

# Programação Orientada a Objetos em C++

Fluxos

**Agostinho Brito**

2020

1 Entrada e saída de dados

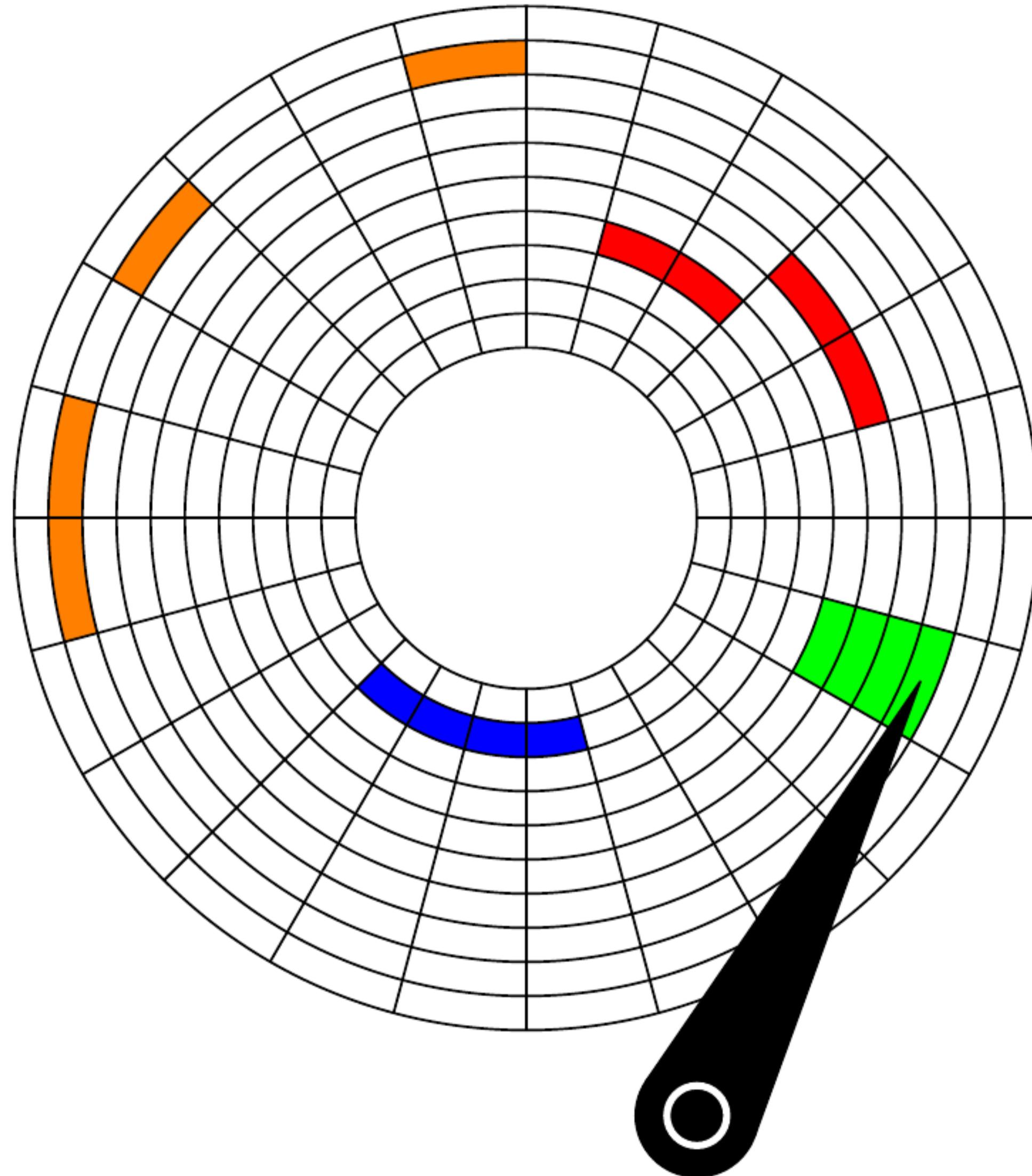
2 Aprimorando a leitura/escrita em fluxos - caracteres

3 Aprimorando a leitura/escrita em fluxos - literais

4 A classe `stringstream`

# Entrada e saída de dados

# Um disco rígido



- Um arquivo é uma seção de armazenamento que recebe uma denominação específica.
- Os arquivos mais comuns são os arquivos de bloco, que ficam armazenados em unidades físicas: discos rígidos, unidades de estado sólido, fitas, CDs, DVDs, Blu-Rays...
- Arquivos são armazenados em sequências de blocos (fragmentos) de tamanho fixo agrupando setores.

# O que são arquivos?

- Cada bloco pode conter informações adicionais para que o S.O. saiba como tratá-los.
- Parte do espaço em disco é usado para armazenar os dados úteis, parte é usada para preparar a infra-estrutura de indexação dos blocos.
- Num sistema Linux, os blocos possuem tamanho mínimo de 4096 bytes, e cada transferência disco ↔ memória totaliza 512 bytes, o tamanho de um setor.
- Agrupar setores (não necessariamente contíguos) em blocos diminui a demanda por endereços, permitindo discos maiores.

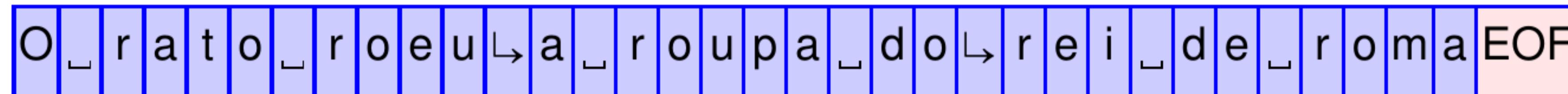
# O que são arquivos?

- Cada bloco pode conter informações adicionais para que o S.O. saiba como tratá-los.
- Parte do espaço em disco é usado para armazenar os dados úteis, parte é usada para preparar a infra-estrutura de indexação dos blocos.
- Num sistema Linux, os blocos possuem tamanho mínimo de 4096 bytes, e cada transferência disco ↔ memória totaliza 512 bytes, o tamanho de um setor.
- Agrupar setores (não necessariamente contíguos) em blocos diminui a demanda por endereços, permitindo discos maiores.

# A visão do programador

- Decidir o que fazer com tantos blocos (que podem ser descontínuos) é algo além da alcada do programador de aplicativos comuns.
- Normalmente, desenvolvedores do kernel é quem cuidam dessa tarefa.
- É papel do sistema operacional facilitar para usuários e programadores o acesso às estruturas de um arquivo.
- Para isso, o S.O. oferece visões mais simplificadas de um arquivo regular no computador.

O rato roeu  
a roupa do  
rei de roma



- O arquivo nada mais é que uma sequência linear de caracteres, onde o final é apenas uma condição existente na estrutura do sistema de arquivos.

# Modo binário e modo texto

- As bibliotecas do C/C++ padrão permitem que um arquivo seja visto de dois modos diferentes: binário e texto.
- Quando um arquivo é aberto no modo texto, algumas traduções são feitas para permitir interoperabilidade entre S.O.
- Por exemplo, no Linux, apenas o caractere `\n` (newline) é usado para marcar um fim de linha. No Windows, é usada a combinação `\r\n` (carriage return/newline).
- Quando o texto é aberto em modo **texto**, a sequência é lida apenas como `\n` (newline).
- Quando o texto é aberto em modo **binário**, a sequência é lida como `\r\n` (carriage return/newline).

## modo texto

Arquivos com texto pleno  
(Códigos fonte, Texto simples...)

## modo binário

Demais arquivos (Mídia, Texto codificado, como .doc ou .pdf)

# Modo binário e modo texto

- As bibliotecas do C/C++ padrão permitem que um arquivo seja visto de dois modos diferentes: binário e texto.
- Quando um arquivo é aberto no modo texto, algumas traduções são feitas para permitir interoperabilidade entre S.O.
- Por exemplo, no Linux, apenas o caractere `\n` (newline) é usado para marcar um fim de linha. No Windows, é usada a combinação `\r\n` (carriage return/newline).
- Quando o texto é aberto em modo **texto**, a sequência é lida apenas como `\n` (newline).
- Quando o texto é aberto em modo **binário**, a sequência é lida como `\r\n` (carriage return/newline).

## modo texto

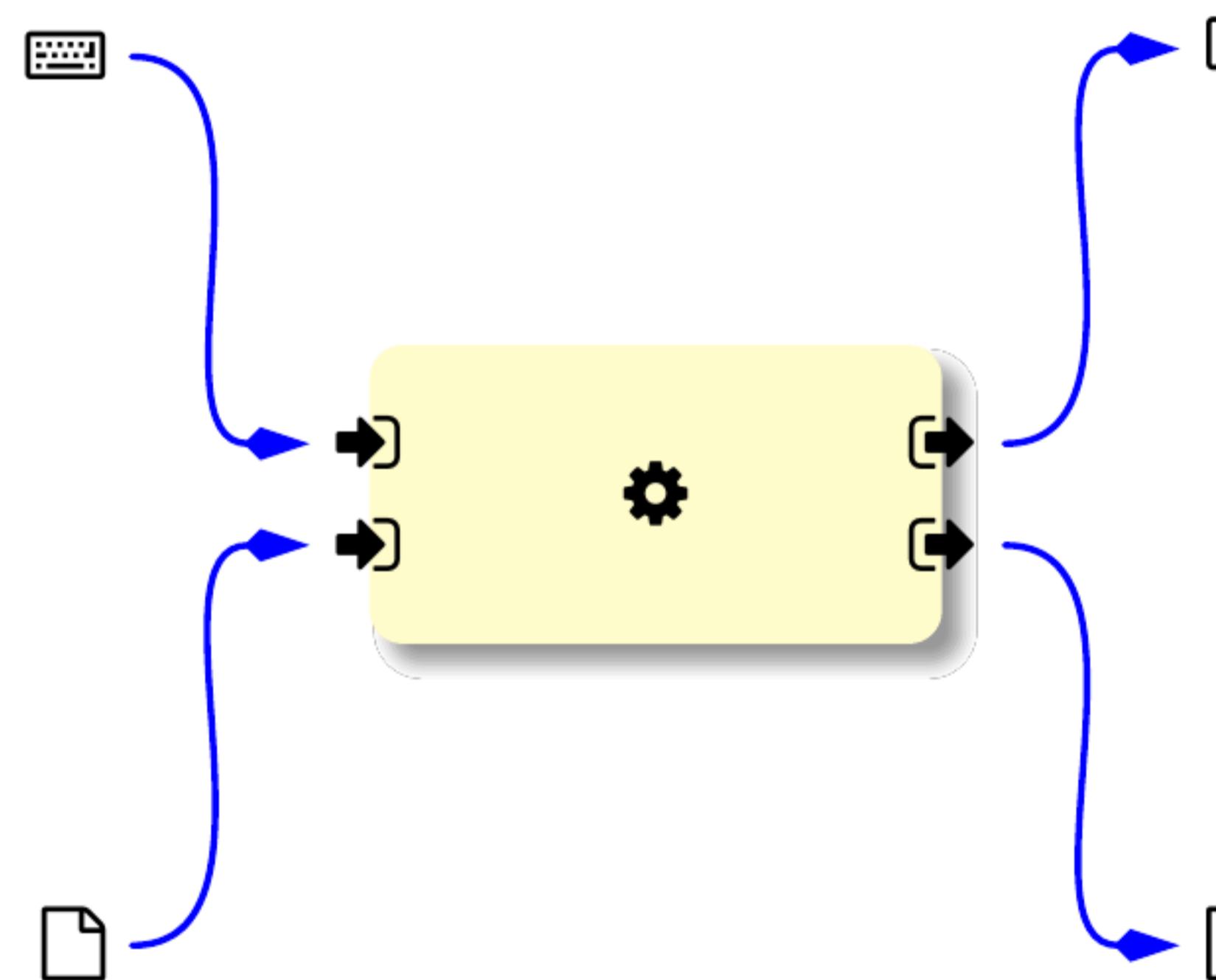
Arquivos com texto pleno  
(Códigos fonte, Texto simples...)

## modo binário

Demais arquivos (Mídia, Texto codificado, como .doc ou .pdf)

# Conceituando fluxo

- Em C++, um fluxo representa um dispositivo com o operações de entrada e saída (ambas em conjunto, inclusive) são realizadas.
- São associados com o destino físico de caracteres, como um disco, uma tela (console) ou o teclado, de/para onde os caracteres fluem.
- Os fluxos são abstrações criadas para facilitar essa manipulação.



# Classes de fluxos da biblioteca padrão

- C++ provê algumas classes para tratar fluxos de entrada e saída. As mais conhecidas são descritas no header `<iostream>`.
- São elas: `istream`, `ostream`, `iostream`, `ofstream`, `ifstream` e `stringstream`.
- Alguns objetos de algumas dessas classes são bem conhecidos:
  - `cin` entrada padrão (teclado).
  - `cout` saída padrão (terminal).
  - `cerr` saída padrão para erros.
  - `clog` saída padrão para registros.
- As interações com os fluxos de `entrada` são feitas com a sobrecarga do operador `>>`.
- As interações com os fluxos de `saída` são feitas com a sobrecarga do operador `<<`.

# Abrindo arquivos para escrita

- A classe `ofstream` é a responsável por manter uma conexão entre um *buffer* interno com o fluxo de dados e prover operações de I/O com esse buffer.
- Está definida no header `<fstream>`.
- Etapas do processo de escrita:
  - 1 Abrir um arquivo, associando-o com o fluxo.
  - 2 Enviar dados para o fluxo.
  - 3 Fechar o arquivo.
- Superada a etapa de abertura do arquivo, a interação com o fluxo se comporta de modo exatamente idêntico a `cin/cout`.

# Gravando o nome em um arquivo

```
1 #include <fstream>
2 #include <cstdlib>
3 int main(void) {
4     std::ofstream fout;
5     fout.open("nome.txt");
6     if (!fout.is_open()) {
7         exit(1);
8     }
9     fout << "Agostinho Brito \n";
10    fout.close();
11 }
```



# Gravando o nome em um arquivo

Para `exit()`

```
1 #include <fstream>
2 #include <cstdlib> ←
3 int main(void) {
4     std::ofstream fout;
5     fout.open("nome.txt");
6     if (!fout.is_open()) {
7         exit(1);
8     }
9     fout << "Agostinho Brito \n";
10    fout.close();
11 }
```



# Gravando o nome em um arquivo

Para `exit()`

O fluxo de saída

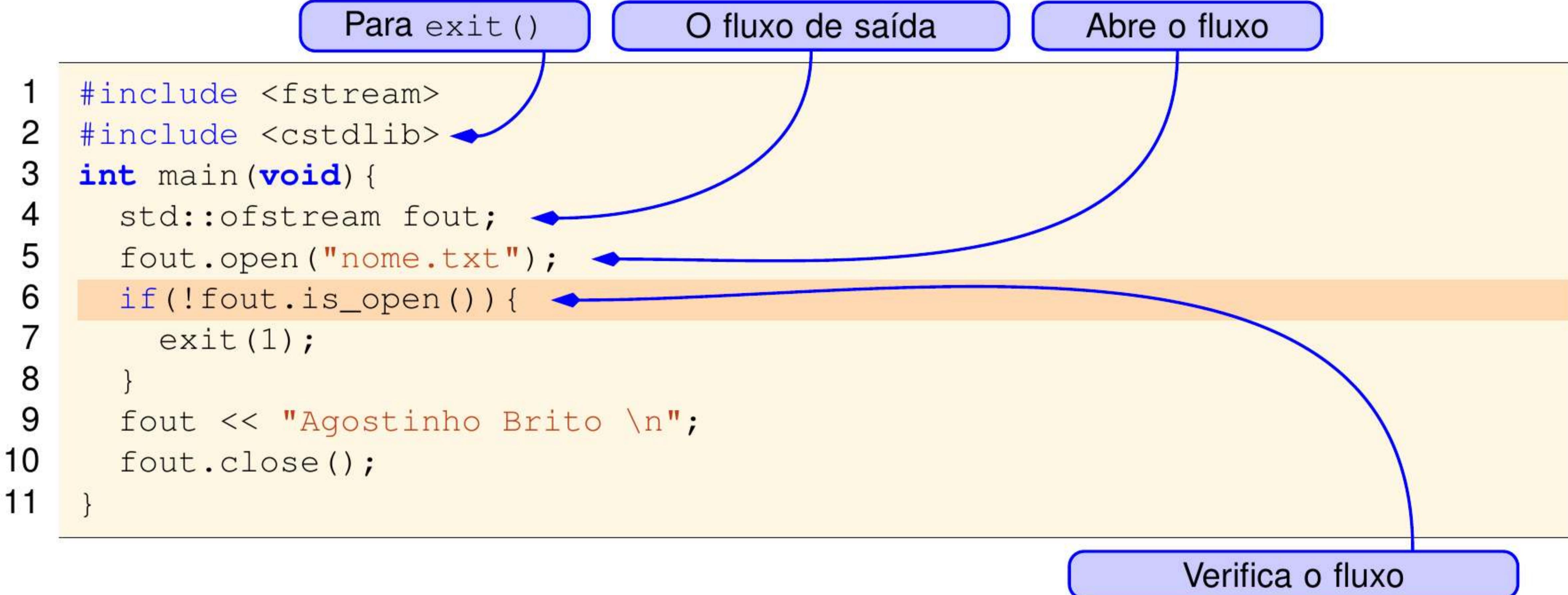
```
1 #include <fstream>
2 #include <cstdlib>
3 int main(void) {
4     std::ofstream fout; ←
5     fout.open("nome.txt");
6     if (!fout.is_open()) {
7         exit(1);
8     }
9     fout << "Agostinho Brito \n";
10    fout.close();
11 }
```

# Gravando o nome em um arquivo

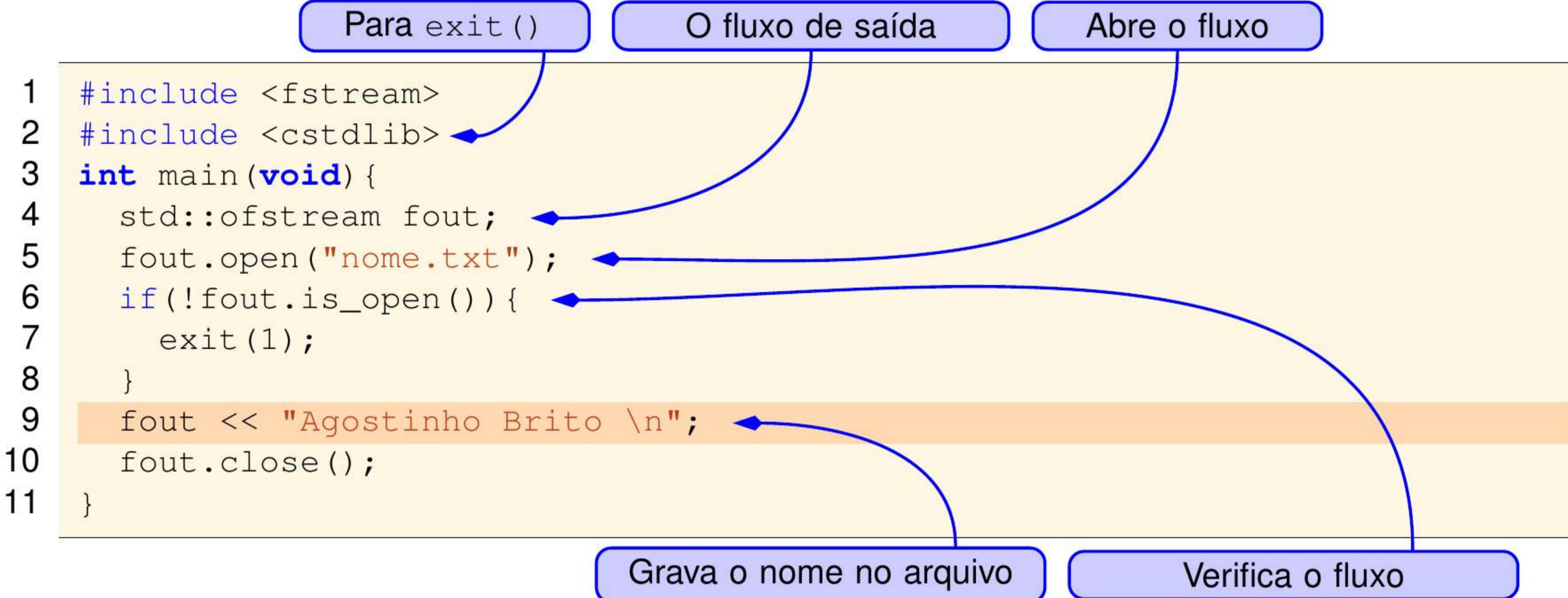
```
Para exit ()          O fluxo de saída          Abre o fluxo
1 #include <fstream>
2 #include <cstdlib>
3 int main(void) {
4     std::ofstream fout;
5     fout.open("nome.txt");
6     if(!fout.is_open()) {
7         exit(1);
8     }
9     fout << "Agostinho Brito \n";
10    fout.close();
11 }
```

The diagram illustrates the flow of control and data handling in the provided C++ program. It features three callout boxes at the top: 'Para exit ()' pointing to the final brace of the main function, 'O fluxo de saída' pointing to the output statement, and 'Abre o fluxo' pointing to the file opening operation. The code itself is annotated with numbers 1 through 11 on the left, corresponding to each line of the program.

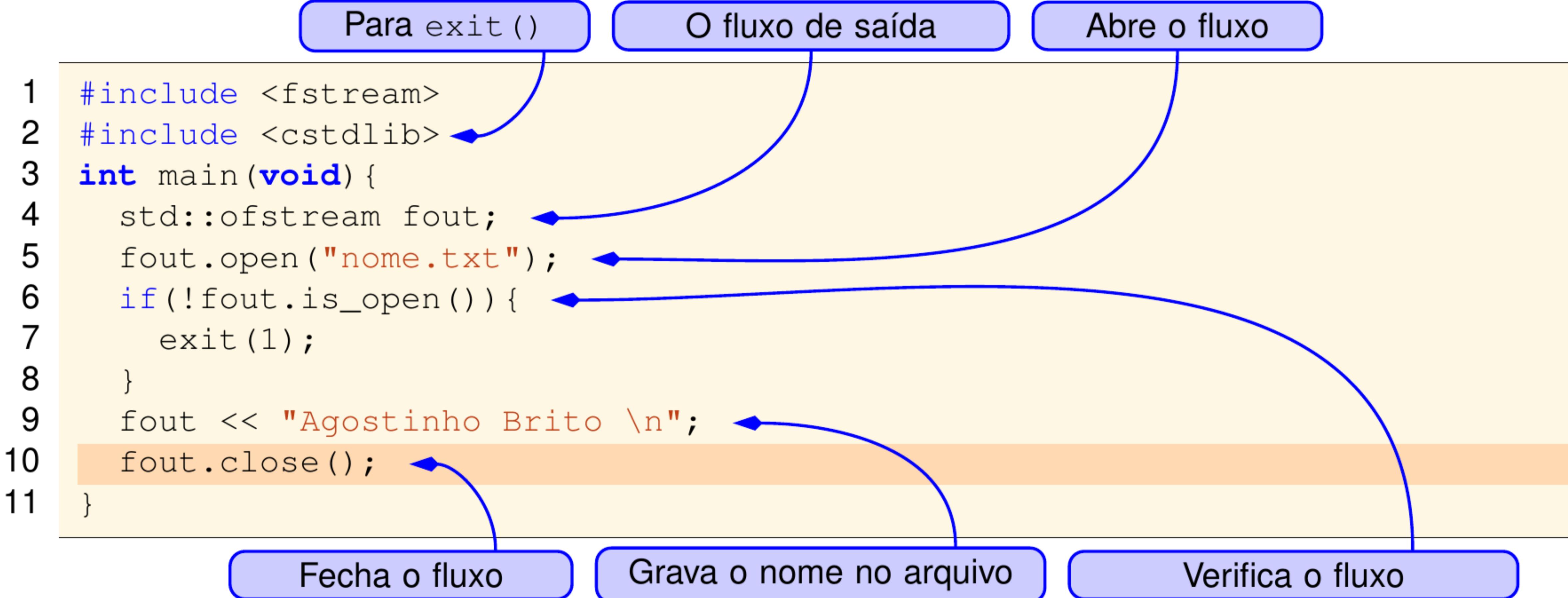
# Gravando o nome em um arquivo



# Gravando o nome em um arquivo



# Gravando o nome em um arquivo



# Lendo o nome de um arquivo

```
1 #include <fstream>
2 #include <string>
3 int main(void) {
4     std::ifstream fin;
5     std::string s;
6     fin.open("nome.txt");
7     if(!fin.is_open()) {
8         exit(1);
9     }
10    fin >> s;
11    std::cout << "leu: " << s << "\n";
12    fin.close();
13 }
```



# Lendo o nome de um arquivo

Para string

```
1 #include <fstream>
2 #include <string> ←
3 int main(void) {
4     std::ifstream fin;
5     std::string s;
6     fin.open("nome.txt");
7     if (!fin.is_open()) {
8         exit(1);
9     }
10    fin >> s;
11    std::cout << "leu: " << s << "\n";
12    fin.close();
13 }
```



# Lendo o nome de um arquivo

Para string

O fluxo de entrada

```
1 #include <fstream>
2 #include <string>
3 int main(void) {
4     std::ifstream fin; ← O fluxo de entrada
5     std::string s;
6     fin.open("nome.txt");
7     if(!fin.is_open()) {
8         exit(1);
9     }
10    fin >> s;
11    std::cout << "leu: " << s << "\n";
12    fin.close();
13 }
```

# Lendo o nome de um arquivo

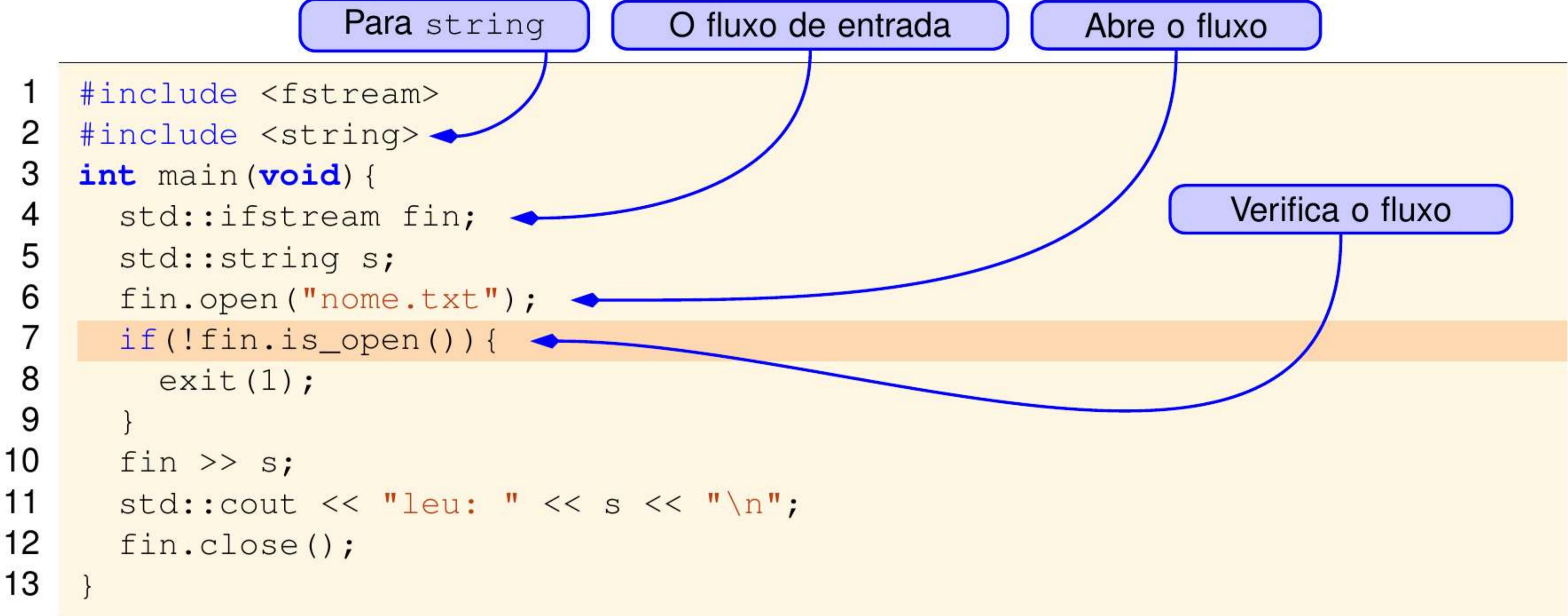
```
1 #include <iostream>
2 #include <string>
3 int main(void) {
4     std::ifstream fin;
5     std::string s;
6     fin.open("nome.txt");
7     if(!fin.is_open()) {
8         exit(1);
9     }
10    fin >> s;
11    std::cout << "leu: " << s << "\n";
12    fin.close();
13 }
```

Para string

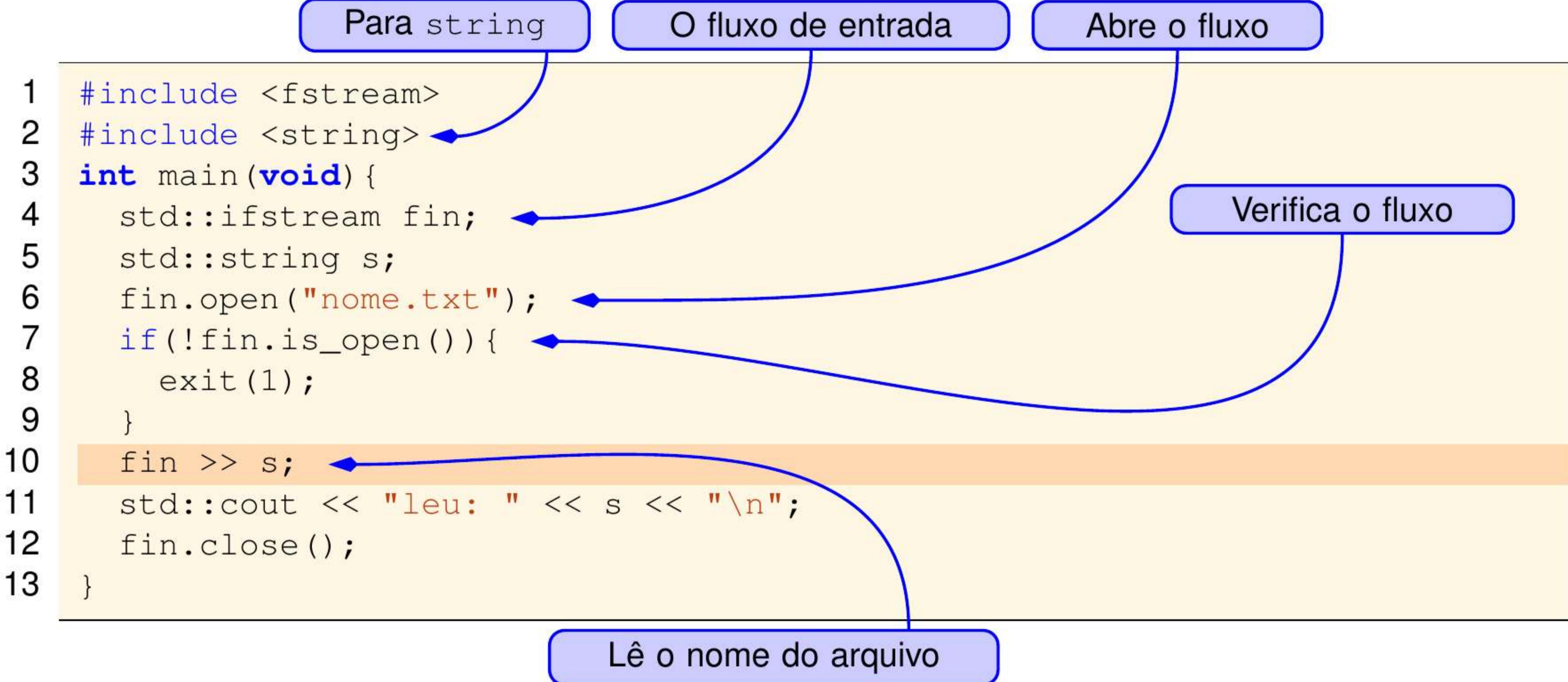
O fluxo de entrada

Abre o fluxo

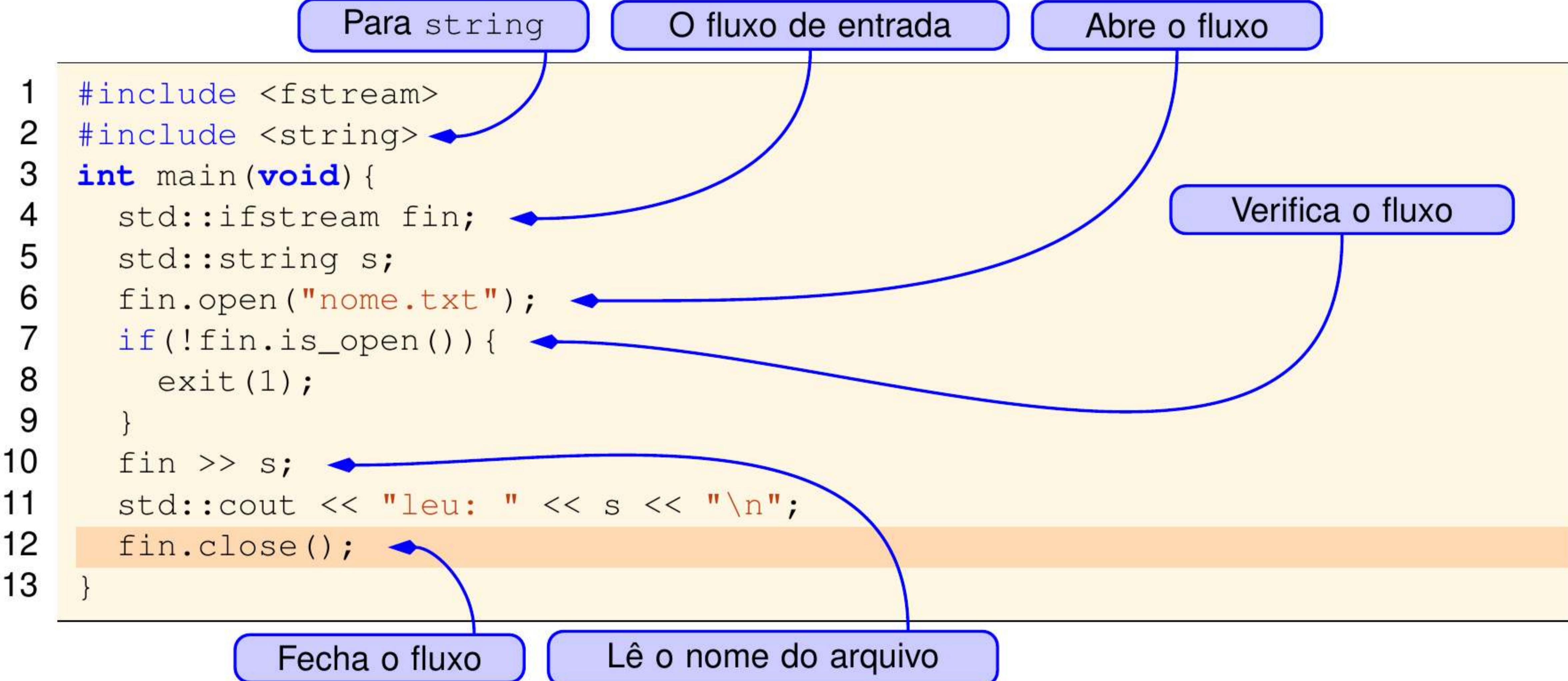
# Lendo o nome de um arquivo



# Lendo o nome de um arquivo



# Lendo o nome de um arquivo





Obrigado

## Aprimorando a leitura/escrita em fluxos - caracteres

- A leitura de caracteres de fluxos de entrada pode ser feita com sobrecargas do método `get()` para a classe `ifstream`.
- A escrita de caracteres em fluxos de saída pode ser feita com o método `put()` para a classe `ofstream`.

```
ifstream fin;
ofstream fout;
char ch;
ch = fin.get();
fout.put(ch);
```

# Fazendo a cópia de um arquivo...

```
1 #include <cstdlib>
2 #include <fstream>
3 #include <iostream>
4
5 int main(void) {
6     std::ifstream fin;
7     std::ofstream fout;
8     char ch;
9     fin.open("entrada.txt");
10    fout.open("saída.txt");
11    while (fin.get(ch)) {
12        fout.put(ch);
13    }
14    fin.close();
15    fout.close();
16 }
```



Obrigado

## Aprimorando a leitura/escrita em fluxos - literais

- Geralmente, os processos de leitura de literais são mais complexos que os de escrita, posto que normalmente não é possível precisar o que deveria ser lido. Por exemplo, a composição nome/sobrenome de um indivíduo.
- A leitura de literais (*strings*) pode ser feita também com o uso do método `get()` da classe `ifstream`.
- Entretanto, a classe `string` pode ser usada em conjunto com a classe `ifstream`, usando a função `std::getline()`.

```
ifstream fin;
char buffer[50];
string str;
fin.get(buffer, 50);
std::getline(fin, str);
std::getline(fin, str, ','');
```



## Praticando fluxos em C++...



Obrigado

## A classe `stringstream`

## A classe `stringstream` - situação-problema

- Suponha que se deseja criar uma ferramenta capaz de ler as linhas de um arquivo e realizar ações conforme os comandos presentes em cada linha.
- O formato de arquivo prevê o seguinte comportamento: o `string` do início da linha especifica o comando, seguido dos seus respectivos argumentos.
- Exemplo: guiar um veículo.

```
1 adiante 30
2 para
3 direita 45
4 atras 20
5 para
6 esquerda -45
```

- Como processar as linhas individualmente, decidindo quando ler ou não os possíveis argumentos que seguem os comandos?
- Solução: ler a linha inteira e processá-la argumento a argumento.
- A classe `stringstream` provê suporte para isso.

# Usando a classe `stringstream`

- A classe `stringstream` permite transformar um `string` em um fluxo de entrada ou transformar um fluxo de saída em um `string`.
- Os operadores `<<` e `>>` funcionam de forma semelhante aos fluxos de saída e entrada.
- Está definida em `<sstream>`.