Arquivos e Persistência de dados

:)

January 14, 2026

1 Introdução

Este documento visa explicar como funciona a manipulação de arquivos no nosso Portugol. As abordagens, especialmente as subrotinas (funções e procedimentos), apresentadas neste documento e suas respectivas aplicações não são idênticas a todas as linguagens de programação, embora os conceitos relacionados sejam totalmente válidos para compreender como podemos lidar com os dados da memória secundária de um computador.

Até o momento, desde Algoritmos I, trabalhamos manipulando os dados da memória primária (memória RAM) e os lendo e escrevendo por meio dos procedimentos `leia()` e `escreva()`. Essa unidade expande a nossa compreensão ao apresentar um novo tipo de dado que permite acessar o conteúdo de arquivos da memória secundária tanto para entrada (leitura) quanto para saída (escrita).

2 Entrada e saída

É fundamental para a compreensão da manipulação de arquivos entender detalhadamente os conceitos de entrada e saída, bem como sua relação com os procedimentos que já estamos acostumados, como `leia()` (para entrada) e `escreva()` (para saída) .

Tanto o procedimento `leia()` quanto o procedimento `escreva()` possuem "parâmetros implícitos" de arquivos que representam para o sistema operacional dispositivos de entrada e saída, respectivamente. Esses parâmetros implícitos, assumidos quando não passamos um parâmetro do tipo sequência, representam **dispositivos padrões**, podendo esses serem tanto de entrada quanto de saída. Para o `leia()` o dispositivo padrão de leitura é o teclado, enquanto para o `escreva()` o dispositivo padrão de escrita é a tela.

Nesse sentido, podemos entender que, ao passar um parâmetro do tipo sequência para quaisquer dos dois procedimentos, estamos desconsiderando os parâmetros implícitos (ou seja, os arquivos que representam os dispositivos padrões) e priorizando o arquivo associado ao parâmetro do tipo sequência passado para o procedimento em questão. Esse uso é demonstrado posteriormente nas subseções 4.2.4 e 4.2.5.

3 Dispositivos de armazenamento

A partir de agora vamos trabalhar com dados da memória secundária, mas, afinal, o que é a memória secundária ou, genericamente, um dispositivo de armazenamento secundário? Em contraste com a memória primária (RAM), a principal característica da memória secundária é a **não volatilidade** (persistência de dados mesmo sem energia constante). Com base nessa característica, podemos perceber que alguns exemplos de dispositivos de armazenamento secundário incluem HDs, SSDs e memórias USB.

Além disso, o nome do nosso novo tipo de dado (sequência) se deve a forma como os dados são organizados e acessados em arquivos da memória secundária. Estamos acostumados a acessar o conteúdo das variáveis da memória primária de forma direta e aleatória, mas na memória secundária o acesso ao conteúdo dos arquivos geralmente é realizado em ordem **sequencial**.

É importante enfatizar que memória primária é mais rápida, mas armazena menos dados do que a memória secundária. Portanto, nunca tente copiar um conteúdo integral da memória secundária para a memória primária.

4 O tipo de dado sequência

Considere a existência de um novo tipo de dado estruturado homogêneo chamado sequência. O tipo de dado sequência será utilizado para representar arquivos. A declaração do tipo sequência deve ser feita da seguinte maneira:

```
var
    f: sequência de tipo
```

4.1 O manipulador de arquivo

O manipulador de arquivo é um apontador para uma determinada posição da sequência. Apesar de ter utilizado o termo "apontador", não estamos lidando com o tipo de dado ponteiro. Na próxima subseção divagaremos acerca de várias funções e procedimentos para lidar com arquivos, sendo um deles a função `abre()` que trata-se de uma função lógica que faz a tentativa de abertura de um arquivo. Se for possível fazer a abertura do arquivo, então o manipulador do arquivo passará a apontar para a posição 1 da sequência.

Figure 1: Arquivo de inteiro

O manipulador do arquivo se move apenas de duas maneiras: utilizando o procedimento `leia()` e o procedimento `escreva()`. Ao utilizar qualquer um dos dois procedimentos, o apontador passará a apontar para a posição seguinte da sequência. Caso o apontador esteja na última posição o apontador passará a pontar para o final da sequência. A consequência deste fato é que é possível utilizar o procedimento `escreva()` no final da sequência e isto fará com que a sequência aumente de tamanho. ATENÇÃO: uma sequência não pode diminuir de tamanho.

4.2 Funções e procedimentos

4.2.1 A função abre()

```
função abre(f, nome): lógico
```

Sempre que precisamos lidar com arquivos precisamos abri-los, porém nem sempre o arquivo abre, seja por não existir, seja por estar corrompido. Por isso, precisamos fazer uma verificação para identificar se o arquivo existe no dispositivo de armazenamento secundário e pode ser aberto.

A função `abre()` carrega dois parâmetros: o manipulador do arquivo (a variável f) e o seu nome (do tipo string). Cada arquivo possui o seu próprio manipulador e um nome único. Caso a tentativa de abrir o arquivo seja sucesso (retorne `VERDADEIRO`), então o manipulador do arquivo passará a apontar para a posição 1 da sequência.

Foi dito que sempre deve-se verificar se o arquivo existe. Transformando isso em código, teriamos algo como:

```
var
    f: sequência de inteiro
início
    ...
    se abre(f, nome) então
        ...
    fim se
...
```

Uma pergunta razoável seria: o que fazer caso o arquivo não abra? Depende do exercício, mas você deve tratar o erro como instruído no primeiro assunto da disciplina de Algoritmos II.

4.2.2 A função cria()

```
função cria(f, nome): lógico
```

A função `cria()` serve para criar um novo arquivo no dispositivo de armazenamento secundário. Geralmente um arquivo novo é criado para remover elementos de um arquivo, tarefa que não seria possível sem a criação de um novo arquivo.

A função deve receber como parâmetros o manipulador do arquivo e o nome que desejar. Trata-se de uma função do tipo lógico que retornará `VERDADEIRO` se, e somente se, o arquivo for criado. Nesse caso, comumente usamos um nome temporário em vez do nome recebido como parâmetro e depois o renomeamos com o nome do arquivo original recebido do usuário. O nome temporário deve incluir a extensão do arquivo (como .dat ou .bin, por exemplo).

Caso o arquivo seja criado com sucesso, o manipulador do arquivo passará a apontar para o final da sequência. Geralmente se utiliza esta função da mesma maneira que se utiliza a função `abre()`, pois ambas essencialmente associam um manipulador a um arquivo.

```
var
    f: sequência de inteiro
início
    ...
    se cria(f, "temp.dat") então
        ...
    fim se
...
```

Detalhe: se utilizar a função `cria()` com o mesmo nome que outro arquivo, então o já existente será sobrescrito e dará lugar ao arquivo recém-criado. Isso significa que perderá todo o conteúdo do arquivo.

4.2.3 A função fds()

```
função fds(f): lógico
```

Na subseção 4.1 foi feito uma divagação acerca do manipulador de arquivo. Foi citado que o manipulador pode apontar para o final da sequência, que corresponde a posição posterior à posição do último elemento da sequência. Em geral, queremos saber quando o apontador chega ao fim da sequência e isso é feito com a função `fds()` que significa "fim de sequência". Esta é uma função lógica que recebe apenas o manipulador do arquivo como parâmetro.

Pelo fato da função ser do tipo lógico, isso significa que podemos fazer coisas como:

```
var
    f: sequência de inteiro
início
    ...
    enquanto não fds(f) faça
        ...
    fim enquanto
...
```

4.2.4 O procedimento leia()

```
procedimento leia(f, variáveis)
```

Esse procedimento é o mesmo usado desde Algoritmos I. A única diferença fica em seu uso para ler de arquivos (em vez do teclado), onde um manipulador associado a um arquivo deve ser passado como o primeiro parâmetro antecedendo a variável (ou as variáveis, se houver mais de uma) de leitura. A leitura deve ser realizada numa variável do mesmo tipo da sequência.

4.2.5 O procedimento escreva()

```
procedimento escreva(f, expressão)
```

Esse procedimento é o mesmo usado desde Algoritmos I. Assim como o `leia()`, a diferença é que para escrever em arquivos (em vez da tela) precisamos passar um manipulador associado a um arquivo como o primeiro parâmetro. Uma vez que o primeiro parâmetro seja um manipulador válido associado a um arquivo, todas as expressões seguintes serão escritas no arquivo.

4.2.6 O procedimento fecha()

```
procedimento fecha(f)
```

O procedimento `fecha()` DEVE sempre ser utilizado em conjunto com as funções `abre()` e `cria()`. O procedimento deve receber o manipulador associado ao arquivo que deseja fechar.

4.2.7 O procedimento reinicia()

```
procedimento reinicia(f)
```

Esse procedimento serve para fazer o manipulador do arquivo retornar para a primeira posição da sequência sem ter que fechar e abrir novamente o arquivo. É recomendável que esse procedimento seja usado apenas após chegar ao fim da sequência.

4.2.8 O procedimento renomeia()

```
procedimento renomeia(nome_atual, novo_nome)
```

O procedimento `renomeia()` serve apenas para alterar o nome do arquivo. Geralmente é utilizado em exercícios que envolvem criação de arquivos.

4.2.9 O procedimento apaga()

```
procedimento apaga(nome)
```

O procedimento `apaga()` serve para apagar o arquivo.

4.3 Exemplo

Esta subseção contém apenas um único exemplo de exercício para que entenda como aplicar a teoria nos exercícios.

Exemplo 4.1. Faça o algoritmo de uma função real que receba o nome de uma sequência de inteiros e retorne a média dos valores nela contidos.

```
função Média(s: string): real
var
    qnt, soma, x: inteiro
    f: sequência de inteiro
início
    qnt ← 0
    soma ← 0
    se abre(f, s) então
        enquanto não fds(f) faça
            leia(f, x)
            soma ← soma + x
            qnt ← qnt + 1
        fim enquanto
        fecha(f)
        se qnt ≠ 0 então
```

```
            retorne soma/qnt
        senão
            retorne -99999999
        fim se
    senão
        retorne -99999999
    fim se
fim função
```

Este exercício serve apenas para que você entenda como começar a utilizar as funções, procedimentos e o manipulador de arquivo. Pontos a se atentar: nunca se esqueça de fechar o arquivo e tomar cuidado com o tratamento caso o arquivo não abra.