Orientação a Objetos
-
Programação em C++

Arquivos Binários

Prof. Dr. Jean Marcelo SIMÃO – DAINF / UTFPR - Ctba

Monitor: Vitor C. M. Corrêa – discente (2018) de Engenharia de Computação DAINF / DAELN

Arquivos Binários

Para ler e escrever arquivos binários
utiliza-se as classes:

ifstream – para ler
ofstream – para escrever
fstream – para ler e escrever

Elas estão presentes no arquivo <fstream>,
no *namespace "std"*

Para abrir um arquivo utiliza-se a função “open”

```
ofstream arquivo;

arquivo.open (“meu_arquivo.bin”, ios::binary | ios::out );
```

Pode-se também utilizar a construtora com parâmetros

```
ofstream arquivo (“meu_arquivo.bin”, ios::binary | ios::out );
```

Após o seu uso, pode-se utilizar a função “close” para fechar o arquivo aberto

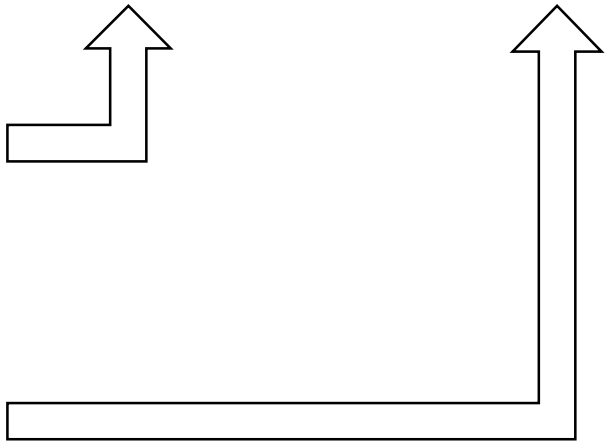
```
arquivo.close ( );
```

Não obstante, de qualquer forma, o arquivo será fechado automaticamente pelo destrutor (~ofstream) ao sair de seu escopo.

Abrindo um arquivo

```
ofstream arquivo;

arquivo.open ( "meu_arquivo.bin", ios::binary | ios::out );
```



1º parâmetro:
Caminho relativo (ao executável) e nome do arquivo a ser salvo

2º parâmetro:
Flags que definem os modos de operação do arquivo

Flags que podem ser utilizadas:

out = arquivo de saída (ofstream) **in** = arquivo de entrada (ifstream)	**binary** = arquivo binário **trunc** = discarta o conteúdo antigo do arquivo e substitui pelo novo

Obs: para incluir mais de uma flag deve-se utilizar o operador *bitwise OR* (|)

Para escrever no arquivo utiliza-se a função “write” da classe ofstream

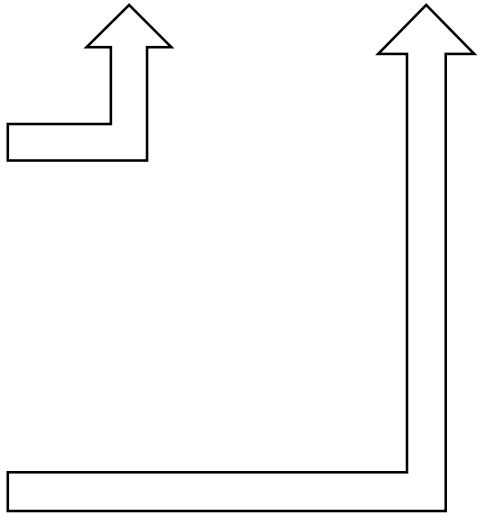
```
ofstream arquivo( “meu_arquivo.bin”, ios::binary | ios::out );

char caractere = ‘a’;

arquivo.write( (char*) &caractere, sizeof ( caractere ) );
```

1º parâmetro:
Endereço da variável a ser salva, que deve ser convertida para o tipo char* por meio de um *cast*

2º parâmetro:
Quantidade de bytes que devem ser utilizados para salvar a variável. para isso é recomendado o uso do operador *sizeof*



Para ler um arquivo utiliza-se a função “read” da classe ifstream

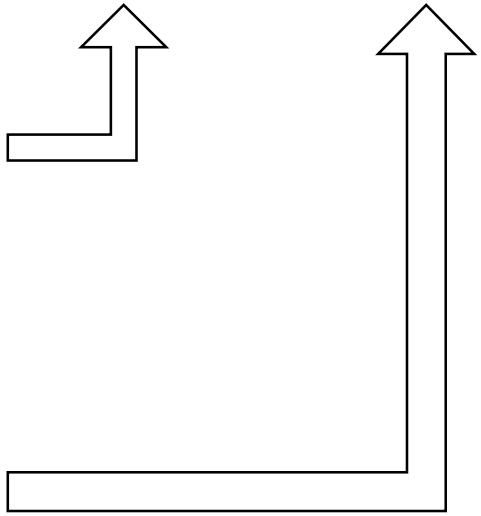
```
ifstream arquivo( “meu_arquivo.bin”, ios::binary | ios::in );

char caractere;

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

1º parâmetro:
Endereço da variável em que o conteúdo lido será salvo, que deve ser convertida para o tipo char* por meio de um *cast*

2º parâmetro:
Quantidade de bytes que serão extraídos do arquivo e salvos na variável. Para isso é recomendado o uso do operador *sizeof.*



Um exemplo simples

Essa função, se executada, irá ler 3 entradas e gerar um arquivo com nome “meu_arquivo.bin” no mesmo diretório do executável compilado, com 9 bytes, contendo-as (sendo char=1byte, int=4bytes e float=4bytes).

```
void gravar()
{
	cout << "digite um charactere" << endl;
	char var1;
	cin >> var1;

	cout << "digite um numero inteiro" << endl;
	int var2;
	cin >> var2;

	cout << "digite um numero real" << endl;
	float var3;
	cin >> var3;

	std::ofstream arquivo("meu_arquivo.bin", std::ios::binary | std::ios::out);

	arquivo.write( (char*) &var1, sizeof( var1 ) );

	arquivo.write( (char*) &var2, sizeof( var2 ) );

	arquivo.write( (char*) &var3, sizeof( var3 ) );
}
```

Um exemplo simples

Para a leitura do conteúdo do arquivo, devemos extrair as informações na mesma ordem em que ela foi gravada.

```
void ler()
{

	char var1;
	int var2;
	float var3;

	std::ifstream arquivo ( "meu_arquivo.bin", std::ios::binary | std::ios::in );

	arquivo.read( (char*) &var1, sizeof( var1 ) );
	arquivo.read( (char*) &var2, sizeof( var2 ) );
	arquivo.read( (char*) &var3, sizeof( var3 ) );

	cout << "1a variavel salva: " << var1 << endl;

	cout << "2a variavel salva: " << var2 << endl;

	cout << "3a variavel salva: " << var3 << endl;
}
```

Atenção

Deve-se tomar cuidado também com ponteiros e referências, que serão corrompidos pelo processo de gravação/leitura

Além disse o processo de armazenamento de objetos mais complexos é um pouco diferente

Para armazenar instâncias de tipos não-primitivos deve-se fazer antes um processo chamado de serialização, isto é, a transformação de um objeto ou uma estrutura de dados em um formato armazenável

Um meio simples de fazer isso é criar dois métodos dentro da classe que terão essa função

Também devemos armazenar o tamanho de uma estrutura com tamanho variável (ex. String)

Um outro exemplo

gravação de não-primitivos
e serialização

obs: antes estudar grupo de slides sobre *string*

Classe *String*

```
#include <iostream>
#include <string>
using namespace std;

int _tmain ( int argc, _TCHAR* argv[] )
{
    string s1 ( “bom dia! " ), s2;

    s2 = s1;                                // Atribui s1 a s2 com =

    cout << "S1: " << s1 << endl;
    cout << "S2: " << s2 << endl;
    cout << endl;

     s2[ 0 ] =  'B';                        // modifica S2 e S3.

    cout << "S1: " << s1 << endl;
    cout << "S2: " << s2 << endl;
    cout << endl;

    int tam = s2.length();

    for (int x = 0; x < tam; ++x)
    {
        cout << s2 [ x ];                   // demonstrando o operador de colchetes
    }
    cout << endl;
}
```

Função para facilitar a gravação de strings em arquivo

```
void grava_string ( string str, ofstream& arquivo)
{

        int tamanho = str.size ( );

        arquivo.write ( (char*) &tamanho, sizeof( tamanho ) );

        arquivo.write ( (char*) &str[0], tamanho );

}
```

Função para facilitar a leitura de strings de um arquivo

```
string le_string ( ifstream& arquivo)
{

        string str;

        int tamanho;

        arquivo.read ( (char*) &tamanho, sizeof( tamanho ));

        str.resize( tamanho );

        arquivo.read ( (char*) &str[0], tamanho );

        return str;
}
```

Outro exemplo ainda

gravação de não-primitivos
e serialização

Classe Principal

```
#pragma once
#include “Pessoa.h“
#include <iostream>
#include <string>
#include <fstream>
using namespace std;

class Principal
{
private:
        int tamanho_grupo;

        // vetor alocado dinamicamente
        Pessoa* grupo;

public:
        Principal();
        ~Principal();

        void executar();
        void cadastrarPessoas();
        void mostrarPessoas();

        void salvarGrupo();
        void carregarGrupo();
};
```

```
Principal::Principal()
{
   tamanho_grupo = 0;
   grupo = NULL;
}

Principal::~Principal()
{
   if ( grupo )
      delete[] grupo;
}

void Principal::executar()
{
        // Cadastra todas as pessoas no vetor grupo
        cadastrarPessoas();

        // Salva o grupo cadastrado em um arquivo
        salvarGrupo();

        // Apaga o grupo
        delete[] grupo; grupo = NULL;

        // Carrega o grupo do arquivo salvo
        carregarGrupo();
}
```

- Para manter a simplicidade do exemplo, o grupo será salvo em arquivo, apagado e depois recuperado do arquivo na mesma função executar.
- Na prática, a opção de salvar ou carregar deveria ser dada ao usuário, assim como a opção de salvar/carregar múltiplos arquivos.

```
void Principal::cadastrarPessoas()
{
    cout << "digite o tamanho do grupo: ";

    cin >> tamanho_grupo;

     grupo = new Pessoa [tamanho_grupo];

    // Cadastra as pessoas do grupo

    for ( int i = 0; i < tamanho_grupo; i++)
    {
        system ("cls");

        cout << "Digite o nome da " << i+1 << "a pessoa: " << endl;
        string c_nome;
        cin >> c_nome;
        grupo[i].setNome(c_nome);

        cout << "Digite a idade da " << i+1 << "a pessoa: " << endl;
        int c_idade;
        cin >> c_idade;
        grupo[i].setIdade(c_idade);

        cout << "Digite o CPF da " << i+1 << "a pessoa: " << endl;
        int c_CPF;
        cin >> c_CPF;
        grupo[i].setCPF(c_CPF);
    }

}
```

Função para salvar um grupo de pessoas em um arquivo

```
void Principal::salvarGrupo()
{

        // Abre o arquivo
        ofstream arquivo;

        arquivo.open("grupo1.dat", ios::binary |  ios::out);

        // Escreve o tamanho do grupo
        arquivo.write( (char* ) &tamanho_grupo, sizeof ( tamanho_grupo ) );

        // Escreve todos os objetos Pessoa
        for ( int i = 0; i < tamanho_grupo; i++ )
        {
                grupo[i].gravar ( arquivo );
        }

}
```

Função para carregar um grupo de pessoas de um arquivo

```
void Principal::carregarGrupo()
{

        ifstream arquivo;

        // Abre o arquivo
        arquivo.open ( "grupo1.dat", ios::binary | ios::in );

        // Lê o tamanho do grupo
        arquivo.read ( (char*)&tamanho_grupo, sizeof( tamanho_grupo ) );

        // Aloca espaço e lê as informações do arquivo
        grupo = new Pessoa [ tamanho_grupo ];

        for ( int i = 0; i < tamanho_grupo; i++ )
        {
                grupo[i].carregar(arquivo);
        }

        mostrarPessoas();

}
```

Função para mostrar um grupo de pessoas de um arquivo

```
void Principal::mostrarPessoas ( )
{

      // Mostra as informações lidas

     system ( "cls" );

     if ( grupo )
     {

         for ( int i = 0; i < tamanho_grupo; i++ )
         {

              cout  << "Pessoa n" << i + 1
                    << " nome: " << grupo[i].getNome()
                    << " idade: "  << grupo[i].getIdade()
                    << " CPF: "   << grupo[i].getCPF()
                    << endl;

          }

     }

   system("pause");

}
```

Serialização simples da classe Pessoa

Pessoa.h

```
#pragma once
#include <iostream>
#include <string>
#include <fstream>
using namespace std;

class Pessoa
{
private:
   string nome;
   int idade;
   int CPF;

public:
   Pessoa ( string c_nome = "",
            int c_idade = -1,
            int c_CPF = -1 );

   ~Pessoa ( ) ;

   void gravar (ofstream& arq);

   void ler (ifstream& arq);

   //métodos Setters e Getters
   . . .

};
```

Pessoa.cpp

```
#include "Pessoa.h"

void grava_string ( string str, ofstream& arquivo ) { . . . }

string le_string ( ifstream& arquivo ) { . . . }

Pessoa::Pessoa ( string c_nome, int c_idade, int c_CPF )
{
   nome = c_nome;     idade = c_idade;       CPF = c_CPF;
}

Pessoa::~Pessoa ( ) { }

void Pessoa::gravar (ofstream& arquivo)
{
        grava_string ( nome, arquivo );

        arquivo.write ( (char*) &idade, sizeof( idade ) ) ;
        arquivo.write ( (char*) &CPF, sizeof( CPF )    );
}

void Pessoa::carregar (ifstream& arquivo)
{
        nome = le_string ( arquivo );

        arquivo.read( (char*) &idade, sizeof( idade ) );
        arquivo.read( (char*) &CPF, sizeof( CPF ) );
}
```