Tratamento de Arquivos

Prof. Joaquim Quinteiro Uchôa

Arquivos - i

O armazenamento de dados em variáveis é temporário, isto é, os dados se perdem quando o programa termina sua execução.

Para manter grandes quantidades de dados de forma permanente são usados arquivos.

Arquivos são armazenados em dispositivos de memória secundária, tais como: discos rígidos, CD, DVD, pendrives, cartões de memória, etc.

Arquivos - II

Dispositivos de memória secundária podem reter grandes volumes de dados por longos períodos de tempo.

Processos de entrada e saída de dados em arquivos são executados por funções especialmente criadas para esta finalidade e disponibilizadas em bibliotecas específicas.

Arquivos - iii

Dados em arquivos podem ser manipulados em dois diferentes tipos de fluxos de dados (**stream**):

- **Fluxo de texto**: composto por uma sequência de bytes que denotam uma sequência de caracteres. Para isso, utiliza-se geralmente caracteres da tabela ASCII.
- **Fluxo binário**: composto por uma sequência de bytes lidos, sem tradução, diretamente do dispositivo externo. Os dados são gravados exatamente como estão organizados na memória do computador. Dados estão sujeitos às convenções dos programas que os geraram.

Streams

Em C++, arquivos são entendidos como streams cujos dados estão guardados em um dispositivo de armazenamento secundário. Existem três tipos de streams para manipulação de arquivos:

– `ifstream` (para entrada/leitura de dados)
– `ofstream` (para saída/escrita de dados)
– `fstream` (para entrada e saída)

Para ter acesso, deve-se incluir a biblioteca `<fstream>`.

Operações comuns em arquivos

- Abertura e fechamento de arquivos;
- Apagar um arquivo;
- Leitura e escrita;
- Indicação de que o fim de arquivo foi atingido;
- Posicionar o arquivo em um ponto determinado.

➔ Para escrever ou ler, é necessário abrir o arquivo.
➔ Ao final das operações necessárias o programa deve fechar o arquivo.
➔ Quando um arquivo é fechado, o conteúdo dos buffers é descarregado para o dispositivo externo.

Arquivos texto

Arquivos são abertos em modo texto por padrão.

Sobre a escrita:

- ➔ Se o arquivo não existir, o mesmo será criado;
- ➔ Se o arquivo já existir, seu conteúdo será apagado;
- ➔ Escrevemos no arquivo de modo similar ao uso de **`cout`**.

Sobre a leitura:

- ➔ Se o arquivo não existir, o mesmo não será criado;
- ➔ Lemos no arquivo de modo similar ao uso de **`cin`**.

Exemplo

```
#include <fstream>

using namespace std;

int main(){
    ofstream arquivo(“meus_dados.txt”);
    arquivo << “Escrevendo no arquivo...” << endl;
    arquivo.close();
    return 0;
}
```

Abertura de Arquivos

O trecho

```
ofstream arquivo(“meus_dados.txt”);
```

é equivalente a:

```
ofstream arquivo;
arquivo.open(“meus_dados.txt”);
```

A opção é preferida, sempre que possível, uma vez que alguns compiladores otimizam o código.

Assegurando Abertura

Na prática, para garantir que o arquivo foi criado corretamente, poderíamos fazer:

```
if (arquivo) {
   /* Bloco de instruções 1
      Arquivo apto para uso */
} else {
   /* Bloco de instruções 2
      Arquivo inapto para uso */
}
```

Lendo do Arquivo

```
#include <fstream>
using namespace std;
int main(){
    ifstream arquivo("meus_dados.txt");
    string dados;
    if (arquivo) {
        while ( arquivo >> dados ) {
            cout << dados << endl;
        }
        arquivo.close();
    }
    return 0;
}
```

Observação

O teste

```
while ( arquivo >> dados ) {
    (...)
```

verifica se foi possível ler do arquivo, verificando, entre outros, se não chegou ao final do arquivo.

Esse teste é mais efetivo e eficiente que:

```
while ( !arquivo.eof() ) {
    arquivo >> dados
    (...)
```

Modos de Abertura

Um arquivo pode ser aberto com diferentes configurações. Em C++, os seguintes modos estão disponíveis:

- `ios::app` ⇒ Não trunca o arquivo, só permite escrever no fim.
- `ios::ate` ⇒ Abre o arquivo, posiciona a cabeça de leitura/escrita na posição final. Permite mover a cabeça de leitura/escrita.
- `ios::binary` ⇒ Abre o arquivo em modo binário
- `ios::in` ⇒ Abre para leitura.
- `ios::out` ⇒ Abre para escrita.
- `ios::trunc` ⇒ Trunca (apaga o conteúdo) o arquivo.

Exemplos

➔ Abertura de arquivo para escrita no fim:

```
ofstream arquivo("meus_dados.txt", ios::app)
```

➔ Abertura de arquivo em modo binário:

```
ofstream arquivo("meus_dados.dat", ios::binary)
```

Para maioria das necessidades, não é necessário o uso desse tipo de recurso.

Arquivos Binários

A nível de Sistema Operacional, os dados de arquivos são obtidos em bloco, como se fossem uma sequência (stream) de bytes. Exemplo:

```
#include <unistd.h>
#include <fcntl.h>
int main(){
    int fd;
    fd = open("teste.txt", O_CREAT|O_RDWR);
    write(fd, "teste\n", 7);
    close(fd);
    return 0;
}
```

C++ e streams binárias

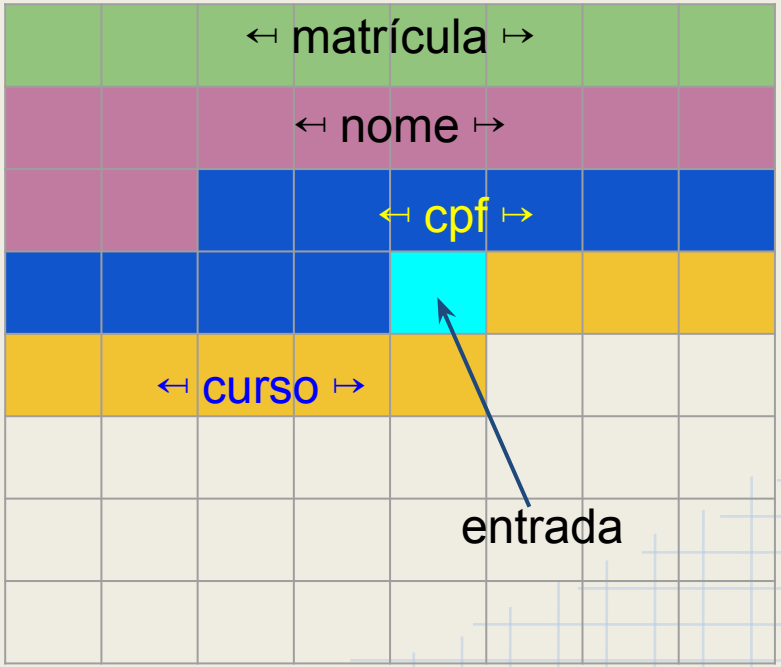
Os tipos fstream, ifstream e ofstream suportam métodos para leitura e escrita de sequências de bytes em arquivos. Ex:

```
ifstream entrada(“teste.txt”,ios::binary);
ofstream saida(“resultado.txt”,ios::binary);
char teste[10];
entrada.read(teste,10);
saida.write(“teste\n”,7);
```

Arquivos Binários e Memória

Arquivos binários permitem gravar os dados como são representados na memória. Ex:

```
struct aluno {
        int matricula;
        char nome [10];
        char cpf[10];
        char entrada;
        int curso;
};
aluno jose;
```



Escrevendo dados binários

Podemos acessar a memória como se fosse um vetor de bytes (caracteres) e gravar os dados acessados em arquivo:

```
arquivo.write((const char *) (&variavel),
sizeof(tipo_dado_variavel));
```

ou

```
arquivo.write(reinterpret_cast<const char *> (&variavel),
sizeof(tipo_dado_variavel));
```

Exemplo:

```
aluno jose;
arquivo.write((const char *) (&jose), sizeof(aluno));
```

Lendo dados binários

Os dados em arquivos binários são lidos como vetores de bytes (caracteres):

```
arquivo.read((char *) (&variavel), sizeof(tipo_dado_variavel));
```

ou

```
arquivo.read(reinterpret_cast<char *> (&variavel),
sizeof(tipo_dado_variavel));
```

Exemplo:

```
aluno jose;
arquivo.read((const char *) (&jose), sizeof(aluno));
```