Entrada e saída em C++ e Java

EDUARDO HABIB BECHELANE MAIA

HABIB@CEFETMG.BR

Introdução

Entrada e saída em C++ são implementados através da biblioteca iostream

#include <iostream>

- objetos cin, cout, cerr, endl, etc

Para usar os recursos de entrada e saída da biblioteca iostream em C++, é preciso incluir o comando **using namespace std**.

Um namespace permite:

Evitar duplicidade com, por exemplo, outras implementações com nomes semelhantes.

Por definição, a linguagem C++ utiliza o namespace std para definir todas as funções da biblioteca padrão.

Exemplo Cout

```
#include <iostream>
using namespace std;
int main()
{
  cout << "Imprimindo o famoso HELLO WORLD com cout!!!\n";

  // imprimindo uma linha usando múltiplos comandos
  cout << "Teste com ";
  cout << "dois couts\n";

  //Imprimindo fazendo append usando Strings e números
  cout << "Teste do " << "append." << 17 <<"\n";

  // usando o manipulador endl
  // endl gera um caractere nova linha, e também descarrega o buffer de saída
  cout << "Escrevendo uma linha..." << endl;

  cout << "Mais uma vez...\n";
  cout << flush;    // agora apenas esvaziando o buffer de saída, sem gerar
nova linha

  return 0;
}
```

Entrada e saída em java

- Em Java, a entrada e saída padrão são implementadas através da classe System.
  - Objetos: System.in, System.out, e System.err.

- Ao contrário do C++, em Java não é necessário utilizar diretivas de pré-processamento, como #include, nem é preciso declarar um "namespace". Entretanto, para utilizar conteúdo que está em outras classes, usa-se o **import**

- Em vez de "namespaces" como em C++, Java utiliza pacotes para:
  - Evitar conflitos de nomes entre classes e interfaces, graças à organização hierárquica dos pacotes.
  - A maioria das classes padrão em Java pertence a pacotes, como java.lang, java.util, entre outros. Por padrão, todas as classes no pacote java.lang são importadas automaticamente em qualquer programa Java.

- Nota: As classes para entrada e saída em Java estão principalmente nos pacotes java.io e java.nio, e ao contrário do iostream em C++, você precisará importar explicitamente as classes que deseja usar desses pacotes (a menos que você se refira a elas usando seu nome completo, incluindo o nome do pacote). Por exemplo: java.io.BufferedReader.

Exemplo Java

```
package exemplos;
public class HelloWorld {
public static void main(String[] args) {
     System.out.print("Imprimindo o famoso HELLO WORLD com
System.out.print!!!\n");

// imprimindo uma linha usando múltiplos comandos
System.out.print("Teste com ");
System.out.print("dois System.out.prints\n");

// Imprimindo fazendo concatenação usando Strings e números
System.out.print("Teste do " + "concat." + 17 + "\n");

// usando System.out.println para gerar uma nova linha
System.out.println("Escrevendo uma linha...");
System.out.print("Mais uma vez...\n");
System.out.flush(); // agora apenas esvaziando o buffer de
saída, sem gerar nova linha
}
}
```

Notas

- Em C++, o cout é usado para imprimir na tela, enquanto em Java, usamos System.out.print e System.out.println.
- Em Java, para concatenar strings com outros valores, você simplesmente usa o operador +.
- Em Java, o método System.out.println() imprime o texto e adiciona uma nova linha ao final. Se você não quiser adicionar uma nova linha, pode usar System.out.print().
- O método System.out.flush() em Java é usado para esvaziar (ou "flush") o buffer de saída, semelhante ao cout << flush; em C++.

Exemplo cin

(Esse programa soma 2 números dgitados pelo usuário utilizando o comando cin)

```
#include <iostream>
#include <locale>
using namespace std;
int main()
{
  float num1=0;
  float num2=0;

  cout << "Lendo do teclado utilizando o cin";

  cout << "\nDigite um número:";
  cin >>  num1;

  cout << "\nDigite outro número:";
  cin >>  num2;

  cout << "\nValor da soma:" << num1+num2;

}
```

Exemplo em Java - Scanner

```
import java.util.Scanner;
import java.util.Locale;

public class ExemploEntrada {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        float num1 = 0;
        float num2 = 0;

        System.out.println("Lendo do teclado utilizando o Scanner:");

        System.out.print("\nDigite um número: ");
        num1 = scanner.nextFloat();

        System.out.print("\nDigite outro número: ");
        num2 = scanner.nextFloat();

        scanner.close();

        System.out.println("\nValor da soma: " + (num1 + num2));
    }

}
```

Tipo de dados string

As linguagens C++ e Java oferecem uma série de facilidades para a construção de programas mais longos

Um desses recursos é a manipulação de *strings*, ou cadeias de caracteres.

Em C, *strings* são implementadas através de vetores de caracteres, cujo último elemento deve ser um caractere nulo (*0*).

Já em C++ e Javao usuário pode empregar a palavra **string** para declarar uma *string* de tamanho variável.

Porém, em c++ deve ser incluído o *header* **string**. A partir da declaração, a utilização de *strings* é simples.

```
#include <string>
```

String

```
#include <iostream>
#include <string>
#include <locale>
using namespace std;
int main()
{
  float num1=0;
  float num2=0;

  string nome;

  cout << "\n\n\nLendo do teclado utilizando o cin";

  cout << "\nDigite um número:";
  cin >>  num1;

  cout << "\nDigite outro número:";
  cin >>  num2;

  cout << "\nQuem está somando: ";
  cin >> nome;

  cout << "\nValor da soma feita por "<< nome <<":" << num1+num2;

}
```

Exemplo em Java

```
import java.util.Scanner;

public class ExemploString {

    public static void
main(String[] args) {
        Scanner scanner = new
Scanner(System.in);

        float num1 = 0;
        float num2 = 0;
        String nome;

        System.out.println("\n\n\nL
endo do teclado utilizando o
Scanner");

        System.out.print("\nDigite
um número: ");
        num1 = scanner.nextFloat();

        System.out.print("\nDigite
outro número: ");
        num2 = scanner.nextFloat();
// Ao digitar um número e depois
//pressionar Enter, um caractere de
//nova linha é adicionado ao
//buffer. A linha abaixo limpa o
//buffer do Scanner após a leitura
//do número.

   scanner.nextLine();

   System.out.print("\nQuem está
somando: ");
   nome = scanner.nextLine();

   System.out.println("\nValor da
soma feita por " + nome + ": " +
(num1 + num2));

   scanner.close();
  }
}
```

Utilização do namespace em c++

Se não utilizarmos o comando using, será necessário especificar explicitamente o namespace utilizado, como por exemplo:

```
#include <iostream>

int main()
{
        std::cout << "Exemplo de saída na tela" << std::endl;
        ...
}
```

Namespace

Ao digitar namespace std, se o compilador encontar o comando string, por exemplo:

- ele saberá que você pode estar se referindo a std::string

Se encontrar um vector,

- saberá que você está se referindo a std::vector.

Mesmo assim é necessário incluir os arquivos de cabeçalho.

Se não colocar o namespace, terá que usar a nomenclatura completa:

- std::string ou std::vector nos exemplos acima.

Entrada e saída com arquivo em c++

- Para poder utilizar a entrada e saída via arquivo, é necessário usar:
  - #include <fstream>
- Para abrir um arquivo apenas para saída, definimos um objeto da classe ofstream:
  - ofstream outFile2("teste.out");
- Antes de tentar escrever ou ler em um arquivo, é sempre uma boa idéia verificar se ele foi aberto sem problemas. Pode-se testá-lo usando:

```
if(!outFile2 ) { // Abertura falhou ...
      cerr << " copy.out não pode ser aberto para saída\n";
      exit(-1);
}
```

Entrada e saída em Java

- Para utilizar leitura e escrita de arquivos em Java, é necessário importar as classes adequadas. As mais comuns são:

```
import java.io.File;
import java.io.FileWriter;
import java.io.FileReader;
import java.io.IOException;
```

- Para abrir um arquivo apenas para escrita, criamos uma instância de FileWriter:

- FileWriter outFile2 = new FileWriter("teste.out");

- Antes de tentar escrever ou ler de um arquivo, é sempre uma boa ideia verificar se ocorreu algum problema ao tentar abri-lo. Em Java, ao lidar com arquivos, tratamos potenciais problemas com uma estrutura try-catch:

```
try {
    FileWriter outFile2 = new FileWriter("teste.out");
    // (Aqui viria o código para escrever no arquivo)
} catch (IOException e) {
    System.err.println("teste.out não pode ser aberto para escrita.");
    e.printStackTrace();
}
```

Entrada e saída com arquivo

- Em c++:
  - Para abrir um arquivo apenas para entrada, um objeto da classe ***ifstream*** é utilizado. O programa do próximo slide lê um arquivo especificado pelo usuário e o copia para a saída padrão e outro arquivo, também especificado pelo usuário.
- Em java:
  - Para abrir um arquivo apenas para leitura, utiliza-se a classe **FileReader**. O programa do próximo slide lê um arquivo especificado pelo usuário e o copia para a saída padrão e outro arquivo, também especificado pelo usuário.

Exemplo entrada_saída em C++

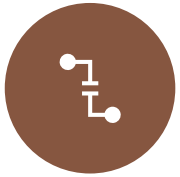
```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main()
{
		cout << "Nome do arquivo de entrada: ";
		string nome_arquivo_entrada;
		cin >> nome_arquivo_entrada;

		ifstream inFile (nome_arquivo_entrada.c_str( ) );

		if(!inFile ) {
			cerr << "Não foi possivel abrir o arquivo de entrada : " << nome_arquivo_entrada << " Saindo do programa! \n";
			 return -1;
		}
		cout << "\nNome do arquivo de saída: ";
		string nome_arquivo_saida;
		cin >> nome_arquivo_saida;
		ofstream outFile (nome_arquivo_saida.c_str( ) );

		if(!outFile ) {
				cerr << "Não foi possivel abrir o arquivo de saida : " <<
				nome_arquivo_saida << " Saindo do programa! \n";
				return -1;
		}

		char ch;
		while (inFile.get(ch) ) {
				cout.put (ch);
				//outFile.put(ch);
				outFile << ch;
		}
}
```

Exemplo em java

```
import java.util.Scanner;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class ManipuladorArquivos {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);

      System.out.print("Nome do arquivo de entrada: ");
      String nomeArquivoEntrada = scanner.nextLine();

      FileInputStream inFile = null;
      try {
         inFile = new FileInputStream(new
File(nomeArquivoEntrada));
      } catch (IOException e) {
         System.err.println("Não foi possível abrir o
arquivo de entrada: " + nomeArquivoEntrada + " Saindo
do programa!");
         scanner.close();
         return;
      }

      System.out.print("\nNome do arquivo de saída: ");
      String nomeArquivoSaida = scanner.nextLine();

      FileOutputStream outFile = null;
      try {
         outFile = new FileOutputStream(new
File(nomeArquivoSaida));
      } catch (IOException e) {
         System.err.println("Não foi possível abrir o
arquivo de saída: " + nomeArquivoSaida + " Saindo do
programa!");
         scanner.close();
         return;
      }

      int ch;
      try {
         while ((ch = inFile.read()) != -1) {
            System.out.print((char) ch);
            outFile.write(ch);
         }
      } catch (IOException e) {
         System.err.println("Erro ao ler ou escrever no
arquivo!");
      } finally {
         try {
            inFile.close();
            outFile.close();
            scanner.close();
         } catch (IOException e) {
            System.err.println("Erro ao fechar arquivos!");
         }
      }
   }
}
```

Abrir o arquivo posteriormente

Objetos das classes *ofstream* e *ifstream* podem ser declarados sem estarem associados a um arquivo.



Um arquivo pode ser conectado posteriormente, chamando a função membro *open( )*

Um arquivo pode ser desconectado de um programa, chamando a função membro *close( )*;

Abrir o arquivo posteriormente

```
#include <fstream>
#include <iostream>
using namespace std;
const int fileCnt = 3;
string fileTable [ fileCnt ] = { "JorgeAmado.txt", "GuimaraesRosa.txt", "CarlosDrummond.txt"};

int main()
{
        ifstream inFile; // não está associado a nenhum arquivo
        for (int i = 0; i < fileCnt; i++)
        {
                string palavra;
                inFile.open (fileTable[i].c_str() );

                while (!inFile.eof()) // verifica se não está no fim do arquivo
                {
                        inFile >> palavra;
                        cout << palavra << endl;
                }
                cout << "\n";
                inFile.close( );
        }
}
```

Exercício

Escreva um algoritmo, **em C++**, e **em Java** que solicite ao usuário a inserção de um número n e calcule, com isso, o número da sequencia de Fibonacci correspondente a esse número digitado pelo usuário. A fórmula do cálculo da sequencia de Fibonacci é:

$$Fib\ n = \begin{cases} Fib\ (0) = 1 \\ Fib\ (1) = 1 \\ Fib\ (n) = Fib(n-1) + Fib(n-2) \end{cases}$$

Imprima o resultado em um arquivo e na tela.

Solução recursiva em c++

```
#include <stdio.h>

int fibonacci(int n) {
  int x;

  if (n == 1) {
    return(1);
  }
if (n == 2) {
    return(1);
  }

  x = fibonacci(n-1) + fibonacci(n-2);
  return(x);
}

int main() {
  int n,i;

  cout << "Digite o numero de termos da sequencia: ";
  cin >> n;
  while(n <= 0) {
    cout << "Numero incorreto. Digite o numero de termos da sequencia: ";
     cin >> n
}

for (i = 1; i <= n; i++) {
    cout << fibonacci(i);
  }
  cout << endl;
  return(0);
}
```

Solução recursiva em Java

```
import java.util.Scanner;

public class Fibonacci {

    public static int fibonacci(int n) {
        if (n == 1 || n == 2) {
            return 1;
        }
        return fibonacci(n - 1) + fibonacci(n - 2);
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite o número de termos da sequência: ");
        int n = scanner.nextInt();

        while (n <= 0) {
            System.out.print("Número incorreto. Digite o número de termos da sequência: ");
            n = scanner.nextInt();
        }

        for (int i = 1; i <= n; i++) {
            System.out.print(fibonacci(i) + " ");
        }

        System.out.println(); // equivalent to "cout << endl;" in C++
        scanner.close();
    }
}
```