ESTRUTURA DE DADOS

Professora **Fernanda**

- Revisão de Algoritmo
- Overview de C++
- Tipos de Variáveis
- Entrada e Saída
- Blocos Condicionais
- Blocos de Repetição

FATEC
Faculdade de Tecnologia do Estado de São Paulo

REVISÃO DE ALGORITMO

Algoritmo é comumente definido como uma sequência finita de etapas lógicas e sequenciais que visam atingir um objetivo definido.

Os casos a seguir podem ser considerados exemplos de algoritmos:

- Substituir uma lâmpada;
- Calçar um sapato;
- Resolver uma equação polinomial;
- Uma receita de bolo.

Todos os casos anteriores, assim como qualquer algoritmo, podem ser expressos por passos ordenados e finitos.

O slide seguinte ilustra o exemplo do primeiro caso analisado como um algoritmo.

FATEC
Faculdade de Tecnologia do Estado de São Paulo

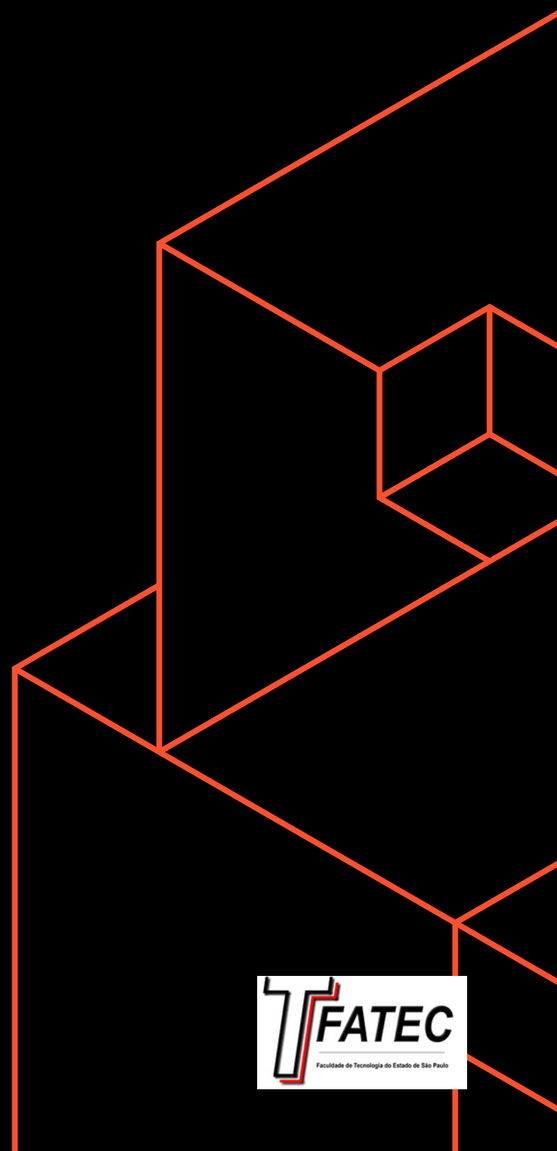
Algoritmo: Substituir uma lâmpada queimada

Passo 1: Selecione uma nova lâmpada;
Passo 2: Remova a lâmpada queimada;
Passo 3: Insira a nova lâmpada.

Embora o algoritmo esteja descrito por passos lógicos e sequenciais, eles podem ser fragmentados em etapas menores, também lógicas e sequenciais, como ilustrado a seguir:

Passo 1.1: Selecione uma nova lâmpada com mesma potência da lâmpada queimada;
Passo 2.1: Gire a lâmpada queimada no sentido anti-horário, até que se solte;
Passo 3.1: Coloque a nova lâmpada no orifício correspondente;
Passo 3.2: Rode a lâmpada no sentido horário até que ela fique presa;

Esta é uma técnica chamada **top-down**, que consiste em dividir um sistema em subsistema, e que pode ser perfeitamente aplicada no ramo de Algoritmos ao dividir um problema ou objetivo em partes menores.



FATEC
Faculdade de Tecnologia do Estado de São Paulo

OVERVIEW DE C++

A linguagem de programação C++ foi desenvolvida em meados de 1983 (originalmente com o nome C with classes) por Bjarne Stroustrup. É uma linguagem orientada a objetos e considerada, em algumas bibliografias, como de **médio nível**, pois oferece mecanismos encontrados em linguagens de baixo e alto nível.

A estrutura de um programa em C++ pode ser definida como uma coleção de objetos que se comunicam via chamadas de seus métodos.

O exemplo de código abaixo ilustra o clássico programa Hello World.

```
#include <iostream>

using namespace std;

//É no método main() que o programa inicia sua execução

int main() {
        cout << "Hello World!";

        return 0;
}
```

No programa anterior, é importante destacar alguns pontos:

- Em aplicativos escritos em C++, é comum encontrar inclusões de arquivos de cabeçalho. Estes arquivos contém informações ou recursos que são necessários ou úteis ao programa. No programa anterior, o cabeçalho **iostream** é necessário;

- A linha **usingnamescape std**; informa ao compilador que a área de nomes chamada **std** será utilizada. Através dela é possível acessar o objeto **cout**;

- •A linha que inicia com // demarca um comentário de única linha em C++;

- •A próxima linha **int main()** denota o método principal da aplicação. A partir dele que a aplicação inicia sua execução;

- •A linha seguinte **cout << “Hello World!”** fará com que o programa exiba a mensagem Hello World! na tela;

- •A última linha **return 0**; concluirá o método main() retornando um código de status.

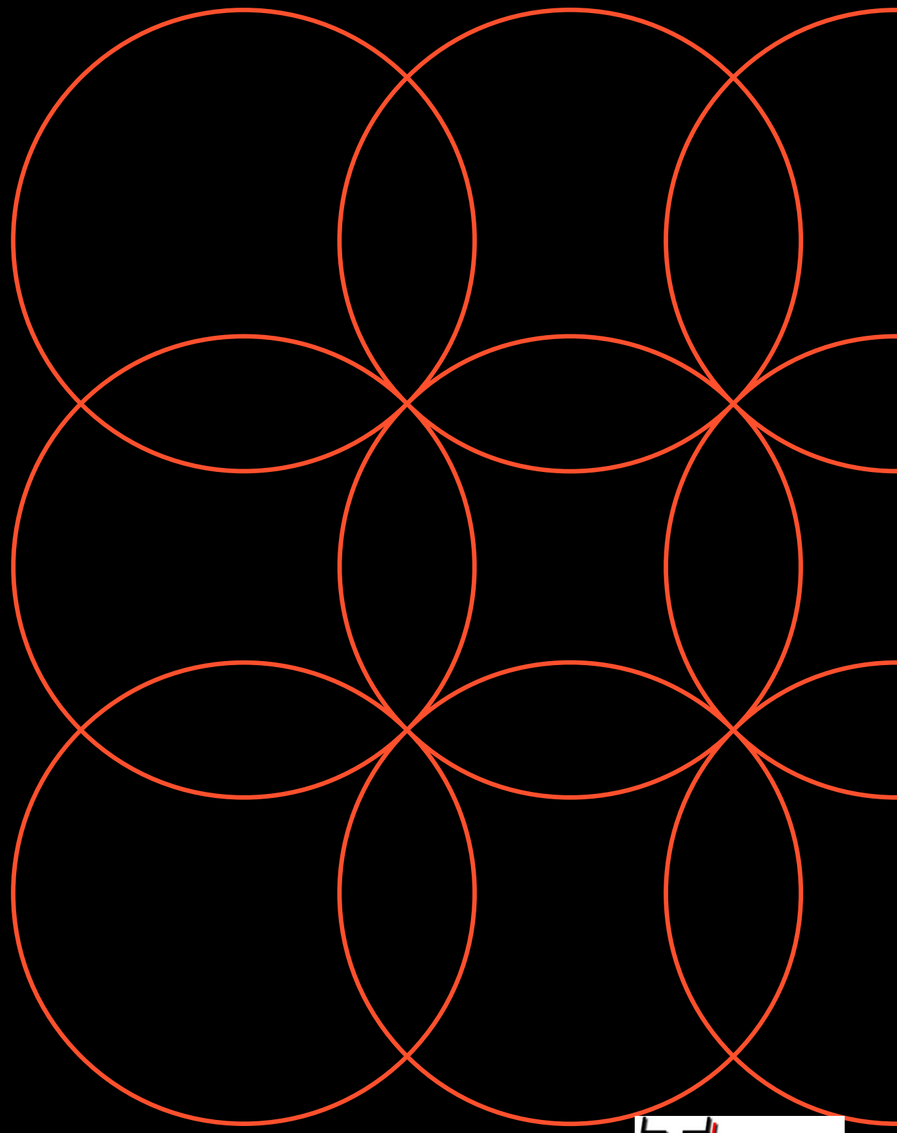
TIPOS DE VARIÁVEIS

Algoritmos computacionais normalmente precisam manipular dados ao longo de sua execução para que o objetivo imaginado seja alcançado.

Neste caso, faz-se necessário ter tipos distintos de dados para suprir diferentes necessidades que os passos lógicos de um algoritmo podem oferecer.

Em C++, os tipos primitivos são os seguintes:

Nome	Descrição
bool	Tipo booleano, que assume apenas os valores true ou false.
char	Caractere representado por um octeto de bits (1 byte) ASCII. É um tipo de int.
int	Número inteiro, cujo tamanho pode variar conforme a arquitetura do processador.
float	Número em ponto flutuante de precisão simples.
double	Igualmente ao float, porém com precisão dupla.
void	Representa a absência de tipo (não é utilizado para declarar variáveis).
wchar_t	Armazena caracteres mais amplos para suportar internacionalização.



FATEC
Faculdade de Tecnologia do Estado de São Paulo

Em C++ existem quatro modificadores que podem ser combinados com os tipos primitivos **char**, **int** e **double**. Os modificadores são:

- **short**: indica uma variável com range menor (aplicável a **int**);
- **long**: indica uma variável com range maior (aplicável a **int** e **double**);
- **signed**: indica um número que pode ser negativo ou positivo (aplicável a **int** e **char**);
- **unsigned**: indica que um número só pode ser positivo (aplicável a **int** e **char**).

O uso dos modificadores permite potencializar a capacidade de armazenamento de variáveis, além de enrijecer a estrutura de um tipo.

Exemplos de uso dos modificadores (alguns intervalos abaixo podem variar de acordo com a arquitetura da máquina):

unsigned short int var1; //Aceita valores entre 0 a 65.535
long double var2; //Possui 10 dígitos de precisão, contra 8 do double convencional
signed char var3; //Varia entre -127 a 127, igualmente ao char convencional
long int var4; //Aceita valores entre -2.147.483.648 a 2.147.483.647

FATEC
Faculdade de Tecnologia do Estado de São Paulo

ENTRADA E SAÍDA

A constituição dos passos algorítmicos, por ora, podem envolver a entrada ou saída de dados. Em C++, operações I/O ocorrem em **streams**, que processam sequências de bytes.

Se o fluxo de bytes tem origem em um teclado, um disco rígido ou uma conexão de rede e tem destino para a memória principal do computador, então diz-se que a essa é uma **operação de entrada.**

Se os bytes estão alocados na memória principal e têm como destino um monitor de vídeo, uma impressora ou uma conexão de rede, então essa é uma **operação de saída.**

As operações de entrada e saída padrão em C++ são realizadas através dos objetos **cout** e **cin**, que são instâncias das classes **ostream** e **istream**, respectivamente. Para utilizá-las, é necessário incluir o arquivo header iostream e explicitar o uso da área de nomes chamada std, conforme os exemplos dos slides seguintes.

FATEC
Faculdade de Tecnologia do Estado de São Paulo

- Exemplo com **cout**

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

int main() {
        char str[] = "Ola C++";
        cout << "O valor de str e: " << str << end;

        return 0;
}
```

- A saída deste programa será:

```
O valor de str e: Ola C++
```

O uso de **endl** fará com que o programa insira uma nova linha após o término da linha corrente.

É possível utilizar mais de uma vez o operador << em uma mesma linha de código, como ilustrado no exemplo acima.

- Exemplo com **cin**

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

int main() {
        char nome[] 50;

        cout << "Por favor, entre com o seu nome: ";
        cin >> nome;
        cout << "Seu nome e: " << nome << endl;

        return 0;
}
```

Uma possível saída deste programa pode ser:

```
Por favor, entre com o seu nome: C++
Seu nome e: C++
```

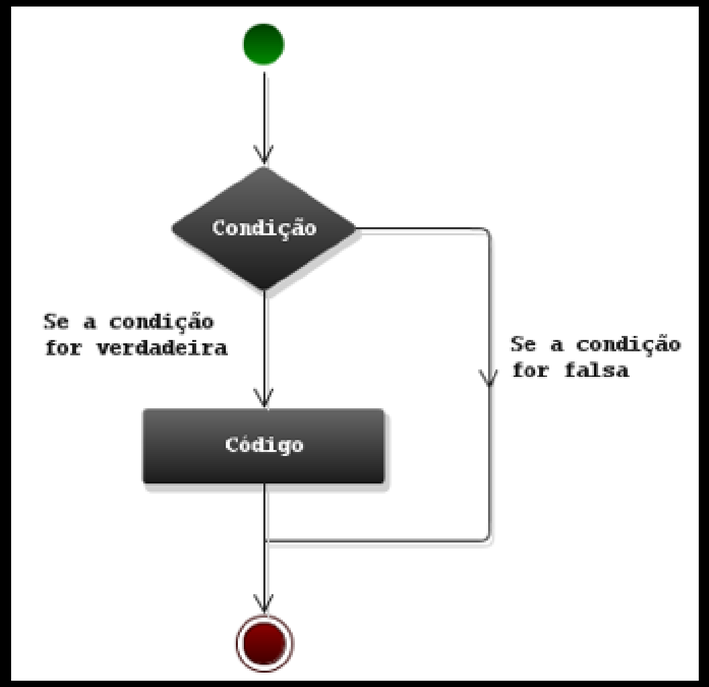
É possível inserir mais de um operador >> em uma mesma linha de código, o que causará ao programa solicitar mais de uma vez a entrada de valores. Exemplo:

```
cin >> var1 >> var2;
```

BLOCOS CONDICIONAIS

Também conhecido como bloco de tomada de decisão, os blocos condicionais são estruturas que envolvem o teste condicional de uma expressão, de tal forma que se a condição for satisfeita, uma determinada tarefa será executada, caso contrário, uma outra tarefa poderá ser realizada.

Um bloco condicional pode ser ilustrado como um diagrama (exemplo abaixo).



Comando **IF**

Executa um bloco de código caso a condição booleana a ser testada retornar **true**. Exemplo abaixo:

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

int main() {
        int n = 10;

        if (n < 20) {
                cout << "n e menor que 20" << endl;
        }

        cout << "Valor de n: " << n << endl;

        return 0;
}
```

Comando **IF...ELSE**

Executa um bloco de código caso a condição booleana a ser testada retornar **true**. Se for falsa, executa outro bloco de código, conforme exemplo a seguir:

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

int main() {
        int n = 10;

        if (n < 20) {
                cout << "n e menor que 20" << endl;
        } else {
                cout << "n nao e menor que 20" << endl;
        }

        cout << "Valor de n: " << n << endl;

        return 0;
}
```

Comando **IF TERNÁRIO**

O comando **IF...ELSE** pode ser escrito como um IF ternário em condições onde haja uma atribuição de variável, conforme exemplo do código abaixo.

```
#include <iostream>

using namespace std;

int main() {
        int n;
        char c = 'A';

        //Atribui à variável n o valor 10 se c for igual a A
        //Caso contrário, o valor de n será igual a 20
        n = (c == 'A' ? 10 : 20);

        return 0;
}
```

Comando **IF...ELSE IF...ELSE**

Executa um bloco de código caso a sua condição correspondente retornar **true**. O uso de **else** nesta estrutura é opcional.

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

int main() {
        int n = 10;

        if (n < 20) {
                cout << "n e menor que 20" << endl;
        } else if (n < 30) {
                cout << "n e menor que 30" << endl;
        } else {
                cout << "n nao e menor que 20 e nem 30" << endl;
        }

        cout << "Valor de n: " << n << endl;

        return 0;
}
```

Comando **SWITCH**

Esta estrutura testa uma variável a partir de uma lista de valores, sendo que cada valor é conhecido como **case**. Se a variável for igual a algum valor, o comando **switch** irá executar o bloco de código correspondente ao **case**. Exemplo:

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

int main() {
        int n = 10;
        switch (n) {
                case 10:
                        cout << "O valor e 10" << endl;
                        break;
                case 20:
                        cout << "O valor e 20" << endl;
                        break;
                default:
                        cout << "Valor nao e 10 e nem 20" << endl;
        }
        return 0;
}
```

COMANDO SWITCH

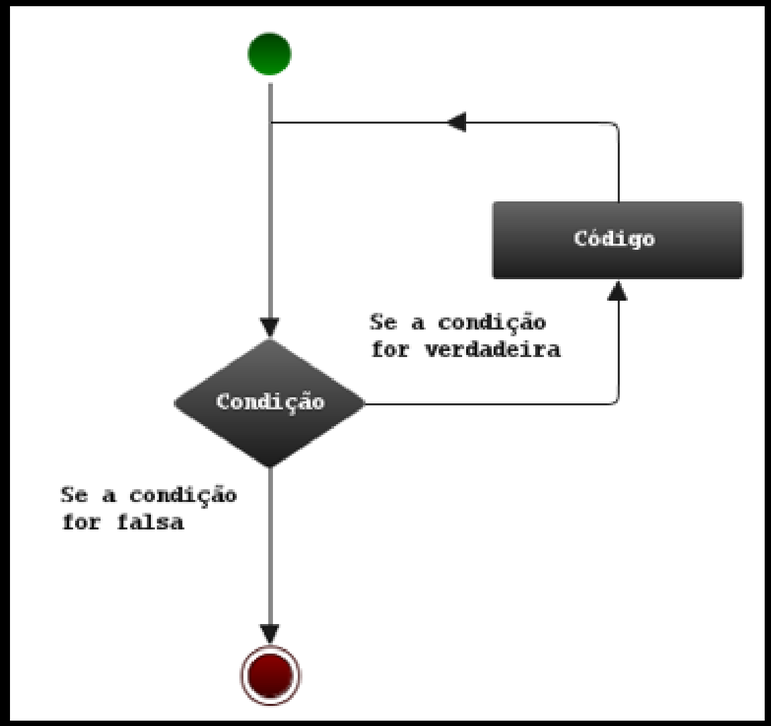
O comando **switch** pode ser comparado com uma estrutura **IF...ELSEIF...ELSE**, porém é importante frisar os seus pontos peculiares:

- A variável a ser testada deve ser um tipo inteiro ou enumerado;
- É possível inserir quantos casos forem necessários, onde a sintaxe deve conter a palavra **case** <valor> e dois pontos;
- O <valor> deve ser do mesmo tipo da variável. Além disso deve ser necessariamente uma constante ou um literal;
- O uso de **break** é opcional. Seu propósito é de interromper o fluxo de código, forçando-o a executar a próxima linha de código depois do bloco **switch**;
- Um bloco **switch** pode ter um caso **default**, que é o equivalente ao **else** de uma estrutura **if**, ou seja, é executado apenas quando todos os casos anteriores falharem. Por ser o último caso, não há necessidade do comando **break**.

BLOCOS DE REPETIÇÃO

Há situações durante a execução de um algoritmo em que é necessário executar blocos de código repetidamente.

Um bloco de repetição processará uma ou várias linhas de código enquanto a condição booleana que a mantém retornar **true**. O diagrama a seguir ilustra a ideia de um bloco de repetição:



Comando **WHILE**

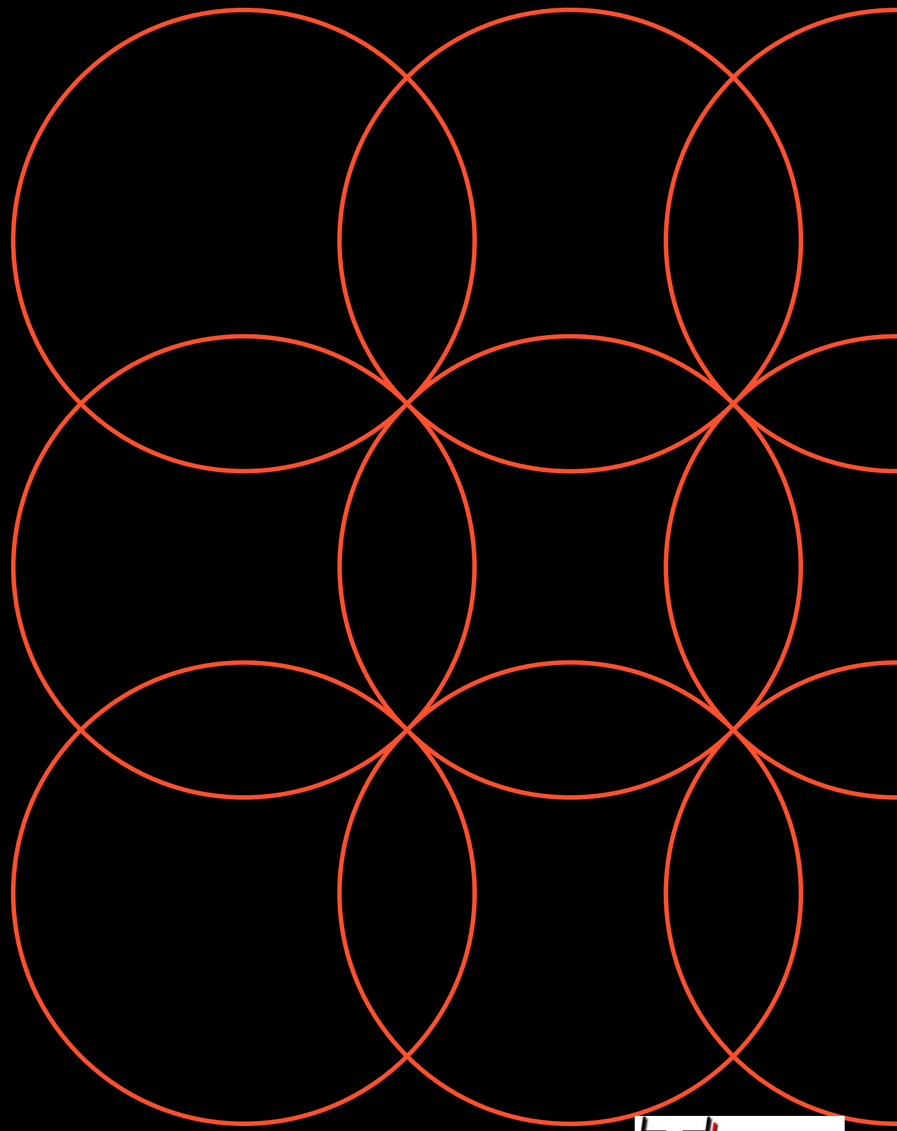
Processa um bloco de código repetidamente até que a sua condição retorne **true**.
O código de exemplo abaixo ilustra o uso do comando **while**.

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

int main() {
        int n = 1;

        while (n <= 10) {
                cout << "O valor de n e " << n << endl;
                n++;
        }

        return 0;
}
```



Comando **WHILE**

Quando o programa anterior for compilado e executado, a saída será a seguinte:

```
O valor de n e 1
O valor de n e 2
O valor de n e 3
O valor de n e 4
O valor de n e 5
O valor de n e 6
O valor de n e 7
O valor de n e 8
O valor de n e 9
O valor de n e 10
```

No código anterior, a rotina irá incrementar o valor de **n** até que este chegue a 11. Quando isso acontecer, a condição no laço **while** retornará **false** (pois 11 não é menor e nem igual a 10) e, consequentemente, o bloco de repetição se encerrará.

Comando **FOR**

Este tipo de bloco de repetição permite controlar de forma mais eficiente a quantidade de iterações a ser realizada, conforme exemplo abaixo.

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

int main() {
        for (int n = 1; n <= 10; n++) {
                cout << "O valor de n e " << n << endl;
        }

        return 0;
}
```

Este programa imprimirá o mesmo resultado do código do exemplo anterior. A notável diferença está em relação ao controle da quantidade de iterações, que fica implícita no próprio comando **for**.

Comando **DO...WHILE**

Diferentemente dos casos anteriores, onde o teste da condição é realizado primeiro, neste tipo de bloco de repetição a condição é verificada somente ao fim do bloco.

A diferença principal é que o código será executado pelo menos uma vez, conforme exemplo abaixo (cuja saída será igual aos dois códigos anteriores).

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

int main() {
        int n = 1;

        do {
                cout << "O valor de n e " << n << endl;
                n++;
        } while (n <= 10);

        return 0;
}
```

CURIOSIDADE

Assim como em outras linguagens de programação, C++ oferece suporte à internacionalização (também conhecida como I18N) e localização, que são conceitos relacionados à adaptação de um software para uma língua ou cultura de um país ou região.

Ao longo dos exemplos desta aula, as mensagens de saída dos programas não apresentaram caracteres acentuados, pois se assim fossem apresentadas, a saída não seria exatamente como o esperado, conforme exemplo abaixo:

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

int main() {
        cout << "Esta mensagem contém caractere com acento";

        return 0;
}
```

O que poderia gerar a seguinte saída:

CURIOSIDADE

Isto ocorre devido ao conjunto de caracteres (muito conhecido como charset) utilizado para a apresentação dos caracteres na saída do programa, o que de certa forma está relacionado com os conceitos de internacionalização e localização.

Uma forma de corrigir esta questão é através da seguinte instrução:

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

int main() {
        //Define a localização para Português-Brasil
        setlocale(LC_CTYPE, "portuguese_brazil");
        cout << "Esta mensagem contém caractere com acento";

        return 0;
}
```

Que basicamente informa para a aplicação que o conjunto de caracteres deverá suportar a acentuação e demais regras do idioma português do Brasil.