# Manipulação de Arquivos

## C++

Delano M. Beder

DC - UFSCar

# Arquivos

- Os arquivos são utilizados para armazenamento de dados fora da memória principal do computador, por exemplo em discos
- Usamos arquivos quando:
  - Informações são muito numerosas para caber na memória principal
  - Necessidade de armazenamento permanente de informações

# Arquivos

- A palavra **stream** é usada para indicar **fluxo de bytes**. Assim, todo objeto que tem a capacidade de **receber** ou **transferir** bytes de ou para a memória do computador é chamado de **objeto stream**

- **cin** e **cout** são exemplos de **objetos stream**

# Arquivos

- C++ fornece três classes para lidar com arquivos:
  - **ofstream:** para escrever em arquivos ("**o**" = output)
  - **ifstream:** para ler de arquivos ("**i**" = input)
  - **fstream:** para ler e/ou escrever em arquivos
- Para usar essas classes precisamos incluir a biblioteca <**fstream**>

# Arquivo de Saída

- Declaração

```
ofstream arqOut;
```

- Abrir: Conectar a stream arqOut ao arquivo "*nomeArquivo.txt*"

```
arqOut.open(“nomeArquivo.txt”);
```

- Fechar: Desconectar a stream arqOut do arquivo "*nomeArquivo.txt*"

```
arqOut.close();
```

- **Operador de inserção** <<: Comportamento idêntico ao **cout**

```
arqOut << “Gravando no arquivo”;
```

# Arquivo de Entrada

- Declaração

```
ifstream arqIn;
```

- Abrir: Conectar a stream arqIn ao arquivo "*nomeArquivo.txt*"

```
arqIn.open("nomeArquivo.txt");
```

- Fechar: Desconectar a stream arqIn do arquivo "*nomeArquivo.txt*"

```
arqIn.close();
```

- **Operador de extração** >>: Comportamento idêntico ao **cin**

```
arqIn >> x;
```

# Outras funções de manipulação de arquivos

**OBSERVAÇÃO:** O operador de **extração de fluxo** (>>) **pula os caracteres de espaço em branco** como espaços, tabulações e nova linha no fluxo de entrada

# Arquivos

- **Exemplo 1:** Lê três números do arquivo "*numeros.dat*", soma os números e escreve o resultado no arquivo "*soma.dat*"

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

main(){

   //declaração dos streams de entrada e saída
   ifstream arq_entrada;
   ofstream arq_saida;

   //abertura dos arquivos
   arq_entrada.open(‘‘numeros.dat’’);
   arq_saida.open(‘‘soma.dat’’);

   ...
```

# Arquivos

```
    ...
    int num1, num2, num3, soma;

    //leitura dos três números
    arq_entrada >> num1 >> num2 >> num3;

    soma = num1 + num2 + num3;

    //escrita do resultado
    arq_saida << "A soma eh: " << soma << endl;

    //fecha os arquivos
    arq_entrada.close();
    arq_saida.close();

}
```

# Modos de abertura de arquivos

- Especificam **como** o arquivo deve ser aberto e **o que pode ser feito** com ele

| Modos | Descrição |
|---|---|
| ios::in | abre para leitura |
| ios::out | abre para gravação |
| ios::app | grava a partir do fim do arquivo |
| ios::trunc | abre e apaga todo o conteúdo do arquivo |
| ios::ate | abre e posiciona no final do arquivo |

# Modos de abertura de arquivos

- O método **open()** aceita a inclusão de um segundo argumento indicando o modo de abertura do arquivo. **Exemplos:**
- Declaração

```
fstream arq;
```

- Abrir arquivo só para leitura

```
arq.open(“nomeArquivo.txt”, ios::in);
```

- Abrir arquivo só para escrita

```
arq.open(“nomeArquivo.txt”, ios::out);
```

- Abrir arquivo para leitura e escrita

```
arq.open(“nomeArquivo.txt”, ios::in|ios::out);
```

# Modos de abertura de arquivos

- **ifstream** e **ofstream** possuem modos de abertura de arquivo **default**

# Modos de abertura de arquivos

- **ifstream** e **ofstream** possuem modos de abertura de arquivo **default**
- Portanto, o segundo parâmetro da função **open()** é opcional quando se utiliza **ifstream** e **ofstream**

# Modos *default*

- **ofstream:**
  - Abertura somente para escrita
  - Não são permitidas leituras
  - Se não existe, o arquivo é criado
  - Se o arquivo já existe, o seu conteúdo anterior é apagado

# Modos *default*

- **ifstream:**
  - Abertura somente para leitura
  - Não é permitido escrever no arquivo
  - A abertura falha caso o arquivo não exista

# Outras funções de manipulação de arquivos

- Os objetos de **leitura** ou **gravação** de arquivos possuem métodos para a **verificação de fim de arquivo**. Considere os objetos:

```
ifstream inArq;
ofstream outArq;
```

- Para verificar o fim de arquivo usamos a função **eof()**. Que retorna **true** se o marcador de arquivo se encontra no **fim** do arquivo e **false** caso contrário.

    - ```
      inArq.eof();
      outArq.eof();
      ```
      ```

# Outras funções de manipulação de arquivos

- Para verificar se um arquivo foi aberto com sucesso pode-se usar a função **is_open()** que retorna **true** se o arquivo encontra-se aberto e **false** caso contrário.

```
inArq.is_open();
outArq.is_open();
```

# Outras funções de manipulação de arquivos

- **Exemplo 2:** Ler todos os dados do arquivo "*dados.txt*" utilizando o operador de extração >> e imprima-os na tela separados por espaço

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

main(){

    ifstream arq_entrada;
    string palavra;

    //tentativa de abertura do arquivo a ser lido
    arq_entrada.open("dados.txt");

            ...
```

# Outras funções de manipulação de arquivos

```
    if(arq_entrada.is_open()){

        while(!arq_entrada.eof()){

            arq_entrada >> palavra;
            cout << palavra << " ";

        }

    }
    else
        cout<<"O arquivo nao pode ser aberto.";

    arq_entrada.close();
    system("pause");

}
```

# Outras funções de manipulação de arquivos

- Se o arquivo **dados.txt** não estiver no mesmo diretório do programa, por exemplo, a mensagem ***"O arquivo nao pode ser aberto."*** será exibida
  - ```
    if(arq_entrada.is_open()){
       ...
    }
    else
       cout<<"O arquivo nao pode ser aberto.";
    ```
- Caso o arquivo seja aberto ele será lido até o seu fim
  - ```
    while(!arq_entrada.eof()){

       arq_entrada >> palavra;
       cout << palavra << " ";

    }
    ```

# Lendo e gravando um caracter por vez no arquivo

- Para trabalharmos com um único caracter por vez, usamos os métodos **put()** (da classe **ostream**) e **get()** (da classe **istream**)
  - **inArq.get(carac):** Extrai o próximo caracter da stream inArq e coloca na variável **char** carac;
  - **outArq.put(carac):** Insere o caracter carac na stream de saída outArq.

# Lendo e gravando um caracter por vez no arquivo

```
#include <fstream>
using namespace std;
int main() {

  // Abertura arquivos
  ifstream ifs("entrada.txt");
  ofstream ofs("saida.txt");

  char carac;
  ifs.get(carac); // Leitura de um caractere
  while (!ifs.eof()) {
    ofs.put(carac);  // Escrita de um caractere
    ifs.get(carac); // Leitura de um caractere
  }

  // Fecha arquivos
  ifs.close();
  ofs.close();

  return 0;
}
```

# Lendo e gravando uma linha por vez no arquivo

```
#include <fstream>
using namespace std;
int main() {

  // Abertura arquivos
  ifstream ifs("entrada.txt");
  ofstream ofs("saida.txt");

  string linha;
  // Leitura uma linha por vez
  while (getline(ifs, linha)) {
      ofs << linha << endl; // Escrita linha
  }

  // Fecha arquivos
  ofs.close();
  ifs.close();

  return 0;
}
```

# Manipulação de arquivos em **modo binário**

- Muitas vezes não desejamos gravar ou ler um caractere ou uma linha por vez, mas sim manipular uma quantidade maior de caracteres.
  **Exemplo:** Ler e gravar **objetos** em disco

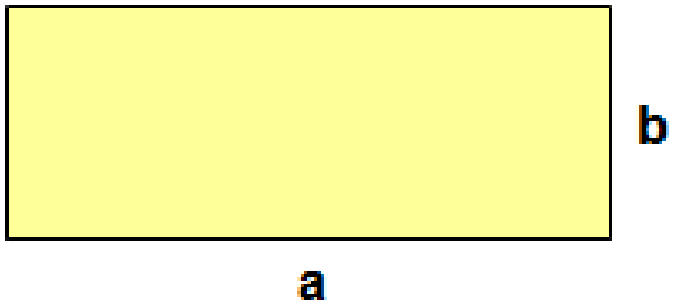
# Manipulação de arquivos em **modo binário**

- Muitas vezes não desejamos gravar ou ler um caractere ou uma linha por vez, mas sim manipular uma quantidade maior de caracteres.
  **Exemplo:** Ler e gravar **objetos** em disco

- Nestes casos as funções apropriadas para **gravação** e **leitura** são, respectivamente: **write()** e **read()**

# Manipulação de arquivos em **modo binário**

**Exemplo:** Implementar uma classe para representar um retângulo



# Manipulação de arquivos em **modo binário**

```
class Retangulo{
   private:
      double ladoa;
      double ladob;

   public:
      void setDim(double a, double b);
      void imprimir();
};
```

**Arquivo: Retangulo.h**

# Manipulação de arquivos em **modo binário**

```
#include “Retangulo.h”
#include <iostream>
using namespace std;

void Retangulo::setDim(double a, double b){
    ladoa = a;
    ladob = b;
}

void Retangulo::imprimir(){
    cout << “Retangulo com lados: ”
         << ladoa << “ e ” << ladob << endl
         << “Area: ” << area() << endl << endl;
}
```

**Arquivo: Retangulo.cpp**

# Manipulação de arquivos em **modo binário**

**Exemplo:** Lendo e gravando 3 objetos **Retangulo** de um mesmo arquivo (***info.dat***)

# Manipulação de arquivos em **modo binário**

```
#include "Retangulo.h"
#include <iostream>
#include <fstream>
using namespace std;

void main(){

    double x, y;
    Retangulo r;
    ofstream saida("info.dat", ios::binary);

    for(int i=0; i<3; i++){
        cout << "\nDigite base e altura: ";
        cin >> x >> y;
        r.setDim(x, y);

        saida.write(reinterpret_cast<char *>(&r), sizeof
            (Retangulo));
    }
```

# Manipulação de arquivos em **modo binário**

```
    saida.close(); //fecha arquivo onde foram gravados
        os objetos

    ifstream entrada("info.dat", ios::binary);

    cout<<"\n\nDADOS DO ARQUIVO: \n\n";
     for(int i=0; i<3; i++){
         entrada.read(reinterpret_cast<char *>(&r),
             sizeof(Retangulo));
         r.imprimir();
    }

    system("pause");
}
```

# Manipulação de arquivos em **modo binário**

Sobre as funções **write()** e **read()**:

- Trabalham em **modo binário**
- Transferem blocos de memória de/para o disco
- Recebem como parâmetros:
  1. Endereço de memória dos dados
  2. Seu tamanho em bytes

# Manipulação de Arquivos

## C++

Delano M. Beder

DC - UFSCar