Estruturas de Dados I

Introdução

Prof. Bruno Azevedo

Instituto Federal de São Paulo

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus Catanduva

Type Casting

- Podemos querer converter um valor de um tipo primitivo para outro tipo primitivo.
- Esta conversão chamada de *type casting*. Em C++, temos dois tipos principais: implicito ou explícito. O segundo pode ser dividido em vários subtipos.
- No *type casting* implícito ou automático, o compilador irá automaticamente alterar um tipo de dado para outro.

  Por exemplo, se você atribuir um valor inteiro a uma variável de ponto flutuante, o compilador irá inserir código para converter o inteiro em um número de ponto flutuante.

```
#include <iostream>
using namespace std;
int main() {
    float a = 20.70;
    int b;
    b=a;
    cout << "Resultado: " << b << endl;
    return 0;
}
```

- Neste exemplo, será exibido o valor 20, descartando a parte após a vírgula (ponto).
- Ou seja, fizemos um *type casting* implícito, onde o compilador converteu um valor em ponto flutuante para um valor inteiro.

Type Casting

- Já o *type casting* explícito é composto de cinco abordagens distintas.
- Tradicional, dynamic cast, static cast, reinterpret cast, e const cast.
- De início, iremos aprender o tradicional (C-like) e o static cast. Os outros exigem um maior conhecimento da linguagem.
- Ambos realizam a conversão em tempo de compilação.
- O tradicional pode ser escrito de duas formas similares.

```
double w = 10.3, x = 12.5;
int y,z;
y = int (w); // convertendo double para int
z = (int) x; // convertendo double para int
```

- Já o static cast possui a seguinte sintaxe: static_cast <new_type> (expressão);

```
int num = 10;
double numDouble = static_cast<double>(num); // convertendo int para doubl
```

- Qual a diferença entre fazer *type castings* implícitos e explícitos?
- O implícito é mais conveniente em alguns casos, pois não exige a escrita explícita do casting; o explícito é vantajoso quando se deseja ter mais controle sobre a conversão.

Type Casting

Exercícios (3).

⇒ Cada programa deve ser exibir as informações solicitadas igual o exemplo dado abaixo do exercício.

- Escreva um programa que solicite ao usuário que digite um número real representando uma medida em metros. O programa deve converter essa medida para centímetros (inteiro) e imprimir o resultado. Deve obrigatoriamente usar o static_cast.

  Exemplo:

```
Digite a medida em metros: 2.5
2.5 metros equivalem a 250 centímetros.
```

Nomeando Variáveis

- Como prática geral, evite nomes como J23qrsnf e restrinja o uso de nomes de variáveis de apenas uma letra para variáveis que são usadas apenas brevemente.
- Busque usar nomes expressivos, como minhaldade e sualdade.
- C++ distingue entre maiúsculas e minúsculas, portanto, **minhaldade** é uma variável diferente de **minhaidade**.
- Nomes assim são mais fáceis de entender três meses depois, quando você estiver tentando descobrir o que planejava quando escreveu aquela linha de código.
- Por exemplo, considere o seguinte código e identifique o que ele faz.
- Sabemos que calcula a multiplicação de duas variáveis, e daí?
- O que estávamos calculando com esse código?

```
int main() {
    unsigned short x = 10;
    unsigned short y = 11;
    unsigned short z = x * y;
    return 0;
}
```

Nomeando Variáveis

- Como prática geral, evite nomes como J23qrsnf e restrinja o uso de nomes de variáveis de apenas uma letra para variáveis que são usadas apenas brevemente.
- Busque usar nomes expressivos, como minhaldade e sualdade.
- C++ distingue entre maiúsculas e minúsculas, portanto, **minhaldade** é uma variável diferente de **minhaidade**.
- Nomes assim são mais fáceis de entender três meses depois, quando você estiver tentando descobrir o que planejava quando escreveu aquela linha de código.
- Por exemplo, considere o seguinte código e identifique o que ele faz.
- Sabemos que calcula a multiplicação de duas variáveis, e daí?
- O que estávamos calculando com esse código?

```
int main() {
    unsigned short largura = 10;
    unsigned short comprimento = 11;
    unsigned short area = largura * comprimento;
    return 0;
}
```

Nomeando Variáveis

- Vocês podem usar algo como a notação Húngara ou outra notação que acharem interessante.
- Mas também podem criar variáveis sem nenhuma notação específica, desde que busquem seguir a regra geral de clareza e expressividade no nome da variável.

Operadores em C++

- Um operador é um símbolo que atua sobre um valor para realizar cálculos matemáticos ou lógicos.
- Um operador opera sobre os operandos. Por exemplo, considere a instrução *int c = a + b;*.
- Aqui, o símbolo + é o operador de adição. a e b são os operandos que estão sendo somados.
- Vamos conhecer cinco tipos de operadores nesta aula:
  1. Operadores de atribuição.
  2. Operadores aritméticos.
  3. Operadores relacionais.
  4. Operadores lógicos.
  5. Operadores bit-a-bit (bitwise).
- Esses operadores são fundamentais quando manipulando dados e fazendo cálculos em programas escritos em C++.

Operadores de Atribuição em C++

- Esses operadores são usados para atribuir um valor a uma variável.
- O operando do lado esquerdo do operador de atribuição é uma variável, enquanto o operando do lado direito pode ser uma variável ou uma constante.
- O valor do lado direito deve ser do mesmo tipo de dados que a variável do lado esquerdo, caso contrário o compilador avisará.
- Operador de Atribuição '=': atribui o valor do lado direito à variável do lado esquerdo

```
int a = 2; // armazenamos o valor 2 em a
```

- Operador de Adição e Atribuição '+=': Adiciona o valor atual da variável do lado esquerdo ao valor do lado direito e, em seguida, atribui o resultado à variável do lado esquerdo.

```
int a = 2, b = 4;
a += b; // a == 6
```

- Operador de Subtração e Atribuição '-=': Análogo ao anterior.

```
int a = 2, b = 4;
a -= b; // a == -2
```

Operadores de Atribuição em C++

- Operador de Multiplicação e Atribuição '*='. Análogo ao de Adição e Atribuição.

```
int a = 2, b = 4;
a *= b; // a == 8
```

- Operador de Divisão e Atribuição '/='. Análogo ao anterior.

```
int a = 4, b = 2;
a /= b; // a == 2
```

Operadores Aritméticos em C++

- Esses operadores são usados para realizar operações aritméticas nos operandos.
- Por exemplo, o símbolo '+' é usado para adição.
- Podem ser unários (um operando) ou binários (dois operandos).
- Operador de Incremento (++): Aumenta o valor inteiro da variável em um.

```
a = 5;
a++; // a == 6
```

- Operador de Decremento (–): Diminui o valor inteiro da variável em um.

```
a = 5;
a--; // a == 4
```

- Equivalentes a fazer a = a + 1 e a = a - 1.

Operadores Aritméticos em C++

- O operador unário de incremento pode ser colocado antes e depois da variável.
- Em a++, valor da variável é usado na expressão antes de ser incrementado.
- Em ++a, o valor da variável é incrementado antes de ser utilizado na expressão.
- O mesmo acontece para o operador de decremento.

```
#include <iostream>
using namespace std;
int main() {
    int a = 10, b = 15;
    cout << "a++ é " << a++ << endl;
    cout << "++a é " << ++a << endl;
    cout << "b-- é " << b-- << endl;
    cout << "--b é " << --b << endl;
    return 0;
}
```

- Vocês sabem me dizer o que será exibido neste programa, sem testá-lo no compilador?

Operadores Aritméticos em C++

- O operador unário de incremento pode ser colocado antes e depois da variável.
- Em a++, valor da variável é usado na expressão antes de ser incrementado.
- Em ++a, o valor da variável é incrementado antes de ser utilizado na expressão.
- O mesmo acontece para o operador de decremento.

```
#include <iostream>
using namespace std;
int main() {
    int a = 10, b = 15;
    cout << "a++ é " << a++ << endl;
    cout << "++a é " << ++a << endl;
    cout << "b-- é " << b-- << endl;
    cout << "--b é " << --b << endl;
    return 0;
}
```

- Vocês sabem me dizer o que será exibido neste programa, sem testá-lo no compilador?

```
a++ é 10
++a é 12
b-- é 15
--b é 13
```

Operadores Aritméticos em C++

- Operadores Binários: Esses operadores operam ou trabalham com dois operandos. Por exemplo: Adição (+), Subtração (-), etc.
- Operador de Adição '+'. Soma dois operandos.

```
int a = 3, b = 6;
int c = a + b; // c == 9
```

- Subtração '-'. Subtrai o segundo operando do primeiro.

```
int a = 9, b = 6;
int c = a - b; // c == 3
```

- Multiplicação '*'. Multiplica dois operandos.

```
int a = 3, b = 6;
int c = a * b; // c == 18
```

- Divisão '/'. Divide o primeiro operando pelo segundo operando.

```
int a = 12, b = 6;
int c = a / b; // c == 2
```

- Operação de Módulo '%'. Retorna o resto da divisão inteira.

```
int a = 8, b = 6;
int c = a % b; // c == 2
```

Operadores Relacionais em C++

- São usados para a comparação dos valores de dois operandos.
- Por exemplo, o símbolo '>' verifica se um operando é maior que o outro ou não. O resultado retorna um valor Booleano, ou seja, verdadeiro (true) ou falso (false).
- Operador Igual a '=='. Verifica se ambos os operandos são iguais.

```
#include <iostream>
using namespace std;
int main() {
    int a = 3, b = 6;
    bool result = a == b;
    cout << "O resultado de a == b é: " << result << endl;
    return 0;
}
```

- Notem que esse código exibirá 0 (zero), porque em C++ false é igual a zero.

Operadores Relacionais em C++

- Se reescreverem o código assim:

```
#include <iostream>
using namespace std;
int main() {
    bool result = true;
    cout << "O resultado é: " << result << endl;
    return 0;
}
```

- Esse código exibirá 1.
- true e false são constantes de valor 1 e 0, respectivamente.
- Se vocês fizerem:

```
#include <iostream>
using namespace std;
int main() {
    int result = true + true;
    cout << "O resultado é: " << result << endl;
    return 0;
}
```

- O resultado será 2.

Operadores Relacionais em C++

- Operador Maior Que '>'. Verifica se o primeiro operando é maior que o segundo operando.

```
int a = 3, b = 6;
bool result = a > b;
cout << "O resultado de a > b é: " << result << endl; // retorna 0
```

- Operador Maior ou Igual '>='. Verifica se o primeiro operando é maior ou igual ao segundo operando.

```
int a = 3, b = 6;
bool result = a >= b;
cout << "O resultado de a >= b é: " << result << endl; // retorna 0
```

- Operador Menor Que '<'. Verifica se o primeiro operando é menor que o segundo operando.

```
int a = 3, b = 6;
bool result = a < b;
cout << "O resultado de a < b é: " << result << endl; // retorna 1
```

Operadores Relacionais em C++

- Operador Menor ou Igual '<='. Verifica se o primeiro operando é menor ou igual ao segundo operando.

```
int a = 3, b = 6;
bool result = a <= b;
cout << "O resultado de a <= b é: " << result << endl; // retorna 1
```

- Operador Diferente de '!='. Verifica se ambos os operandos são diferentes.

```
int a = 3, b = 6;
bool result = a != b;
cout << "O resultado de a != b é: " << result << endl; // retorna 1
```

Operadores Lógicos em C++

- São usados para combinar duas ou mais condições ou para complementar a avaliação da condição original em consideração.
- O resultado retorna um valor booleano, ou seja, verdadeiro ou falso.
- AND Lógico '&&'. Retorna verdadeiro apenas se todos os operandos são verdadeiros, ou seja, diferentes de zero.

```
int a = 3, b = 6;
bool result = a && b;
cout << "O resultado de a && b é: " << result << endl; // retorna 1
```

- OR Lógico '||'. Retorna verdadeiro se pelo menos um dos operandos é verdadeiro, ou seja, diferente de zero.

```
int a = 3, b = 6;
bool result =  a || b;
cout << "O resultado de a || b é: " << result << endl; // retorna 1
```

- NOT Lógico '!'. Retorna verdadeiro se o operando é falso, ou seja, igual a zero. Caso contrário, retorna falso (zero).

```
int a = 3;
bool result = !a;
cout << "O resultado de !a é: " << result << endl; // retorna 0
```

Operadores Lógicos em C++

- AND Binário '&'. Copia um bit para o resultado avaliado se ele existir em ambos os operandos.

```
int a = 2, b = 3;
cout << "(a & b) retorna: " << (a & b) << endl;
```

- OR Binário '|'. Copia um bit para o resultado avaliado se ele existir em qualquer um dos operandos.

```
int a = 2, b = 3;
cout << "(a | b) retorna: " << (a | b) << endl;
```

- XOR Binário '^'. Copia o bit para o resultado avaliado se ele estiver presente em qualquer um dos operandos, mas não em ambos.

```
int a = 2, b = 3;
cout << "(a ^ b) retorna: " << (a ^ b) << endl;
```

- Deslocamento à Esquerda '<<'. Desloca o valor à esquerda pelo número de bits especificado pelo operando à direita.

```
int a = 2, b = 3;
cout << "(a << 1) retorna: " << (a << 1) << endl;
```

- Deslocamento à Direita '>>'. Desloca o valor à direita pelo número de bits especificado pelo operando à direita.

```
int a = 2, b = 3;
cout << "(a >> 1) retorna: " << (a >> 1) << endl;
```

Operadores Lógicos em C++

- AND Binário '&'. Faz a operação AND bit a bit entre os dois operandos.

```
int a = 2, b = 3;
cout << "(a & b) retorna: " << (a & b) << endl; // retorna 2 (10 AND 11 == 10)
```

- OR Binário '|'. Faz a operação OR bit a bit entre os dois operandos.

```
int a = 2, b = 3;
cout << "(a | b) retorna: " << (a | b) << endl; // retorna 3 (10 OR 11 == 11)
```

- XOR Binário '^'. Faz a operação XOR bit a bit entre os dois operandos.

```
int a = 2, b = 3;
cout << "(a ^ b) retorna: " << (a ^ b) << endl; // retorna 1 (10 OR 11 == 01)
```

- Deslocamento à Esquerda '<<'. Desloca o valor à esquerda pelo número de bits especificado pelo operando à direita.

```
int a = 2, b = 3;
cout << "(a << 1) retorna: " << (a << 1) << endl; /*retorna 4 (10 deslocado
                                                    para esquerda 1 bit == 100)*/
```

- Deslocamento à Direita '>>'. Desloca o valor à direita pelo número de bits especificado pelo operando à direita.

```
int a = 2, b = 3;
cout << "(a >> 1) retorna: " << (a >> 1) << endl; /*retorna 1 (10 deslocado
                                                    para esquerda 1 bit == 01)*/
```

Operadores em C++

- Existem outros operadores, como o operador ternário ou condicional, o operador ->, o operador *, o operador & (não é o bitwise, este é unário), o operador ., o operador sizeof (não é uma função, apesar de muitos acharem que é – é um operador em tempo de compilação), etc.
- Esta não é uma lista exaustiva, mas um início em seu aprendizado de C++.
- Aprenderemos mais operadores futuramente.

Precedência de Operadores em C++

- O que é executado primeiro nessa expressão? O resultado mudará, dependendo da precedência.

```
x = 5 + 3 * 8;
```

- Em C++, o operador de multiplicação tem precedência sobre o operador de adição.
- Quando dois operadores tem a mesma precedência eles são executados da esquerda para a direita.

```
x = 5 + 3 + 8 * 9 + 6 * 4;
```

- Temos que 8*9 == 72 e 6*4 == 24.
- Agora a adição: (esquerda para a direita) 5+3 == 8; 8+72 == 80; 80+24 == 104.
- Vocês podem encontrar tabelas em livros, e na internet, detalhando as precedências entre operadores em C++.

Precedência de Operadores em C++

- Por exemplo, essa tabela:

Rank	Name	Operator
1	Scope resolution	`::`

2	Member selection, subscripting, function calls, postfix increment and decrement	`. ->`
		`()`
		`++ --`

3	`Sizeof`, prefix increment and decrement, complement, and, not, unary minus and plus, address-of and dereference, `new`, `new[]`, `delete`, `delete[]`, casting, `sizeof()`	`++ --`
		`^ !`
		`- +`
		`& *`
		`()`

4	Member selection for pointer	`.* ->*`
5	Multiply, divide, modulo	`* / %`
6	Add, subtract	`+ -`
7	Shift (shift left, shift right)	`<< >>`
8	Inequality relational	`< <= > >=`
9	Equality, inequality	`== !=`
10	Bitwise AND	`&`
11	Bitwise exclusive OR	`^`
12	Bitwise OR	`|`
13	Logical AND	`&&`
14	Logical OR	`||`
15	Conditional	`?:`

16	Assignment operators	`= *= /= %=`
		`+= -= <<=`
		`>>=`
		`&= |= ^=`

17	Comma	`,`

- Para não ter que decorar essa tabela, use parênteses para definir a precedência como desejado.

A Declaração if

- A declaração 'if' permite que você teste uma condição e desvie, dependendo do resultado.
- Ou seja, é um **desvio condicional**.
- Podemos usar desvios condicionais para alterar o fluxo de nosso programa.
- Para isso precisamos avaliar **condições**.
- Uma condição é uma expressão que, quando avaliada, gera um resultado verdadeiro (1) ou falso (0).
- Podemos usar, por exemplo, os operadores relacionais que aprendemos para construir condições que usaremos com o nossa declaração if.
- A sintaxe da declaração if é a seguinte:

```
if (condicao) {
    // Código a ser executado se a condição for verdadeira
}
```

- Nessa declaração, *condicao* é a expressão que será avaliada.
- Se esta for verdadeira, o bloco de código dentro das chaves será executado.
- Caso contrário, o bloco será ignorado e o programa continuará a partir da próxima linha após o bloco 'if' (lembrem que usamos chaves em C++ para delimitar blocos de código).

A Declaração if

- Podemos também conectar condições usando os operadores lógicos que aprendemos.

```
if (condicaoUm && condicaoDois) {
    // Código a ser executado se ambas as condições forem verdadeiras
}
```

- Vocês precisam lembrar as tabelas-verdade dos operadores lógicos para conseguir usar corretamente os operadores lógicos como conectivos de condições.
- Vamos ver um exemplo prático de um uso de 'if'.

```
#include <iostream>
using namespace std;
int main() {
    int idade;
    cout << "Digite a sua idade: ";
    cin >> idade;
    if(idade >= 18)
        cout << "\nVocê possui a maioridade civil no Brasil!";
    cout << "\nVocê tem " << idade << " anos";
    return 0;
}
```

- Notem que não é necessário usar as chaves neste caso, porque temos apenas uma instrução.

A Declaração else

- A declaração 'else' permite definirmos um outro fluxo para o nosso programa, caso a condição do 'if' for avaliada como falsa.
- O else é utilizado junto com o if, com a seguinte sintaxe.

```
if (condicao) {
    // Código a ser executado se a condição for verdadeira
}
else {
    /* Código que será executado se a condição do if foi avaliada
       como falsa*/
}
```

- Notem que não avaliamos nenhuma condição no else.
- O fluxo do código irá desviar para dentro do else se o fluxo **não** foi desviado para o if.
- Da mesma forma, se o código desviar para dentro do if, ele não desviará em seguida para dentro do else.
- Ou seja, se não entrou no if, entrará no else. Se entrar no if, não entrará no else.

As Declarações if e else

- Vamos ver um exemplo de uso dessas declarações.

```
#include <iostream>
using namespace std;
int main() {
    int idade;
    cout << "Digite a sua idade: ";
    cin >> idade;
    if(idade >= 18)
        cout << "\nVocê possui a maioridade civil no Brasil!";
    else
        cout << "\nVocê ainda não possui a maioridade civil no Brasil.";
    cout << "\nVocê tem " << idade << " anos";
    return 0;
}
```

As Declarações if e else

- Outro exemplo.

```
#include <iostream>
using namespace std;
int main() {
    int nota;
    cout << "Digite a nota do aluno: ";
    cin >> nota;
    if (nota >= 60)
        cout << "Aprovado" << endl;
    else {
        if (nota >= 40)
            cout << "IFA" << endl;
        else
            cout << "Reprovado" << endl;
    }
    return 0;
}
```

Começando a Programar em C++

Exercícios (4).

⇒ Declarações if else

- Escreva um programa que solicite um número ao usuário e verifique se é par ou ímpar.
- Escreva um programa que solicite um número ao usuário e verifique se é positivo, zero, ou negativo.
- Escreva um programa que solicite três números ao usuário e verifique qual deles é o maior e qual deles é o menor. Caso dois deles, ou os três, sejam iguais, deve fornecer essa informação.
- Escreva um programa que solicite um ano ao usuário e verifique se é um ano bissexto (descubram como!).

Começando a Programar em C++

Exercícios (5).

⇒ Operadores Lógicos

- Escreva um programa que solicite ao usuário que insira três números: os dois primeiros definirão um intervalo, e o terceiro número será verificado para determinar se está dentro deste intervalo. O programa utilizará operadores lógicos conectando condições.
- Escreva um programa que verifique se um número é positivo e não é múltiplo de 3. O programa utilizará operadores lógicos conectando condições.

Começando a Programar em C++

Exercícios (6).

⇒ Operadores bit-a-bit

- Escreva um programa que solicite ao usuário um número e o multiplique por dois usando operadores bit-a-bit.
- Escreva um programa que solicite ao usuário um número e verifique se é par ou ímpar usando operadores bit-a-bit.
- Estude o código a seguir e explique o que ele faz e como funciona.

```
#include <iostream>
using namespace std;
int main() {
    int a, b;
    cout << "Digite o valor de a: ";
    cin >> a;
    cout << "Digite o valor de b: ";
    cin >> b;
    a = a ^ b;
    b = a ^ b;
    a = a ^ b;
    return 0;
}
```