Introdução a linguagem C++

Profa. Regiane Kawasaki

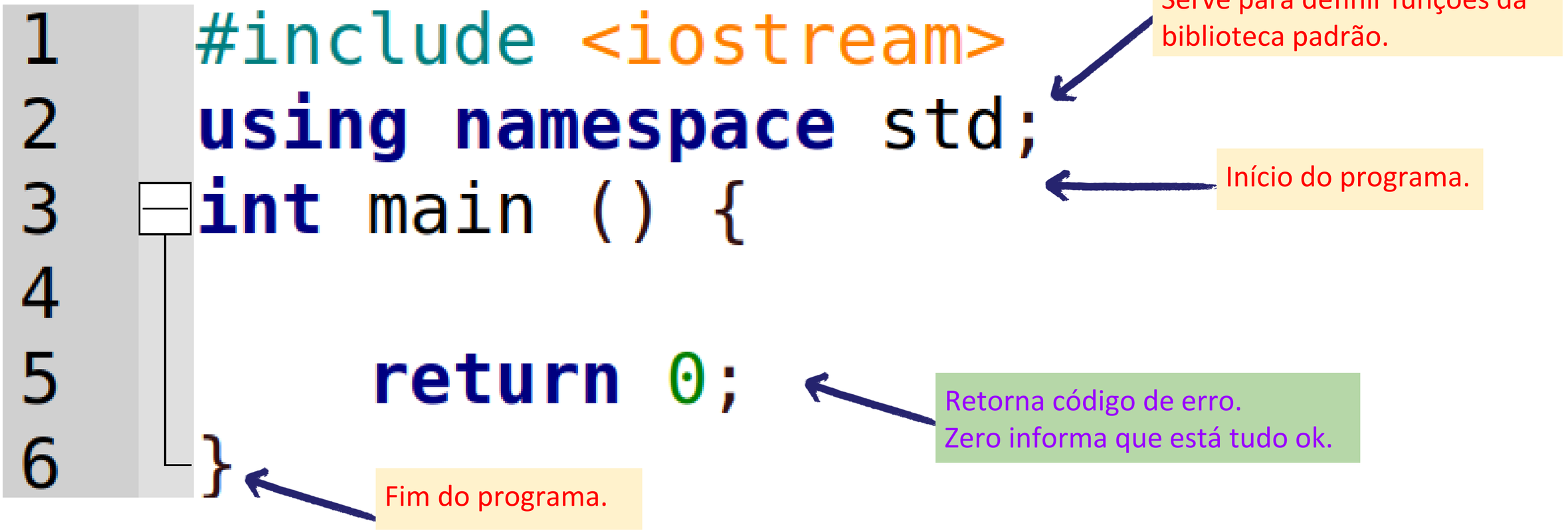
kawasaki@ufpa.br

Estrutura básica de um programa em C++

```
1  #include <iostream>
2  using namespace std;
3  int main () {
4
5      return 0;
6  }
```

Estrutura básica de um programa em C++

Declaração de bibliotecas usadas no programa.



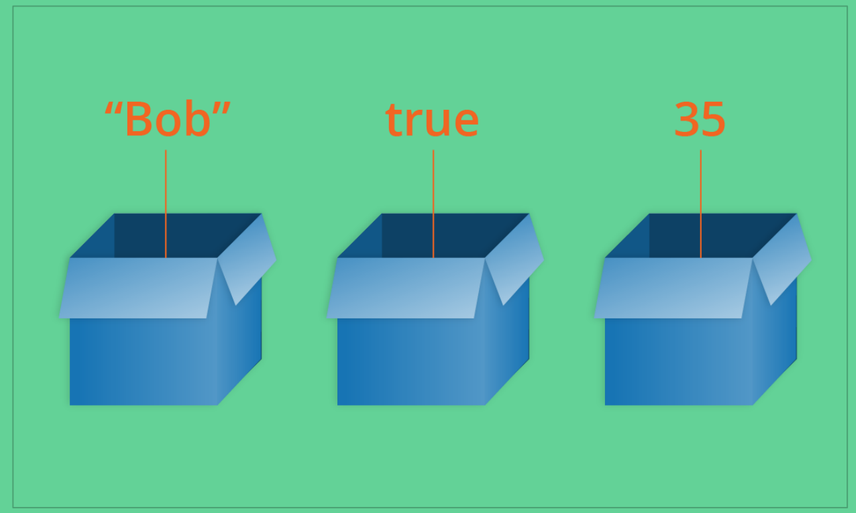
Serve para definir funções da biblioteca padrão.

Início do programa.

Retorna código de erro.
Zero informa que está tudo ok.

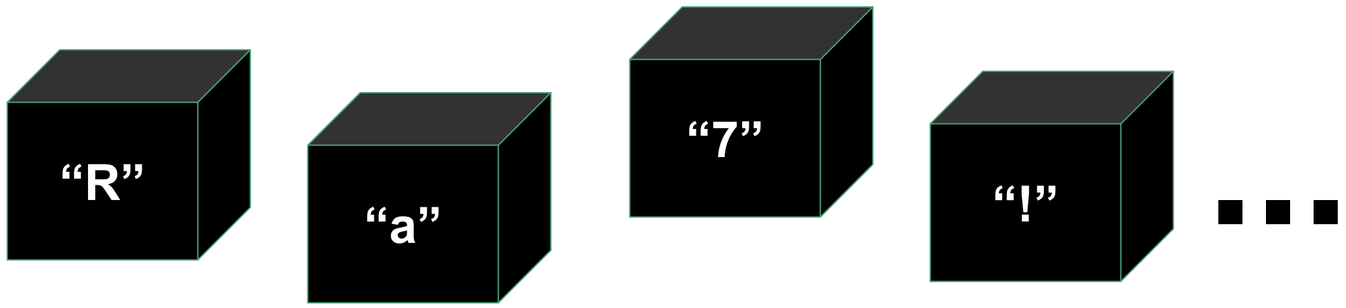
Fim do programa.

Tipos de dados



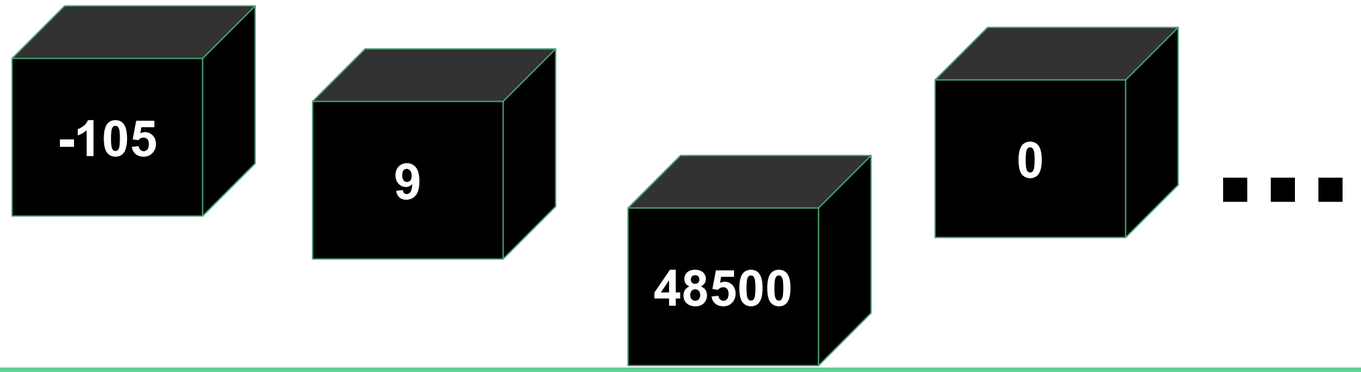
Tipos de dados fundamentais em C++

- **char**
  - Utilizado para representar valores alfanuméricos unitários (ou seja: letras, algarismos e símbolos), também chamados de caracteres;
  - Espaço ocupado na memória: geralmente 1 byte;
  - Internamente, caracteres são representados por números inteiros compreendidos no intervalo [-128, 127].



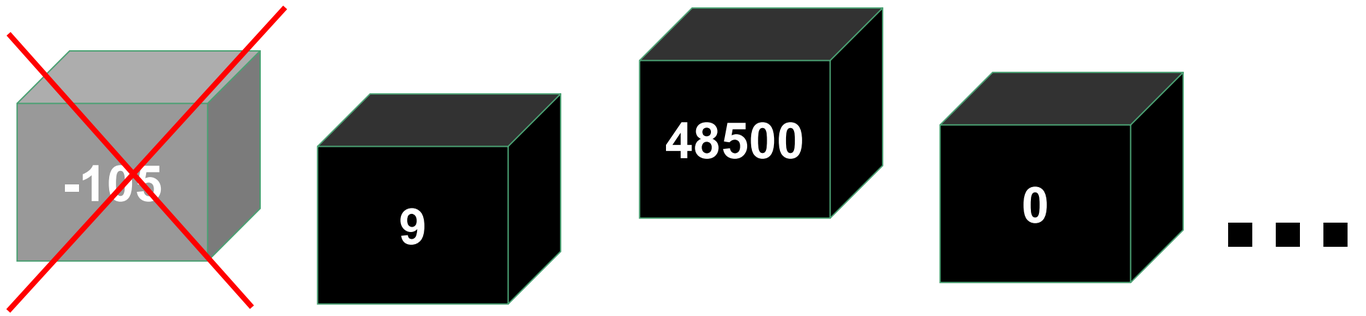
Tipos de dados fundamentais em C++

- **int**
  - Utilizado para representar valores numéricos inteiros ($\mathbb{Z}$);
  - Embora possa existir diferenças entre versões distintas de compiladores, usualmente o tipo int é usado para denotar números inteiros no intervalo de -2.147.483.648 até 2.147.483.647;
  - Padrão especifica ao menos 2 bytes para representação.



Tipos de dados fundamentais em C++

- **unsigned (ou unsigned int)**
  - Utilizado para representar valores numéricos naturais (ℕ);
  - Embora possa existir diferenças entre versões distintas de compiladores, usualmente o tipo unsigned é usado para denotar números naturais no intervalo de 0 até 2.147.483.647.



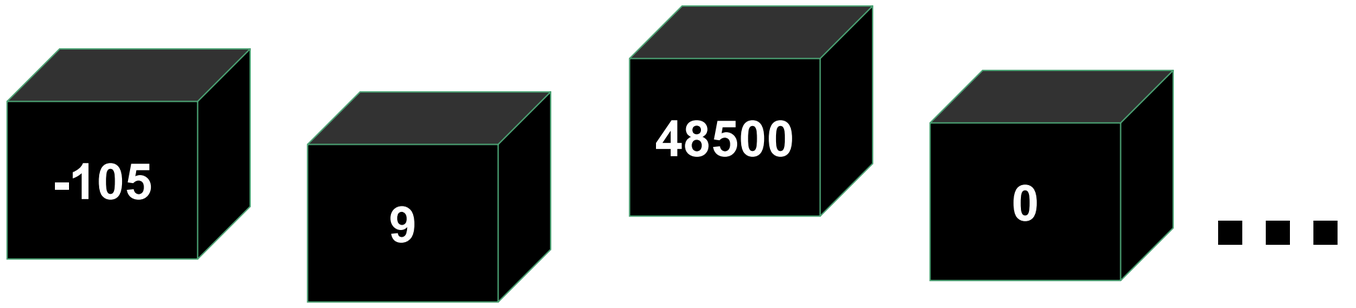
Tipos de dados fundamentais em C++

- **short (ou short int)**

Utilizado para representar valores numéricos inteiros com uma faixa de precisão menor que int (depende do compilador);

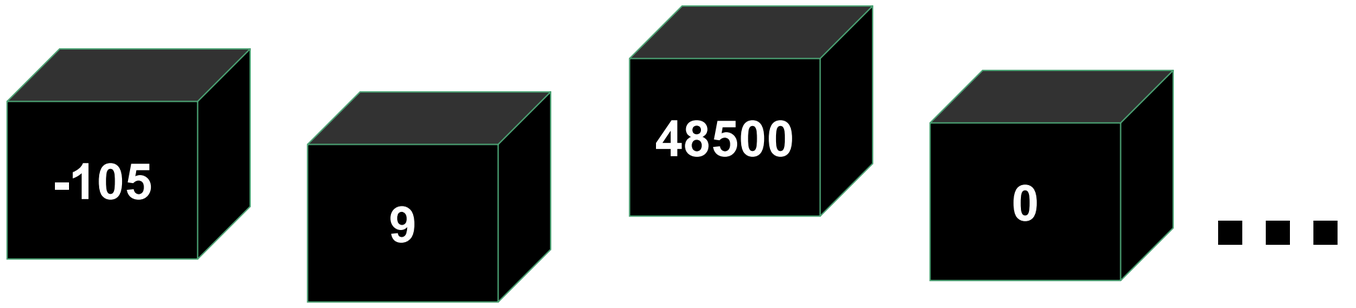
Também pode ser usado sem sinal: **unsigned short**;

Ao menos 2 bytes para representação.



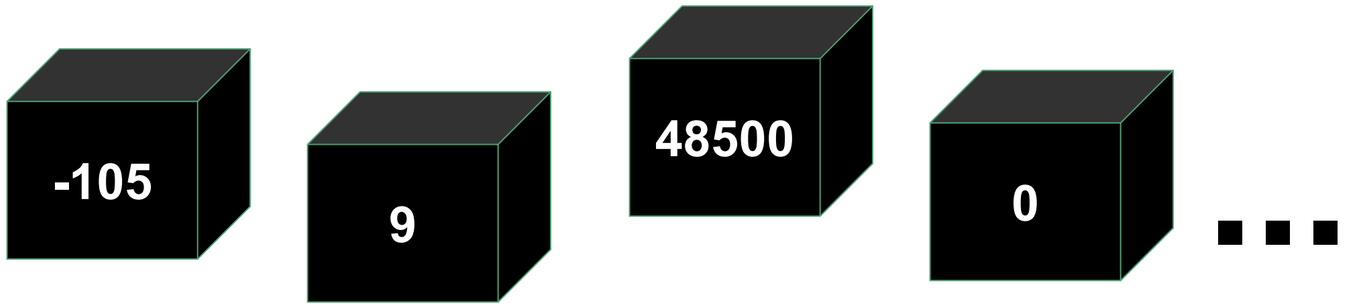
Tipos de dados fundamentais em C++

- **long (ou long int)**
  - Utilizado para representar valores numéricos inteiros com uma faixa de precisão maior que int (depende do compilador);
  - Também pode ser usado sem sinal: **unsigned long**;
  - Ao menos 4 bytes para representação.



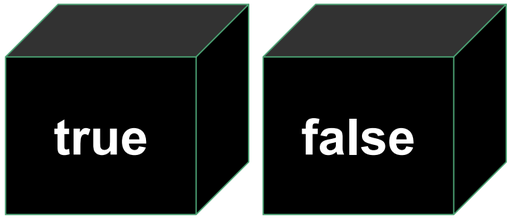
Tipos de dados fundamentais em C++

- **long long (ou long long int)**
  - Utilizado para representar valores numéricos inteiros com uma faixa de precisão maior que long (depende do compilador);
  - Também pode ser usado sem sinal: **unsigned long long**;
  - Ao menos 8 bytes para representação.



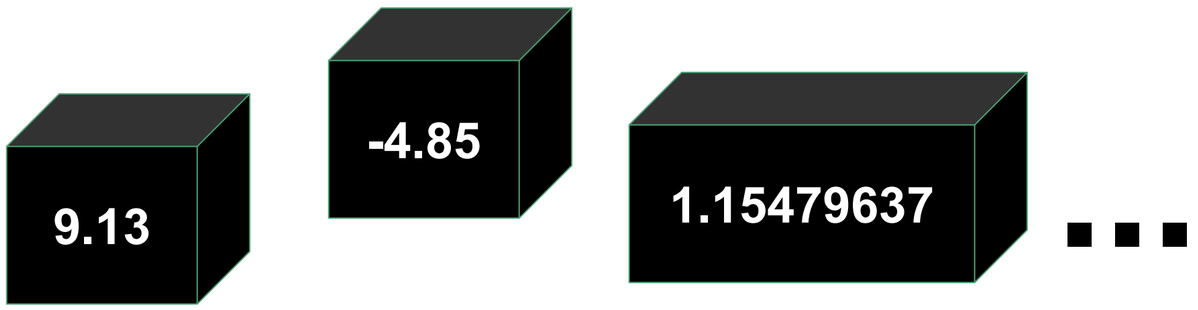
Tipos de dados fundamentais em C++

- **bool**
  - Utilizado para representar dados lógicos. Também chamados de booleanos (referência à teoria de George Boole);
  - Há apenas dois valores possíveis: true (verdadeiro) ou false (falso);
  - Dados lógicos geralmente são empregados para expressar o resultado de uma condição. Por exemplo: o fato de que 10 > 9 é verdadeiro (true).



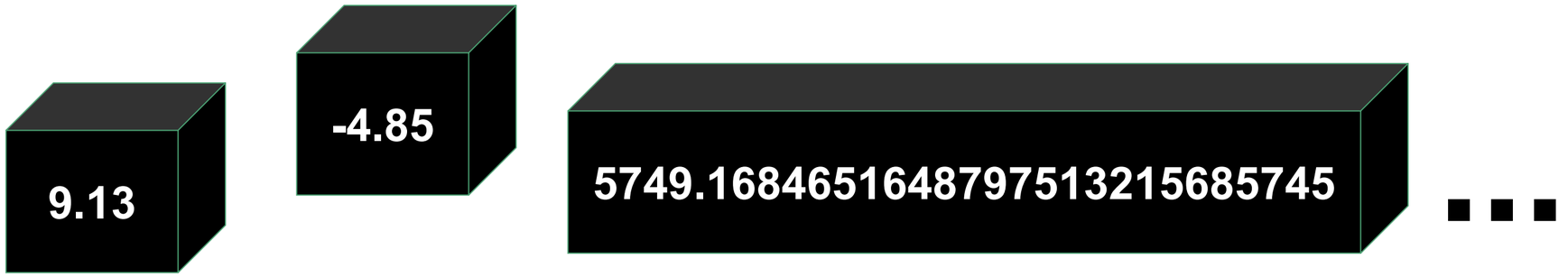
Tipos de dados fundamentais em C++

- **float**
  - Utilizado para representar valores numéricos em ponto flutuante com precisão simples, representados com pelo menos 4 bytes;
  - Embora possa existir diferenças entre versões distintas de compiladores, a faixa de valores do tipo float varia de 1.2×10^{-38} até 3.4×10^{38}.



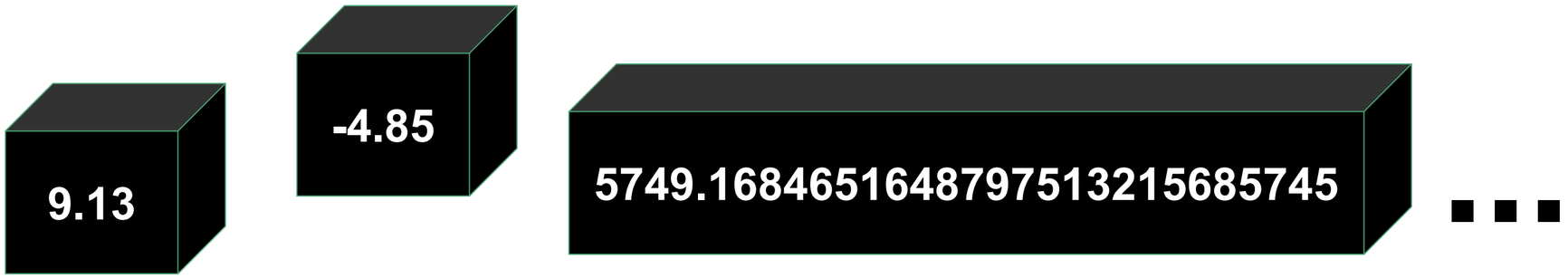
Tipos de dados fundamentais em C++

- **double**
  - Utilizado para representar valores numéricos em ponto flutuante com precisão dupla (pelo menos 8 bytes);
  - Embora possa existir diferenças entre versões distintas de compiladores, a faixa de valores do tipo double varia de 2.2×10^{-308} até 1.8×10^{308}.



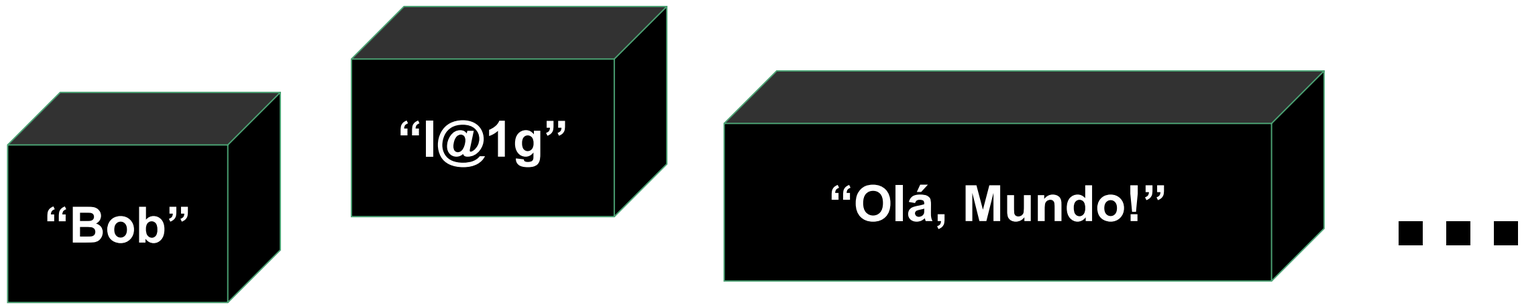
Tipos de dados fundamentais em C++

- **long double**
  - Utilizado para representar valores numéricos em ponto flutuante com precisão maior que a dupla (pelo menos 10 bytes).



Tipos de dados fundamentais em C++

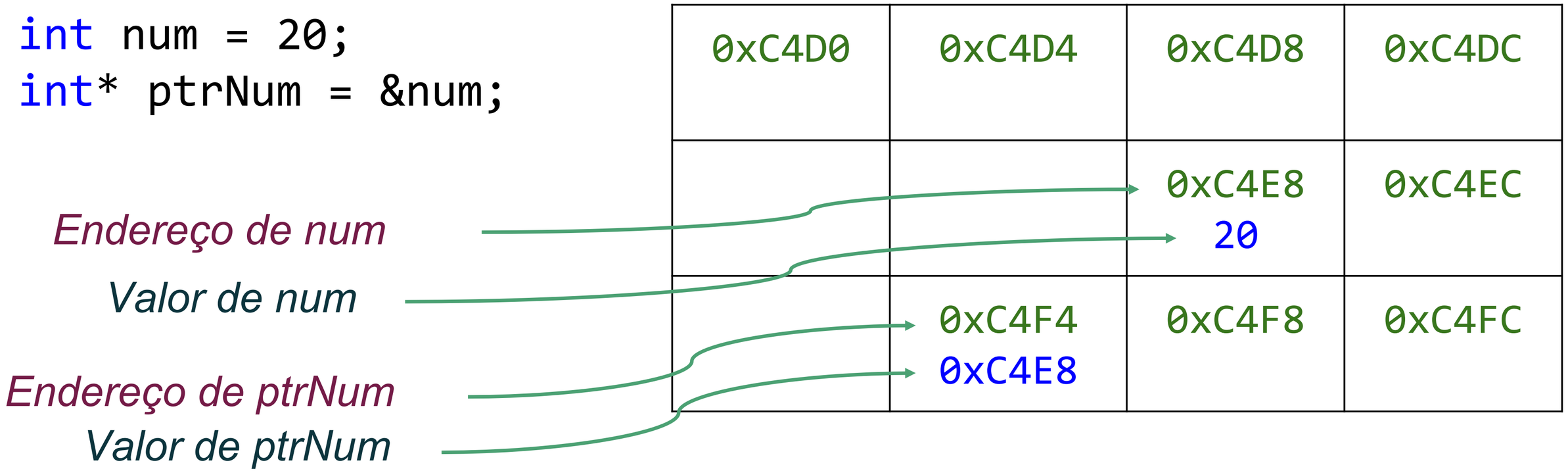
- **string**
  - Utilizado para representar textos;
  - A diferença entre os tipos fundamentais string e char está no tamanho da memória utilizado para armazenar a informação e nas operações que podem ser realizadas sobre cada tipo.



Tipos de dados fundamentais em C++

- **ponteiro**
  - São variáveis que armazenam endereços de memória (ou seja, o endereço de outras variáveis).

```
int num = 20;
int* ptrNum = &num;
```



Uso de Ponteiros

Existem diversos usos para ponteiros em programação, destacando-se:

- Acesso indireto a valores, muitas vezes sem necessidade de uma variável
- Casting dinâmico de variáveis - (acesso de uma variável de um tipo como se fosse de outro)
- Alocação e gerenciamento dinâmicos de memória

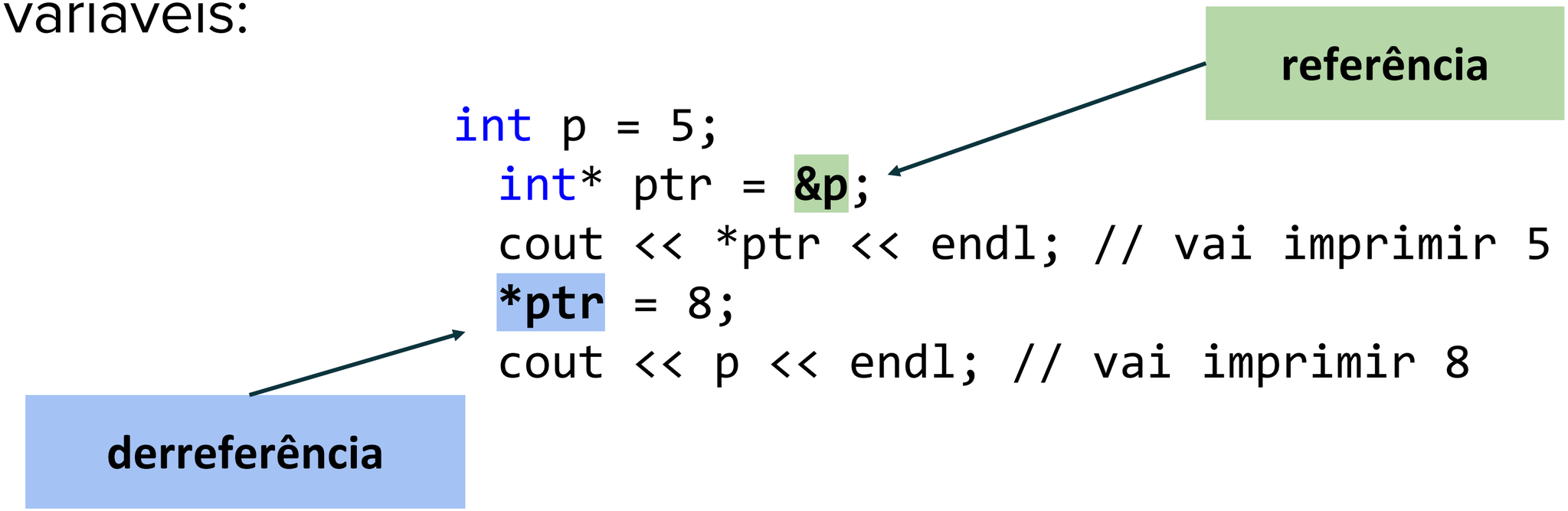
Indireção

É possível utilizar ponteiros para acessar os valores de outras variáveis:

```
int p = 5;
  int* ptr = &p;
  cout << *ptr << endl; // vai imprimir 5
  *ptr = 8;
  cout << p << endl; // vai imprimir 8
```

Indireção

É possível utilizar ponteiros para acessar os valores de outras variáveis:

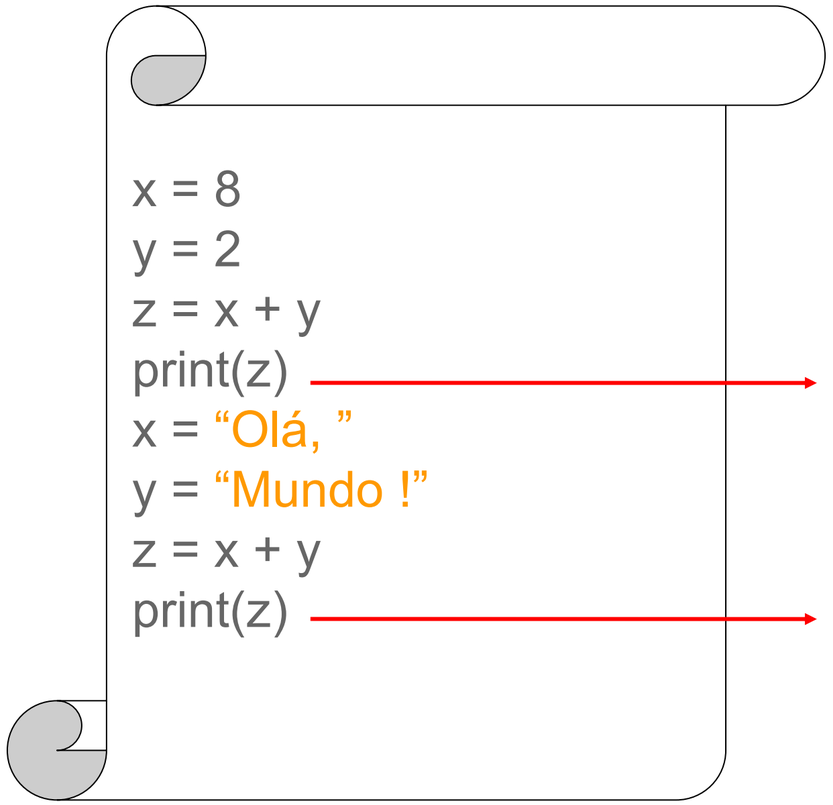


Tipificação (forte ou fraca)

- Diferentes linguagens de programação podem tratar a tipificação das variáveis de formas distintas:
  - Podem permitir que as **variáveis assumam valores de diferentes tipos** de dados ao longo da execução do programa (conhecidas como linguagens fracamente tipadas);
  - Ou, podem exigir que uma vez que os **tipos das variáveis sejam definidos**, os mesmos não possam ser modificados durante a execução do programa (conhecidas como linguagens fortemente tipadas).

Tipificação (forte ou fraca)

- Python 3.5



A linguagem permite mudar o tipo de dado armazenado em uma variável durante a execução do programa.

***Mensagem exibida:* 10**

***Mensagem exibida:* “Olá, Mundo!”**

Tipificação (forte ou fraca)

- C++

A linguagem **não** permite mudar o tipo de dado armazenado em uma variável.

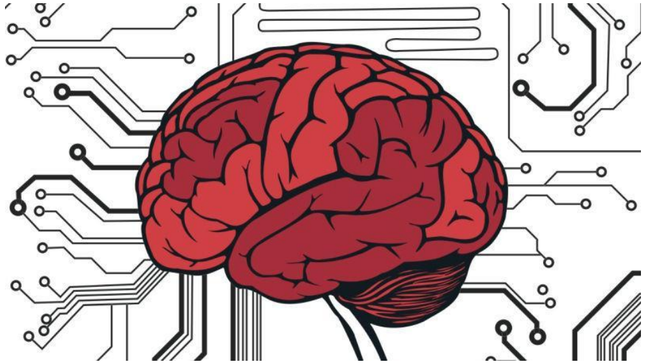
```
#include <iostream>
using namespace std;

int main()
{
    int x, y;
    x = 2;
    y = 8;
    cout << x;
    x = “Olá, “;
    return 0;
}
```

Erro de sintaxe!

*De acordo com a declaração da variável **x**, a mesma pode armazenar apenas valores numéricos inteiros. Portanto não é possível fazê-la armazenar uma informação do tipo textual. Neste tipo de situação, o próprio compilador de C++ acusa que o programa possui um erro de sintaxe que precisa ser obrigatoriamente corrigido.*

Declaração de variáveis

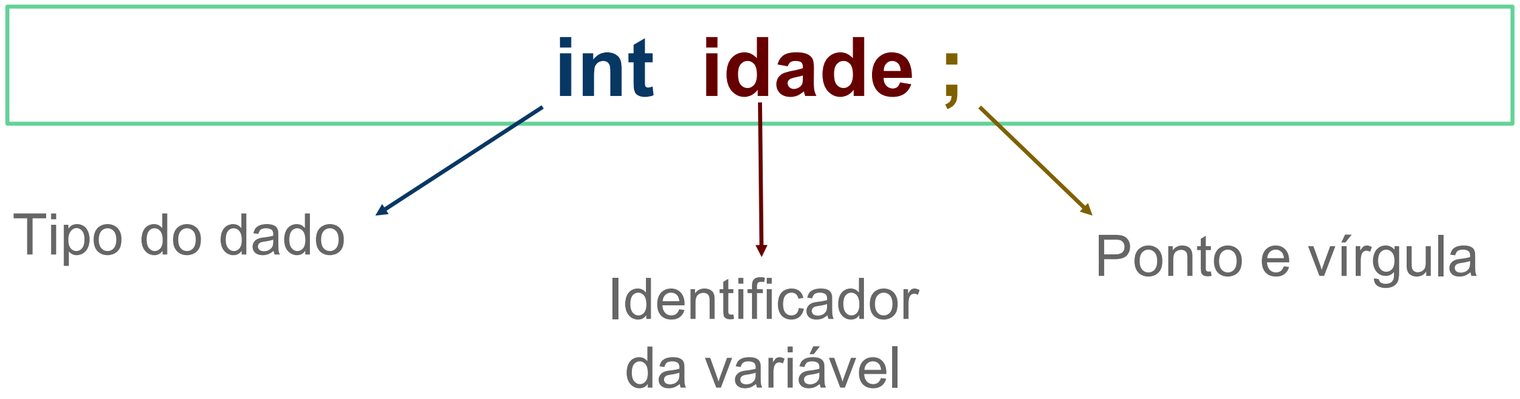
Declaração de variáveis



- **Definição**
  - Instrução em um programa que reserva um espaço na memória, por meio de um identificador, para o armazenamento de uma determinada informação.
- Em C++, para se declarar uma variável deve-se definir (obrigatoriamente):
  - O tipo de dado que a variável armazenará;
  - Um identificador (nome) para a variável.

Exemplo de declaração de uma variável

Assuma, por exemplo, que precisamos declarar uma variável para armazenar a informação da idade de um usuário. Para realizar a declaração desta variável, poderíamos fazer em C++:



Inicialização de variáveis

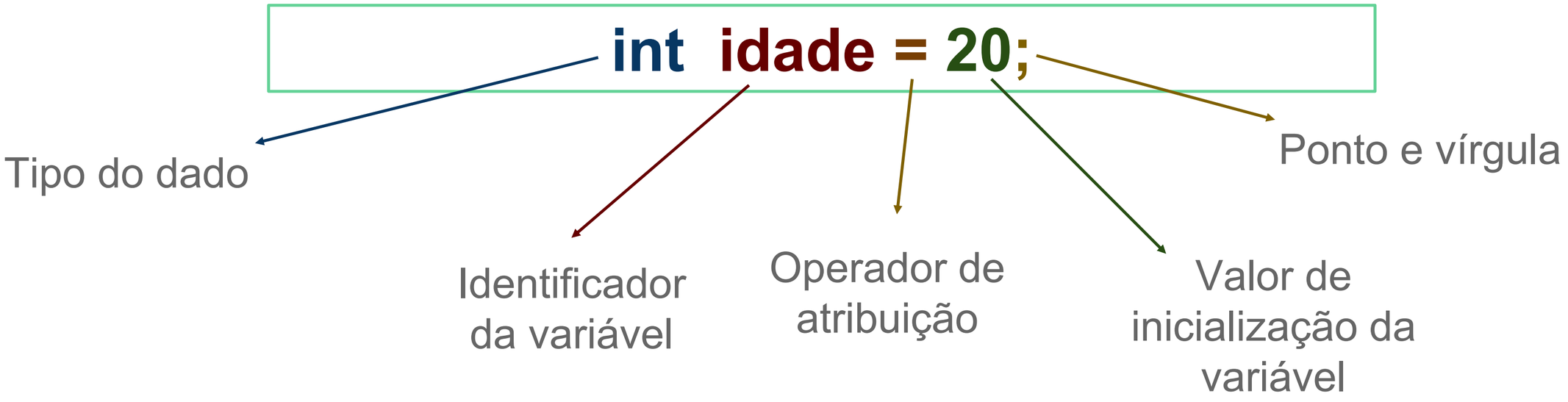
Além da declaração anterior, em C++ também é possível declarar uma variável **armazenando um valor inicial em seu endereço de memória** (processo este chamado de inicialização). Tal valor pode ser futuramente alterado, desde que a nova informação possua a mesma natureza do tipo de dado declarado originalmente.

Sintaxe em C++ com inicialização:

```
tipo_do_dado  identificador_da_variável = valor_inicial;
```

Exemplo de declaração com inicialização

Assuma, novamente, que precisamos declarar uma variável para representar a idade de um usuário. Contudo, considere desta vez, que também desejamos que esta variável armazene o valor numérico 20. Em C++ teríamos então:



Constantes

Em determinadas situações, é interessante garantir em nível de programação que o valor de inicialização de uma determinada variável não será alterado durante a execução do programa. Para isto, basta declarar uma variável do gênero constante, inserindo a palavra reservada **const** antes do tipo de dado da variável.

Sintaxe em C++ para declaração de constantes:

```
const   tipo_do_dado   identificador_da_variável   =   valor;
```

Constantes - Exemplo

Considere, por exemplo, que estamos interessados em fazer um programa para lidar com relações trigonométricas. Neste nosso programa, poderíamos então criar uma variável para representar o valor em ponto flutuante do número pi (Π). Como o valor desta variável não precisará ser alterado ao longo da execução do programa, poderíamos declara-la como sendo uma constante. Em C++ teríamos então:

```
const float pi = 3.14159;
```

Desta forma, qualquer tentativa de alterar o valor da variável pi provocaria um erro de sintaxe, uma vez que a mesma é uma constante.

Definição de identificadores de variáveis

- Para definir o nome de uma variável, um programador C++ deve seguir obrigatoriamente um conjunto de regras. As regras básicas para definição de identificadores são:
  - (1º): Todo identificador pode conter apenas: letras do alfabeto romano, números e o símbolo de sublinhado (*underscore*);
  - (2º): Um identificador não pode começar com um dígito (número);
  - (3º): Exceto em casos especiais, duas variáveis diferentes não podem ter o mesmo nome.

Identificadores - exemplos

Exemplo de identificadores que **são** válidos em C++:

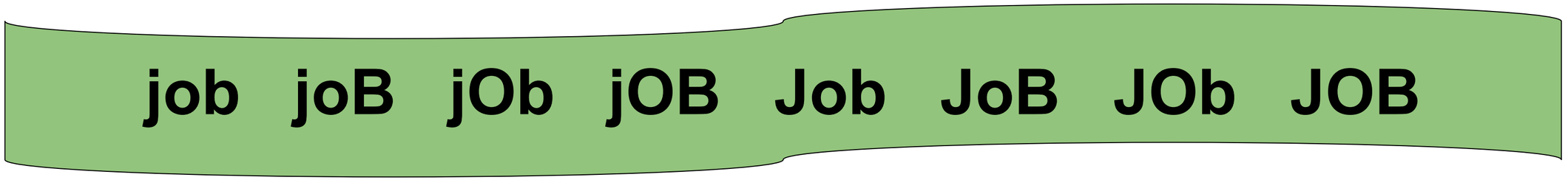
idade **pi** **x** **_nome** **valor_Funcao**

Exemplo de identificadores que **não** são válidos em C++:

id@de **p!** **1x** **%nome** **valor..Funcao**

Definição de identificadores de variáveis

Cuidado 1: C++ diferencia letras maiúsculas e minúsculas. Identificadores diferentes podem ser criados apenas trocando a capitalização das letras. Por exemplo, todos os identificadores abaixo são diferentes:



Note que como os identificadores acima são diferentes entre si, cada um deles poderia ser utilizado em um programa para armazenar informações completamente diferentes, inclusive de naturezas (tipos de dados) diferentes.

Definição de identificadores de variáveis

Cuidado 2: C++ utiliza palavras reservadas que possuem significado semântico especial para a linguagem. Deste modo, estas palavras reservadas **não** podem ser utilizadas como identificadores. Por exemplo, todas as palavras a seguir são reservadas e não podem ser utilizadas como identificadores:

if main break int float for while

Estes são apenas alguns exemplos, existem muitas outras palavras reservadas em C++.

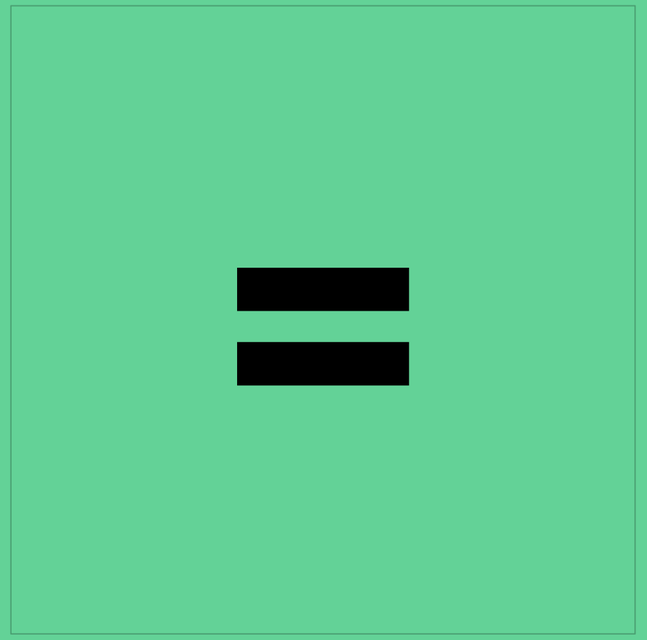
Recomendação

Procure usar nomes de variáveis significativos, auto-explicativos.

Por exemplo: `totalFuncionarios`, `salarioPrincipal`, `soma_intermediaria` são nomes de variáveis + sugestivos que `tf`, `s` e `SI`, por exemplo.

Use letras maiúsculas ou sublinhado para separar palavras ao escrever variáveis que teriam mais que uma palavra no nome.

Operador de atribuição



Operador de atribuição

- Para que uma variável possa ser utilizada em um programa, ela deve ter um valor associado a ela.
- Em programação, o ato de associar um determinado valor a uma variável é denominado atribuição.
- **Sintaxe em pseudocódigo:**

identificador_da_variável ← valor ;

Operador de atribuição

Operador de atribuição

- Para que uma variável possa ser utilizada em um programa, ela deve ter um valor associado a ela.
- Em programação, o ato de associar um determinado valor a uma variável é denominado atribuição.
- **Sintaxe em C++:**

```
identificador_da_variável = valor ;
```

Operador de atribuição

Operador de atribuição

- Exemplos:

```
nome =  “Belém”;
idade = 20;
pi = 3.14159;
```

- Estes valores permanecem associados às variáveis correspondentes até que o programa os altere por meio de uma nova atribuição ou até que o programa como um todo se encerre.

Operador de atribuição

identificador_da_variável = valor ;

- De modo geral, o lado direito de um operador de atribuição pode ser:
  - Um valor fixo. Exemplo:

```
idade = 20;
```

Variável **idade** recebe o valor fixo 20

  - Uma outra variável. Exemplo:

```
ladoA = 7;
ladoB = ladoA;
```

Variável **ladoA** recebe o valor fixo 7

Variável **ladoB** recebe o valor que está armazenado na variável **ladoA**

Operador de atribuição

identificador_da_variável = valor ;

- De modo geral, o lado direito de um operador de atribuição pode ser:
  - Uma expressão formada por valores fixos e/ou variáveis. Exemplo:

```
base = 2;
altura = 9;
area = (base*altura)/2;
```

Variável **base** recebe o valor fixo 2

Variável **altura** recebe o valor fixo 9

Variável **area** recebe o valor resultante da multiplicação dos valores armazenados nas variáveis **base** e **altura** dividido pelo valor fixo 2

Cuidados ao fazer uma atribuição

- (1º) Números em ponto flutuante contêm um ponto para denotar as casas decimais (notação americana para separar casas decimais).

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

int main()
{
    float pi;
    pi = 3.14;
    return 0;
}
```

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

int main()
{
    float pi;
    pi = 3,14;
    return 0;
}
```

Cuidados ao fazer uma atribuição

- (2º) Quando se atribui um valor em ponto-flutuante à uma variável inteira, ocorre uma conversão e podem ocorrer perdas.

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

int main()
{
    int x;
    x = 20.743;
    return 0;
}
```

Variável **x** passa a armazenar o valor inteiro 20. Neste caso, a parte decimal é perdida e o valor é truncado. Note que não há arredondamento.

Cuidados ao fazer uma atribuição

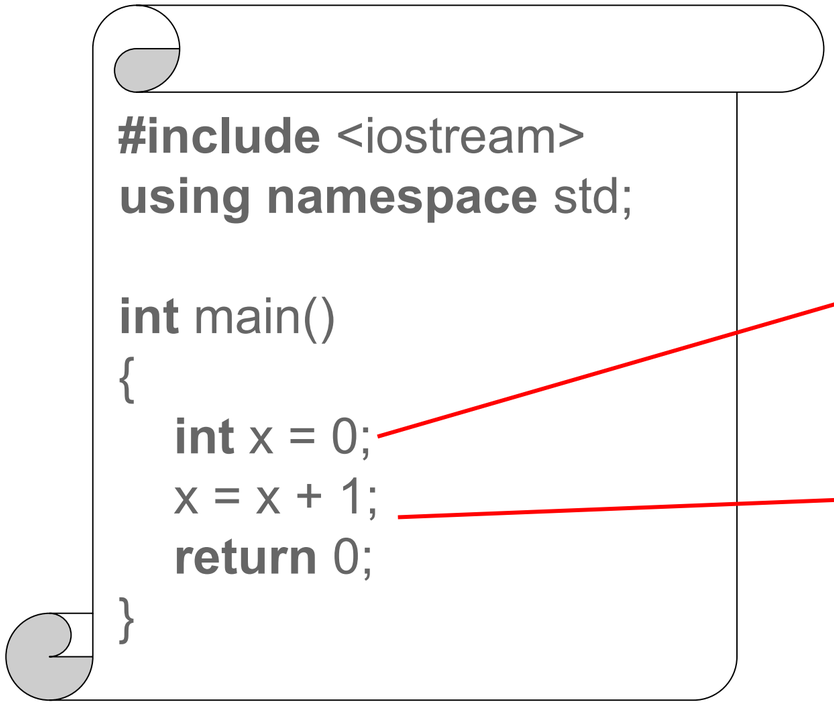
- (3º) Quando se atribui um valor inteiro a uma variável em ponto-flutuante, ocorre uma conversão e podem ocorrer perdas.

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

int main()
{
    int i = 20;
    float x;
    x = i;
    return 0;
}
```

Variável **x** passa a armazenar o valor flutuante 20. Neste caso, a representação é feita usando aproximações e podem ocorrer erros caso seja necessário um valor inteiro, para comparação, por exemplo.

Cuidados ao fazer uma atribuição

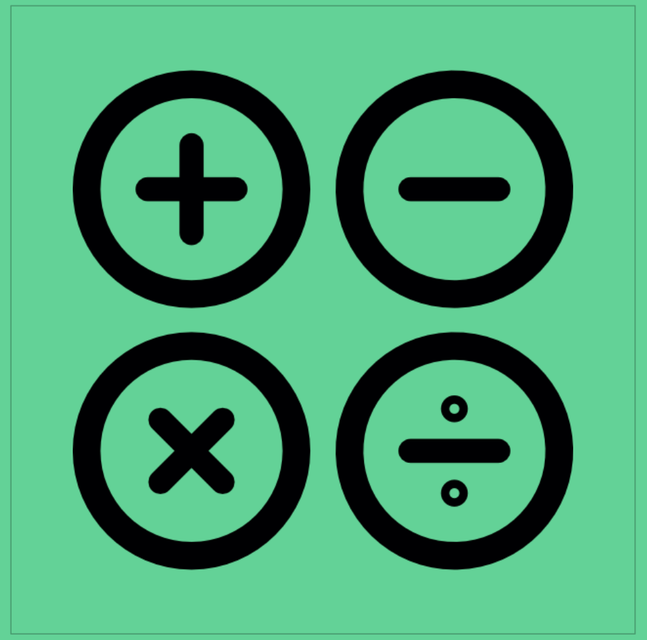
- (4º) O identificador de uma variável pode aparecer simultaneamente tanto a esquerda quanto a direita do operador de atribuição.



Variável **x** é inicializada com o valor inteiro 0

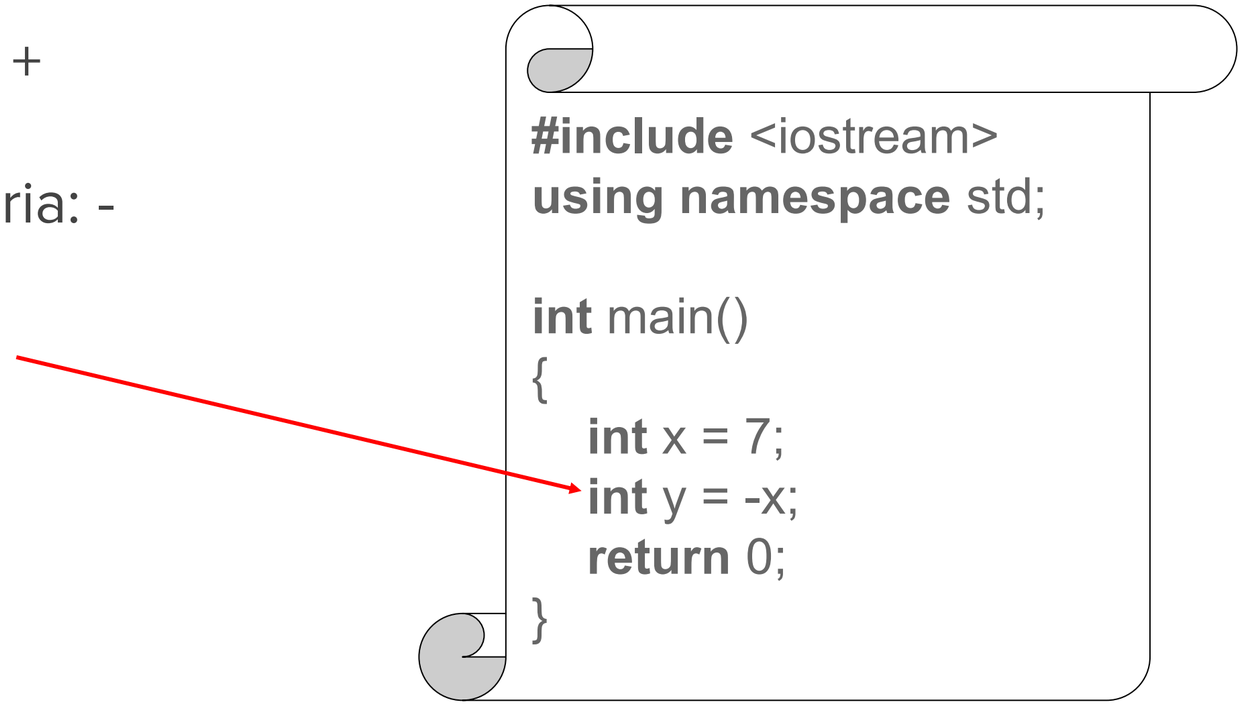
Variável **x** passa a armazenar o valor resultante do somatório do valor armazenado na própria variável **x** mais o valor inteiro 1. Ou seja, a variável **x** recebe o resultado da expressão **0 + 1**.

Operadores aritméticos



Operadores aritméticos

- Operadores aritméticos podem ser divididos em duas classes: operadores unários e operadores binários.
  - Operadores unários: possuem um único operando.
    - Adição unária: +
    - Subtração unária: -
      - Exemplo:



Operadores aritméticos

- Operadores binários: possuem dois operandos em notação infixa.
  - Adição: +
  - Subtração: -
  - Multiplicação: *
  - Divisão: /
  - Resto (módulo): %
    - Exemplo:

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

int main()
{
    int x = 7;
    int y = x % 5;
    return 0;
}
```

Neste exemplo, a variável **y** armazena o valor resultante do resto (módulo) da divisão de 7 por 5. Ou seja, **y** armazena o número inteiro 2.

Particularidades dos operadores aritméticos

- Sobre números inteiros e números em ponto flutuante:
  - Com **exceção** do operador de resto (**%**), todos os demais operadores aritméticos permitem operandos inteiros ou em ponto flutuante;
    - Quando inteiros e números em ponto flutuante são misturados, o resultado da expressão sempre será um número em ponto flutuante. Exemplo: **9 + 2.5** terá como resultado o número **11.5**.
    - A divisão de dois números inteiros sempre resultará em um número também inteiro. Exemplo: **7 / 2** terá como resultado o número **3**.
  - O operador de resto (%) apenas permite operandos do tipo inteiro. Qualquer tentativa do contrário gerará um erro de sintaxe em C++.

Divisão de números inteiros

Como comentado anteriormente, a divisão de dois números inteiros sempre resultará em um número inteiro. Exemplo:

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

int main()
{
    float x = 7 / 2;
    return 0;
}
```

Mesmo a variável **x** sendo declarada como **float**, o resultado da divisão de dois números inteiros sempre será um número inteiro. Portanto, neste exemplo, a variável **x** armazena o valor **3** e não o valor **3.5**.

Divisão de números inteiros

Para fazermos com que o resultado da divisão apresente casas decimais, é necessário garantirmos que um dos operandos da expressão seja um número em ponto flutuante. Exemplos:

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

int main()
{
    float x = 7.0 / 2;
    return 0;
}
```

Em ambos os exemplos, a variável **x** armazena o valor **3.5**.

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

int main()
{
    float x = 7 / 2.0;
    return 0;
}
```

Divisão de números inteiros - exemplos

Código	Saída

`int a,b,c;`
`float p;`
`p=(a+b+c)/2;`
	Entrada: 2 3 4
	Saída de p: 4
	Mesmo p sendo float, a divisão de variáveis inteiras, gera um valor inteiro, truncando a parte decimal.

`int a,b,c;`
`float p;`
`p=float(a+b+c)/2;`
	Entrada: 2 3 4
	Saída de p: 4.5
	É feito um casting para a saída ser de fato um float

`int a,b,c;`
`float p;`
`p=(a+b+c)/2.0;`
	Entrada: 2 3 4
	Saída de p: 4.5
	É feito um casting para a saída ser de fato um float

`float p=2/3;`	Saída será 0.
`float p=2/3.0;`	Saída será 0.666667

Divisão de números inteiros

Uma outra forma para garantirmos que um dos operandos da expressão seja um número em ponto flutuante é utilizarmos **estratégias de conversão para transformar um número inteiro em um em ponto flutuante**.

Existem duas abordagens de conversão em C++, uma é utilizando a operação de conversão (**casting**) e a outra é utilizando uma função especial, chamada construtor, de número flutuante.

É importante ressaltar que apesar de estarmos tratando o caso de conversão de inteiro para ponto flutuante, essas duas abordagens podem funcionar em outras situações, mas deve-se verificar a disponibilidade do recurso antes.

Divisão de números inteiros

Na primeira forma, operador de conversão (**casting**), utilizamos o tipo desejado entre parênteses, antes do valor a ser convertido. Exemplos:

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

int main()
{
    float x = (float) 7 / 2;
    return 0;
}
```

Em ambos os exemplos, a variável **x** armazena o valor **3.5**.

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

int main()
{
    float x = 7 / (float) 2;
    return 0;
}
```

Divisão de números inteiros

Na segunda forma, uso de construtor, utilizamos o tipo desejado como se fosse uma função, com o valor a ser convertido entre parênteses. Exemplos:

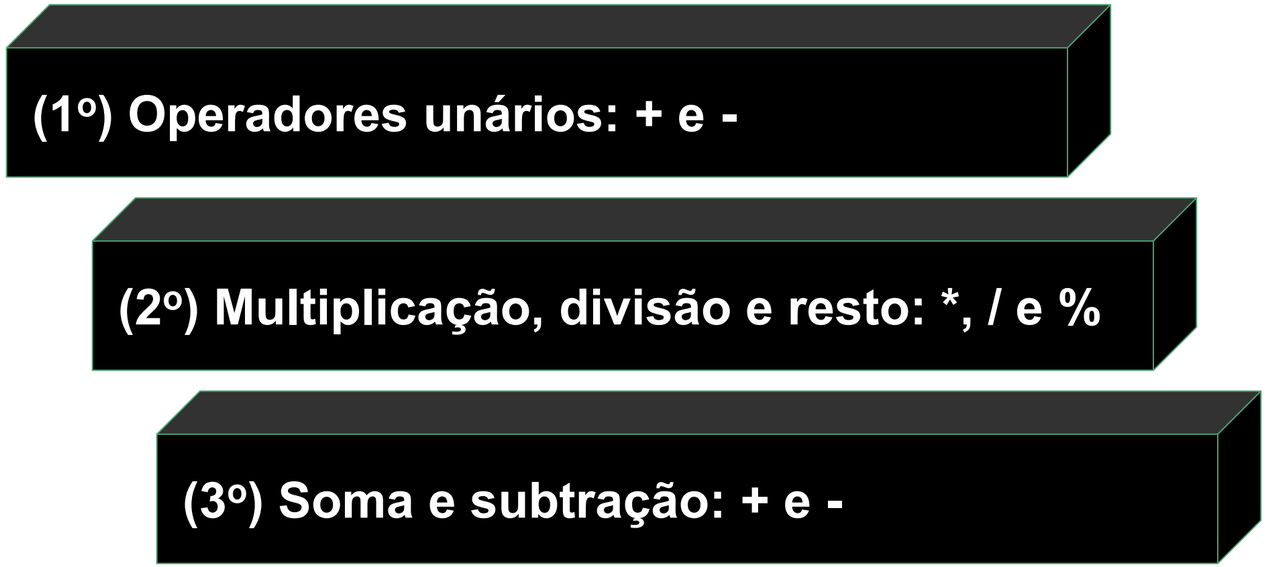
```
#include <iostream>
using namespace std;

int main()
{
    float x = float(7) / 2;
    return 0;
}
```

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

int main()
{
    float x = 7 / float(2);
    return 0;
}
```

Em ambos os exemplos, a variável **x** armazena o valor **3.5**.

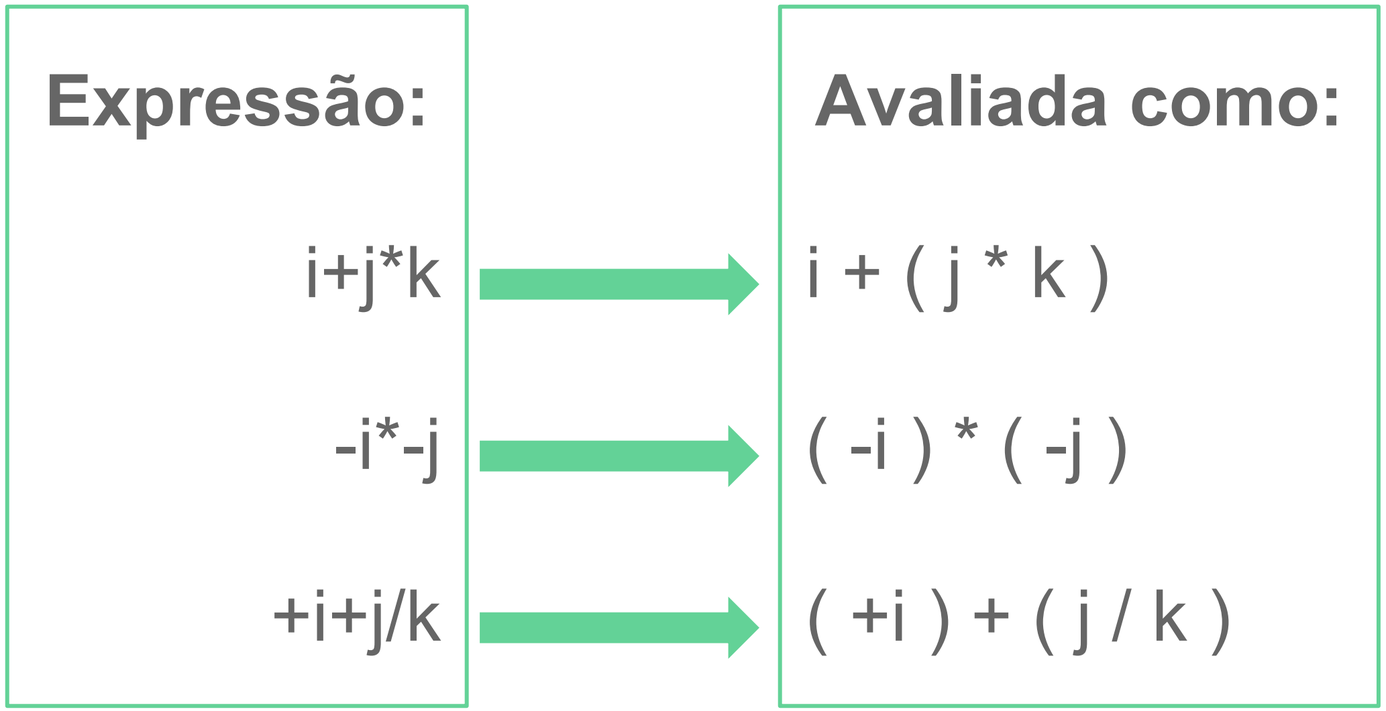
Precedência de operadores aritméticos

- Quando uma expressão aritmética precisa ser avaliada em um programa, o mesmo processa a expressão dando prioridade a certos operadores em detrimento de outros.
- Operações com maior prioridade (precedência) são avaliadas primeiro. Ordem de precedência dos operadores aritméticos:



Precedência de operadores aritméticos

- Exemplos:



Associatividade de operadores aritméticos

- Quando uma expressão aritmética com operadores com o mesmo nível de precedência precisa ser avaliada em um programa, o mesmo processa a expressão seguindo as regras de associatividade dos operadores.
- Operadores aritméticos possuem duas regras possíveis de associatividade:
  - Operadores associativos à esquerda: as operações mais à esquerda são executadas primeiro;
  - Operadores associativos à direita: as operações mais à direita são executadas primeiro.

Associatividade de operadores aritméticos

- Operadores associativos à esquerda:

Adição binária: **+**
Subtração binária: **-**
Multiplicação: *
Divisão: **/**
Resto: **%**

Exemplo:

Expressão:	**Avaliada como:**
a*b/c*d	(((a * b) / c) * d)

- Operadores associativos à direita:

Adição unário: **+**
Subtração unária: **-**

Exemplo:

Expressão:	**Avaliada como:**
-+i	- (+ i)

Exemplos: precedência e associatividade

EXEMPLO 1:

1 + 10 * 4 - 6

1 + 40 - 6

41 - 6

35

EXEMPLO 2:

(1 + 10) * 4 - 6

(11) * 4 - 6

44 - 6

38

EXEMPLO 3:

(1 + 10) * (4 - 6)

(11) * (4 - 6)

(11) * (-2)

-22

Recomendação em precedência de operadores

- Tentar decorar a precedência de operadores é uma má ideia para programadores iniciantes.
- Sempre use parênteses (e apenas parênteses) para garantir que uma determinada expressão seja avaliada na ordem desejada.

Operadores aritméticos e de atribuição

- Operadores aritméticos e o operador de atribuição podem ser combinados de forma a gerar uma atribuição composta.
- Uma atribuição composta pode ser criada apenas quando a variável a esquerda de um comando de atribuição também aparece em seu lado direito. Sendo assim, instruções do tipo:

```
variável = variável operador valor ;
```

podem ser combinadas de modo a formar instruções com a sintaxe:

```
variável operador= valor ;
```

Operadores aritméticos e de atribuição

- Por exemplo, a instrução:

```
i = i + 2 ;
```

poderia ser combinada, gerando a atribuição composta:

```
i += 2 ;
```

- Outras possibilidades para operadores compostos: -=, *=, /= e %=.

Incremento e Decremento

- Uma das operações mais comuns em programação é aumentar ou diminuir o valor de uma variável inteira em uma unidade. Por exemplo:

```
i = i + 1 ;
j = j - 1 ;
```

- Podemos realizar este tipo de operação por meio de atribuições compostas:

```
i += 1 ;
j -= 1 ;
```

Incremento e Decremento

- Na prática, estas operações são tão comuns que C++ oferece ainda um terceiro recurso para indicá-las, que são os operadores de incremento (++) e de decremento (--).
- Os operadores de incremento e decremento, implicam tanto no aumento ou diminuição de uma variável em uma unidade quanto na atribuição deste novo valor para a mesma variável.
- Nos nossos exemplos anteriores, poderíamos então fazer:

```
i++ ;
j-- ;
```

Biblioteca <cmath>

Biblioteca <cmath>

- Comumente, programas que realizam cálculos matemáticos um pouco mais complexos se utilizam de funções prontas da biblioteca <cmath>.
- **Bibliotecas podem ser vistas como uma coleção de subprogramas** (funções) utilizados no desenvolvimento de softwares.
- Com a biblioteca <cmath> é possível encontrar funções para calcular potências, raiz quadrada, funções trigonométricas, além de constantes para números irracionais, como por exemplo, o número pi (Π).

Biblioteca <cmath>

- Para utilizar funções da biblioteca <cmath>, o programador deve:
  - (1º) Incluir a biblioteca <cmath> no início de seu próprio programa. Para isso, basta inserir a instrução no seu programa:

```
#include <cmath>
```

  - (2º) Utilizar o nome da função de interesse, passando os parâmetros necessários para a mesma. De modo geral, a sintaxe será:

```
nome_da_função(parâmetros)
```

Se houver mais de um parâmetro, os mesmos devem ser separados por vírgula.

Biblioteca <cmath>

- Alguns exemplos de funções:

Funções	O que faz
pow()	Potenciação
sqrt()	Raiz quadrada
cos()	Cosseno
sin()	Seno
tan()	Tangente
ceil()	Arredondamento para cima
floor()	Arredondamento para baixo
abs()	Valor absoluto (módulo)

Biblioteca <cmath>

- Exemplo: cálculo da raiz quadrada de um número.

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

int main()
{
    float numero = 25;
    float resultado = sqrt(numero);
    return 0;
}
```

A biblioteca <cmath> deve ser incluída antes da declaração da função main()

Outras funções da biblioteca podem ser encontradas em:
http://www.cplusplus.com/reference/cmath/

Entrada e Saída

cin >>

Entrada

cout <<

Saída

Entrada e saída de dados

Função	Parâmetros	Interpretação Usual
leia(x)	Variável **x** de qualquer tipo fundamental de dados	Um valor digitado no teclado é armazenado na variável **x**
escreva(x)	Variável, constante ou expressão de qualquer tipo fundamental de dados	O valor definido por **x** é apresentado no monitor.

Entrada e saída de dados

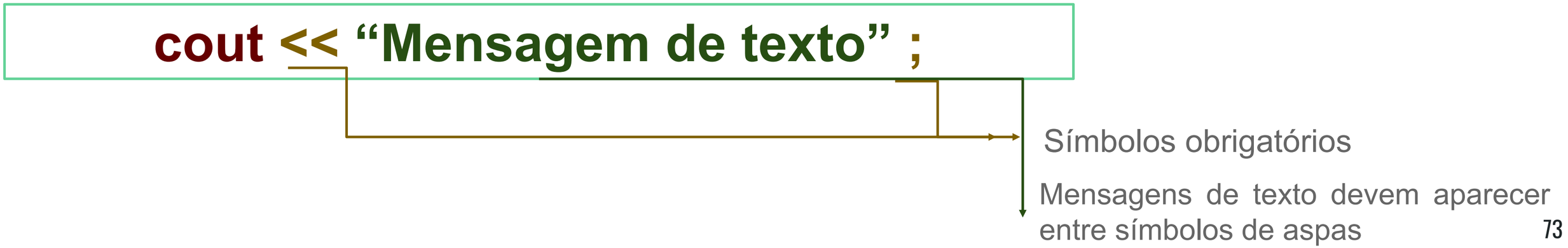
- Em C++, estas duas funções são relacionadas a dois objetos distintos:

Função	Objeto C++	Dispositivo acessado comumente
leia(x)	cin	Teclado
escreva(x)	cout	Monitor

- Vamos ver então, a partir de agora, a sintaxe para estes dois objetos.

Saída de dados

- Sintaxe para **cout** (para mensagens fixas):
  - Para utilizar **cout** deve-se:
    - Incluir a biblioteca para operações de entrada e saída do C++. Nome da biblioteca: **<iostream>**;
    - Utilizar o ambiente de nomeação padrão do C++. Nome do ambiente de nomeação: **std**;



Saída de dados

- Sintaxe para **cout** (para mensagens fixas):
  - Para utilizar **cout** deve-se:
    - Incluir a biblioteca para operações de entrada e saída do C++. Nome da biblioteca: **<iostream>**;
    - Utilizar o ambiente de r ente de nomeação: **std**;

Operador de encadeamento: “recebe”

```
cout << “Mensagem de texto” ;
```

Símbolos obrigatórios

Mensagens de texto devem aparecer entre símbolos de aspas

Saída de dados

- Exemplo com **cout**: mensagem fixa

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

int main()
{
    cout << “Olá mundo!”;
    return 0;
}
```

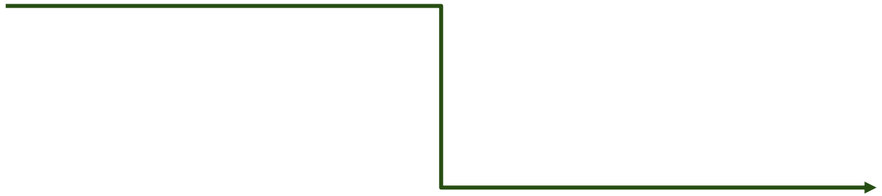
Esquecer de incluir a biblioteca **<iostream>** ou o ambiente de nomeação **std** ocasionará um erro de sintaxe quando o objeto **cout** for utilizado.

Não colocar a mensagem a ser exibida no dispositivo de saída padrão (monitor) entre os símbolos de aspas também gerará um erro de sintaxe.

Saída de dados

- Sintaxe para **cout** (para variáveis ou expressões):

```
cout << nome_da_variável ;
```



Identificadores de variáveis não devem aparecer entre símbolos de aspas

Saída de dados

- Sintaxe para o objeto **cout** (formatações):
  - O objeto **cout** pode exibir simultaneamente no dispositivo de saída padrão múltiplas mensagens fixas e valores de variáveis/expressões.
  - Para exibir múltiplas informações, basta separar cada uma delas por operadores <<.

```
cout << “Mensagem de texto” << nome_da_variável ;
```

  - Podemos forçar uma quebra de linha no dispositivo de saída padrão por meio do manipulador **endl**.

Saída de dados

- Exemplo: raiz quadrada de 25

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

int main()
{
    float numero = 25;
    cout << “A raiz de ” << numero << “ é o valor ”
         << sqrt(numero) << endl;
    return 0;
}
```

Saída de dados

- Exemplo: modificando saída de ponto-flutuante, imprimindo a parte decimal com 10 dígitos:

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

int main() {
    float numero = 25;
    cout.precision(10);
    cout.setf(ios::fixed);
    cout << “A raiz de ” << numero << “ é o valor ”
         << sqrt(numero) << endl;
    return 0;
}
```

Informa precisão (casas decimais)

Informa que parte decimal deve ser impressa, mesmo que seja zero.

Entrada de dados

- Sintaxe para o objeto **cin**:
  - Para utilizar o operador **cin** se deve:
    - Incluir a biblioteca para operações de entrada e saída do C++. Nome da biblioteca: **<iostream>**;
    - Utilizar o ambiente de nomeação padrão do C++. Nome do ambiente de nomeação: **std**;

```
cin >> nome_da_variável ;
```

Símbolos obrigatórios

Identificadores não devem aparecer entre símbolos de aspas

Entrada de dados

- Sintaxe para o objeto **cin**:
  - Para utilizar o operador **cin** se deve:
    - Incluir a biblioteca para operações de entrada e saída do C++. Nome da biblioteca: **<iostream>**;
    - Utilizar o ambiente de r ente de nomeação: **std**;

Operador de encadeamento: “envia”

```
cin >> nome_da_variável ;
```

Símbolos obrigatórios

Identificadores não devem aparecer entre símbolos de aspas

Entrada de dados

- Exemplo com **cin**: raiz quadrada de um número qualquer

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

int main()
{
    float numero;
    cin >> numero;
    cout << sqrt(numero);
    return 0;
}
```

Esquecer de incluir a biblioteca **<iostream>** ou o ambiente de nomeação **std** ocasionará um erro de sintaxe quando o objeto **cin** for utilizado.

Entrada de dados

- Sintaxe para o objeto **cin**:
  - O objeto **cin** permite o armazenamento simultâneo de múltiplos valores em múltiplas variáveis, desde que os valores sejam separados por espaços, tabulação ou linhas em branco no dispositivo de entrada padrão (teclado);
  - Para isso, é necessário que exista uma variável diferente para cada valor a ser lido;
  - Cada nova variável deve ser separada pelos símbolos >>.

```
cin >> nome_da_variável1 >> nome_da_variável2;
```

Entrada de dados

- Exemplo com **cin**: soma de dois números quaisquer

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

int main()
{
    float numero1, numero2;
    cin >> numero1 >> numero2;
    cout << numero1+numero2;
    return 0;
}
```

Inverter os símbolos **>>** do operador **cin** pelos símbolos **<<** do operador **cout** gerará um erro de sintaxe. O contrário também gerará um erro de sintaxe.

Atividades

Prepare sua máquina.
Instalar algum editor de código e um compilador C++.