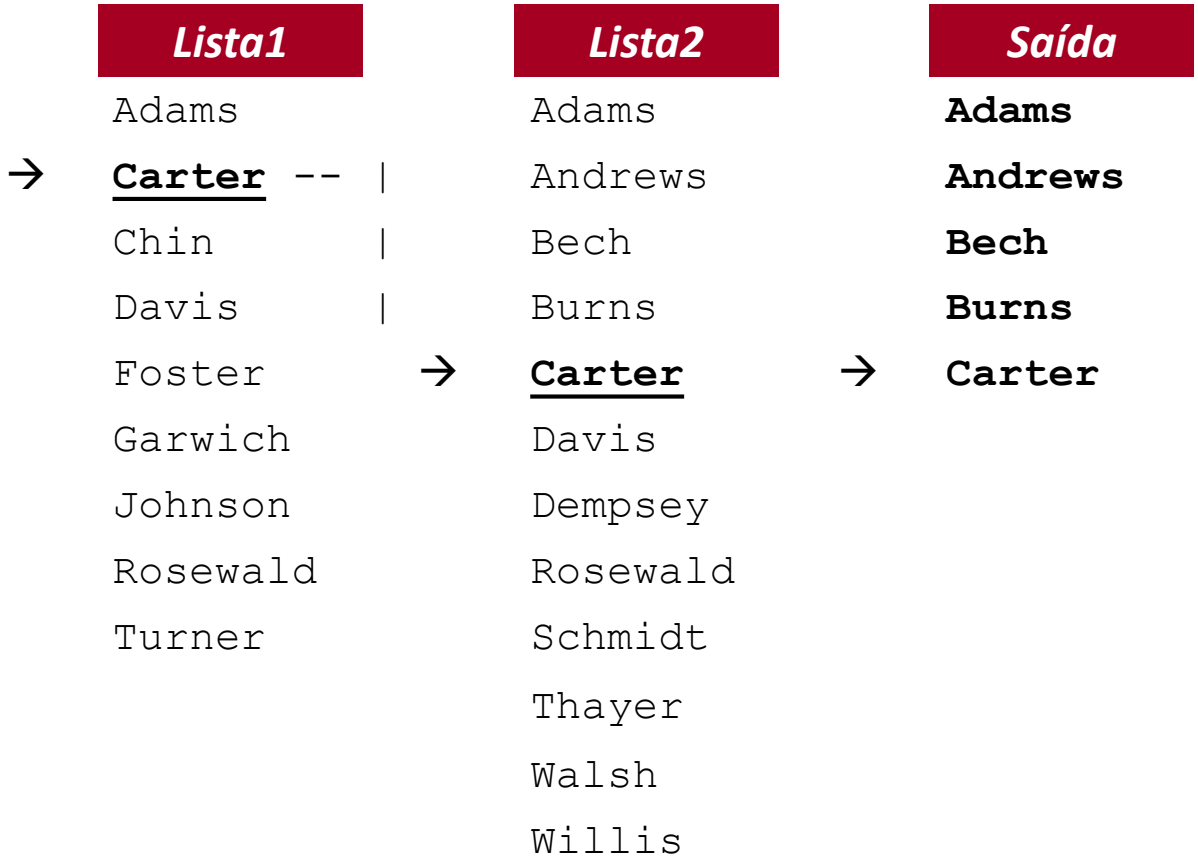
Exemplo

- Intercalação (*Merge*)



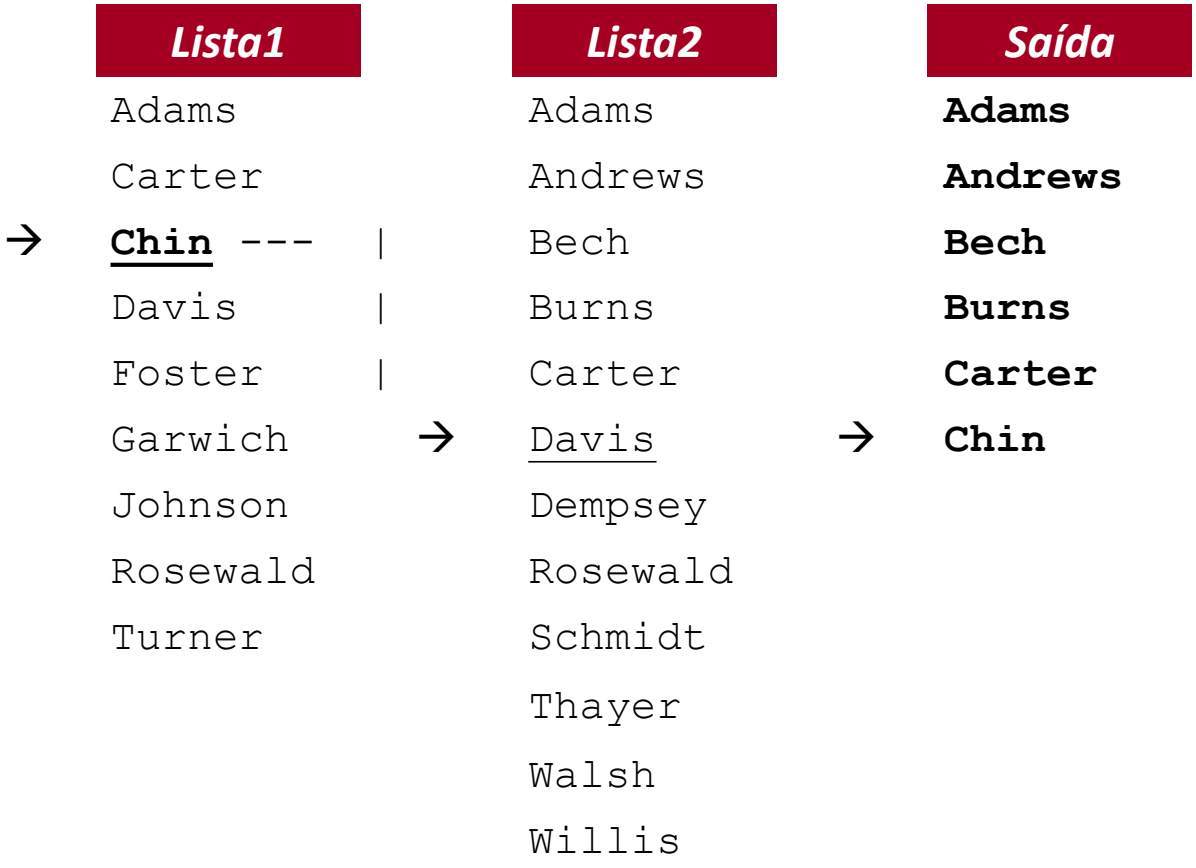
Exemplo

- Intercalação (*Merge*)



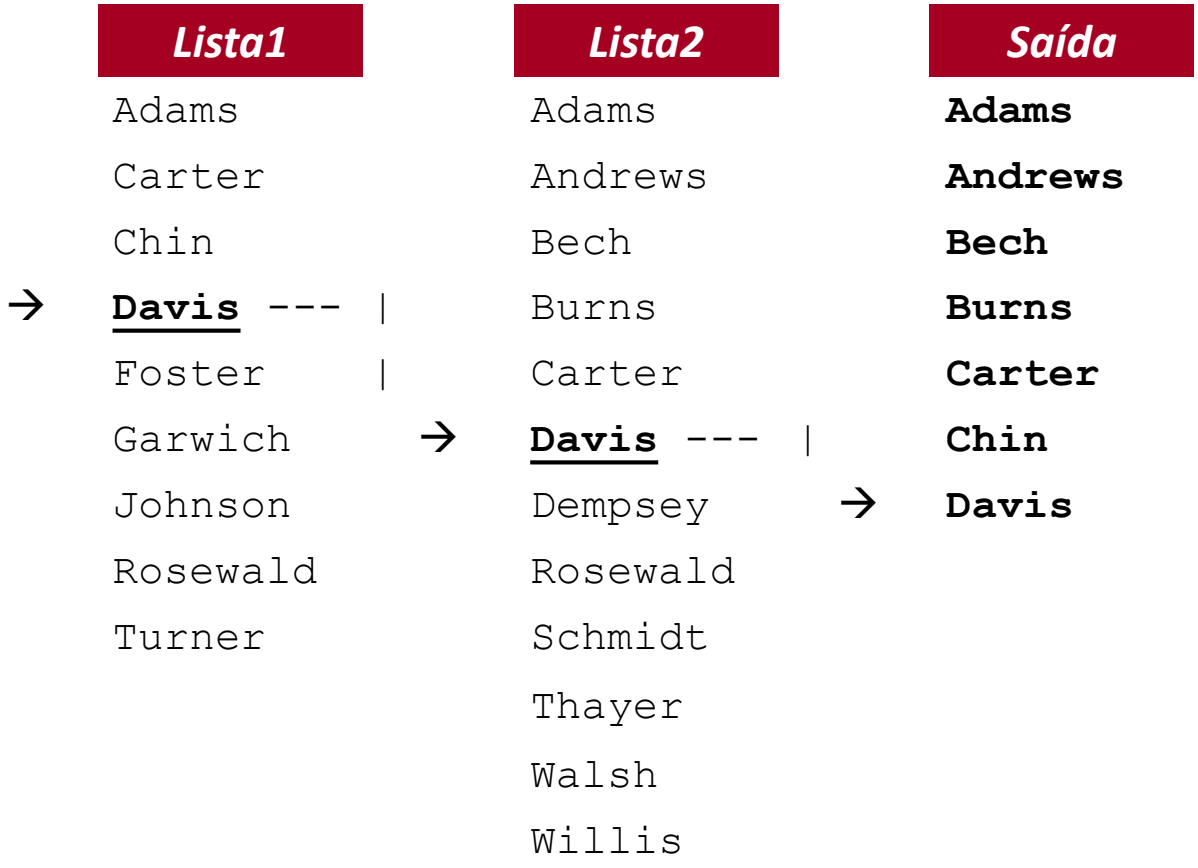
Exemplo

- Intercalação (*Merge*)



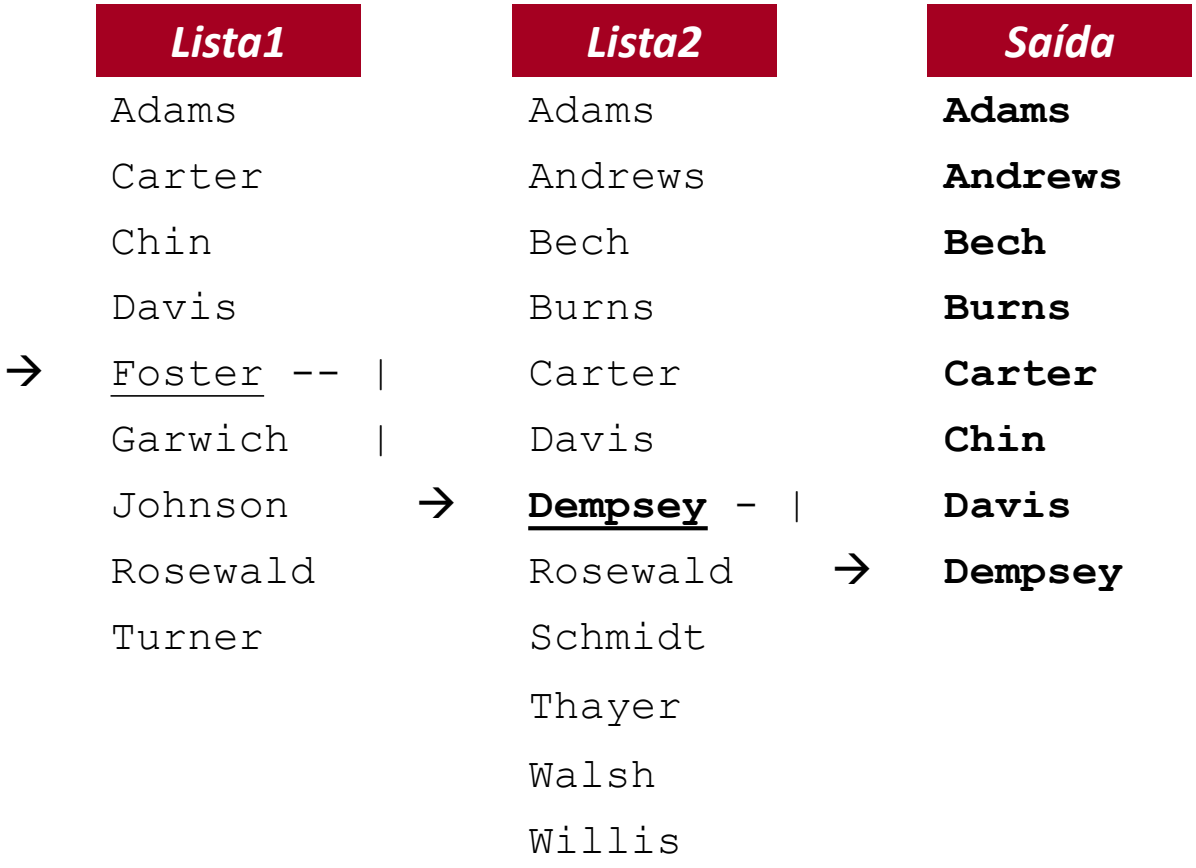
Exemplo

- Intercalação (*Merge*)



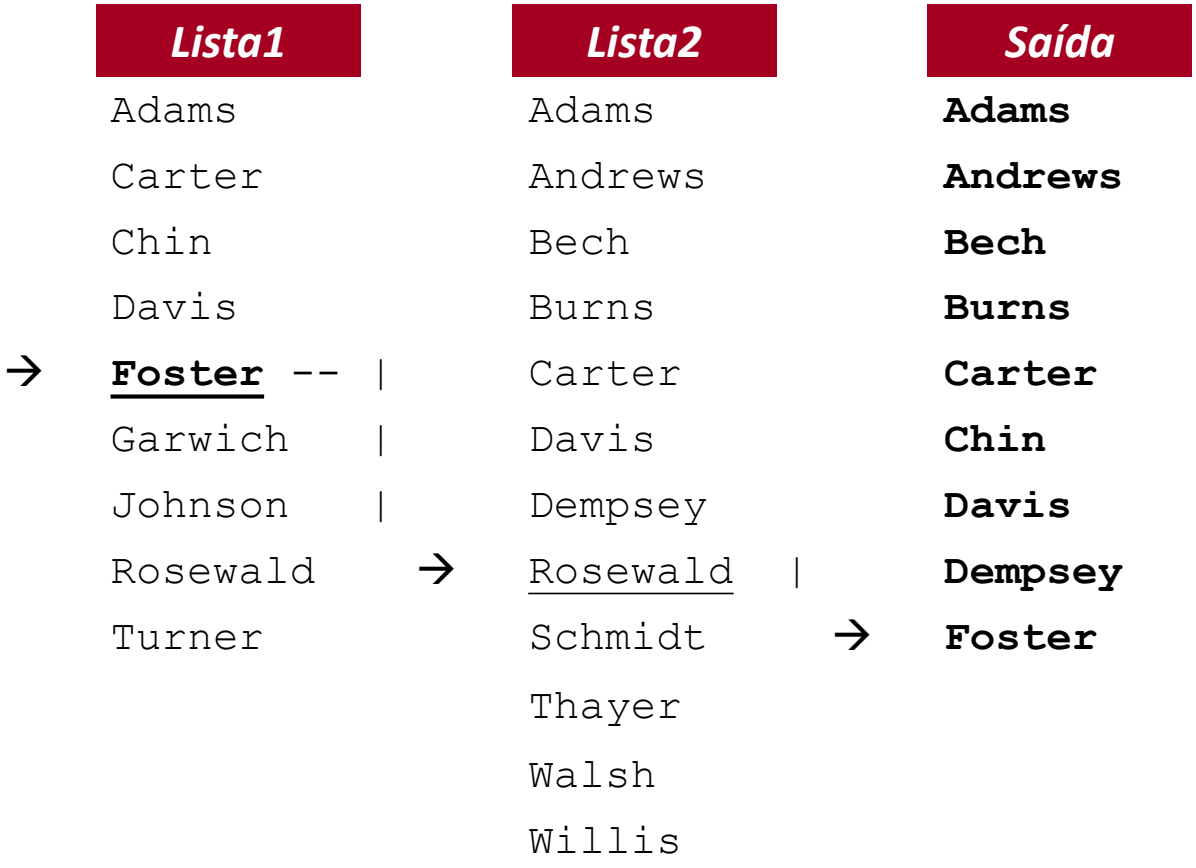
Exemplo

- Intercalação (*Merge*)



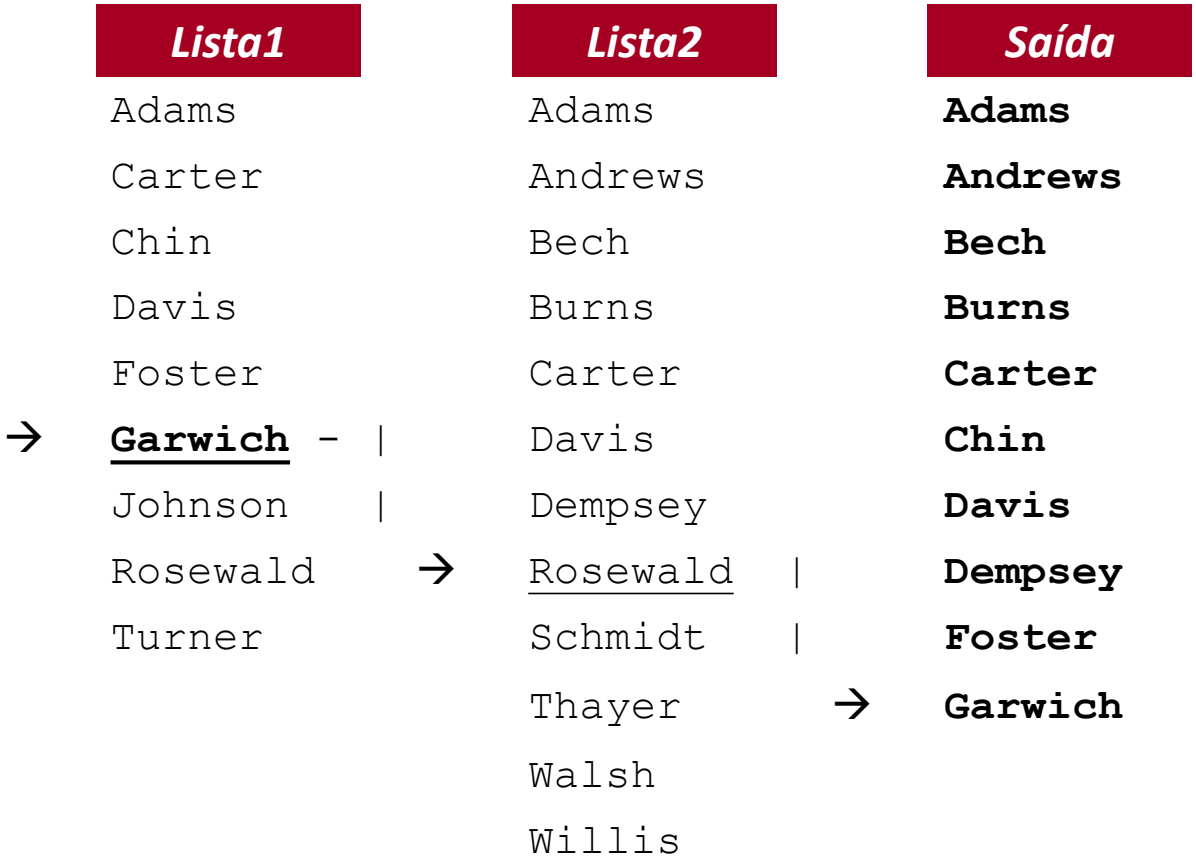
Exemplo

- Intercalação (*Merge*)



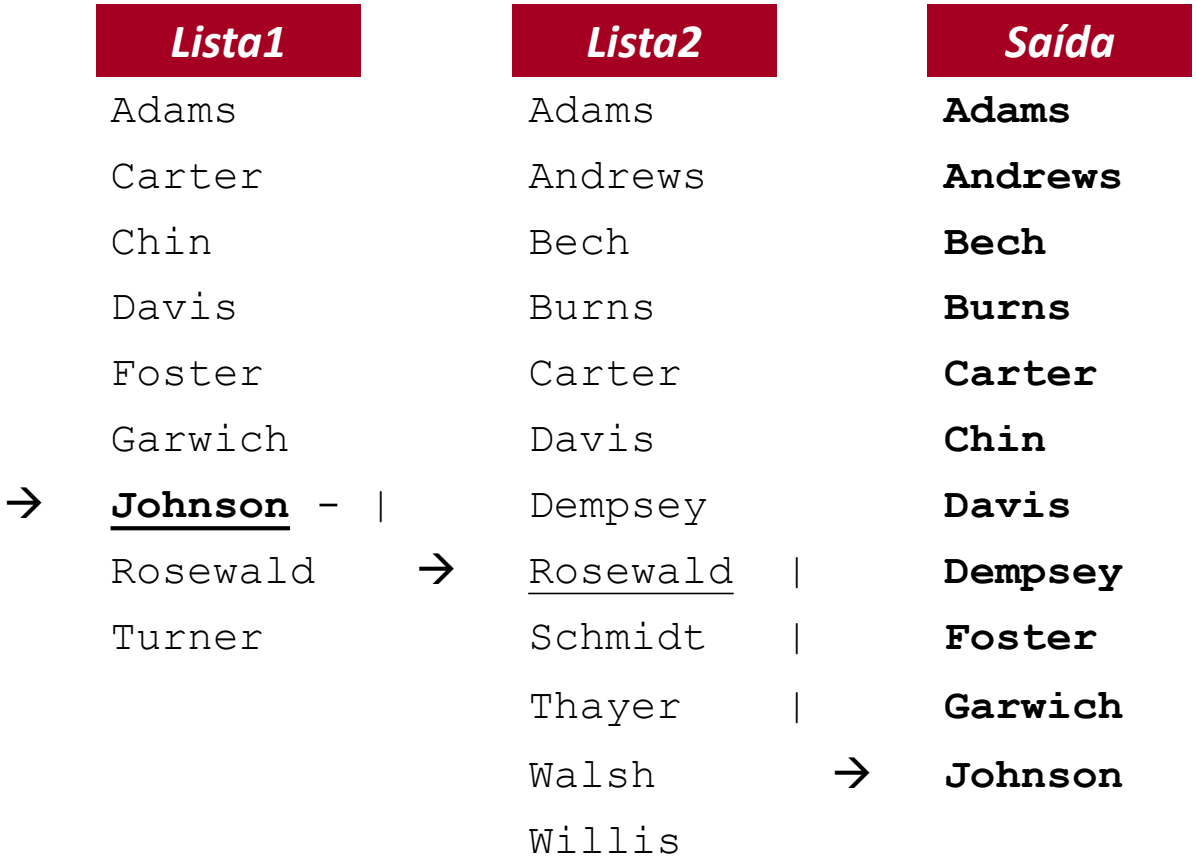
Exemplo

- Intercalação (*Merge*)



Exemplo

- Intercalação (*Merge*)



Exemplo

- Intercalação (*Merge*)

	Lista1		*Lista2*		*Saída*
	Adams		Adams		**Adams**
	Carter		Andrews		**Andrews**
	Chin		Bech		**Bech**
	Davis		Burns		**Burns**
	Foster		Carter		**Carter**
	Garwich		Davis		**Chin**
	Johnson		Dempsey		**Davis**
→	**Rosewald**	→	**Rosewald**	|	**Dempsey**
	Turner		Schmidt	|	**Foster**
			Thayer	|	**Garwich**
			Walsh	|	**Johnson**
			Willis	→	**Rosewald**

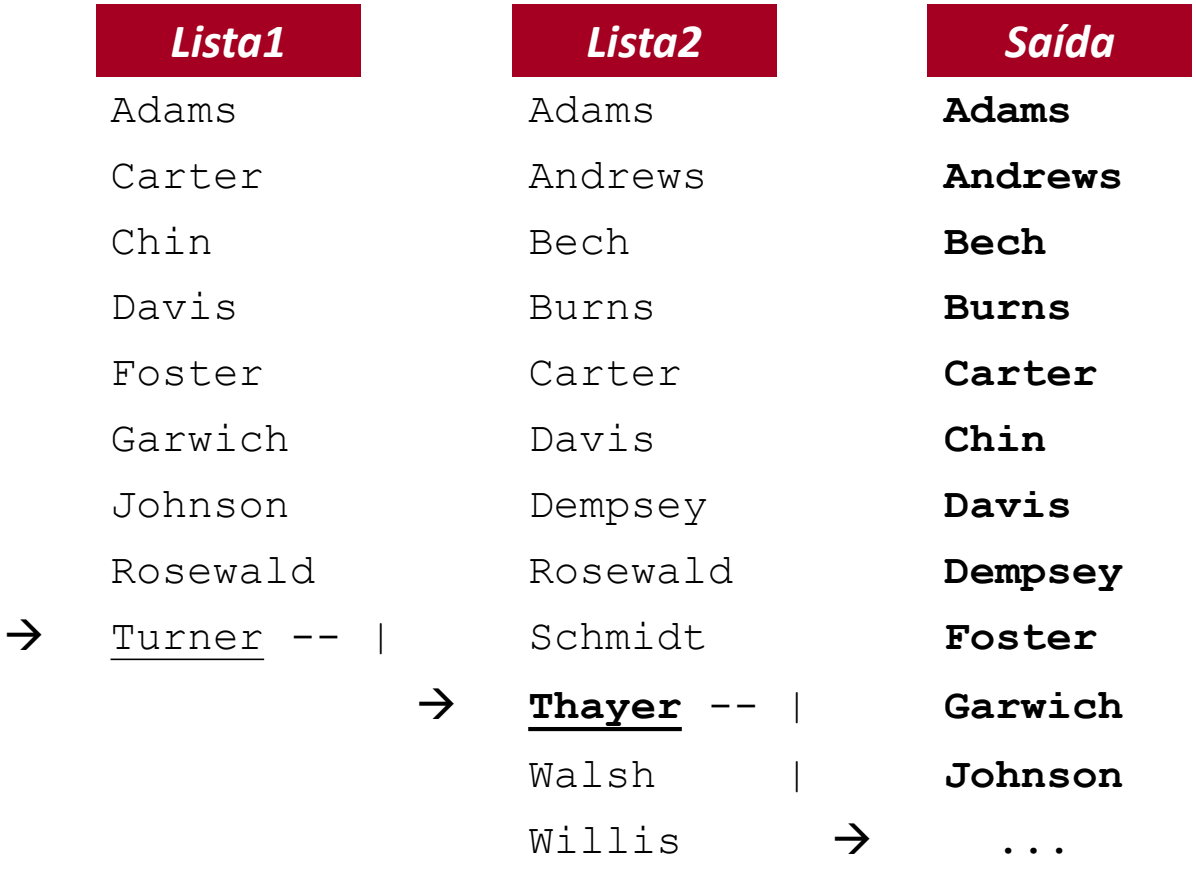
Exemplo

- Intercalação (*Merge*)

Lista1	*Lista2*	*Saída*
Adams	Adams	**Adams**
Carter	Andrews	**Andrews**
Chin	Bech	**Bech**
Davis	Burns	**Burns**
Foster	Carter	**Carter**
Garwich	Davis	**Chin**
Johnson	Dempsey	**Davis**
Rosewald	Rosewald	**Dempsey**
→ Turner	→ **Schmidt** - |	**Foster**
	Thayer |	**Garwich**
	Walsh |	**Johnson**
	Willis →	...

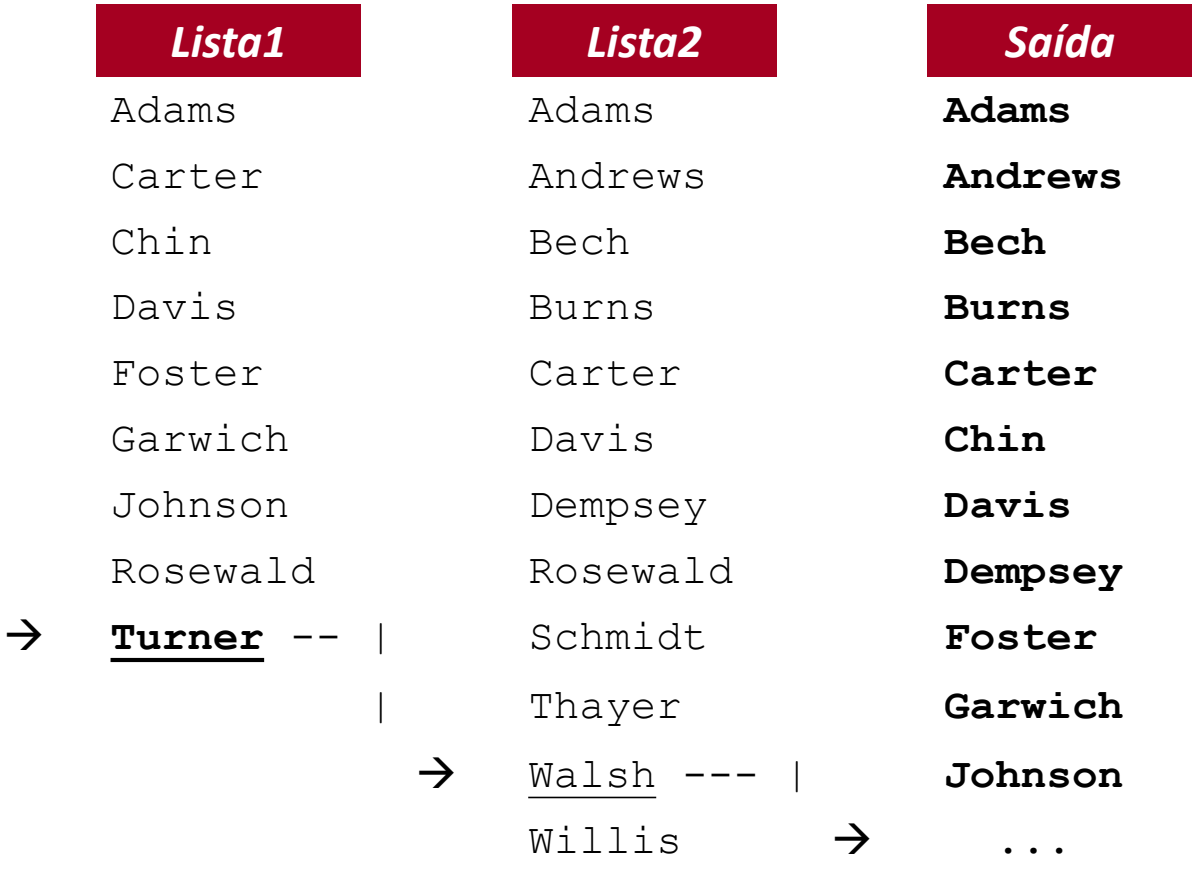
Exemplo

- Intercalação (*Merge*)



Exemplo

- Intercalação (*Merge*)



Exemplo

- Intercalação (*Merge*)

Lista1	*Lista2*	*Saída*
Adams	Adams	**Adams**
Carter	Andrews	**Andrews**
Chin	Bech	**Bech**
Davis	Burns	**Burns**
Foster	Carter	**Carter**
Garwich	Davis	**Chin**
Johnson	Dempsey	**Davis**
Rosewald	Rosewald	**Dempsey**
Turner	Schmidt	**Foster**
EOF	Thayer	**Garwich**
→	**Walsh** --- |	**Johnson**
	Willis →	...

Exemplo

- Intercalação (*Merge*)

Lista1	*Lista2*	*Saída*
Adams	Adams	**Adams**
Carter	Andrews	**Andrews**
Chin	Bech	**Bech**
Davis	Burns	**Burns**
Foster	Carter	**Carter**
Garwich	Davis	**Chin**
Johnson	Dempsey	**Davis**
Rosewald	Rosewald	**Dempsey**
Turner	Schmidt	**Foster**
EOF	Thayer	**Garwich**
	Walsh	**Johnson**
	→ **Willis** -- |	...
		→ **Willis**

Exemplo

- Intercalação (*Merge*)

Lista1	*Lista2*	*Saída*
Adams	Adams	**Adams**
Carter	Andrews	**Andrews**
Chin	Bech	**Bech**
Davis	Burns	**Burns**
Foster	Carter	**Carter**
Garwich	Davis	**Chin**
Johnson	Dempsey	**Davis**
Rosewald	Rosewald	**Dempsey**
Turner	Schmidt	**Foster**
EOF	Thayer	**Garwich**
	Walsh	**Johnson**
	Willis	...
	EOF	**Willis**

Intercalação (*Merge*)

- Sincronização dos arquivos
  - Se *Nome1* < *Nome2*, escrevemos o nome1 na saída lemos o próximo da Lista1
  - Se *Nome1* > *Nome2,* escrevemos o nome2 na saída lemos o próximo da Lista2
  - Se *Nome1* = *Nome2*, escrevemos o nome1 na saída e lemos os próximos nomes das listas 1 e 2

Intercalação

```
FUNÇÃO inicialize()
    # VALOR_BAIXO é um valor constante que sempre será
    # menor que qualquer valor das listas de entrada
     ANTERIOR1 := VALOR_BAIXO
     ANTERIOR2 := VALOR_BAIXO
     abra o arquivo com a lista 1 para leitura como LISTA1
     abra o arquivo com a lista 2 para leitura como LISTA2
     abra o arquivo de saída para escrita como SAIDA
     EXISTEM_MAIS_NOMES := TRUE
     retorne ANTERIOR1, ANTERIOR2, LISTA1, LISTA2, SAIDA, EXISTEM_MAIS_NOMES
fim FUNÇÃO
```

Intercalação

```
FUNÇÃO leia_nome(LISTA, NOME_ANT, NOME_OUTRA_LISTA, EXISTEM_MAIS_NOMES)
    leia NOME do arquivo LISTA
    se EOF então
        # VALOR_ALTO é um valor constante que sempre será maior que qualquer outro
        se NOME_OUTRA_LISTA = VALOR_ALTO então
            EXISTEM_MAIS_NOMES := FALSE          # fim das duas listas
        senão NOME := VALOR_ALTO                 # fim da lista atual
    senão # existem mais nomes
        se NOME <= NOME_ANT então
            gere um erro: 'Erro na checagem de sequência'
    NOME_ANT := NOME
    retorne NOME, NOME_ANT, EXISTEM_MAIS_NOMES
fim FUNÇÃO
```

Intercalação

```
FUNÇÃO merge()
    chame inicialize()
    chame leia_nome() para ler NOME1 da LISTA1
    chame leia_nome() para ler NOME2 da LISTA2
    enquanto EXISTEM_MAIS_NOMES faça
        se NOME1 < NOME2 então # chave da LISTA1 é menor
            escreva NOME1 em SAIDA
            chame leia_nome() para ler NOME1 da LISTA1
        senão se NOME1 > NOME2 então # chave da LISTA2 é menor
            escreva NOME2 em SAIDA
            chame leia_nome() para ler NOME2 da LISTA2
        senão          # as chaves são iguais
            escreva NOME1 em SAIDA
            chame leia_nome() para ler NOME1 da LISTA1
            chame leia_nome() para ler NOME2 da LISTA2
    feche LISTA1, LISTA2 e SAIDA
fim FUNÇÃO
```

Intercalação múltipla *(K-way Merge)*

- **Algoritmo *K-way merge***: intercala *K* arquivos ordenados, gerando um único arquivo ordenado de saída
  - O valor *K* é a ordem do merge
  - A parte mais cara do processo são os testes para verificar em quais arquivos a menor chave ocorreu, para se saber quais arquivos devem ser lidos a seguir

```
enquanto EXISTEM_MAIS_NOMES faça
  NOME_SAIDA := menor(NOME1, NOME2, NOME3, ..., NOMEK)
  escreva NOME_SAIDA em SAIDA
  se NOME1 = NOME_SAIDA então
    chame leia_nome() para ler NOME1 da LISTA1
  se NOME2 = NOME_SAIDA então
    chame leia_nome() para ler NOME2 da LISTA2
  ...
  se NOMEK = NOME_SAIDA então
    chame leia_nome() para ler NOMEK da LISTAK
fim enquanto
```

K-Way Merge

- Condição de parada
  - Pode-se usar uma variável contadora
    - Inicia com 0 e é incrementada cada vez que um dos arquivos chega ao fim
  - A condição de parada se torna verdadeira se ocorrer EOF e o contador atingir o número de arquivos de entrada

```
FUNÇÃO leia_nome()
     ...
     se EOF então
       NOME :=  VALOR_ALTO
       numEOF := numEOF  + 1
       se numEOF = numListas então
           EXISTEM_MAIS_NOMES := FALSE
     ...
fim FUNÇÃO
```

K-Way Merge

- Se for possível garantir que uma chave ocorre somente em uma lista/arquivo, o procedimento ficaria mais simples e eficiente
- Suponha o uso das listas LISTA e NOME
  - LISTA[i] contém os descritores dos arquivos de entrada
  - NOME[i] contém o nome (chave) associado a cada arquivo de entrada

K-Way Merge

- Algoritmo modificado assumindo que não existem chaves duplicadas

```
...
#inicie o processo lendo um nome de cada lista
para i := 1 até K faça
  chame leia_nome() para ler NOME[i] da LISTA[i]
fim para

#inicie a intercalação em K-vias
enquanto EXISTEM_MAIS_NOMES faça
  #encontre o nome de menor valor entre os nomes das K listas
  MENOR := 1
  para i := 2 até K faça
    se NOME[i] < NOME[MENOR] então
      MENOR := i
  fim para
  escreva NOME[MENOR] em SAIDA
  #leia o próximo nome da lista cujo nome foi para a saída
  chame o leia_nome() para ler NOME[MENOR] da LISTA[MENOR]
fim enquanto
```

Quanto maior for o valor de K, mais caro será encontrar o menor