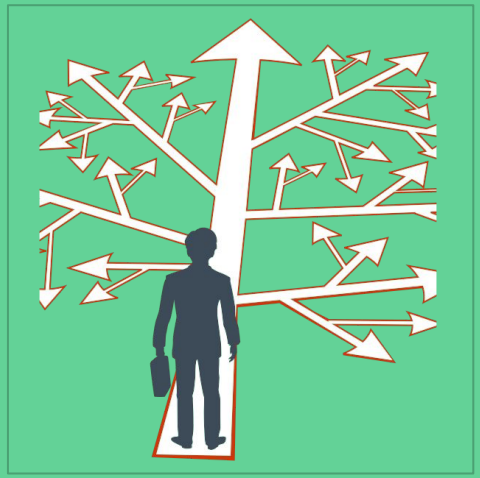
Universidade Federal de Lavras
GCC224 - Introdução aos Algoritmos

Estruturas condicionais

Prof. Joaquim Quinteiro Uchôa
Profa. Juliana Galvani Greghi
Profa. Marluce Rodrigues Pereira
Profa. Paula Christina Cardoso
Prof. Renato Ramos da Silva

- Contextualização
- Operadores relacionais e lógicos
- Seletor Unidirecional
- Seletor Bidirecional

Contextualização



Contextualização

A estrutura sequencial é formada pela execução sequencial dos comandos dispostos entre um início e um fim. Em C++ temos:

```
int main() {

    return 0;
}
```

início

fim

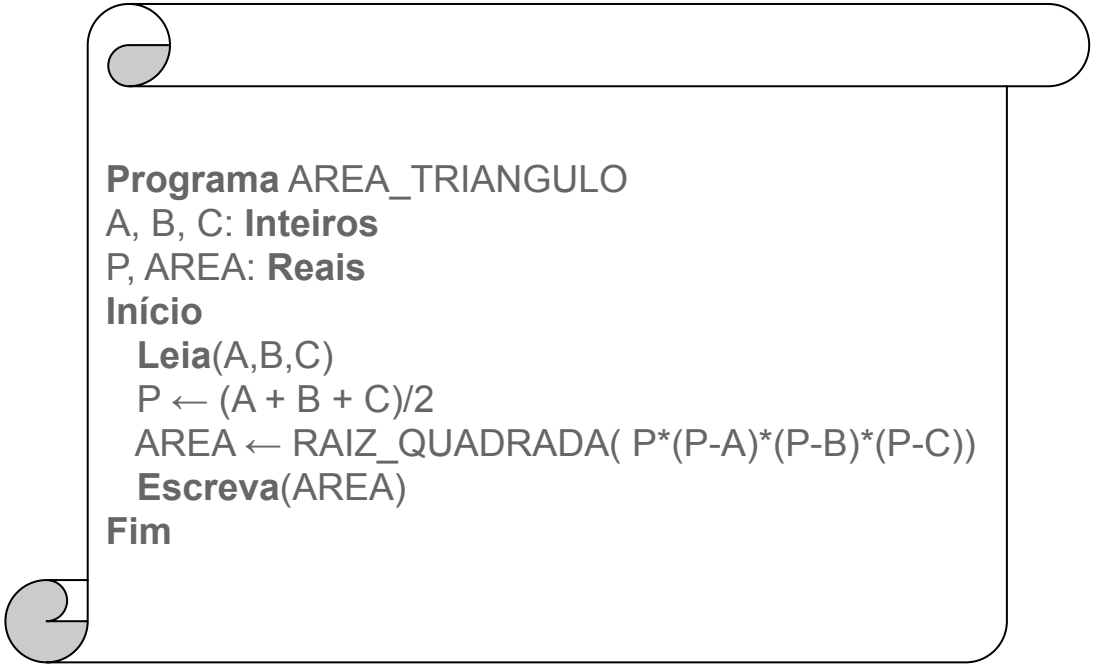
Contextualização

Utilizando o que conhecemos sobre estrutura sequencial, vamos abordar o seguinte problema:

- Construir um algoritmo que calcule a área de um dado triângulo, dados os valores de cada um dos seus lados, representados pelas variáveis A, B e C.

Contextualização

- Exemplo: área de um triângulo (em pseudocódigo)



```
Programa AREA_TRIANGULO
A, B, C: Inteiros
P, AREA: Reais
Início
   Leia(A,B,C)
   P ← (A + B + C)/2
   AREA ← RAIZ_QUADRADA( P*(P-A)*(P-B)*(P-C))
   Escreva(AREA)
Fim
```

Contextualização

- Exemplo: área de um triângulo (em pseudocódigo)

```
Programa AREA_TRIANGULO
A, B, C: Inteiros
P, AREA: Reais
Início
   Leia(A,B,C)
   P ← (A + B + C)/2
   AREA ← RAIZ_QUADRADA( P*(P-A)*(P-B)*(P-C))
   Escreva(AREA)
Fim
```

Exemplo de uso (Situação 1)

A = 3, B = 4, C = 5
P = 6
AREA = RAIZ_QUADRADA(6*(6-3)*(6-4)*(6-5))
AREA = RAIZ_QUADRADA(6 * 3 * 2 * 1)
AREA = RAIZ_QUADRADA(36)
AREA = 6

Contextualização

- Exemplo: área de um triângulo (em pseudocódigo)

```
programa AREA_TRIANGULO
A, B, C: inteiros
SemiPerimetro, AREA: real
Início
   leia(A,B,C)
   P ← (A + B + C)/2
   AREA ← RAIZ_QUADRADA(P*(P-A)*(P-B)*(P-C))
   escreva(AREA)
Fim
```

Exemplo de uso (Situação 2)

A = 3, B = 4, C = 9
P = 8
AREA = RAIZ_QUADRADA(8*(8-3)*(8-4)*(8-9))
AREA = RAIZ_QUADRADA(8 * 5 * 4 * -1)
AREA = RAIZ_QUADRADA(-160)
AREA = ???

Neste caso, não é possível encontrar a área do triângulo.

Contextualização

- Exemplo: área de um triângulo
  - A regra matemática diz que três números inteiros compõem um triângulo se, e somente se, cada lado for menor que a soma dos outros dois lados.
  - Sendo assim, antes de se calcular a área de um determinado triângulo, é necessário verificar se os lados definidos formam um triângulo.

Contextualização

- Ou seja:

Regras a serem satisfeitas

$A < B + C$
$B < A + C$
$C < A + B$

- No nosso exemplo:

Situação 1: forma um triângulo

A = 3, B = 4, C = 5
$A < B + C \rightarrow 3 < 4 + 5 \rightarrow 3 < 9$ ***(V)***
$B < A + C \rightarrow 4 < 3 + 5 \rightarrow 4 < 8$ ***(V)***
$C < A + B \rightarrow 5 < 3 + 4 \rightarrow 5 < 7$ ***(V)***

Situação 2: não forma um triângulo

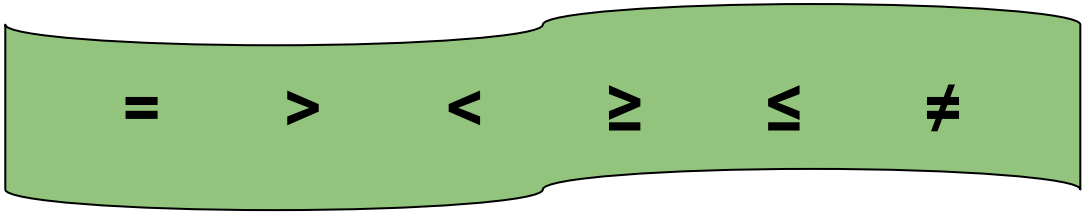
A = 3, B = 4, C = 9
$A < B + C \rightarrow 3 < 4 + 9 \rightarrow 3 < 13$ ***(V)***
$B < A + C \rightarrow 4 < 3 + 9 \rightarrow 4 < 12$ ***(V)***
$C < A + B \rightarrow 9 < 3 + 4 \rightarrow 9 < 7$ ***(X)***

Contextualização

- Quando verificamos em um algoritmo se determinadas condições (ou regras) são satisfeitas antes de realizar um determinado processamento, estamos criando o que chamamos de uma **Estrutura de Decisão** ou **Estrutura Condicional**.

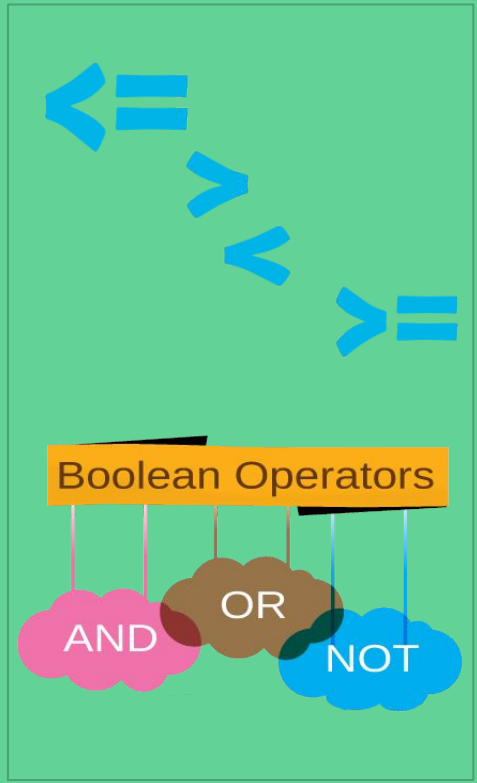
Contextualização

Para verificarmos essas condições, em geral precisamos comparar elementos, utilizando operadores que relacionam esses elementos. Por exemplo, utilizamos esses operadores para verificar se um valor é maior ou menor que outro.



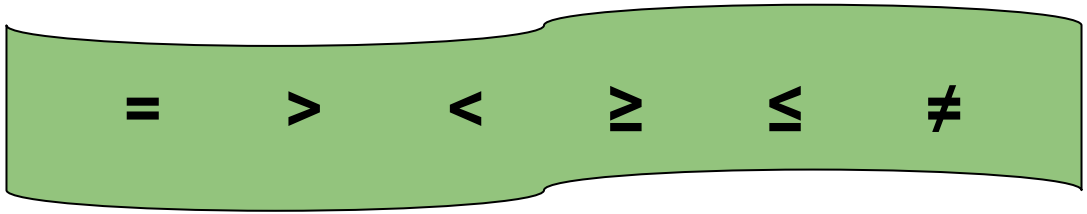
Assim, vamos verificar esses operadores a seguir.

Operadores relacionais e lógicos



Operadores relacionais

- Os operadores relacionais são usados para fazer comparações entre elementos que podem ser variáveis ou valores manipulados por um algoritmo durante sua execução.



- O resultado de uma expressão relacional sempre será um valor booleano (Verdadeiro ou Falso).

Operadores relacionais

Operador	Significado semântico	Sintaxe em C/C++
=	Igualdade	==
>	Maior que	>
<	Menor que	<
≥	Maior ou igual a	>=
≤	Menor ou igual a	<=
≠	Diferença	!=

Todos os operadores relacionais são operadores binários em notação infixa.

Operadores relacionais

Operador	Significado semântico	Sintaxe em C/C++
=	Igualdade	==
>		
<		
≥		
≤	Menor ou igual a	<=
≠	Diferença	!=

Por notação infixa, entende-se que o operador está no meio dos operandos. Por exemplo, para compararmos dois valores representados por `a` e `b`, podemos fazer:

```
(a <= b)
```

Todos os operadores relacionais são operadores binários em notação infixa.

Operadores relacionais

- Exemplo: igualdade entre dois números inteiros

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

int main()
{
    int A, B;
    cin >> A;
    cin >> B;
    cout << (A == B) << endl;
    return 0;
}
```

Caso o usuário digite dois números inteiros iguais ao executar este programa, a mensagem que será exibida será o valor numérico **1** (pois este é o valor utilizado internamente pelo C++ para representar o valor lógico **true**).

Caso o usuário digite dois números inteiros diferentes, a mensagem que será exibida neste caso será o valor numérico **0** (pois este é o valor utilizado internamente pelo C++ para representar o valor lógico **false**).

Operadores relacionais

- **Cuidado em C++:** Não confundir o operador de igualdade (==) com o operador de atribuição (=). Exemplo:

```
#include <iostream>
using namespace std;
int main()
{
    int A, B;
    bool C;
    B = A = 5;
    C = A == 5;
    return 0;
}
```

Neste caso, a variável **A** recebe o valor inteiro 5, enquanto que a variável **B** recebe o valor armazenado por A, ou seja, o mesmo valor 5.

Neste outro caso, compara-se o valor armazenado na variável **A** com o número 5, como os dois valores são iguais, armazena-se o resultado **true** na variável **C**.

Operadores lógicos

- Os operadores lógicos são utilizados para combinar expressões relacionais e/ou para efetuar verificações de valores associados a testes lógicos.

E (conjunção) **OU (disjunção)** **NÃO (negação)**

- O resultado de uma expressão lógica sempre será um valor booleano (Verdadeiro ou Falso).

Operadores lógicos

Operador	Significado semântico	Sintaxe em C++	Sintaxe em C
E	Conjunção lógica	and	&&
OU	Disjunção lógica	or	||
NÃO	Negação (inversor) lógica	not	!

Como a linguagem C++ é uma evolução de C, é possível utilizar em programas escritos em C++ tanto a sintaxe dos operadores lógicos do próprio C++ quanto dos operadores do C.

Para aumentar a legibilidade dos programas, recomenda-se especificamente a sintaxe do C++, uma vez que a mesma é mais clara.

Operadores lógicos

- Tabela verdade para os operadores lógicos
  - Dadas duas proposições **X** e **Y** quaisquer sobre o domínio do problema a ser tratado, a tabela verdade que sumariza os operadores lógicos pode ser descrita como:

X	Y	X and Y	X or Y	not X	not Y
true	true	true	true	false	false
true	false	false	true	false	true
false	true	false	true	true	false
false	false	false	false	true	true

Operadores lógicos

- Exemplo: três números inteiros compõem um triângulo se, e somente se, cada lado for menor que a soma dos outros dois lados

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

int main()
{
    int A, B, C;
    bool verifica;
    cin >> A >> B >> C;
    verifica = (A < (B + C)) and (B < (A + C))
                    and (C < (A + B));
    cout << verifica << endl;
    return 0;
}
```

Caso o usuário digite três números inteiros que satisfaçam as regras que definem se os mesmos formam um triângulo, a mensagem que será exibida neste programa será o valor numérico **1** (**true**).

Caso o usuário digite três números inteiros que **não** satisfaçam as regras, a mensagem que será exibida neste caso será o valor numérico **0** (**false**).

Seletor unidirecional

Seletor unidirecional

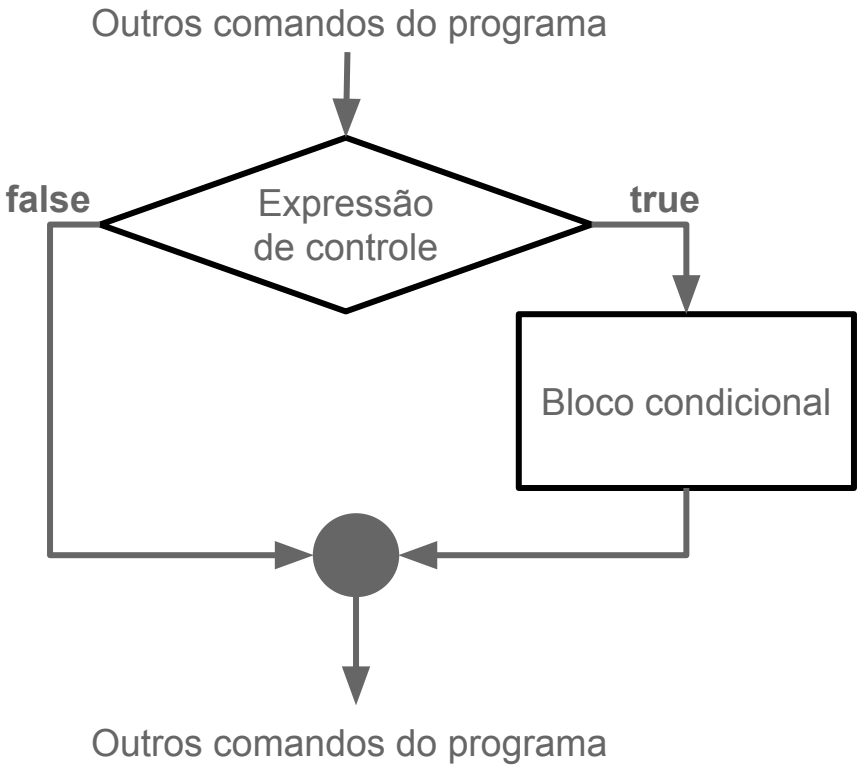
- Uma estrutura de decisão ou condicional permite a escolha do conjunto de comandos a serem executados, ou seja, um determinado conjunto de ações será executado somente se a condição imposta for satisfeita.

- Tais condições são representadas por expressões relacionais ou lógicas. Ou seja, expressões cujos valores finais são sempre booleanos (Verdadeiro ou Falso).

Seletor unidirecional

- A estrutura condicional mais simples que pode ser desenvolvida é a estrutura conhecida como seletor unidirecional.
- Este tipo de estrutura é caracterizada por dois componentes principais, a saber:
  - Expressão de Controle: expressão relacional ou lógica que deve ser avaliada a fim de se obter um valor booleano.
  - Bloco Condicional: conjunto de comandos que serão executados apenas se a expressão de controle assumir o valor **true**.

Seletor unidirecional



Seletor unidirecional

- Sintaxe em pseudocódigo:

```
Se  Expressão_controle Então
Início-Se
    Bloco_condicional
Fim-Se
```

Os comandos que forem descritos no **Bloco_condicional** serão executados apenas se a **Expressão_controle** assumir um valor **true** ao ser avaliada. Caso o valor **false** seja obtido, todos os comandos do **Bloco_condicional** serão completamente ignorados.

Seletor unidirecional

- **Sintaxe em C++:**

```
if  (Expressão_controle) {
    Bloco_condicional
}
```

ou

```
if              (Expressão_controle)
{
    Bloco_condicional
}
```

Em C/C++, os símbolos de chaves, **{** e **}**, são utilizados para designar/delimitar um bloco de código.

Caso um bloco de código possua apenas um único comando, os símbolos de chaves podem ser omitidos se o programador assim o desejar.

Não deixar a **Expressão_controle** entre símbolos de parênteses, **(** e **)**, gerará um erro de sintaxe.

Seletor unidirecional

- Exemplo
  - Vamos voltar ao problema que foi discutido anteriormente e vamos construir um programa em C++ para resolvê-lo.
  - Problema: construir um algoritmo que calcule a área de um dado triângulo, dados os valores de cada um dos seus lados, representados pelas variáveis A, B e C.
  - Contudo, para calcular tal área, devemos garantir que os três lados realmente compõem um triângulo, ou seja, se cada lado é menor que a soma dos outros dois lados.

Exemplo Seletor Unidirecional: calcular área de um triângulo

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

int main()
{
    int A, B, C;
    float P, Area;
    cin >> A >> B >> C;

    if ( (A < (B + C)) and (B < (A + C)) and (C < (A + B)) )
    {
        P = float(A + B + C)/2;
        Area = sqrt(P* (P-A)*(P-B)*(P-C));
        cout << Area << endl;
    }
    return 0;
}
```

Expressão_controle

Bloco_condicional

De acordo com o algoritmo expresso neste programa, a área do triângulo, apenas será exibida no dispositivo de saída padrão (monitor), se a condição de que cada lado do triângulo for menor do que a soma dos outros dois lados for satisfeita.

Seletor unidirecional

- Outro exemplo
  - Suponha que você precise imprimir o maior valor entre três números, sabendo que não existe a possibilidade de dois ou três valores serem iguais.
  - Para isso, basta comparar cada valor com os outros dois, imprimindo-o caso seja maior.

Exemplo Seletor Unidirecional: imprime maior de três valores

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

int main() {
    int A, B, C;
    cin >> A >> B >> C;

    if ((A > B) and  (A > C))
        cout << A << endl;
    if ((B > A) and  (B > C))
        cout << B << endl;
    if ((C > A) and  (C > B))
        cout << C << endl;

    return 0;
}
```

Notem que como o bloco condicional em cada um dos seletores possui um único comando, as chaves não são necessárias.

Entretanto, não é erro incluí-las, sendo inclusive recomendado para programadores iniciantes.

Exemplo Seletor Unidirecional: imprime maior de três valores

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

int main() {
    int A, B, C;
    cin >> A >> B >> C;

    if ((A > B) and  (A > C)) {
        cout << A << endl;
    }
    if ((B > A) and  (B > C)) {
        cout << B << endl;
    }
    if ((C > A) and  (C > B)) {
        cout << C << endl;
    }

    return 0;
}
```

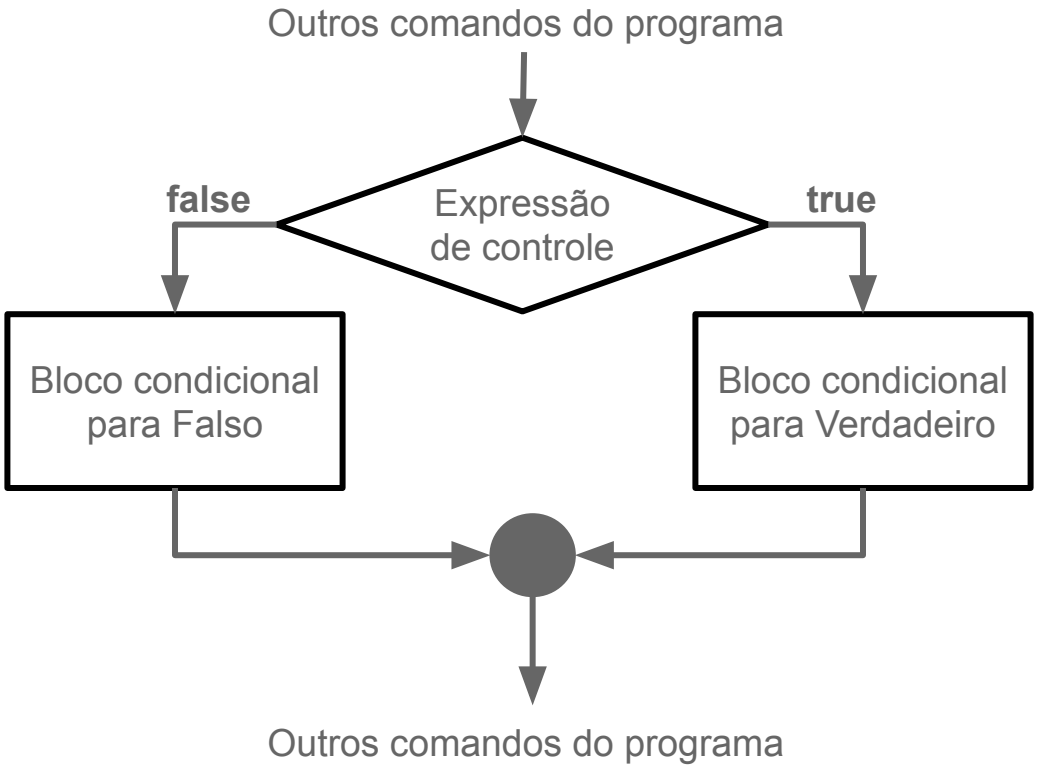
O exemplo ao lado é o mesmo código anterior, mas destacando o bloco condicional entre chaves, o que é recomendado para programadores iniciantes.

Seletor bidirecional

Seletor bidirecional

- Dependendo da situação, pode ser importante definir o que um dado programa deve fazer nos casos em que a expressão de controle de uma estrutura condicional se torne falsa.
- As estruturas condicionais que definem um bloco condicional para o caso da expressão de controle ser verdadeira e um segundo bloco condicional diferente para o caso dela ser falsa são chamadas seletores bidirecionais.

Seletor bidirecional



Seletor bidirecional

- Sintaxe em pseudocódigo:

```
Se  Expressão_controle Então
Início-Se
    Bloco_condicional_para_Verdadeiro
Fim-Se
Senão
Início-Senão
    Bloco_condicional_para_Falso
Fim-Senão
```

É importante observar que o **Bloco_condicional_para_Verdadeiro** e o **Bloco_condicional_para_Falso** são excludentes, ou seja, apenas um deles pode ser executado.

Seletor bidirecional

- Sintaxe em C++:

```
if  (Expressão_controle) {
    Bloco_condicional_para_Verdadeiro
} else {
    Bloco_condicional_para_Falso
}
```

Caso **Expressão_controle** seja verdadeira (**true**), os comandos do **Bloco_condicional_para_Verdadeiro** serão executados.

Caso **Expressão_controle** seja falsa (**false**), os comandos descritos no **Bloco_condicional_para_Falso** serão executados.

Reescrever a **Expressão_controle** ao lado da palavra **else** gerará um erro de sintaxe.

Seletor bidirecional

- Exemplo
  - Problema: construir um algoritmo que calcule a área de um dado triângulo, dados os valores de cada um dos seus lados, representados pelas variáveis A, B e C.
  - Contudo, desta vez, caso os três lados dados não formem um triângulo, informar ao usuário do programa que não é possível calcular a área para os valores fornecidos.

Exemplo Seletor Bidirecional: calcular área de um triângulo

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

int main(){
    int A, B, C;
    float P, Area;
    cin >> A >> B >> C;
    if ( (A < (B + C)) and (B < (A + C)) and (C < (A + B)) ) {
        P = float(A + B + C)/2;
        Area = sqrt(P*(P-A)* (P-B)*(P-C));
        cout << Area << endl;
    }
    else {
        cout << "Os valores fornecidos não formam um triângulo." << endl;
    }
    return 0;
}
```

Seletores aninhados

Seletores aninhados

- Dependendo da situação, pode ser interessante construir uma estrutura condicional que permita a seleção entre mais do que apenas dois conjuntos de comandos.
- Uma das formas de se construir estruturas condicionais com múltiplos caminhos de execução é por meio do aninhamento de estruturas condicionais bidirecionais.
- Para cada estrutura condicional aninhada é necessário definir uma nova expressão de controle para a mesma.

Seletores aninhados

- **Sintaxe em C++:**

```
if  (Expressão_controle_1) {
    Bloco_condicional_1
} else if  (Expressão_controle_2) {
    Bloco_condicional_2
} else if  (Expressão_controle_3) {
    Bloco_condicional_3
} //...Outras regras de decisão
else {
    Bloco_condicional_N
}
```

Ao aninhar seletores, as expressões de controle são avaliadas na ordem em que foram definidas.

A primeira expressão de controle que, durante a sua avaliação, assumir o valor **true** será aquela cujo bloco condicional será executado.

Uma vez que um bloco condicional seja executado, todos os outros blocos da estrutura condicional em questão são automaticamente ignorados.

Seletores aninhados

- Exemplo:
  - Problema: desenvolver um programa em C++ que, dado um nível de alerta de risco, classifique-o quanto ao seu respectivo nível de gravidade. Os níveis de alerta seguem a seguinte regra:
    - 0 - 3: nível de gravidade baixo;
    - 4 - 6: nível de gravidade médio;
    - 7 - 10: nível de gravidade alto;
    - Caso o valor informado para o nível de alerta esteja fora dos limites estabelecidos, escreva uma mensagem de erro.

Exemplo Seletores Aninhados: definir nível de gravidade

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

int main(){
    int nivel_alerta;
    cin >> nivel_alerta;

    if ((nivel_alerta >= 0) and (nivel_alerta <= 3))
        cout <<  “Nível de gravidade baixo.” << endl;
    else if ((nivel_alerta >= 4) and (nivel_alerta <= 6))
        cout <<  “Nível de gravidade médio.” << endl;
    else if ((nivel_alerta >= 7) and (nivel_alerta <= 10))
        cout <<  “Nível de gravidade alto.” << endl;
    else
        cout <<  “O nível de alerta de risco informado está fora do intervalo.” << endl;

    return 0;
}
```

Seletor múltiplo

Seletor múltiplo

- Algumas linguagens de programação oferecem, como alternativa aos seletores aninhados, um seletor múltiplo.
- O seletor múltiplo permite tomar decisões a partir de um conjunto de valores discretos.
- Dependendo da linguagem de programação e do compilador escolhido, seletores múltiplos podem ser formas mais eficientes de se realizar condicionais com múltiplos caminhos de execução.

Seletor múltiplo

- Diferentemente das outras estruturas condicionais, em um seletor múltiplo não precisamos definir diretamente uma expressão de controle booleana, cujo resultado final é Verdadeiro ou False.

- Ao invés disso, precisamos definir uma variável/expressão de análise e descrever os possíveis valores que esta variável/expressão pode assumir. Para cada valor possível, definimos então um bloco de comandos que deverá ser executado para o caso da variável/expressão de análise assumir o valor em questão.

Seletor múltiplo

- **Sintaxe em pseudocódigo:**

```
Caso  (Variável_Expressão_Análise)
   Igual Valor_1
        Bloco_para_Valor_1
   Igual Valor_2
        Bloco_para_Valor_2
   //... demais possíveis caminhos
   Caso padrão
        Bloco_para_caso_padrão
Fim-Caso
```

Ao construir um seletor múltiplo, o programador pode definir um **caso padrão** (último caso da estrutura).

A definição de um caso padrão não é obrigatória, mas é fortemente recomendada.

O caso padrão é definido para garantir que pelo menos um dos blocos de comandos será executado.

O caso padrão é executado quando a variável/expressão de análise não assumir nenhum dos valores definidos para ela (**Valor_1**, **Valor_2**, …).

Seletor múltiplo

- **Sintaxe em C++:**

```
switch  (Variável_Expressão_Análise) {
  case Valor_1:
      Bloco_para_Valor_1
      break;
  case Valor_2:
      Bloco_para_Valor_2
      break;
  //... demais possíveis casos
  default:
      Bloco_para_caso_padrão
}
```

Para garantir que cada bloco de comandos (**case**) seja executado exclusivamente é necessário terminar cada um deles com o comando **break**.

Esquecer de colocar o comando **break** ao final de um bloco fará com que mais de um **case** seja executado.

Em C++, a **variável_expressão_análise** obrigatoriamente precisa ser definida como um valor do tipo de dado enumerável, como **char, short, long** ou **int**. Informações de qualquer outra natureza, como **float** ou **string**, gerarão um erro de sintaxe,

- Exemplo:
  - Problema: desenvolver um programa em C++ que receba dois números inteiros e realiza uma das quatro operações básicas (adição, subtração, multiplicação e divisão), a ser escolhida pelo usuário do programa, escrevendo o resultado da operação no dispositivo de saída padrão. Assuma que o usuário definirá qual operação deve ser executada por meio de um caracter, seguindo as seguintes regras de valores:
    - +: a operação de adição deve ser calculada;
    - -: a operação de subtração deve ser calculada;
    - * ou x: a operação de multiplicação deve ser calculada;
    - /: a operação de divisão inteira deve ser calculada.

Exemplo Seletores Múltiplo: operações matemáticas básicas


```
#include <iostream>
using namespace std

int main() {
    int numero1, numero2;
    char operacao;

    cin >> numero1 >> numero2 >> operacao;

    switch (operacao) {
        case '+':
            cout << numero1 + numero2 << endl;
            break;
        case '-':
            cout << numero1 - numero2 << endl;
            break;
```

Caracteres utilizam aspas simples. Se utilizar aspas duplas, será interpretado como string e dará erro de compilação pois não é mais um tipo enumerável.

```
        case '*':
        case 'x':
            cout << numero1 * numero2 << endl;
            break;
        case '/':
            if (numero2 != 0)
                cout << numero1 / numero2 << endl;
            else
                cout << "Não é possível dividir por 0." << endl;
             break;
        default:
             cout << "Operação inválida." << endl;
    }

    return 0;
}
```

Como o *case '*'* não tem break, isso implica que ele vai fazer as ações do próximo, ou seja *case 'x'*.

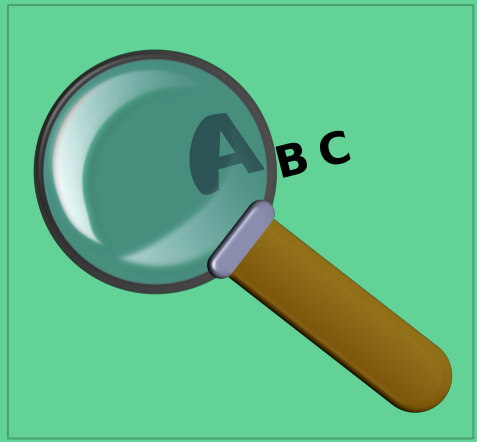
Cuidados com o uso de break

O comando **break** pode ser utilizado em outras situações, que não a de interromper um bloco condicional de um **switch**.

Isso leva ao uso de técnicas de programação não-estruturada, não sendo uma boa prática de programação e piorando em muito a manutenibilidade de código.

Para a equipe responsável por este curso, o comando **break** deve ser utilizado **apenas** como final de bloco condicional em um **case** do seletor múltiplo **switch**.

Especificidades de estruturas condicionais



Zero é falso, tudo mais é verdadeiro!

Uma característica interessante do C/C++ é que ao fazermos testes lógicos, ele considera que zero é falso e qualquer outro valor é verdadeiro.

Ou seja, **5**, '**a**', e **3.2** são todos verdadeiros.

Vejamos um exemplo.

Exemplo - testes condicionais

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


int main() {
    if (5)
        cout << "5 é verdadeiro" << endl;
    if ('a')
        cout << "'a' é verdadeiro" << endl;
    if (3.2)
        cout << "3.2 é verdadeiro" << endl;

    return 0;
}
```

```
5 é verdadeiro
'a' é verdadeiro
3.2 é verdadeiro
```

É tudo culpa do hardware

C e C++ foram implementados com a filosofia de buscarem otimizar ao máximo os recursos oferecidos pelo hardware, são linguagens para implementação de sistemas operacionais, por exemplo.

Processadores geralmente possuem instruções que fazem com que, caso uma variável em um dado registrador seja zero, então o programa é desviado para um outro conjunto de instruções, saindo do fluxo normal. Ou seja, um condicional se um valor é zero. Se for zero, desvia, caso contrário, continua a sequência normal do código.

É tudo culpa do hardware

Assim, C e C++ simplesmente aproveitam dessa instrução de desvio (**JZ** em x86 ou **BEQ** em ARM, por exemplo), para implementar estruturas condicionais.

Para isso, o compilador converte os testes lógicos em testes que permitem verificar se na verdade uma dada variável é ou não zero.

Se usado adequadamente, este recurso torna o código mais legível e eficiente.

Exemplo - impacto adicional

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


int main() {
    int m = 0;
    int n = 1;

    if (m = 1)
        cout << "m vale 1" << endl;

    if (n = 0)
        cout << "n vale 0" << endl;

    cout << m << endl << n << endl;

    return 0;
}
```

```
m vale 1
m = 1
n = 0
```

Exemplo - impacto adicional

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


int main() {
    int m = 0;
    int n = 1;

    if (m = 1)
        cout << "m vale 1" << endl;

    if (n = 0)
        cout << "n vale 0" << endl;

    cout << m << endl << n << endl;

    return 0;
}
```

```
m vale 1
m = 1
n = 0
```

Valores não foram comparados (==) e sim atribuídos (=).

Vamos comparar ponto-flutuante?

Lembram-se do que já dissemos sobre números em ponto-flutuante serem representados de forma aproximada no computador.

O que será que podemos esperar de um resultado que depende de uma comparação exata entre valores em ponto-flutuante?

```cpp
#include <iostream>

using namespace std;

int main() {
    float x = 1.1;

    if  (2*x == 2.2) {
        cout << "passou no teste" << endl;
    } else {
        cout << "tem algo errado" << endl;
    }

    return 0;
}
```

```cpp
#include <iostream>

using namespace std;

int main() {
    float x = 1.1;

    if  (2*x == 2.2) {
        cout << "passou no teste" << endl;
    } else {
        cout << "tem algo errado" << endl;
    }

    return 0;
}
```

```
tem algo errado
```

```cpp
#include <iostream>

using namespace std;

int main() {
    float x = 1.1;

    if  (2*x == 2.2) {
        cout << "passou no teste" << endl;
    } else {
        cout << "tem algo errado" << endl;
    }

    return 0;
}
```

```
tem algo errado
```

Dependendo do compilador e do hardware, pode ser que não ocorra esse erro com esses valores, mas é fácil conseguir, utilizando outros valores... Isso é esperado, por conta da forma como números em ponto-flutuante são armazenados.

Como corrigir isso?

A única forma de fazer testes condicionais usando valores em ponto-flutuante é considerando uma precisão, um erro aceitável.

Ao invés de verificar se uma variável z vale 7.5, por exemplo, é melhor verificar se o valor dela menos 7.5 é menor ou igual a um certo erro: $(z - 7.5) \leq \varepsilon$.

Como esse resultado pode ser negativo, é necessário usar a função de módulo, abs( ), da biblioteca <cmath>, pois precisamos desse cálculo com módulo: $| z - 7.5 | \leq \varepsilon$.

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


int main() {
    float x = 1.1;

    if (abs(2*x - 2.2) <= 0.00001) {
        cout << "passou no 2o. teste"
             << endl;
    }

    return 0;
}
```

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


int main() {
    float x = 1.1;

    if (abs(2*x - 2.2) <= 0.00001) {
        cout << "passou no 2o. teste"
             << endl;
    }

    return 0;
}
```

```
passou no 2o. teste
```

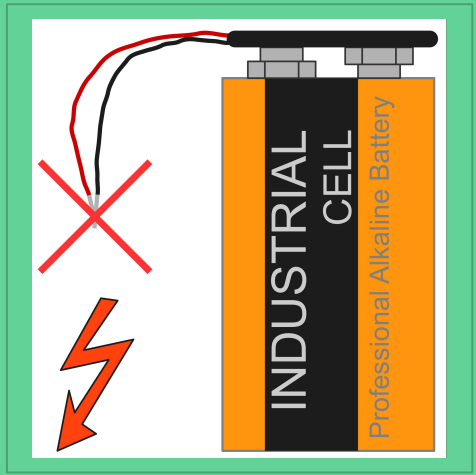
```
#include <iostream>
#include <cmath>
using namespace std;


int main() {
    float x = 1.1;

    if (abs(2*x - 2.2) <= 0.00001) {
        cout << "passou no 2o. teste"
             << endl;
    }

    return 0;
}
```

```
passou no 2o. teste
```

Podemos escolher um erro adequado ao problema em questão, de acordo com a capacidade do tipo.

Em geral, o erro é definido como constante, para facilitar a alteração.

Lógica de curto-circuito



Lógica de curto-circuito

Podemos utilizar mais de uma condição em estruturas condicionais:

```
if ((cont == 20) or (soma >= 200))
```

O C++ possui um recurso interessante e poderoso chamado lógica de curto-circuito nesses casos.

Curto-circuito com and

Uma expressão `t1` **`and`** `t2` **`and`** `t3` **`and`** **`…`** **`and`** `tn`, de vários testes combinados por **`and`**, só é verdadeira se todas os testes forem verdadeiras.

Não adianta ficar conferindo todos os testes, caso já tenha sido encontrado um falso, o resultado será obviamente falso.

Exatamente isso que o C++ faz: caso um teste dê falso, ele não realiza os outros testes, retornando falso para a expressão.

Curto-circuito com or

Uma expressão `t1 or t2 or t3 or … or tn`, de vários testes combinados por **`or`**, só é falso se todas os testes forem falsos.

Não adianta ficar conferindo todos os testes, caso já tenha sido encontrado um verdadeiro, o resultado será obviamente verdadeiro.

Exatamente isso que o C++ faz: caso um teste dê verdadeiro, ele não realiza os outros testes, retornando verdadeiro para a expressão.

Lógica de curto-circuito - Exemplo

Exemplo:

```
if ((cont == 20) or (soma >= 200))
```

Caso cont seja igual a 20, então a segunda expressão retornará verdadeiro, sem verificar o valor de soma.

Sobre o Material

Sobre este material

Material produzido coletivamente, principalmente pelos seguintes professores do DCC/UFLA:

- Janderson Rodrigo de Oliveira
- Joaquim Quinteiro Uchôa
- Juliana Galvani Greghi
- Renato Ramos da Silva

Inclui contribuições de outros professores do Departamento de Computação Aplicada - DAC/UFLA.