MC-102 — Aula 23
Arquivos, Parâmetros do Programa

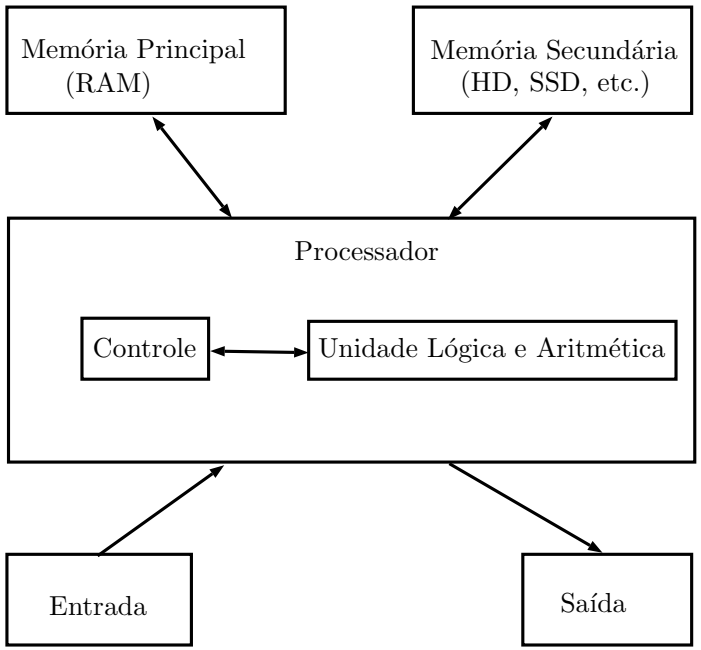
Instituto de Computação – Unicamp

3 de Novembro de 2016

Roteiro

Tipos de Memória

- Quando vimos a organização básica de um sistema computacional, havia somente um tipo de memória.
- Mas na maioria dos sistemas, a memória é dividida em dois tipos:



Tipos de Memória

- A memória principal (Random Access Memory) utilizada na maioria dos computadores, usa uma tecnologia que requer alimentação constante de energia para que informações sejam preservadas.

Tipos de Memória

- A memória secundária (como Hard Disks ou SSD) utilizada na maioria dos computadores, usa uma outra tecnologia que NÃO requer alimentação constante de energia para que informações sejam preservadas.

Tipos de Memória

- Todos os programas executam na RAM, e por isso quando o programa termina ou acaba energia, as informações do programa são perdidas.
- Para podermos gravar informações de forma *persistente*, devemos escrever estas informações em arquivos na memória secundária.
- A memória secundária possui algumas características:
  - É muito mais lenta que a RAM.
  - É muito mais barata que a memória RAM.
  - Possui maior capacidade de armazenamento.
- Sempre que nos referirmos a um arquivo, estamos falando de informações armazenadas em memória secundária.

Nomes e extensões

- Arquivos são identificados por um nome.
- O nome de um arquivo pode conter uma extensão que indica o conteúdo do arquivo.

Algumas extensões

arq.txt	arquivo texto simples
arq.py	código fonte em Python
arq.pdf	*portable document format*
arq.html	arquivo para páginas WWW *(hypertext markup language)*
arq*	arquivo executável (UNIX)
arq.exe	arquivo executável (Windows)

Tipos de arquivos

Arquivos podem ter o mais variado conteúdo, mas do ponto de vista dos programas existem apenas dois tipos de arquivo:

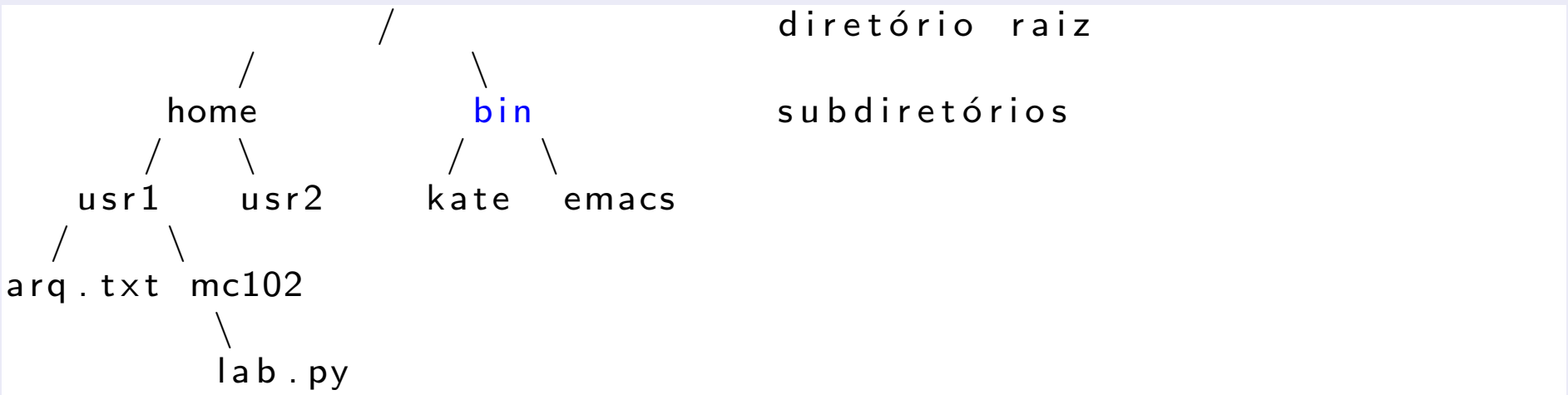
Arquivo texto: Armazena caracteres que podem ser mostrados diretamente na tela ou modificados por um editor de textos simples. Exemplos: código fonte C, documento texto simples, páginas HTML.

Arquivo binário: Seqüência de bits sujeita às convenções dos programas que o gerou, não legíveis diretamente. Exemplos: arquivos executáveis, arquivos compactados, documentos do Word.

Diretório

- Também chamado de pasta.
- Contém arquivos e/ou outros diretórios.

Uma hierarquia de diretórios



Caminhos absolutos ou relativos

O nome de um arquivo pode conter o seu diretório, ou seja, o caminho para encontrar este arquivo a partir da raiz. Os caminhos podem ser especificados de duas formas:

Caminho absoluto: descrição de um caminho desde o diretório raiz.

```
/bin/emacs
/home/usr1/arq.txt
```

Caminho relativo: descrição de um caminho a partir do diretório corrente.

```
arq.txt
mc102/lab.py
```

Arquivos texto

- Para se trabalhar com arquivos devemos abri-lo e associá-lo com uma variável.
- A variável será um objeto do tipo **file** que contém métodos para ler e escrever no arquivo.
- O primeiro passo então é abrir o arquivo com o comando **open**:

```
var_arquivo = open("nome_do_arquivo", "modo")
```

- O **nome_do_arquivo** pode ser relativo ou absoluto.
- O **modo** pode ser "r" (leitura), "r+" (leitura e escrita), "w" (escrita), "a" (append).

```
arq = open("teste.txt", "r")
```

No exemplo acima, **arq** está associado com o arquivo **teste.txt** que foi aberto apenas para leitura.

Arquivos texto

```
arq = open("teste.txt", "r")
```

- O primeiro parâmetro para **open** é uma string com o nome do arquivo
  - Pode ser absoluto, por exemplo: "/user/eduardo/teste.txt"
  - Pode ser relativo como no exemplo acima: "teste.txt"
- O segundo parâmetro é uma string informando como o arquivo será aberto.
  - Se para leitura ou gravação de dados, ou ambos.
  - Se é texto ou se é binário.
  - No nosso exemplo o **r** significa que abrimos um arquivo texto para leitura.

Abrindo um arquivo texto para leitura

- Se abrirmos um arquivo para leitura e ele não existir, ocorrerá um erro:

```
>>> arq = open("naoExiste.txt", "r")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory: 'naoExiste.tx
```

Abrindo um arquivo texto para leitura

- Se ao abrir um arquivo ele não existir, podemos tratar o erro usando os comandos **try except** de Python.
- Todo erro em python gera o que chamamos de **exceção**.
- Quando comandos são executados dentro de um bloco **try**, se ocorrer uma exceção automaticamente passa a ser executado os comandos do bloco **except**.

```
try:
  comandos que podem gerar exceção
except:
  comandos executados se houver alguma exceção
```

Abrindo um arquivo texto para leitura

- Ao se trabalhar com arquivos é bom colocar a abertura do arquivo no bloco **try**, e o tratamento da exceção no bloco **except**.

```
try:
    arq = open("teste.txt", "r")
    print("Abri arquivo com sucesso")
except:
    print("Não foi possível abrir o arquivo")
```

Lendo dados de um arquivo texto

- Para ler dados do arquivo aberto, usamos o método **read**.
  - **read(num_bytes)**: Retorna uma string contendo os próximos **num_bytes** do arquivo.
  - **read()**: Sem parâmetro é retornado uma string contendo todo o arquivo! Se o arquivo tiver Gigas de tamanho o problema será seu para lidar com isso!

```
try:
    arq = open("teste.txt", "r")
    conteudo = arq.read()
except:
    print("Arquivo não pode ser aberto")
```

Lendo dados de um arquivo texto

- Quando um arquivo é aberto, um **indicador de posição** no arquivo é criado, e este recebe a posição do início do arquivo.
- Para cada dado lido do arquivo, este indicador de posição é automaticamente incrementado para o próximo dado não lido.
- Eventualmente o indicador de posição chega ao fim do arquivo:
  - O método **read** devolve uma string vazia caso o indicador de posição esteja no fim do arquivo.

O exemplo abaixo mostra o conteúdo do arquivo **teste.txt** na tela.

```
try:
    arq = open("teste.txt", "r")
    while True:
        s = arq.read(1)
        print(s,end="")
        if(s == ""):
            break
    arq.close()
except:
    print("Arquivo teste.txt não pode ser aberto")
```

Lendo dados de um arquivo texto

```
try:
    arq = open("teste.txt", "r")
    while True:
        s = arq.read(1)
        print(s,end="")
        if(s == ""):
            break
    arq.close()
except:
    print("Arquivo teste.txt não pode ser aberto")
```

- O método **close** deve sempre ser usado para fechar um arquivo que foi aberto.
  - Quando escrevemos dados em um arquivo, este comando garante que os dados serão efetivamente escritos no arquivo.
  - Ele também libera recursos que são alocados para manter a associação da variável com o arquivo.

Lendo dados de um arquivo texto

- O programa anterior pode ser alterado para ler todo o arquivo de uma vez.
  - Mas lembre-se que se o arquivo for muito grande isto pode acarretar em uma sobrecarga da memória do seu computador fazendo com que este fique lento ou mesmo trave.

```
try:
    arq = open("teste.txt", "r")
    s = arq.read()
    print(s,end="")
    arq.close()
except:
    print("Arquivo teste.txt não pode ser aberto")
```

Lendo dados de um arquivo texto

- Uma maneira mais eficiente do que se ler um *byte* por vez e menos arriscada do que se ler todo o arquivo de uma única vez, é ler uma linha por vez.
- Para isso usamos o método **readline()** que devolve uma linha do arquivo em formato string.

```
try:
    arq = open("teste.txt", "r")
    while True:
        s = arq.readline()
        print(s,end="")
        if(s == ""):
            break
    arq.close()
except:
    print("Arquivo teste.txt não pode ser aberto")
```

Lendo dados de um arquivo texto

- Notem que ao realizar a leitura de um caractere, ou uma linha, automaticamente o indicador de posição do arquivo se move para o próximo caractere (ou linha).
- Ao chegar no fim do arquivo o método **read (readline())** retorna a string vazia.
- Para voltar ao início do arquivo novamente você pode fechá-lo e abrí-lo mais uma vez, ou usar o método **seek**.
- **seek(offset, from_what)**: o primeiro parâmetro indica quantos bytes se mover a partir do valor inicial **from_what**.
- Os valores de **from_what** podem ser:
  - 0: indica início do arquivo.
  - 1: indica a posição atual no arquivo.
  - 2: indica a posição final do arquivo.

Lendo dados de um arquivo texto

- O programa abaixo imprime duas vezes o conteúdo do arquivo **teste.txt**.

```
try:
    arq = open("teste.txt", "r")
    while True:
        s = arq.readline()
        print(s,end="")
        if(s == ""):
            break

    arq.seek(0,0) #mover indicador de posição
                  #0 bytes a partir do início
    while True:
        s = arq.readline()
        print(s,end="")
        if(s == ""):
            break

    arq.close()
except:
    print("Arquivo teste.txt não pode ser aberto")
```

Escrevendo dados em um arquivo texto

- Para escrever em um arquivo, ele deve ser aberto de forma apropriada usando o modo **w**, **a** ou **r+**.
- **arq = open("nome_arquivo", "modo")**
  - **w**: se o arquivo existir ele será sobreescrito, ou seja todo o conteúdo anterior será apagado.
  - **a**: o indicador de posição ficará no fim do arquivo, e dados escritos serão adicionados no fim do arquivo.
  - **r+**: o indicador de posição ficará no início do arquivo, e dados serão escritos sobre dados anteriores.

Escrevendo dados em um arquivo texto

- O programa abaixo vai sobre-escrever o início do arquivo **teste.txt**.

```
try:
    arq = open("teste.txt", "r+")
    arq.write("Alterei o comeco do arquivo\n")
    arq.close()
except:
    print("Problemas no arquivo teste.txt")
```

- Já o programa abaixo vai incluir mais um texto no fim do arquivo **teste.txt**.

```
try:
    arq = open("teste.txt", "a")
    arq.write("Adicionei este texto no fim do arquivo\n")
    arq.close()
except:
    print("Problemas no arquivo teste.txt")
```

- Este outro apagará todo conteúdo anterior e escreverá um novo texto.

```
try:
    arq = open("teste.txt", "w")
    arq.write("boa noite\nola turma de mc102\neste eh um arquivo texto\n")
    arq.close()
except:
    print("Problemas no arquivo teste.txt")
```

Lembre-se: open

```
open(nome_do_arquivo, modo);
```

Modos de abertura de arquivo texto

modo	operações	indicador de posição começa
r	leitura	início do arquivo
r+	leitura e escrita	início do arquivo
w	escrita	início do arquivo
a	(append) escrita	final do arquivo

Lembre-se: open

- Se um arquivo for aberto para leitura (**r**) e ele não existir, **open** gera um erro.
- Se um arquivo for para escrita (**w**) e existir ele é sobrescrito. Se o arquivo não existir um novo arquivo é criado.
- Se um arquivo for aberto para leitura/escrita (**r+**) e existir ele NÃO é apagado. Se o arquivo não existir, **open** gera um erro.

Alterando um Texto

- Podemos ler todo o texto de um arquivo e fazer qualquer alteração que julgarmos necessária.
- O texto alterado pode então ser sobrescrito sobre o texto anterior.
- Como exemplo vamos fazer um programa para alterar um texto substituindo toda ocorrência da letra 'a' por 'A'.
- Como uma string é imutável primeiro transformaremos esta em lista, alteramos o que precisar, depois transformamos a lista em string novamente para então escrever em arquivo.

Transformando strings em listas e vice-versa.

```
>>> a = "abc"
>>> a = list(a)
>>> a
['a', 'b', 'c']
>>> a = "".join(a)
>>> a
'abc'
```

Alterando um Texto

Programa que altera arquivo texto trocando ocorrências de 'a' por 'A'.

```
try:
    arq = open("teste.txt", "r+")
    t = arq.read()
    t = list(t) #transformamos em lista
    for i in range(len(t)):
        if(t[i] == 'a'):
            t[i] = 'A'
    arq.seek(0,0)
    t = "".join(t)
    arq.write(t)
    arq.close()
except:
    print("Problemas no arquivo teste.txt")
```

sys.argv

- É possível um programa em Python receber parâmetros diretamente da linha de comando quando o programa é executado.
- Para isso devemos importar o módulo **sys** e ler os dados armazenados na lista **sys.argv**
  - O primeiro parâmetro na lista **sys.argv** é o nome do arquivo que contém o programa.
  - Os demais parâmetros aparecem na mesma ordem em que foram digitados na linha de comando.

O programa abaixo imprime os parâmetros da linha de comando, um por linha.

```
import sys
print("Voce executou o programa com ", len(sys.argv), " parâmetros!")
print("Os parâmetros foram")
for p in sys.argv:
    print(p)
```

Argc e Argv

- Seu uso é útil em programas onde dados de entrada são passados via linha de comando.
- Exe: dados a serem processados estão em um arquivo, cujo nome é passado na linha de comando.

```
#este programa mostra o conteúdo de um arquivo
#o nome do arquivo deve ser passado como parâmetro
import sys
if(len(sys.argv)!=2):
    print("Execute\npython more.py nome_do_arquivo")
else:
    try:
        arq = open(sys.argv[1], "r")
        while True:
            t = arq.readline()
            print(t,end="")
            if(t==""):
                break
        arq.close()
    except:
        print("Arquivo não existe!")
```