**Entrada e Saída de Dados**

A maioria dos programas de computador trabalham com arquivos. Processadores de texto criam e editar arquivos de texto; navegadores de internet interpretam e exibem arquivos HTML; compiladores leêm arquivos-fonte e geram arquivos executáveis. Um arquivo nada mais é do que uma sequência de bytes armazenada em um dispositivo, seja esse dispositivo ,por exemplo, um disco rígido, um CD ou um disquete. Tipicamente, o sistema operacional gerencia os arquivos presentes em um computador, mantendo registro de onde estão armazenados, quais os seus tamanhos, quando foram criados, etc.

Quando trabalhamos com arquivos em programação, precisamos de meios para conectar um programa à um arquivo, de modo que o programa possa ler e escrever dados dentro deste arquivo, e também meios para criar novos arquivos e salvá-los em um dispositivo. A linguagem C++ possui um pacote de classes e funções que trabalham com arquivos de forma bastante semelhante às classes cout e cin já estudadas.

**10.1 – A biblioteca fstream**

O primeiro passo para manipular um arquivo em C++ é adicionar a biblioteca específica para a manipulação de dados em arquivos ao cabeçalho de nosso programa. Esta biblioteca é chamada de “fstream”, de “file stream” (fluxo de arquivos). Para adicioná-la ao cabeçalho, fazemos:

#include <iostream> #include <fstream> using namespace std;

Note que os nomes das duas bibliotecas são parecidos: ambas contém classes que trabalham com entrada e saída de dados. No entanto, a biblioteca iostream trabalha apenas com o fluxo de dados via monitor/teclado/periféricos, enquanto que a biblioteca fstream trabalha com o fluxo de dados de arquivos. (Na verdade, atualmente a biblioteca fstream engloba todas as classes contidas em iostream, mas nem todos os compiladores já adotaram essa modificação).

**10.2 – Os objetos de fstream**

Uma vez adicionada a biblioteca, podemos criar objetos em nossos programas que servirão de intermediários entre o programa e os arquivos manipulados. A biblioteca fstream define três tipos de objeto para esta função, cada um com objetivo definido:

* ofstream: objetos que escrevem dados em um arquivo.
* ifstream: objetos que leêm dados em um arquivo.
* fstream: objetos que podem tanto ler como escrever em um arquivo.

Explicaremos o uso de cada um deles, cada vez nos aprofundando mais no assunto. Por enquanto, veremos como criar um arquivo e escrever algumas strings nele utilizando um objeto ofstream.

**10.3 – Escrevendo em um arquivo**

Podemos resumir assim as etapas necessárias para escrever em um arquivo através de um programa em C++:

- Cria-se um objeto do tipo ofstream.  
- Associa-se este objeto com um arquivo em particular (seja criando ou abrindo um arquivo já existente).  
- Usa-se o objeto para enviar dados para este arquivo, de forma bem parecida como usamos o comando cout. A diferença é que os dados vão para o arquivo, ao invés de serem exibidos na tela.

Para criar um objeto ofstream, declaramos seu nome de maneira parecida com a declaração de uma variável:

ofstream <nome do objeto>

Por exemplo:

ofstream escreve;

A linha acima cria o objeto “escreve”, do tipo ofstream, capaz de escrever em arquivos. O próxima passo é associar este objeto a um arquivo. Para isto, utilizamos a função open( ), que abre o arquivo desejado ou cria um arquivo no disco rígido. A sintaxe de open( ) é a seguinte:

<objeto>.open(“nome do arquivo”);

Por exemplo, vamos utilizar o objeto “escreve” para criar um arquivo chamado “strings.txt”. Para fazer isso, utilizamos a função open da seguinte maneira:

escreve.open(“strings.txt”);

Dessa forma, estamos associando o objeto escreve ao arquivo strings.txt. Agora podemos enviar dados através do objeto “escreve”, e estes dados serão escritos no arquivo “strings.txt”. Para fazer isso, utilizamos o objeto que criamos da mesma forma que utilizamos o comando cout. Da mesma forma como escreveríamos na tela do computador com o comando cout utilizando variáveis, strings, strings literais, formatação, etc, podemos utilizar estes recursos todos também com os objeto do tipo ofstream, com a diferença que estas informações serão gravadas em um arquivo. O exemplo abaixo reúne todos os passos anteriores e utiliza o objeto “escreve” para escrever diversos dados em um arquivo, para dar uma idéia melhor de suas possibilidades.

#include <iostream>  
#include <fstream>  
#include <string>  
using namespace std;  
  
int main(){  
  
string frase;  
cout<<"Escreva uma frase para ser escrita no arquivo string.txt:";  
getline(cin, frase);  
cout<<"Obrigado. Escrevendo dados no arquivo strings.txt...\n";  
ofstream escreve;  
escreve.open("strings.txt");  
escreve << "Utilizamos os objetos ofstream para escrever em arquivos\n";  
escreve<< "Note que podemos utilizar os caracteres \\n pra quebrar a linha, como fazíamos em cout\n";  
int numero = 100;  
escreve<<"Podemos escrever o valor de variaveis numericas: " <<numero <<"\n";  
int matriz[3] = {42, 19, 99};  
escreve<<"Podemos também escrever matrizes!";  
for (int i=0; i < 3; i++){  
escreve.width(6);  
escreve<<matriz[i]<<" ";  
}  
escreve<<"\n";  
escreve<<"Finalmente, podemos receber dados via cin e escrever estes dados no arquivo!\n";  
escreve<<"A frase que você digitou durante a execução do programa: "<<frase<<"\n";  
escreve.close();  
cout<<"Dados escritos no arquivo. Fim do Programa!";  
system("PAUSE");  
return 0;  
}

Procure o arquivo strings.txt em seu sistema, e abra ele (ele provavelmente está no mesmo diretório onde está o arquivo executável do programa que compilamos. Por definição, estes arquivos ficam no diretório onde está o compilador, a menos que tenhamos configurado o compilador para salvá-los em outro lugar.). Percebeu a semelhança entre a utilização do objeto ofstream com o comando cout? Todos os métodos que vimos com cout para exibir um dado na tela podem ser utilizado para gravar um dado em um arquivo.

**10.4 – Checando se o arquivo abriu sem problemas**

Sempre existe a possibilidade de erros quando trabalhamos com arquivos. Talvez o arquivo que desejamos ler tenha sido apagado, ou renomeado, ou esteja sendo usado por outro programa. A biblioteca de C++ possui uma função que checa se um objeto ofstream/ifstream conseguiu abrir um determinado arquivo, e se continua conectado corretamente à este arquivo: é a função is\_open( ). Esta função checa o estado do objeto, e retorna 0 para o programa caso tudo esteja certo. Caso um valor diferente de 0 seja enviado ao programa, isso indica que o arquivo não pode ser aberto pelo objeto.

O código abaixo pode ser utilizado em qualquer programa que utilize leitura e escrita de arquivos (basta, logicamente, substituir os nomes do objeto e do arquivo) para checar o estado do objeto. Caso um erro seja encontrado, o programa indica que o arquivo não pode ser aberto e a execução é terminada. Caso contrário, o programa continua sua execução normalmente.

if(!leitura.is\_open( ))  
{  
cout<<”Não foi possível abrir arquivo! Programa será terminado!\n”;  
leitura.clear( ); //reseta o objeto leitura, para limpar memória do sistema  
return 0;  
}

**10.5 – Fechando um Arquivo**

Utilizamos a função close( ) no programa acima após escrever todos os dados desejados no arquivo. Como você deve ter adivinhado, a função close simplesmente fecha o arquivo a que o objeto estava associado, liberando o objeto e a memória do arquivo. Após fecharmos o arquivo antigo, podemos associar o objeto à outro arquivo no mesmo programa. Dessa forma, um objeto ofstream (na verdade, qualquer um dos objetos derivados de fstream) pode trabalhar com múltiplos arquivos dentro de um mesmo programa, porém não simultaneamente (caso precisemos trabalhar com múltiplos arquivos simultaneamente, é preciso criar mais de um objeto fstream). A sintaxe da função close( ) é a seguinte:

<objeto>.close( );

Note que não é necessário especificar o arquivo à ser fechado, pois cada objeto só consegue estar conectado à um único arquivo por vez. No programa acima, por exemplo, utilizamos o seguinte comando para fechar o arquivo string.txt:

escreve.close( );

**10.6 – Lendo os dados de um arquivo**

Para ler os dados de um arquivo em um programa, é preciso antes de mais nada criar um objeto do tipo ifstream, capaz de ser o intermediário entre o arquivo à ser lido e o programa. A criação do objeto ifstream é muito semelhante à criação de um objeto ofstream: no exemplo de código abaixo, cria-se um objeto ifstream chamado “leitura” e conecta-se ele ao arquivo string.txt criado previamente.

ifstream leitura;  
leitura.open(“string.txt”);

Após criarmos o objeto e conectarmos ele à um arquivo, podemos começar a ler através deste arquivo. A leitura de dados de um arquivo é parecida com a leitura de dados enviados através do teclado: criamos uma variável para armazenar os dados recebidos, e utilizamos o objeto ifstream para ler e enviar os dados do arquivo para o programa (no caso da leitura via teclado, o objeto cin receberia os dados enviados pelo usuário e os repassaria para o programa). Assim, podemos utilizar qualquer um dos métodos de leitura que estudamos para cin, seja para ler caracteres, palavras ou frases inteiras.

Para lermos somente um caractere de um arquivo, por exemplo, nossa melhor opção é utilizar o método .get, que serve justamente para esta função. Criamos uma variável do tipo char para armazenar este caractere, e utilizamos o objeto ifstream da forma mostrada no exemplo abaixo:

char armazena;  
leitura.get( armazena);

O exemplo de código acima utiliza o objeto “leitura” e o método .get para ler um caractere de um arquivo, e armazena este caractere na variável “armazena”. Poderíamos criar um laço de repetição para obter todos os caracteres do arquivo, como vemos no exemplo abaixo. O laço while continuará a executar o método .get para ler o arquivo, até que seja encontrado o fim do arquivo. Quando o fim do arquivo é encontrado, o comando “leitura.get(armazena) retorna “falso” para o laço while, terminando a repetição. Lembre-se que o arquivo strings.txt deve estar no mesmo diretório onde estamos executando o programa, caso contrário erros ocorrerão!

#include <iostream>  
#include <fstream>  
#include <string>  
using namespace std;  
  
int main(){  
  
char letra;  
ifstream leitura;  
leitura.open("strings.txt");  
  
if(!leitura.is\_open( ))   
{  
cout<<"Não foi possível abrir arquivo! Programa será terminado!\n";  
leitura.clear( ); //reseta o objeto leitura, para limpar memória do sistema}  
}  
  
while (leitura.get(letra)) {cout << letra;}  
leitura.close();  
system("PAUSE");  
return 0;  
}

Para lermos uma palavra inteira de um arquivo, ao invés de utilizarmos uma simples variável de tipo char, utilizamos uma matriz do tipo char. Da mesma maneira que o comando cin, o objeto lerá todos os caracteres em seu caminho, até que a matriz atinja seu tamanho máximo especificado OU encontre um espaço em branco, uma quebra de linha ou o fim do arquivo.

char matriz\_chars[80];  
leitura >> matriz\_chars;

Finalmente, para lermos uma linha inteira de um arquivo, utilizamos o método .getline, que já havíamos estudado no módulo 5. O método getline lê uma linha inteira de entrada, até que o tamanho máximo especificado seja atinjido ou até encontrar uma quebra de linha ou o fim de arquivo. Relembrando sua sintaxe:

<nome do objeto>.getline ( <matriz\_destino>, <limite de caracteres>);

Por exemplo:

char matriz\_chars[80];  
leitura.getline( matriz\_chars, 80);

O comando acima leria todos os caracteres de uma linha, até encontrar a quebra da linha. Como vimos com o método .get, podemos utilizar um laço de repetição para lermos todas as linhas de um arquivo utilizando o método getline. No exemplo abaixo, à cada repetição uma linha do arquivo será escrita na tela.

#include <iostream>  
#include <fstream>  
#include <string>  
using namespace std;  
  
int main(){  
  
char depot[300];  
ifstream leitura;  
leitura.open("strings.txt");  
  
if(!leitura.is\_open( ))  
{  
cout<<"Não foi possível abrir arquivo! Programa será terminado!\n";  
leitura.clear( ); //reseta o objeto leitura, para limpar memória do sistema}  
}  
  
while (leitura.getline(depot, 300)) {cout << depot <<"\n";}  
leitura.close();  
system("PAUSE");  
return 0;  
}

Uma pequena diferença que vemos entre esses dois métodos de ler um arquivo inteiro, é que o método .get lê todos os caracteres até o fim de arquivo, incluindo quebras de linha. O método getline utiliza as quebras de linha para determinar o fim de sua leitura, e descarta elas. Assim, no exemplo logo acima, tivemos que instruir o programa à adicionar uma quebra de linha após exibir na tela cada uma das linhas lidas. (Entretanto, lembre-se que para arquivos grandes, o método getline é mais eficiente e mais rápido. Tenha em mente que cada acesso de um objeto fstream à um arquivo demanda um certo processamento: em um arquivo grande, se temos centenas de linhas, então temos milhares de caracteres, o que significa que o método get acessará muito mais vezes o mesmo arquivo do que o método getline).

**10.7 – Modos de Arquivo**

Quando abrimos um arquivo, podemos querer fazer diferentes coisas com o conteúdo deste arquivo. Podemos querer continuar à escrever no fim do arquivo, dando continuidade aos dados já escritos nele. Podemos querer apagar todos os dados já escritos e começar do zero novamente. Os modos de arquivo servem justamente para isto: para definir como deve ser o comportamento de um arquivo quando acessado pelo programa. Podemos definir que o arquivo será utilizado somente para leitura ou somente escrita, se será utilizado para anexar dados ou se será reescrito totalmente. Definimos o modo do arquivo quando abrimos ele com um objeto, adicionando um parâmetro novo ao método open. Veja a sintaxe abaixo:

<objeto ofstream>.open(“nome do arquivo”, ofstream::<modo de arquivo>);

ou

<objeto ifstream>.open(“nome do arquivo”, ofstream::<modo de arquivo>);

A tabela abaixo resume os modos de arquivo disponíveis em C++.

ios\_base::in Abre arquivo para leitura.  
ios\_base::out Abre arquivo para escrita.  
ios\_base::ate Procura o final do arquivo ao abrir ele.  
ios\_base::app Anexa os dados à serem escritos ao final do arquivo.  
ios\_base::trunc Trunca os dados existentes no arquivo.  
ios\_base::binary Abre e trabalha com arquivos em modo binário.

Note que um arquivo aberto por um objeto ofstream não necessita que definamos o modo de arquivo ofstream::out, pois este modo já é definido para este tipo de objeto por definição. O mesmo ocorre com o modo ifstream::in e os objetos ifstream.

Além disso, por definição, um arquivo aberto por um objeto ofstream irá truncar os dados já existentes no arquivo, escrevendo os dados novos “por cima” dos antigos. Porém, o modo ofstream::app nos permite anexar dados ao final de um arquivo. O exemplo abaixo utliza o modo ofstream::app para criar uma espécie de “agenda” simplificada em um arquivo de texto.

#include <iostream>  
#include <fstream>  
#include <string>  
using namespace std;  
  
int main(){  
int dia, mes;  
char letra[1000];  
cout<<"PROGRAMA AGENDA Versão 0.00042\n";  
  
ofstream agenda;  
agenda.open("agenda.txt", ofstream::app);  
cout<<"Digite o compromisso no espaço abaixo (): \n";  
cin.getline(letra, 1000);   
cout << "Digite o dia do compromisso:\n";  
cin >> dia;  
cout << "Digite o mês do compromisso:\n";  
cin >> mes;  
  
agenda << "Compromisso marcado para dia "<<dia<<" de "<<mes<<": ";  
agenda << letra;  
agenda<<"\n";  
cout<<"Obrigado! Sua agenda foi atualizada com sucesso!";  
agenda.close();  
system("PAUSE");  
return 0;  
}