Algoritmos e Programação II

https://evandro-crr.github.io/alg2

Arquivos

Um arquivo é uma **coleção de dados** armazenados. Ele pode conter diferentes tipos de informações, como texto, imagens, áudio ou dados binários. Os arquivos são organizados de maneira sequencial ou estruturada, permitindo que programas acessem, leiam e modifiquem seu conteúdo de forma persistente, mesmo após o encerramento da aplicação.

Manipulação de Arquivos em C++

A manipulação de arquivos envolve três passos principais:

Etapas Básicas

- Abrir o arquivo
  - Conecta o arquivo ao programa
- Operar com o arquivo
  - Leitura ou escrita de dados
- Fechar o arquivo
  - Garante a consistência dos dados

Formas de Uso

- Arquivo de entrada
  - Utilizado para ler dados de um arquivo.
- Arquivo de saída
  - Utilizado para escrever dados em um arquivo.

Nome do Arquivo

O nome de um arquivo é composto por **nome base** e **extensão**.

Nome Base

- É a parte que identifica o arquivo.
- Pode conter letras, números e alguns caracteres especiais, **mas evita-se o uso de espaços**.
- Exemplos:
  - `dados`
  - `relatorio_final`
  - `foto_de_gatinho`
  - `entrega_final`

Extensão

- Define o tipo de conteúdo do arquivo.
- Composta usualmente por três ou mais letras que vêm após o ponto.
- Exemplos:
  - `.txt` : arquivo de texto
  - `.cpp` : código-fonte C++
  - `.csv` : valores separados por vírgula

Exemplo de Nome de Arquivo

- Nome do arquivo: `relatorio_financeiro.csv`
  - Nome Base: `relatorio_financeiro`
  - Extensão: `.csv` (arquivo de dados separados por vírgula)
- A extensão **auxilia** o sistema operacional a identificar o tipo do arquivo.
- A extensão não é obrigatória, mas ajuda o usuário a identificar o tipo do arquivo.

Tipos de Arquivos

Arquivos de Texto:

- Contêm dados codificados como caracteres legíveis.
- Usados para armazenar textos simples, como `.txt` e `.csv`.
- Exemplo: cada linha pode ser uma string que termina com um caractere de nova linha (`\n`).

Arquivos Binários:

- Armazenam dados em formato bruto, diretamente como uma sequência de bytes.
- Usados para dados mais complexos, como imagens, vídeos ou estruturas de dados mais eficientes.
- Exemplos: `.png`, `.dat`, `.exe`.

Métodos de Leitura

Leitura Sequencial:

- Os dados são lidos na ordem em que estão armazenados.
- Utilizada principalmente em arquivos de texto e em alguns arquivos binários.
- Exemplo: processar uma linha de cada vez de um arquivo `.txt`.

Leitura Direta (Acesso Aleatório):

- Permite acessar dados em qualquer posição do arquivo diretamente, sem ter que percorrer todo o arquivo.
- Mais comum em arquivos binários, onde se pode pular para posições específicas.
- Exemplo: ler um registro específico em um arquivo binário de dados.

Arquivo de Texto com Leitura Sequencial

`cin` e `cout` Comportamento como Arquivos

Em C++, `cin` e `cout` são objetos que representam fluxos de entrada e saída, respectivamente, e se comportam de forma similar a arquivos.

Fluxos de Dados:

- Ambos são fluxos de dados que podem ser lidos e escritos sequencialmente, como arquivos.
- `cin` lê dados do console, enquanto `cout` escreve dados no console.

Uso de Operadores:

- Assim como em arquivos, usamos operadores de inserção (`<<`) e extração (`>>`) para manipular dados.
- O mesmo conceito aplicado a `cin` e `cout` se aplica a arquivos.

Streams de Arquivos em C++

```
#include <fstream>
```

Tipo	Descrição
`ofstream`	Stream de arquivo de saída. Você cria um objeto deste tipo quando deseja criar um arquivo e escrever dados nele.
`ifstream`	Stream de arquivo de entrada. Você cria um objeto deste tipo quando deseja abrir um arquivo existente e ler dados dele.
`fstream`	Stream de arquivo. Objetos deste tipo podem ser usados para abrir arquivos para leitura, escrita ou ambos.

Arquivo de Entrada

```cpp
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

int main() {
    // Declara um objeto ifstream
    ifstream arquivo;
    // Abre o arquivo "dados.txt" para leitura
    arquivo.open("dados.txt");

    // Verifica se o arquivo foi aberto corretamente
    if (!arquivo) {
        cerr << "Erro ao abrir o arquivo!\n";
        return 1;
    }

    string palavra;

    // Lê o arquivo palavra por palavra
    while (arquivo >> palavra) {
        cout << palavra << "\n";
    }

    arquivo.close();  // Fecha o arquivo
    return 0;
}
```

Arquivo: `dados.txt`

```
Olá, mundo!
Este é um exemplo de arquivo de texto.
Vamos ler cada palavra desse arquivo.
```

Saída do terminal

```
Olá,
mundo!
Este
é
um
exemplo
de
arquivo
de
texto.
Vamos
ler
cada
palavra
desse
arquivo.
```

Leitura de Números de um Arquivo de Texto

```cpp
#include <iostream>
#include <fstream>

using namespace std;

int main() {
    // Abre o arquivo "numeros.txt" para leitura
    ifstream arquivo("numeros.txt");

    // Verifica se o arquivo foi aberto corretamente
    if (!arquivo) {
        cerr << "Erro ao abrir o arquivo!\n";
        return 1;
    }

    int numero;

    // Lê o arquivo número por número
    while (arquivo >> numero) {
        cout << "Número lido: " << numero << "\n";
    }

    arquivo.close();  // Fecha o arquivo
    return 0;
}
```

Arquivo: `numeros.txt`

```
10
20
30
40
50
```

Saída do terminal

```
Número lido: 10
Número lido: 20
Número lido: 30
Número lido: 40
Número lido: 50
```

Arquivo de Saída

```cpp
#include <iostream>
#include <fstream>

using namespace std;

int main() {
    // Declara um objeto ofstream
    ofstream arquivo;
    // Abre ou cria o arquivo "mensagens.txt" para escrita
    arquivo.open("mensagens.txt");

    // Verifica se o arquivo foi aberto corretamente
    if (!arquivo) {
        cerr << "Erro ao abrir o arquivo!\n";
        return 1;
    }

    // Escreve mensagens no arquivo
    arquivo << "Olá, mundo!\n";
    arquivo << "Este é um exemplo de escrita em arquivo.\n";

    arquivo.close();  // Fecha o arquivo
    cout << "Mensagens escritas no arquivo 'mensagens.txt'.\n";
    return 0;
}
```

Se o arquivo `mensagens.txt` já existir, ele será sobrescrito.

Arquivo: `mensagens.txt`

```
Olá, mundo!
Este é um exemplo de escrita em arquivo.
```

Saída do terminal

```
Mensagens escritas no arquivo 'mensagens.txt'.
```

Escrita de Números em um Arquivo de Texto

```cpp
#include <iostream>
#include <fstream>

using namespace std;

int main() {
    // Abre ou cria o arquivo "saida.txt" para escrita
    ofstream arquivo("saida.txt");

    // Verifica se o arquivo foi aberto corretamente
    if (!arquivo) {  // Corrigido para verificar se houve erro na abertura
        cerr << "Erro ao abrir o arquivo!\n";
        return 1;
    }

    // Escreve números de 1 a 5 no arquivo
    for (int i = 1; i <= 5; ++i) {
        // Escreve cada número em uma nova linha
        arquivo << i << "\n";
    }

    arquivo.close();  // Fecha o arquivo
    cout << "Dados escritos no arquivo 'saida.txt'.\n";
    return 0;
}
```

Arquivo: `saida.txt`

```
1
2
3
4
5
```

Saída do terminal

```
Dados escritos no arquivo 'saida.txt'.
```

Manipulação de Arquivos com `fstream`

O `fstream` permite abrir arquivos para leitura e escrita. Você pode usar diferentes flags para especificar o comportamento ao abrir arquivos.

Flag	Descrição
`ios::in`	Abre o arquivo para leitura.
`ios::out`	Abre o arquivo para escrita.
`ios::trunc`	Se o arquivo já existir, seu conteúdo será truncado (apagado).
`ios::app`	Abre o arquivo para escrita e move o ponteiro para o final do arquivo. Novos dados serão adicionados ao final.
`ios::ate`	Abre o arquivo e move o ponteiro para o final. O arquivo pode ser lido ou escrito em qualquer parte.
`ios::binary`	Abre o arquivo em modo binário (não usa conversões de texto).

Flags de Acesso

```cpp
#include <iostream>
#include <fstream>

using namespace std;

int main() {
    // Declara um objeto fstream
    fstream dataFile;

    // Abre o arquivo "info.txt" para escrita
    dataFile.open("info.txt", ios::out);
    // Escreve dados no arquivo
    dataFile << "Exemplo de uso de fstream.\n";
    dataFile.close();  // Fecha o arquivo

    // Abre o arquivo "info.txt" para leitura
    dataFile.open("info.txt", ios::in);

    string linha;
    while (getline(dataFile, linha)) {
        cout << linha << "\n";
    }

    dataFile.close();  // Fecha o arquivo
    return 0;
}
```

- Se o arquivo já existir, novos dados serão adicionados ao final, sem apagar o que já está lá.

```cpp
fstream("ex.txt", ios::out | ios::app);
```

- O ponteiro é posicionado no final do arquivo ao ser aberto.

```cpp
fstream("ex.txt", ios::in | ios::ate);
```

- Se o arquivo já existir, seu conteúdo será apagado, permitindo que novos dados sejam escritos desde o início.

```cpp
fstream("ex.txt", ios::in | ios::out | ios::trunc);
```

Arquivo Binário

```
int main() {
    ofstream file("num.dat");
    short x = 1297;
    file << x;
    file.close();
}
```

Arquivo `num.dat`

Byte	1	2	3	4	
Texto	'1'	'2'	'9'	'7'	<EOF>
ASCII	49	50	57	55	<EOF>

Se usar arquivo binário:

Byte	1	2	
	5	17	<EOF>

Escrita de Dados Binários

```cpp
#include <iostream>
#include <fstream>

using namespace std;

int main() {
    // Abre o arquivo "dados.dat" para escrita em modo binário
    fstream arquivo("dados.dat", ios::out | ios::binary);

    // Verifica se o arquivo foi aberto corretamente
    if (!arquivo) {
        cerr << "Erro ao abrir o arquivo!\n";
        return 1;
    }

    // Escreve um caractere 'A' no arquivo
    char letra = 'A';
    arquivo.write(&letra, sizeof(letra));

    // Exemplo de um inteiro
    int numero = 123456;
    arquivo.write((char *) &numero, sizeof(numero));

    arquivo.close();  // Fecha o arquivo
    cout << "Dados escritos no arquivo 'dados.dat'.\n";
    return 0;
}
```

Método `write`

```cpp
arquivo.write(const char* ptr, streamsize size);
```

- **`ptr`** : Um ponteiro para o bloco de memória que contém os dados que você deseja escrever. Esse bloco deve ser tratado como um array de caracteres (`char`).
- **`size`** : O número de bytes que você deseja escrever a partir do endereço especificado por `ptr` .

Leitura de Dados Binários

```cpp
#include <iostream>
#include <fstream>

using namespace std;

int main() {
    // Abre o arquivo "dados.dat" para leitura em modo binário
    fstream arquivo("dados.dat", ios::in | ios::binary);

    // Verifica se o arquivo foi aberto corretamente
    if (!arquivo) {
        cerr << "Erro ao abrir o arquivo!\n";
        return 1;
    }

    // Lê um caractere do arquivo
    char letra;
    arquivo.read(&letra, sizeof(letra));

    // Lê um inteiro do arquivo
    int numero;
    arquivo.read((char *) &numero, sizeof(numero));

    arquivo.close();  // Fecha o arquivo
    cout << "Dados lidos do arquivo 'dados.dat': "
         << letra << ", " << numero << "\n";
    return 0;
}
```

Método `read`

```cpp
arquivo.read(char* ptr, streamsize size);
```

- **`ptr`** : Um ponteiro para o bloco de memória onde os dados lidos serão armazenados. Esse bloco deve ser tratado como um array de caracteres (`char`).
- **`size`** : O número de bytes que você deseja ler e armazenar no bloco de memória apontado por `ptr` .

Escrita de um Array em um Arquivo

```cpp
#include <iostream>
#include <fstream>

using namespace std;

int main() {
    int numeros[] = {1, 2, 3, 4, 5};

    // Abre "array.dat" para escrita em modo binário
    fstream arquivo("array.dat", ios::out | ios::binary);

    // Escreve o array no arquivo
    arquivo.write((char *) numeros, sizeof(numeros));

    arquivo.close();  // Fecha o arquivo
    return 0;
}
```

Verificação com `hexdump`:

```
$ hexdump array.dat -X
0000000  01  00  00  00  02  00  00  00  03  00  00  00  04  00  00  00
0000010  05  00  00  00
0000014
```

Leitura de um Array de um Arquivo

```cpp
#include <iostream>
#include <fstream>

using namespace std;

int main() {
    int numeros[5];

    // Abre "array.dat" para leitura em modo binário
    fstream arquivo("array.dat", ios::in | ios::binary);

    // Lê o array do arquivo
    arquivo.read((char *) numeros, sizeof(numeros));

    arquivo.close();  // Fecha o arquivo

    // Exibe os dados lidos
    cout << "Array lido do arquivo: ";
    for (int i = 0; i < 5; i++) {
        cout << numeros[i] << " ";
    }
    cout << "\n";

    return 0;
}
```

⚠️ Arquivos binários podem não ser compatíveis entre plataformas.

- O tamanho de tipos de dados como `int` e `long` pode variar entre plataformas.
- A ordem dos bytes (endianness) pode variar entre sistemas.
  - **big-endian**
  - **little-endian**
- Estruturas podem ter diferentes requisitos de alinhamento em diferentes plataformas.

Leitura e Escrita de `struct`

Escrita

```cpp
#include <iostream>
#include <fstream>

using namespace std;

struct Pessoa {
    char nome[50];
    int idade;
};

int main() {
    Pessoa p = {"Nicolas", 30};

    ofstream arquivo("pessoa.dat", ios::out | ios::binary);

    arquivo.write((char *) &p, sizeof(p));
    cout << "Escrito: " << sizeof(p) << "B\n";

    arquivo.close();
    return 0;
}
```

Saída do terminal:

```
Escrito: 56B
```

Verificação com `hexdump`:

```
$ hexdump pessoa.dat -c -X
0000000   N   i   c   o   l   a   s  \0  \0  \0  \0  \0  \0  \0  \0  \0
0000000  4e  69  63  6f  6c  61  73  00  00  00  00  00  00  00  00  00
0000010  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
0000010  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
*
0000030  \0  \0   i   b 002  \0  \0  \0
0000030  00  00  69  62  1e  00  00  00
0000038
```

Leitura

```cpp
int main() {
    Pessoa p;

    ifstream arquivo("pessoa.dat", ios::in | ios::binary);

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

    arquivo.close();

    cout << "Nome: " << p.nome
         << ", Idade: " << p.idade << "\n";

    return 0;
}
```

Passando `fstream` para Função

```cpp
#include <iostream>
#include <fstream>

using namespace std;

// Função que recebe um fstream por referência
void manipular_arquivo(fstream &arquivo) {
    string linha;

    // Lê e exibe o conteúdo do arquivo linha por linha
    while (getline(arquivo, linha)) {
        cout << linha << "\n";
    }
}

int main() {
    fstream arquivo("dados.txt", ios::in);

    // Chama a função passando o arquivo por referência
    manipular_arquivo(arquivo);

    arquivo.close();  // Fecha o arquivo
    return 0;
}
```

Objetos do tipo `fstream` não podem ser copiados. Para evitar erros e garantir que as operações de leitura e escrita afetem o objeto original, é necessário passá-los por referência.

Questionário

1. Explique a diferença entre os modos de abertura `ios::in`, `ios::out`, e `ios::app` em `fstream`. Quando você usaria cada um desses modos?
2. Descreva como a escrita e leitura de dados em arquivos binários difere da escrita e leitura de dados em arquivos de texto. Inclua exemplos de quando cada tipo pode ser mais apropriado.
3. O que acontece quando você tenta abrir um arquivo com `ios::trunc` e o arquivo já existe?
4. O que é endianness em sistemas de computação? Explique como isso pode afetar a leitura e escrita de arquivos binários em diferentes plataformas.

Questionário

5. Considere a seguinte estrutura:

```
struct Produto {
    char nome[50];
    float preco;
    int quantidade;
};
```

   Escreva um código C++ que permita gravar um array de `Produto` em um arquivo binário e, em seguida, leia esses produtos de volta e os exiba.
6. Descreva o método `write()` da classe `fstream`. Quais são suas principais características e como ele deve ser usado corretamente? Forneça um exemplo de código.

Questionário

7. Explique o que pode acontecer se você não fechar um arquivo após abrir e realizar operações nele. Quais problemas podem surgir devido a essa prática?
8. Analise o seguinte trecho de código e explique o que ele faz. Quais dados estão sendo escritos no arquivo e como eles seriam lidos posteriormente?

```
ofstream arquivo("dados.dat", ios::out | ios::binary);
int numero = 42;
arquivo.write((char *) &numero, sizeof(numero));
arquivo.close();
```

Algoritmos e Programação II

https://evandro-crr.github.io/alg2

Evandro Chagas Ribeiro da Rosa