Atividade Prática:
Merge e K-way Merge

Organização e Recuperação de Dados

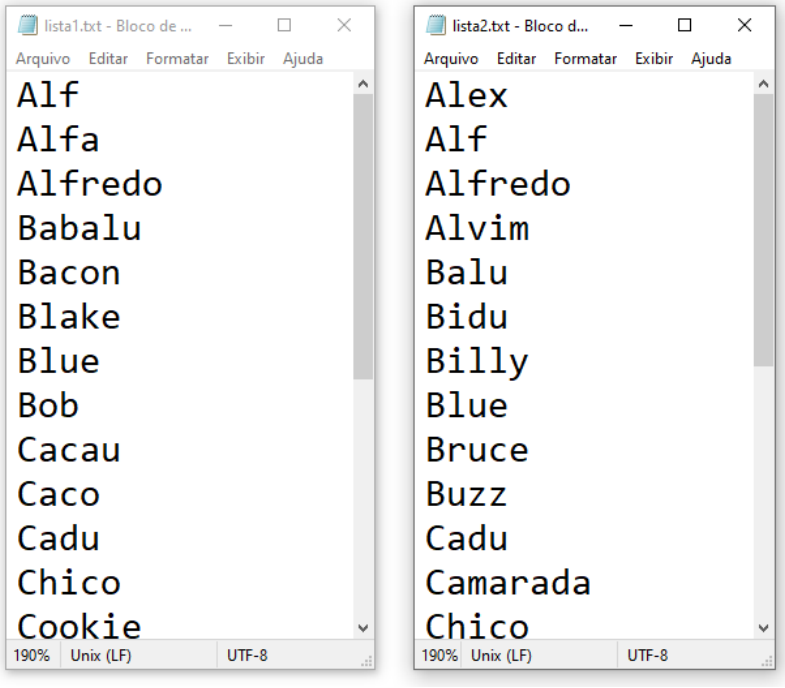
Profa. Valéria

UEM – CTC – DIN

Slides preparados com base no Cap. 4 do livro FOLK, M.J. & ZOELLICK, B. *File Structures*. 2nd Edition, Addison-Wesley Publishing Company, 1992

Exercício 1

- O **Exercício 1** um consiste na implementação do ***merge*** (intercalação) de duas listas ordenadas de nomes
  - Vamos supor que as listas estão armazenadas nos arquivos de texto *lista1.txt* e *lista2.txt* (disponíveis no *Classroom*)
  - Nos dois arquivos, cada nome ocupa exatamente uma linha



Merge de duas listas ordenadas

Lembre-se que as funções **inicialize** e **leia_nome** retornarão tuplas de valores, logo você deve usar uma tupla de valores para receber o retorno dessas funções

Estrutura geral do programa

```
import io
# CONSTANTES
VALOR_BAIXO = ''
VALOR_ALTO = '~'

def inicialize() -> tuple[str, str, io.TextIOWrapper, io.TextIOWrapper,
                                                    io.TextIOWrapper, bool]:
    ...
    return ant1, ant2, lista1, lista2, saida, existem_mais_nomes


def leia_nome(lista: io.TextIOWrapper, nome_ant: str, nome_outra_lista: str,
                          existem_mais_nomes: bool) -> tuple[str, str, bool]:
    ...
    return nome, nome_ant, existem_mais_nomes


def merge() -> None:
    ...


if __name__ == '__main__':
    merge()
```

Função de inicialização

Pseudocódigo inicialize

```
def inicialize() -> tuple[str, str, io.TextIOWrapper, io.TextIOWrapper,
                                                    io.TextIOWrapper, bool]:
    inicialize ant1 e ant2 com VALOR_BAIXO
    abra os arquivos "lista1.txt" e "lista2.txt" para leitura em modo texto e
            armazene os respectivos descritores em lista1 e lista2
    abra o arquivo "saída.txt" para escrita em modo texto e armazene em saida
    inicialize existem_mais_nomes com TRUE
    retorne ant1, ant2, lista1, lista2, saída e existem_mais_nomes
```

Função de leitura para o merge de duas listas

Para gerar um erro, use o comando ***raise***

Pseudocódigo leia_nome

```
def leia_nome(lista: io.TextIOWrapper, nome_ant: str, nome_outra_lista: str,
                            existem_mais_nomes: bool) -> tuple[str, str, bool]:
    leia um nome do arquivo lista
    se fim da lista então
        se nome_outra_lista igual a VALOR_ALTO então
            faça existem_mais_nomes receber FALSE
        senão
            faça nome receber VALOR_ALTO
    senão
        se nome for menor ou igual a nome_ant então
            gere um erro "Erro de sequência"
    faça nome_ant receber nome
    retorne nome, nome_ant e existem_mais_nomes
```

Função merge para duas listas

Pseudocódigo merge

```
def merge() -> None:
    chame inicialize() para inicializar anterior1, anterior2,
                                lista1, lista2, saida, existem_mais_nomes
    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 todos os arquivos
```

Exercício 2

- O **Exercício 2** consiste na implementação do ***k-way merge*** (*merge* de k listas), supondo que:
  - As listas a serem intercaladas estão ordenadas
  - **Não há repetição de nomes nas listas**
- Descompacte o arquivo *listas.zip* (disponível no *Classroom*) e use os arquivos (lista0.txt, lista1.txt, ..., lista99.txt) para testar a sua implementação
- Faça com que o **valor de k** (i.e., o número de listas a serem intercaladas) seja passado como um **argumento pela linha da comando**
  - Exemplo:

```
$ python3 kwaymerge.py 100
```

Estrutura geral do programa

```
from sys import argv
import io
# CONSTANTES
VALOR_BAIXO = ''
VALOR_ALTO = '~'
# VAR GLOBAL
numEOF = 0

def inicialize(numListas: int) -> tuple[list[str], list[str], list[io.TextIOWrapper],
                                        io.TextIOWrapper, bool]:
    ...
    return anteriores, nomes, listas, saida, existem_mais_nomes

def finalize(listas: list[io.TextIOWrapper], saida: io.TextIOWrapper, numListas: int) -> None:
    ...

def leia_nome(lista: io.TextIOWrapper, nome_ant: str, existem_mais_nomes: bool, numListas: int) ->
                                                    tuple[str, str, bool]:
    ...
    return nome, nome_ant, existem_mais_nomes

def kwaymerge(numListas: int) -> None:
    ...

def main() -> None:
    ...

if __name__ == '__main__':
    main()
```

Função de inicialização para o kwaymerge

Pseudocódigo inicialize

```
def inicialize(numListas: int) -> tuple[list[str], list[io.TextIOWrapper],
                                        io.TextIOWrapper, bool]:
    inicialize anteriores com uma lista de numListas elementos VALOR_BAIXO
    inicialize nomes com uma lista de numListas elementos VALOR_BAIXO
    inicialize listas com uma lista de numListas elementos NONE
    para i até numlistas faça
        faça nomearq receber o nome da lista i
        abra o arquivo nomearq para leitura ("r") e armazene em descritor
        insira o descritor em listas
    abra o arquivo "saida.txt" para escrita ("w") e armazene em saida
    inicialize exitem_mais_nomes com TRUE
    retorne anteriores, nomes, listas, saída, existem_mais_nomes
```

Função de finalização

Pseudocódigo finalize

```
def finalize(listas: list[io.TextIOWrapper], saida: io.TextIOWrapper,
                                             numListas: int) -> None:
    para i até numlistas faça
        feche o arquivo listas[i]
    feche o arquivo saida
```

Função de leitura para o kwaymerge

Pseudocódigo leia_nome

```
def leia_nome(lista: io.TextIOWrapper, nome_ant: str, existem_mais_nomes: bool,
                                                    numListas: int) -> tuple[str, str, bool]:
    global numEOF
    leia nome do arquivo lista
    se fim da lista então
        faça nome receber VALOR_ALTO
        incremente numEOF
        se numEOF foi igual a numListas então
            faça existem_mais_nomes receber FALSE
    senão
        se nome for menor ou igual a nome_ant então
            gere um erro "Erro de sequência"
    faça nome_ant receber nome
    retorne nome, nome_ant e existem_mais_nomes
```

Função kwaymerge

Pseudocódigo kwaymerge

```
def kwaymerge(numListas: int) -> None:
    # inicialize as variáveis
    chame inicialize() para inicializar anteriores, nomes, listas, saída,
                                                        existem_mais_nomes
    inicialize nomes com uma lista de numListas elementos VALOR_BAIXO
    # leia um nome de cada lista
    para i até numlistas faça
        chame leia_nome para para ler nomes[i] da listas[i]
    enquanto existem_mais_nomes faça
        # encontre o índice do menor nome
        inicialize menor com zero
        para i até numListas faça
            se nomes[i] for menor que nomes[menor] então
            faça menor receber i
        # escreva o menor na saída
        escreva nomes[menor] no arquivo saída
        # leia o próximo nome da listas[menor]
        chame leia_nome para ler nomes[menor] da listas[menor]
    chame finalize() para fechar os arquivos
```

Função main

Programa principal

```
def main() -> None:
    if len(argv) < 2:
        raise TypeError('Numero incorreto de argumentos!')
    kwaymerge(int(argv[1]))
```

argv do módulo **sys** é
uma lista que contém as strings
passadas na linha de comando

```
$ python3 kwaymerge.py 100
```