# IMD0030
# LINGUAGEM DE PROGRAMAÇÃO I

Aula 19 – Manipulação de Arquivos em C++
(material baseado nas notas de aula do Prof. Silvio Sampaio)

IMD INSTITUTO METRÓPOLE DIGITAL
UFRN UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

# Objetivos desta aula

- Introduzir os conceitos e estruturas para a manipulação de arquivos em C++
- Para isso, estudaremos:
  - As principais bibliotecas e objetos do C++ que implementam operações a arquivos
  - As principais operações em arquivos no C++
  - Algumas operações de E/S em ***streams***
- Ao final da aula espera-se que o aluno seja capaz de:
  - Escrever programas em C++ que manipulem arquivos

# Introdução

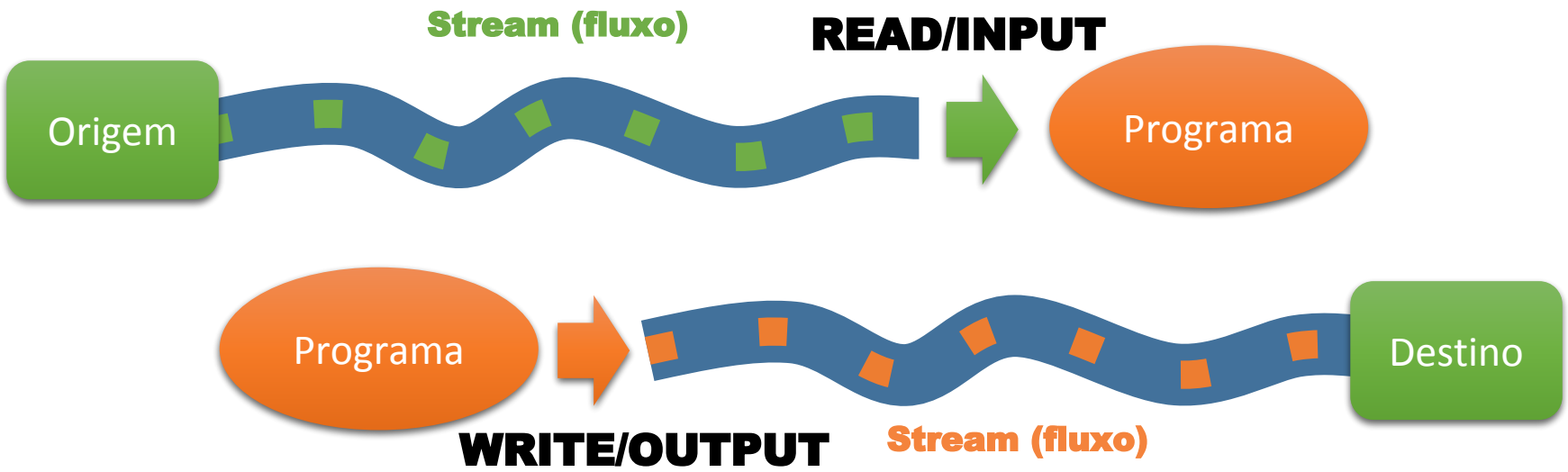
- Como forma de uniformizar as primitivas através das quais um programa invoca ações de I/O (entrada e saída de dados), a linguagem C++ utiliza objetos ***streams***.

- Cada operação de E/S é realizada de maneira sensível ao tipo de dado

- Extensibilidade
  - É possível especificar operações de E/S sobre tipos definidos pelo usuário da mesma forma como é feito para tipos padrão

# O conceito de Stream no C++

- O conceito de ***Stream*** pode ser entendido como um fluxo de informação que pode entrar ou sair de um programa para uma fonte de informação que pode ser um arquivo, a memória, o teclado ou o vídeo
  - Desta forma, a escrita para qualquer um destes meios utiliza basicamente a mesmas classes e métodos, facilitando seu uso.



# O conceito de Stream no C++

- Todos os dispositivos lógicos (streams) são semelhantes em comportamento e bastante independentes dos dispositivos reais
- Distinguem-se dois tipos de streams
  - streams para texto
  - streams para palavras binárias
- Um stream associa-se a um periférico realizando uma operação abertura (*open*) e desassocia-se dele com uma operação de fechamento (*close*).

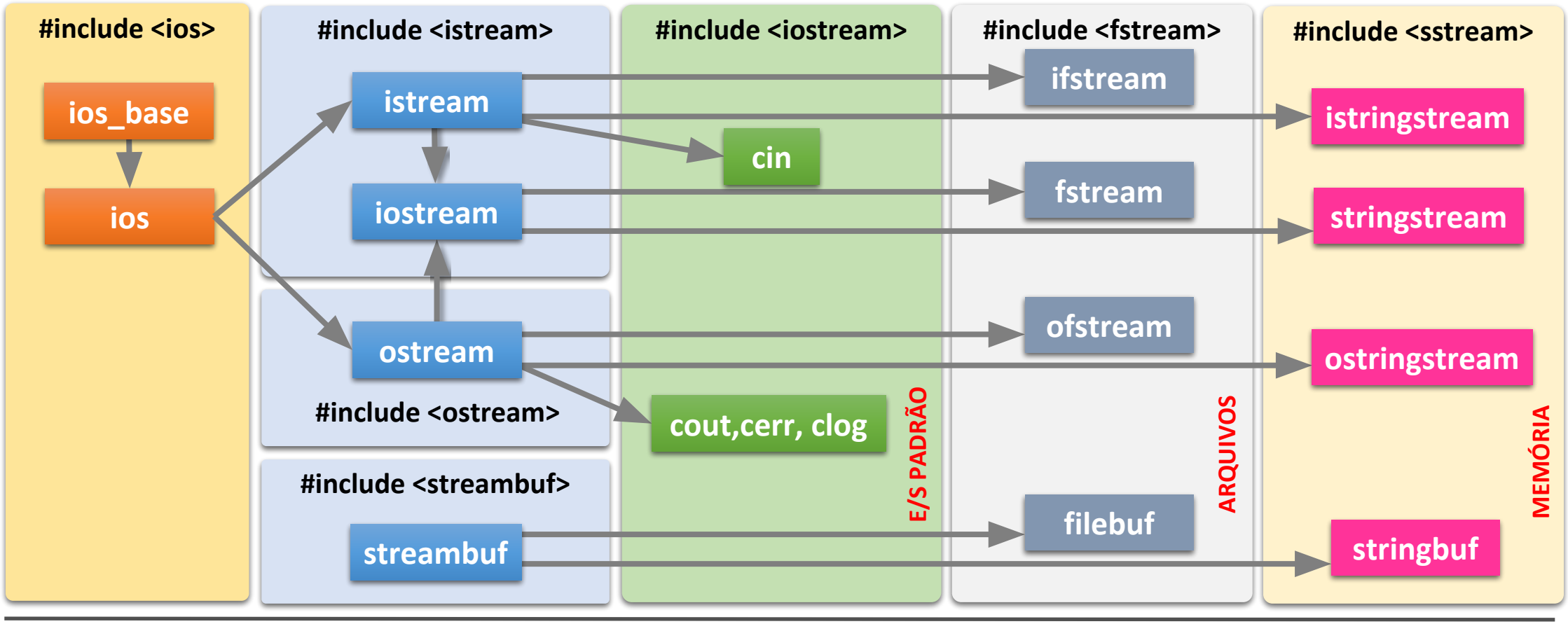
# Buffer

- O buffer consiste em uma área de memória onde as informações são lidas ou escritas.
- O acesso ao buffer ocorre na velocidade da RAM enquanto os dispositivos de entrada e saída (teclado, disco etc.) trabalham em uma velocidade de acesso muito menor.
- Quando parte de um arquivo é necessária, normalmente um volume maior de informações é lido e transferido para o buffer. Assim, quando uma próxima parte desta informação é necessária, o seu conteúdo já se encontra no buffer.
- A utilização de um buffer reduz o número de acessos necessários a disco e desta forma este tipo de I/O é mais eficiente.

# E/S padrão

- Os serviços de E/S mais comuns do C++ (***cin***, ***cout*** e ***cerr***) são implementados através da biblioteca `iostream`
  - No C++, E/S pode ser feita tanto num arquivo quanto na memória (*streams*)
    - Um objeto ***stream*** pode ser entendido como um lugar na memória reservado para o recebimento ou envio de *bytes*
  - Além disso, amplia o conceito de arquivo no sentido de considerar não somente os arquivos em disco mas também o teclado, o vídeo, a impressora e portas de comunicação (serial, USB, etc.)
    - Por exemplo, `cout` está associado ao vídeo (saída) e `cin` está associado ao teclado (entrada)
- O conceito de **herança** é largamente explorado no modelo de classes usado para E/S
  - Evita repetição de código em operações semelhantes
  - Permite a criação de classes e métodos polimórficos

# Hierarquia de entrada/saída em C++



# Entrada e saída em C++

- Operações de E/S:
  - **Entrada** (***Input***):  Um *stream* flui de um dispositivo de entrada para a memória principal
  - **Saída** (***Output***): Um *stream* flui da memória principal para um dispositivo de saída
- E/S de baixo nível (*low-level I/O*)
  - Sem formato definido
  - Opera sobre *bytes* individuais
  - Boa para manipular grande volume de dados com alta velocidade
- E/S de alto nível (*high-level I/O*)
  - Formatada
  - Opera sobre conjunto de *bytes* agrupados em unidades com significado (inteiro, character, etc.)
  - Boa para todo tipo de E/S, exceto processamento de arquivos muito volumosos

# E/S de *streams* em C++

- `ios`:
  - `istream` e `ostream` são classes herdadas de `ios`
  - `iostream` herda características das classes `istream` e `ostream`
    - `#include <iostream>` permite utilizar os *streams* de E/S `cin`, `cout`, `cerr` e `clog`
- << (*left-shift operator*)
  - Operador sobrecarregado que funciona como operador de inserção em *stream*
- >> (*right-shift operator*)
  - Operador sobrecarregado que funciona como operador de extração de *stream*
- Ambos os operadores << e >> são usados com os *streams* já conhecidos `cin`, `cout`, `cerr`, `clog`, assim como com objetos do tipo *stream* definidos pelo usuário

# E/S de *streams* em C++

- `istream`: *input streams*
  - `cin >> varX`
    - `cin` sabe qual o tipo do dado a ser associado a `varX` (baseado no tipo definido para `varX`)
- `ostream`: *output streams*
  - `cout << varX`
    - `cout` sabe qual o tipo do dado a ser associado a `varX` (baseado no tipo definido para `varX`)
  - `cerr << varX`
    - Não armazena em *buffer* (*unbuffered*), ou seja, imprime o valor de `varX` imediatamente

# Manipuladores

- Funções para auxiliar a extração e inserção em *stream*

| Manipulador | *In* | *Out* | Definição |
|---|---|---|---|
| `endl` | | √ | Mudar de linha e *flush* do ostream |
| `ends` | | √ | Inserir '\0' para terminar *string* |
| `flush` | | √ | Esvaziar (*flush*) o *buffer* do ostream |
| `dec` | √ | √ | Conversão para base decimal |
| `hex` | √ | √ | Conversão para base hexadecimal |
| `oct` | √ | √ | Conversão para base octal |
| `ws` | √ | | Eliminar caracteres separadores |
| `setbase(int b)` | √ | √ | Fixar a base de conversão em *b* |
| `resetiosflags(long b)` | √ | √ | Desativar *bit*-vector *flags* de acordo com *b* |
| `setiosflags(long b)` | √ | √ | Ativar *bit*-vector *flags* de acordo com *b* |
| `setfill(int f)` | | √ | Definir o caracter de preenchimento de espaços do campo com `(char)`*f* |
| `setprecision(int n)` | | √ | Situar em *n* dígitos a precisão de um número em ponto-flutuante |
| `setw(int n)` | √ | √ | Colocar em *n* caracteres a largura do campo |

Consulte a lista completa de manipuladores em: http://en.cppreference.com/w/cpp/io/manip

# Acessando arquivos em C++

- Em **C++**, pode-se definir e interagir com objetos associados a arquivos com os mesmos operadores, métodos e manipuladores que se utilizam para `cin` e `cout`
- Entre os vários objetos que podem ser criados para ter acesso a arquivos, destacam-se:
  - `ifstream` - quando queremos abrir um arquivo para leitura
  - `ofstream` - quando queremos abrir um arquivo para escrita
  - `fstream` - quando se deseja que o arquivo possa ser lido e escrito ao mesmo tempo
- Para utilizar E/S em arquivos, utilizar a biblioteca `fstream`
  - `#include <fstream>`

# Manipulando streams

- Um objeto `ifstream` pode ser construído da seguinte forma:
  - `ifstream arqDados;`
    - o objeto `arqDados` é criado mas nenhum arquivo é aberto
  - Para abrir é preciso usar o método `open()`: `arqDados.open(“dados.dat”);`
- Um objeto `ifstream` também pode ser construído da seguinte forma:
  - `ifstream arqDados(“dados.dat");`
    - o objecto `arqDados` é criado e o arquivo `dados.dat` é aberto

# Manipulando streams

- Um objeto `ofstream` pode ser construído da seguinte forma:
  - `ofstream arqDados;`
    - O objeto `arqDados` é criado mas nenhum arquivo é aberto
  - Para abrir é preciso usar o método `open()`: `arqDados.open(“dados.dat” , ios::binary);`
- Um objeto `ofstream` também pode ser construído da seguinte forma:
  - `ofstream arqDados(“dados.dat", ios::binary);`
    - O objeto `arqDados` é criado e o arquivo `dados.dat` é aberto
    - O parâmetro `ios::binary` indica que o arquivo deve ser aberto em modo binário

# Manipulando streams

- Um objeto `fstream` (`ifstream` + `ofstream`) pode ser criado da seguinte forma:
  - `fstream arqDados(“dados.dat", ios::in | ios::out | ios::binary);`
  - Neste exemplo, é criado o objeto `arqDados` para leitura/escrita em modo binário, associado ao arquivo `dados.dat`
- O modo de abertura padrão do fstream é `ios::in | ios::out`
  - Caso o arquivo não exista, a operação de abertura irá falhar, pois a porção `ios::in` irá falhar pelo fato de o arquivo não existir
  - Sempre que for necessário utilizar um `fstream` para leitura e escrita, é necessário **garantir que o arquivo existe**

# Manipulando streams

**Importante lembrar:**

- Um arquivo aberto por um objeto `ofstream` não necessita que definamos o modo de arquivo `ios::out`, pois este modo já é definido para este tipo de objeto por definição
- O mesmo ocorre com o modo `ios::in` e os objetos `ifstream`
- Além disso, por definição, um arquivo aberto por um objeto `ofstream` irá truncar os dados já existentes no arquivo, sobrescrevendo os dados novos aos antigos
  - Porém, o modo `ios::app` permite anexar dados ao final de um arquivo

# Verificando a abertura do arquivo

- A verificação da abertura efetiva de um arquivo deve ser sempre realizada antes de efetuar qualquer operação de E/S sobre o mesmo
- Exemplos de verificação:

```
ifstream arqDados("dados.dat");

if(arqDados.bad()) {
    cerr << "o arquivo nao foi aberto" << endl;
    exit(1);
}

if(!arqDados) {
    cerr << "o arquivo nao foi aberto" << endl;
    exit(1);
}
if(arqDados.is_open() == 0) {
    cerr << "o arquivo nao foi aberto" << endl;
    exit(1);
}
```

# Verificando o fim do arquivo

- O método `eof()` permite saber se foi atingido o fim do arquivo
- Exemplo:

```
ifstream arqDados("dados.dat");
while(!arqDados.eof()) {
    // lê arquivo
    // ...
}
```

# Leitura de arquivos em C++

- Exemplos de leitura:

```
char str[10];
ifstream in_file("dados.dat");
in_file >> str;           // extrai “o tipo de dado” definido por str do arquivo in_file
in_file.read(str, 5);     // lê 5 caracteres do arquivo in_file
in_file.getline(str, 8); // lê uma linha de no máximo 8 caracteres do arquivo in_file
```

# Leitura de arquivos em C++

- Uma forma popular de leitura de arquivos no C++ utiliza o operador de extração
- Exemplo:

```
ifstream arqDados("dados.dat");
while(arqDados >> valor) {
/* O operador de extração retornará 0 (false) quando encontrar EOF e o
      laço de repetição chegará ao fim */
}
```

# Escrita de arquivos em C++

- Exemplos de escrita:

```
char str1[] = "LP1";
fstream out_file("data.dat", ios::out);
out_file << str1;           // Escreve o conteúdo da string str1 no string out_file
out_file.write(str1, 3);    // Escreve, no máximo, 3 caracteres no string out_file
```

# Fechando o arquivo

- Como todo recurso em **C++**, os objetos associados a arquivos devem ser liberados após sua utilização, ou seja, quando já não forem mais necessários
  - No caso de arquivos em **C++**, os objetos que implementam os *streams* já se encarregam disso, através de métodos destrutores, logo **não é necessário fechar um *stream* manualmente**
  - Não há problema em liberar um *stream* manualmente, **mas não é o "estilo C++"** de fazê-lo
- Para liberar um objeto associado a um arquivo, deve-se utilizar o método `close()`
  - Exemplo:

```
ifstream arqDados("dados.dat");
// Utiliza o arquivo
// ...
arqDados.close();
```

# std::getline

- O método std::**getline()** extrai caracteres de um stream até que o final de linha ou um delimitador especificado seja encontrado e armazena numa variável std::string passada como parâmetro.
- Assinatura:

  istream& getline (istream&  is, string& str, char delim);

  - Parâmetros:
    - ***is - o stream de entrada de onde devem ser lidos os dados***
    - ***str - a variável string que será usada para armazenar o valor lido***
    - ***delim - o character a ser usado como delimitador (seu valor padrão é '\n')***

# std::getline

```
// extract to string
#include <iostream>
#include <string>

int main ()
{
  std::string name;

  std::cout << "Please, enter your full name: " ;
  std::getline (std::cin,name);
  std::cout << "Hello, " << name << "!\n";

  return 0;
}
```

```
/* File Handling with C++ using ifstream & ofstream class object*/
/* To write the Content in File*/
/* Then to read the content of file*/
#include <iostream>

/* fstream header file for ifstream, ofstream,
fstream classes */
#include <fstream>

using namespace std;

int main()
{
	// Creation of ofstream class object
	ofstream fout;

	string line;

	// by default ios::out mode, automatically deletes
	// the content of file. To append the content, open in ios:app
	// fout.open("sample.txt", ios::app)
	fout.open("sample.txt");

	// Execute a loop If file successfully opened
	while (fout) {

		// Read a Line from standard input
		getline(cin, line);

		// Press -1 to exit
		if (line == "-1")
			break;

		// Write line in file
		fout << line << endl;
	}

	// Close the File
	fout.close();

	// Creation of ifstream class object to read the file
	ifstream fin;

	// by default open mode = ios::in mode
	fin.open("sample.txt");

	// Execute a loop until EOF (End of File)
	while (fin) {

		// Read a Line from File
		getline(fin, line);

		// Print line in Console
		cout << line << endl;
	}

	// Close the file
	fin.close();

	return 0;
}
```

https://www.geeksforgeeks.org/file-handling-c-classes/

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

using namespace std;

int main()
{
	string a;
	float b;

	ifstream fin;

	fin.open("file.txt");

	while (fin) {

		if(!fin.eof())
		{
			std::getline(fin, a, ';');

			b=stof(a);
			cout << b << endl;
		}
		else
			break;

	}

	return 0;
}
```

# Posicionando o ponteiro do arquivo

- O ponteiro de arquivo indica a posição no arquivo onde será feita a próxima leitura ou escrita
- No C++, há um conjunto de métodos que podem ser usados para movimentar o ponteiro do arquivo
  - `seekg(pos)`
    - Movimenta a posição atual de leitura para `pos`
    - Ex: `in_file.seekg(0); // retorna para o início do arquivo`
  - `seekp(pos)`
    - Movimenta a posição atual de escrita
  - `tellg()`
    - Retorna a posição atual de leitura (em *bytes*), a partir do início do arquivo
  - `tellp()`
    - Retorna a posição atual de escrita (em *bytes*), a partir do início do arquivo

```
// Code to demonstrate the seekg function in file handling
#include <fstream>
#include <iostream>

using namespace std;

int main ()
{
    // Open a new file for input/output operations
    // discarding any current in the file (assumes
    // a length of zero on opening)
    fstream myFile("test.txt", ios::in | ios::out | ios::trunc);

    // Add the characters "Hello World" to the file
    myFile << "Hello World";

    // Seek to 6 characters from the beginning of the file
    myFile.seekg(6, ios::beg);

    // Read the next 5 characters from the file into a buffer
    char A[6];
    myFile.read(A, 5);

    // End the buffer with a null terminating character
    A[5] = 0;

    // Output the contents read from the file and close it
    cout << buffer << endl;

    myFile.close();

    return 0;
}
```

# Lendo e gravando para a memória

- As classes `sstream` interagem com a memória usando a mesma sintaxe usada na manipulação de arquivos em disco
- Exemplo:

```
#include <iostream>
using std::cout;
using std::endl;

#include <string>
using std::string;

#include <sstream>
using std::ostringstream;
```

```
int main() {
    ostringstream oss;
    oss << "Testando a escrita em memoria\n";
    oss << 123 << '\n';
    string s = oss.str();
    cout << s << endl;
    return 0;
}
```

# Exercício

Escreva um programa em C++ que leia um arquivo de texto no formato CSV (valores separados por vírgulas) referente às notas dos alunos de uma turma. O programa deverá computar a média de cada um e a situação de aprovação (aprovação com média maior ou igual a 7.0) ou reprovação e imprimir essas informações tanto na saída padrão (tela) quanto em um outro arquivo de texto. No arquivo de entrada, cada linha deve conter o nome do aluno seguido de três notas. A média deverá ser impressa com apenas uma casa decimal.

Exemplo de entrada:

```
Antonio Silva;10.0;9.0;8.0
Maria Joaquina;10.0;10.0;10.0
Carla Sousa;9.0;5.0;0.0
```

Exemplo de saída:

```
Antonio Silva 9.0 Aprovado
Maria Joaquina 10.0 Aprovado
Carla Sousa 4.7 Reprovado
```