

TAD0102

Algoritmos e programação

**Prof. Dr. Josenalde Barbosa de
Oliveira**

josenalde@eaj.ufrn.br

Aulas: 24T345, 6 CRDS – 90h

Manipulação de arquivos

PERSISTÊNCIA

Adicionar camada de persistência

- Base comum às linguagens de programação
- Permite manipular e interagir com o sistema de arquivos do SO
- Operações comuns: leitura, escrita (com sobrescrita ou adição)

```
#include <fstream> // Já inclui <iostream>
using namespace std;
```

```
int main() {
    ofstream fout; // cria objeto fout para escrita em arquivos
    fout.open("file1.txt");
    // ou num único comando: ofstream fout("file1.txt")
    // caso o arquivo exista, é substituído, se não existe é criado
    fout << "gravando no arquivo" << endl << "noutra linha";
    fout.close(); // encerra conexão ao arquivo
}
```




Usando normalmente como um cout

Adicionar camada de persistência

- Modos de abertura:
 - ios_base::in - LEITURA
 - ios_base::out - ESCRITA
 - ios_base::app – ESCRIVE NO FIM
 - ios_base::binary – BINÁRIO (não ascii)

```
#include <fstream> // Já inclui <iostream>
using namespace std;
```

```
int main() {
    ofstream fout; // cria objeto fout para escrita em arquivos
    fout.open("file1.txt", ios_base::app);
}
```



Caso arquivo não exista, também cria com o ::out

Leitura

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

int main() {
    ifstream fin;
    fin.open("file2.txt"); // ifstream fin("file1.txt")
    char ch;
    fin >> ch; // le um caracter do arquivo
    cout << ch << endl;
    char buffer[80];
    fin >> buffer; // le uma palavra (sem espaços)
    cout << buffer;
    fin.getline(buffer, 80); // le uma linha do arquivo
    cout << buffer;
    string linha;
    getline(fin, linha);
    cout << linha;
}
```

Escrita e leitura caractere a caractere

```
string arquivo;
cout << "Qual o nome do arquivo: ";
cin >> arquivo;
ofstream fout(arquivo.c_str());
// converte de string para char * (C)
fout << "Gravando esta frase no arquivo: " << endl;
cout << "Digite o valor (escrito na tela) ";
float n;
cin >> n;
fout << "O valor digitado na tela e gravado no arquivo: " << n
<< endl;
fout.close();

// leitura
ifstream fin(arquivo.c_str());
cout << "Conteudo do arquivo " << arquivo << ": " << endl;
char ch;
while (fin.get(ch)) cout << ch; // lê caractere a caractere
cout << "Feito " << endl;
fin.close();
```

Teste de leitura

```
fin.open(argv[1]);  
if (fin.fail()) { .... }  
if (!fin) { ... }  
if (!fin.is_open()) { .... }
```

Gravando arquivos binários

```
#define LIM 20
struct Cidade {
    char nome[LIM]; // para gravar binario, não funciona com string
    unsigned int pop;
};

int main() {
    Cidade c1 = {"natal", 884100};
    ofstream fout;
    fout.open("file5.dat", ios_base::out | ios_base::binary);
    // pode ser ios ao invés de ios_base
    fout.write((char *) &c1, sizeof c1);
}
```

grava toda a estrutura como uma unidade, não pode ser lido como texto . Ocupa menos espaço em disco.

write() copia determinado numero de bytes da memoria para o arquivo sem conversões. O endereço é o início do bloco de bytes, mas precisa ser feito o `typecast para ponteiro para char`

Leitura de arquivos binários (file6.cpp)

```
struct Cidade {
    char nome[LIM];
    unsigned int pop;
};

const char *arquivo = "file5.dat";
int main() {
    Cidade c1, c2;
    ofstream fout;
    ifstream fin;
    fin.open(arquivo, ios_base::in | ios_base::binary);
    if (fin.is_open()) { // se arquivo abriu corretamente
        // le os blocos de bytes para cada registro na estrutura
        while (fin.read((char *) &c2, sizeof c2)) {
            cout << setw(12) << "Nome: " << c2.nome << endl;
            cout << setw(12) << "Pop: " << c2.pop << endl;
        }
        fin.close();
    }
}
```

```
// adiciona novo conteúdo no mesmo arquivo
fout.open(arquivo, ios_base::out | ios_base::app | ios_base::binary);
if (!fout.is_open()) {
    cerr << "Erro na abertura de " << arquivo << endl;
    exit(EXIT_FAILURE);
}
cout << "Cadastro: (tecle enter para finalizar)" << endl;
cout << "Nome: ";
cin.get(c1.nome, LIM);
while (c1.nome[0] != '\0') { // se der enter, é string vazia (SAI)
    cout << "Pop. :";
    cin >> c1.pop;
    fout.write((char *) &c1, sizeof c1);
    setbuf(stdin, NULL); // limpa buffer para evitar que \n seja lido pelo cin.get
    cout << "Nome: ";
    cin.get(c1.nome, LIM);
}
fout.close();
}
```


Localizar dados – alterar (modo in, out e binary)

O método **seekg** move o ponteiro de entrada (leitura) para uma localização especificada

O método **seekp** move o ponteiro de saída (escrita) para uma localização especificada

seekg(streamoff, ios_base::seekdir) ou **seekg**(streampos)

streamoff é a quantidade de bytes a ser lida e **seekdir** é o deslocamento a partir da posição indicada (ios_base::**beg** (início), **cur** (posição atual), **end** (final do arquivo));

streampos é a quantidade de bytes a ser lida a partir do início do arquivo

Exemplo	Obs
fin.seekg(30, ios_base::beg)	Lê 30 bytes a partir do início
fin.seekg(-1, ios_base::cur)	Volta 01 byte a partir da posição atual
fin.seekg(0, ios_base::end)	Vai para o fim do arquivo
fin.seekg(0)	Vai para o início do arquivo

O método **tellg** para fluxos entrada (leitura) retorna a localização em bytes desde o início

O método **tellp** para fluxo de saída (escrita)

Localizar dados – alterar (modo in, out e binary) file7

```
const char *arquivo = "file5.dat"; // global
int main() {
    Cidade c1, c2;
    fstream finout; // se aplica a fluxos de leitura e escrita
    finout.open(arquivo, ios_base::in | ios_base::out | ios_base::binary);
    int ct = 0; // numero do registro
    if (finout.is_open()) {
        finout.seekg(0); // vai para inicio do arquivo
        while (finout.read((char *) &c2, sizeof c2)) {
            cout << ct++ << ": " << setw(12) << "Nome: " << c2.nome << endl;
            cout << setw(12) << "Pop: " << c2.pop << endl;
        }
        if (finout.eof()) finout.clear();
        // desabilita set do bit EOF, que eh ativo no fim da leitura.
        // ao usar clear() libera o EOF, para que o arquivo possa continuar
        // a ser manipulado
        else {
            cerr << "Erro na leitura " << endl;
            exit(EXIT_FAILURE);
        }
    } else {
        cerr << "Erro na abertura do arquivo" << endl;
        exit(EXIT_FAILURE);
    }
}
```

Localizar dados – alterar (modo in, out e binary) file7

```
cout << "Registro n. para alterar: ";
long rec;
cin >> rec;
setbuf(stdin, NULL); // limpa nova linha
if (rec < 0 or rec >= ct) {
    cerr << "Registro invalido -- fechando...";
    exit(EXIT_FAILURE);
}
streampos local = rec * sizeof c1;
finout.seekg(local); // pula para o local
finout.read((char *) &c1, sizeof c1);
cout << rec << ": " << setw(12) << "Nome: " << c1.nome << endl;
cout << setw(12) << "Pop: " << c1.pop << endl;
if (finout.eof()) finout.clear();
// permite alterar
cout << "Novo nome: ";
cin.get(c1.nome, LIM);
setbuf(stdin, NULL);
cout << "Nova Pop.: ";
cin >> c1.pop;
setbuf(stdin, NULL);
finout.seekp(local); // retorna ao registro localizado para alterar
finout.write((char *) &c1, sizeof c1) << flush; // garante a gravação
if (finout.fail()) {
    cerr << "Erro ao escrever no arquivo" << endl;
```

Exercícios

- 1) Com base nestes arquivos modelo, proponha uma estrutura de cadastro com membro ID, que funcione como uma espécie de chave (índice) para busca, ou uma busca textual num campo nome, por exemplo. Como isto poderia ser implementado para retornar todos os registros que atendam ao critério de busca?