Manipulação de Arquivos do Tipo Texto

O armazenamento de dados em variáveis ou vetores é temporário, ou seja, ao terminar o programa os dados deixam de existir, já que eles residem na *memória principal*. Para o armazenamento permanente dos dados é necessária utilização de *arquivos* (*files*), que ficam residentes na *memória secundária* tais como discos magnéticos, ópticos ou fitas.

O principal objetivo desta aula consiste em explicar como os arquivos de dados são criados, atualizados e utilizados por programas C++. Serão vistos dois tipos de acesso a arquivos: seqüencial e aleatório (*random*). Todo arquivo possui um *ponteiro de arquivo* que indica a posição do arquivo onde será efetuada a próxima leitura/gravação. O acesso seqüencial de um arquivo implica que uma vez efetuada uma leitura ou gravação, o ponteiro de arquivo automaticamente se posiciona na próxima posição a ser lida. O acesso aleatório implica que o programador deve posicionar inicialmente o ponteiro de arquivo para o local desejado e então efetuar uma leitura ou escrita. Também serão vistas técnicas de processamento de arquivos texto e arquivos binários nesta e nas próximas aulas.

Um arquivo é um conjunto de *registros*. Os registros são formados por unidades de informação denominadas *campos*. As operações elementares que estão associadas a arquivos são: **abrir** (*open*), **fechar** (*close*), **ler** (*read*), **escrever** (*write*), **testar** pelo final-de-arquivo (eof = *end-of-file*) e **posicionar** (*seek*) o arquivo em um determinado registro. Um arquivo pode ser de leitura (*read*), de escrita (*write*) ou de leitura/escrita (*read*/*write*). O acesso aos registros em um arquivo pode ocorrer de forma seqüencial, aleatória ou uma combinação de ambas. Como regra geral nas linguagens de programação, todo arquivo antes de ser utilizado deve ser **aberto**; ao término das operações com o arquivo ele deve ser **fechado**. Entretanto, em C++ o fechamento de arquivo é opcional, pois arquivos são objetos em C++ e, portanto, seu *finalizador* se encarrega de fechar o arquivo quando o objeto deixa de existir. Entretanto é uma boa prática de programação fechar um arquivo quando ele não é mais necessário para economizar recursos do sistema operacional.

Criando um Arquivo Seqüencial

Inicialmente, vamos utilizar um arquivo para armazenar a saída de um programa. No exemplo seguinte a saída será destinada para o arquivo saída.txt ao invés da tela; o número e o nome de funcionários são lidos por meio do teclado e seus valores são escritos no arquivo.

```
 1 #include <iostream>
 2 #include <fstream>
 3 #include <cstdlib>
 4 #include <string>
 5 using namespace std;
 6 int main()
 7 { int numero;
 8   string nome;
 9   ofstream outFile;  // outFile é o arquivo onde a saída será escrita
10
11   outFile.open("saída.txt", ios::out);  // abre o arquivo para escrita
12   if (! outFile)
13   {  cout << "Arquivo saida.txt nao pode ser aberto" << endl;
14      abort();
15   }
16
17   cout << "Entre com o numero e nome do funcionário\n"
18        << "Fim de arquivo (Ctrl-Z) termina a entrada de dados\n\n? ";
19   while(cin >> numero >> nome)
20   {  outFile << numero << " " << nome << '\n';
21      cout << "? ";
22   }
23   outFile.close(); // se o programador omitir a chamada ao método close
```

```
24    return 0;         // o finalizador se encarrega de fechar o arquivo
25 }
```

Observe as operações básicas sobre arquivos de saída neste exemplo. Em primeiro lugar, foi declarada uma variável do tipo arquivo (linha 9) e a variável foi associada a um arquivo em disco, aberto para escrita (linha 11). O construtor do objeto ofstream permite que essas duas operações sejam realizadas em um único comando, ou seja os comandos:

```
ofstream outFile;
outFile.open("saída.txt", ios::out);
```

são equivalentes ao único comando:

```
ofstream outFile("saída.txt", ios::out);
```

O primeiro parâmetro passado para o método *open* (ou ao construtor) é o nome do arquivo ("saída.txt") e o segundo parâmetro é o modo de abertura do arquivo (ios::out). Para um objeto ofstream, pode-se utilizar, dentro outros, os seguintes modos de abertura:

Modo	Descrição
`ios::out`	Cria e abre um novo arquivo; se o arquivo já existe, todos seus dados são apagados
`ios::app`	Abre um arquivo existente, posicionando o ponteiro do arquivo no final, permitindo adicionar linhas (os dados já existentes não são modificados)
`ios::nocreate`	Se o arquivo não existe, a operação de abertura falha
`Ios::noreplace`	Se o arquivo existe, a operação de abertura falha

Por *default*, os objetos ofstream são abertos no modo ios::out, ou seja, o comando:

```
ofstream outFile("saída.txt", ios::out);
```

é equivalente ao comando:

```
ofstream outFile("saída.txt");
```

Após criada uma variável do tipo ofstream e feita uma tentativa de abertura do arquivo, o programa testa se a operação de abertura teve sucesso (linhas 12 a 15):

```
if (! outFile)
{  cout << "Arquivo saida.txt nao pode ser aberto" << endl;
   abort();
}
```

O teste "! outFile" determina se a operação de abertura não teve sucesso. Nesse caso, uma mensagem de erro é fornecida e o programa interrompe sua execução. Alguns erros possíveis na abertura de arquivos consistem na falta de permissão ou quando não há mais espaço em disco.

Em seguida (linhas 17 a 22), o programa solicita que o usuário entre com o número e nome de funcionários de uma empresa qualquer. A condição no comando while (linha 19) é:

```
while(cin >> numero >> nome)
```

Esta condição é verdadeira enquanto o usuário entrar com valores diferentes de final de arquivo. Em ambientes Dos/Windows o final de arquivo é a seqüência de teclas Ctrl-Z. Em ambientes Unix, é a seqüência Ctrl-D.

Na linha 20

```
outFile << numero << " " << nome << '\n';
```

os valores fornecidos pelo usuário são escritos no arquivo saída.txt utilizando o operador << e a variável associada outFile. Quando um final de arquivo é encontrado, o laço termina e então o comando (linha 23)

```
outFile.close();
```

fecha o arquivo saida.txt. Entretanto, o *finalizador* do objeto ofstream realiza essa operação automaticamente. Assim, é possível omitir completamente a linha 23 e mesmo assim o arquivo será fechado. O arquivo saída.txt pode ser lido por qualquer editor de texto. Fechar um arquivo faz com que qualquer informação que tenha permanecido no *buffer* associado ao fluxo de saída seja gravada no disco. Quando você envia dados para serem gravados em um arquivo, eles são armazenados temporariamente em uma área de memória (*buffer*) ao invés de serem imediatamente escritos em disco. Quando o *buffer* estiver cheio, seu conteúdo é escrito no disco de uma vez pelo sistema operacional. A razão para efetuar isso se deve à eficiência na leitura e escrita de arquivos. Se, para cada dado que fosse ser gravado, o sistema operacional tivesse que posicionar a cabeça de leitura/gravação em um ponto específico do disco, apenas para gravar aquele dado, as gravações seriam muito lentas. Assim, estas gravações só são efetuadas quando há um volume razoável de informações a serem gravadas ou quando o arquivo for fechado. Em C++ é possível forçar a gravação dos dados em disco utilizando `endl` (*end line* = fim de linha),ou seja, `endl` além inserir um caracter de fim-de-linha (´\n´) ele descarrega (ou esvazia) (*flush*) o *buffer* de saída no mesmo, mesmo que ele não esteja cheio. O *buffer* de saída também pode ser descarregado pelo do comando:

```
outFile << flush;
```

Como observação final, os comandos das linhas 19-22 poderiam ser substituídos pelo seguinte fragmento de código equivalente:

```
cin >> numero >> nome; // ler dados
while(! cin.eof())     // final de dados?
{ outFile << numero << " " << nome << '\n';
  cout << "? ";
  cin >> numero >> nome; // ler proximos dados
}
```

Resumo das operações básicas sobre arquivos seqüenciais para escrita:

- Abrir: `ofstream arquivo(”nome-do-arquivo”);`
- Escrever: `arquivo << valor1 << “ “ << valor2 << ... << endl;`
- Esvaziar o *buffer* `arquivo << flush;` ou `arquivo << endl;`
- Teste de fim-de-arquivo: `arquivo.eof();`
- Fechar (opcional): `arquivo.close();`

Lendo Dados de um Arquivo Seqüencial

Os dados armazenados em arquivos podem ser recuperados quando necessários. Na seção anterior, foi criado o arquivo seqüencial saída.txt contendo o número e o nome de funcionários de uma empresa. Nesta seção, veremos como ler os dados armazenados em um arquivo seqüencial, por meio do seguinte exemplo:

```
 1 #include <iostream>
 2 #include <fstream>
 3 #include <cstdlib>
 4 #include <iomanip>
 5 #include <string>
 6 using namespace std;
 7 int main()
 8 { int numero;
 9   string nome;
10   ifstream inFile;  // inFile é o arquivo de leitura dos dados
11
12   inFile.open(”saída.txt”, ios::in);    // abre o arquivo para leitura
13   if (! inFile)
14   {  cout << “Arquivo saida.txt nao pode ser aberto” << endl;
15      abort();
16   }
```

```
17
18    cout << setiosflags(ios::left)
19         << setw(10) << "Numero"
20         << setw(50) << "Nome" << endl;
21    while(inFile >> numero >> nome)
22      cout << setiosflags(ios::left)
23           << setw(10) << numero
24           << setw(50) << nome << endl;
25
26    inFile.close(); // se o programador omitir a chamada ao método close
27    return 0;      // o finalizador se encarrega de fechar o arquivo
28  }
```

Em primeiro lugar, foi declarada uma variável do tipo arquivo (linha 10) e a variável foi associada a um arquivo em disco, aberto para leitura (linha 12). O construtor do objeto ifstream permite que essas duas operações sejam realizadas em um único comando, ou seja os comandos:

```
ifstream inFile;
inFile.open("saída.txt", ios::in);
```

são equivalentes ao único comando:

```
ifstream inFile("saída.txt", ios::in);
```

O primeiro parâmetro passado para o método *open* (ou ao construtor) é o nome do arquivo ("saída.txt") e o segundo parâmetro é o modo de abertura do arquivo (ios::in). Por *default*, os objetos ifstream são abertos no modo ios::in, ou seja, o comando:

```
ifstream inFile("saída.txt", ios::in);
```

é equivalente ao comando:

```
ifstream inFile("saída.txt");
```

Após criada uma variável do tipo ifstream e feita uma tentativa de abertura do arquivo, o programa testa se a operação de abertura teve sucesso (linhas 13 a 16):

```
if (! inFile)
{  cout << "Arquivo saida.txt nao pode ser aberto" << endl;
   abort();
}
```

O teste "! inFile" determina se a operação de abertura não teve sucesso. Nesse caso, uma mensagem de erro é fornecida e o programa interrompe sua execução. Alguns erros possíveis na abertura de arquivos consiste na falta de permissão ou quando o arquivo não existe.

Em seguida (linhas 18 a 24), o programa lê os dados do arquivo. A condição no comando while (linha 21) é:

```
while(inFile >> numero >> nome)
```

Esta condição é verdadeira enquanto o programa ler valores diferentes de final de arquivo.. Quando um final de arquivo é encontrado, o laço termina e então o comando (linha 26)

```
inFile.close();
```

fecha o arquivo saida.txt. Entretanto, o *finalizador* do objeto ifstream realiza essa operação automaticamente. Assim, é possível omitir completamente a linha 26 e mesmo assim o arquivo será fechado.

Como observação final, os comandos das linhas 21 e 22 poderiam ser substituídos pelo seguinte fragmento de código equivalente:

```
inFile >> numero >> nome; // ler primeiro registro
```

```
while(! inFile.eof())
{ cout << setiosflags(ios::left) << setw(10) << numero << setw(50) << nome << endl;
  inFile >> numero >> nome; // ler próximo registro
}
```

O método eof() testa se foi atingido o final-de-arquivo (*end-of-file*). É importante salientar que este método somente pode ser chamado após realizado um comando de leitura (ou escrita) no arquivo e uma vez verdadeiro, ele permanece verdadeiro até a utilização do método clear().

Resumo das operações básicas sobre arquivos seqüenciais para leitura:

- Abrir: `ifstream arquivo("nome-do-arquivo");`
- Ler: `arquivo >> variavel1 >> variavel2 >> ... ;`
- Teste de fim-de-arquivo: `arquivo.eof();`
- Fechar (opcional): `arquivo.close();`

Atualizando Dados em um Arquivo Seqüencial

Os dados escritos em arquivos seqüenciais não podem ser modificados sem o risco de destruir outros dados no arquivo. Por exemplo, suponha o seguinte arquivo:

100 Paulo
200 Mara
300 Pedro

e que o nome "Mara" precise ser trocado para "Marelise"; assim o nome antigo não pode ser simplesmente sobreposto. Se o registro para "Mara" for sobreposto na mesma localização no arquivo utilizando o nome mais longo, o registro ficaria

200 Marelise

que contém quatro caracteres a mais que o registro original. Não esqueça que após cada registro existe apenas um '\n', ou seja, o arquivo pode ser escrito como

100 Paulo\n200 Mara\n300 Pedro\n

Portanto, os caracteres após o primeiro "e" em "Marelise" sobrescreveriam o início do próximo registro no arquivo, da seguinte forma:

100 Paulo\n200 Marelise Pedro\n

O problema é que a entrada/saída formatada utilizando << e >> pode variar em tamanho. Por exemplo, embora 5, 10 e –2010 sejam inteiros (ocupando a mesma quantidade de bytes internamente), eles ocupam tamanhos diferentes quando escritos no formato texto em um arquivo em disco. Portanto, a entrada/saída formatada não é usualmente utilizada quando a atualização de registros é necessária. Se realmente for necessário, o processo de atualização requer que todos os registros antes daquele a ser atualizado sejam copiados para um novo arquivo; o novo registro então é escrito ao novo arquivo e, finalmente, os demais registros também devem ser copiados para o novo arquivo.

Consultas em Arquivos Seqüenciais

Como visto na aula anterior, um arquivo é um conjunto de registros, onde cada registro é formado por um ou mais campos. Por exemplo, os registros de um arquivo de funcionários poderiam incluir os seguintes campos:

- Número de matrícula
- Nome
- Cargo

- Grau de escolaridade
- Sexo (M,F)
- Local
- Estado civil (S, C)
- Salário

Uma amostra de dados para um arquivo desse tipo pode ser:

Registro	E#	Nome	Cargo	Esc.	Sexo	Local	E.C.	Salário
1	800	Austin	programador	2	F	Campinas	S	10.000
2	510	William	analista	3	M	Campinhas	C	15.000
$\Rightarrow$ 3	950	Melissa	analista	3	F	Vinhedo	S	12.000
4	750	Hawkins	programador	2	M	Campinas	S	12.000
5	620	Newton	programador	2	M	Vinhedo	C	9.000

Todo arquivo possui um *ponteiro de arquivo* (mostrado como $\Rightarrow$) que indica a posição do arquivo onde:
- Será efetuada a próxima leitura (caso seja utilizado um comando de leitura);
- Será efetuada a próxima escrita ou gravação (caso seja utilizado um comando de escrita/gravação)

O objetivo principal da organização de arquivo é proporcionar meios para a recuperação e atualização dos registros. A atualização de um registro pode incluir sua eliminação, a alteração de alguns de seus campos ou a inserção de um registro completamente novo. Normalmente, alguns campos do registro são reservados como *campos-chave*. Os registros podem ser recuperados especificando os valores de alguns ou todas essas chaves. Um combinação de valores de chave, especificada para recuperação, é denominada *consulta*. Supondo que no exemplo anterior os campos Número de Matrícula, Cargo, Sexo e Salário foram designados como campos-chave. Algumas consultas válidas ao arquivo são:

Recupere os registros de todos os funcionários com:
Q1: Sexo = 'M'
Q2: Salário > 9000
Q3: Salário > média dos salários de todos os funcionários
Q4: (Sexo = 'F' **and** Cargo = 'Programador') **or** (Matricula > 700 **and** Sexo = 'M')

Note que uma consulta do tipo Locação = 'Vinhedo' seria inválida, já que o campo de locação não foi especificado como campo-chave. Os exemplos Q1-Q4 são representativos de tipos de consultas que se poderia fazer. Os quatro tipos de consultas são:

Q1: Consulta simples: especifica-se o valor de uma chave única
Q2: Consulta em intervalo: especifica-se um intervalo de valores para uma chave única
Q3: Consulta funcional: especifica-se alguns valores determinados dentro do arquivo (exemplo: soma, média, desvio-padrão)
Q4: Consulta booleana: uma combinação booleana de Q1-Q3 utilizando os operadores lógicos **and**, **or** e **not**.

É possível efetuar consultas dos tipos Q1-Q4 em arquivos seqüenciais. Nesse caso, os programas normalmente começam a leitura a partir do início do arquivo, lêem todos os registros consecutivamente até que um registro específico seja encontrado. Pode ser necessário processar um arquivo seqüencial várias vezes (a partir do começo dele) durante a execução do programa. Ambas classes ifstream e ofstream possuem métodos para reposicionar o ponteiro de arquivo. Os métodos são seekg (seek get) para ifstrem e seekp (seek put) para ofstream. Cada objeto ifstream possui seu próprio ponteiro de arquivo que indica o byte no arquivo a partir do qual a próxima leitura será efetuada; cada objeto ofstrem também possui seu próprio ponteiro de arquivo que indica o byte no arquivo a partir do qual a próxima escrita será efetuada. O comando:

```
inFile.seekg(0);
```

reposiciona o ponteiro de arquivo para o começo do arquivo (byte 0). O parâmetro de seekg é normalmente um **long int** indicando a quantidade de bytes. Um segundo parâmetro pode ser especificado para indicar a direção do posicionamento: ios::beg (default) para posicionar relativamente a partir do começo do arquivo; ios::cur para posicionar relativamente à posição atual do ponteiro de arquivo e ios::end para posicionar relativamente ao final do arquivo. Por exemplo:

```
// posiciona no n-ésimo byte de inFile (assume ios::beg)
inFile.seekg(n)

// posiciona no n-ésimo byte à frente do ponteiro de arquivo de inFile
inFile.seekg(n, ios::cur)

// posiciona no y-ésimo byte a partir do final do arquivo inFile (de trás para frente)
inFile.seekg(y, ios::end)

// posiciona no final do arquivo inFile
inFile.seekg(0, ios::end)
```

O método seekp é utilizado de forma análoga. Adicionalmente, existem os métodos tellg e tellp que retornam a posição atual do ponteiro do arquivo onde a próxima leitura ou escrita ocorrerá, respectivamente.

O programa seguinte implementa a consulta do tipo Q1, permitindo que o usuário escolha qual o sexo dos funcionários a serem listados.

```
 1 #include <iostream>
 2 #include <fstream>
 3 #include <cstdlib>
 4 #include <iomanip>
 5 #include <string>
 6 using namespace std;
 7 struct funcionario       // estrutura do registro de funcionário
 8 { int   matricula;
 9   string nome;
10   string cargo;
11   int   escolaridade;
12   char  sexo;
13   string local;
14   char  ecivil;  // estado civil
15   float salario;
16 };
17
18 int main()
19 { funcionario f;
20   char sexo;
21   ifstream funcFile("func.dat");
22
23   if (! funcFile)
24   {  cout << "Arquivo func.dat nao pode ser aberto" << endl;
25      abort();
26   }
27
28   cout << "\n\nSexo dos funcionarios a serem listados (M ou F) Ctrl-Z termina? ";
29   while (cin >> sexo)
30   { cout << "Consulta funcionario com sexo = " << sexo << endl;
31     cout << setiosflags(ios::left)
32          << setw(10) << "Matricula"
33          << setw(10) << "Nome"
34          << setw(15) << "Cargo"
35          << setw(5)  << "Esc."
36          << setw(5)  << "Sexo"
37          << setw(10) << "Local"
38          << setw(5)  << "E.C."
39          << setw(10) << "Salario" << endl;
40
41     funcFile.clear();      // limpa "eof = final de arquivo" para proximo uso
42     funcFile.seekg(0);     // posiciona no inicio do arquivo
43     funcFile >> f.matricula >> f.nome >> f.cargo >> f.escolaridade
44              >> f.sexo >> f.local >> f.ecivil >> f.salario;  // ler primeiro registro
```

```
45      while (! funcFile.eof())
46      { if(f.sexo == sexo)      // o registro atende a condição de consulta?
47         cout << setiosflags(ios::left)
48              << setw(10) << f.matricula
49              << setw(10) << f.nome
50              << setw(15) << f.cargo
51              << setw(5)  << f.escolaridade
52              << setw(5)  << f.sexo
53              << setw(10) << f.local
54              << setw(5)  << f.ecivil
55              << setw(10) << f.salario << endl;
56        funcFile >> f.matricula >> f.nome >> f.cargo >> f.escolaridade
57                 >> f.sexo >> f.local >> f.ecivil >> f.salario;  // ler próximo registro
58      }
59      cout << "\n\nSexo dos funcionarios a serem listados (M ou F) Ctrl-Z termina? ";
60    }
61    return 0;
62  }
```

Nas linhas 7-16 é definida a estrutura de dados na qual as informações de um funcionário serão colocadas. Na linha 21, o arquivo **func.dat** é aberto para leitura. Na linha 29, o usuário fornece qual o sexo que ele deseja consultar no arquivo de funcionários (M ou F). Na linha 41, o *flag* de final de arquivo é "limpo", caso contrário uma vez ele sendo verdadeiro ele permanece verdadeiro. Na linha 42, o ponteiro de arquivo é posicionado no começo do arquivo. Nas linhas 43-44 é lido o primeiro registro do arquivo e logo a seguir na linha 45 é verificado se o final de arquivo (eof = *end-of-file*) foi atingido. Lembre-se que antes de testar pelo final de arquivo é necessário efetuar pelo menor um comando de leitura que altera o valor do *flag* de final de arquivo para seu valor adequado. Na linha 46 encontra-se o critério de seleção de registros, ou seja, o critério da consulta; caso ele seja satisfeito, o registro é exibido na tela (linhas 47-55). Em seguida, nas linhas 56-57 o próximo registro do arquivo é lido para continuidade do *loop* na linha 45. Ao encontrar o final de arquivo, o processamento termina e o programa solicita ao usuário (linha 59) novamente qual o sexo dos funcionários a serem listados para continuidade do *loop* na linha 29. Ao digitar Ctrl-Z, o programa termina. Fica como exercício a implementação das consultas Q2-Q4.

Exemplo de execução

Arquivo func.dat

```
800 Austin  programador 2 F Campinas  S 10000
510 William analista    3 M Campinas  C 15000
950 Melissa analista    3 F Vinhedo   S 12000
750 Hawkins programador 2 M Campinas  S 12000
620 Newton  programador 2 M Vinhedo   C  9000
```

Execução do programa anterior com o arquivo func.dat

```
Sexo dos funcionarios a serem listados (M ou F) Ctrl-Z termina? M
Consulta funcionario com sexo = M
Matricula Nome      Cargo          Esc. Sexo Local     E.C. Salario
510       William   analista       3    M    Campinas  C    15000
750       Hawkins   programador    2    M    Campinas  S    12000
620       Newton    programador    2    M    Vinhedo   C    9000


Sexo dos funcionarios a serem listados (M ou F) Ctrl-Z termina? F
Consulta funcionario com sexo = F
Matricula Nome      Cargo          Esc. Sexo Local     E.C. Salario
800       Austin    programador    2    F    Campinas  S    10000
950       Melissa   analista       3    F    Vinhedo   S    12000


Sexo dos funcionarios a serem listados (M ou F) Ctrl-Z termina?
```