

Treinamento em C#

Matheus Neder

matheusneder@gmail.com

Apresentação

- Desenvolvedor Fullstack
- Engenheiro de Computação (em formação)
- Arquiteto de Software

Apresentação - assuntos de interesse

- Padrões de projeto;
- POO;
- Arquitetura de software;
- DDD;
- TDD;
- SOLID;
- DevOps ...

Apresentação - tecnologias UX / UI

- Web
- Java Swing, AWT, SWT e Android
- C GTK+
- C++ GTKmm, QT
- C# Xamarin.Forms
- Objective-C iOS

Apresentação - tecnologias em geral

- ASP
- VB6 (COM+)
- JavaScript
- Java
- C/C++
- Linux
- Bash

Visão geral

- Tipos de dados;
- Gerenciamento de memória;
- Passagem de valores;
- Estruturas de desvio condicional / repetição;
- Delegates e eventos;

- **POO:**

- Tipos genéricos;
- Encapsulamento;
- Polimorfismo;

Visão geral

- POO:
 - Tratamento de exceções (try ... catch);
 - Classes abstratas e interfaces;
 - **SOLID**;
 - IoC e DI;
 - **DDD**;

Visão geral

- Programação funcional:
 - LINQ;
 - Funções lambda;
- ORMs;
- Threads:
 - Tasks;
 - Programação assíncrona (async e await).

Tipos de dados

- Tipos primitivos (bool, char, int ...);
- Tipos complexos;
- Tipos de valor (value types);
- Tipos de referência (reference types).

Tipos primitivos (internos)

- bool
- byte
- sbyte
- char
- decimal
- double
- float
- int
- uint
- long
- ulong
- object
- short
- ushort
- string

Tipos complexos

Definição:

```
struct Coord
{
    public double x;
    public double y;
}
```

Utilização:

```
Coord coord = new Coord();
coord.x = 10;
coord.y = 20;
```

Tipos de dados

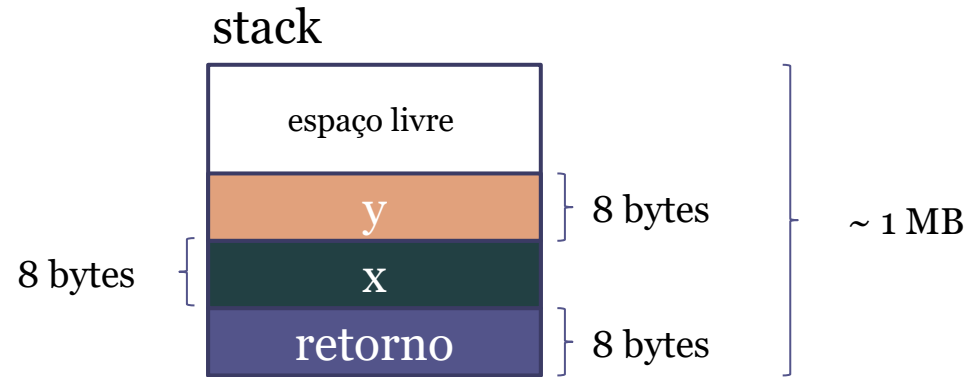
Tipos de valor

VS

Tipos de referência

Gerenciamento de memória

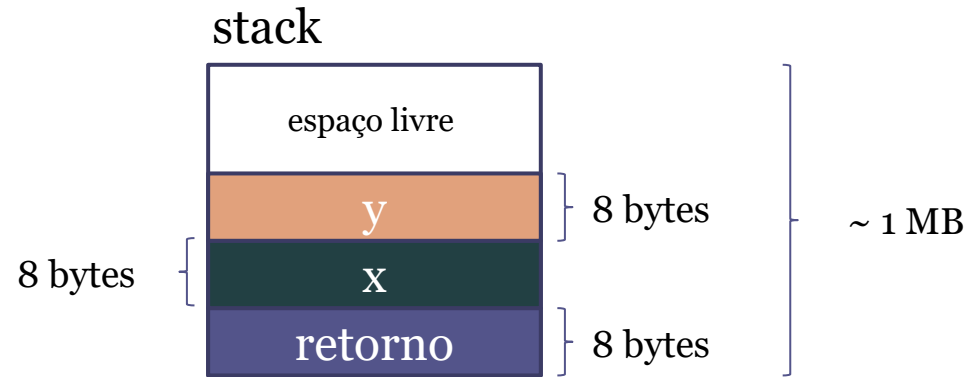
```
public long Soma(long x, long y)
{
    return x + y;
}
```



Observação: 8 bytes = sizeof(long)

Tipos de dados

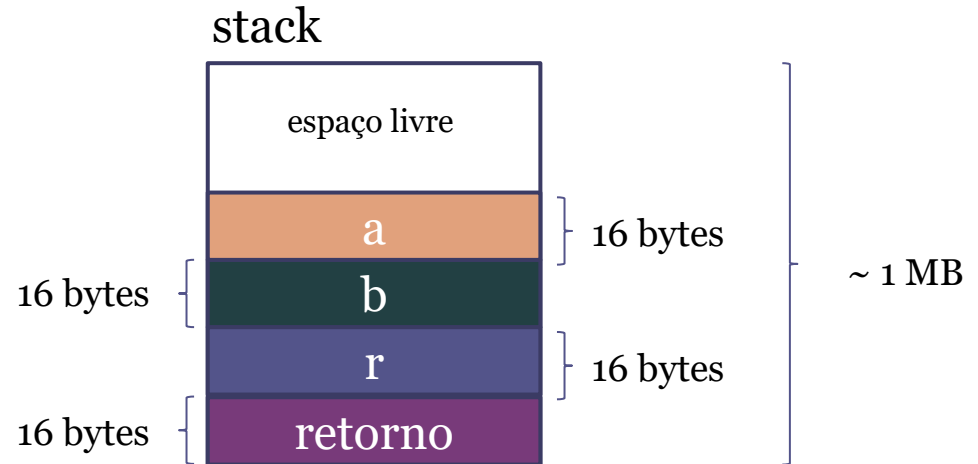
Tipos de valor (value types) são armazenados na pilha (stack).



Observação: 8 bytes = sizeof(long)

Tipos de dados

```
public Coord Midpoint(Coord a, Coord b)
{
    Coord r = new Coord();
    ...
    return r;
}
```



Observação: 16 bytes = sizeof(Coord)

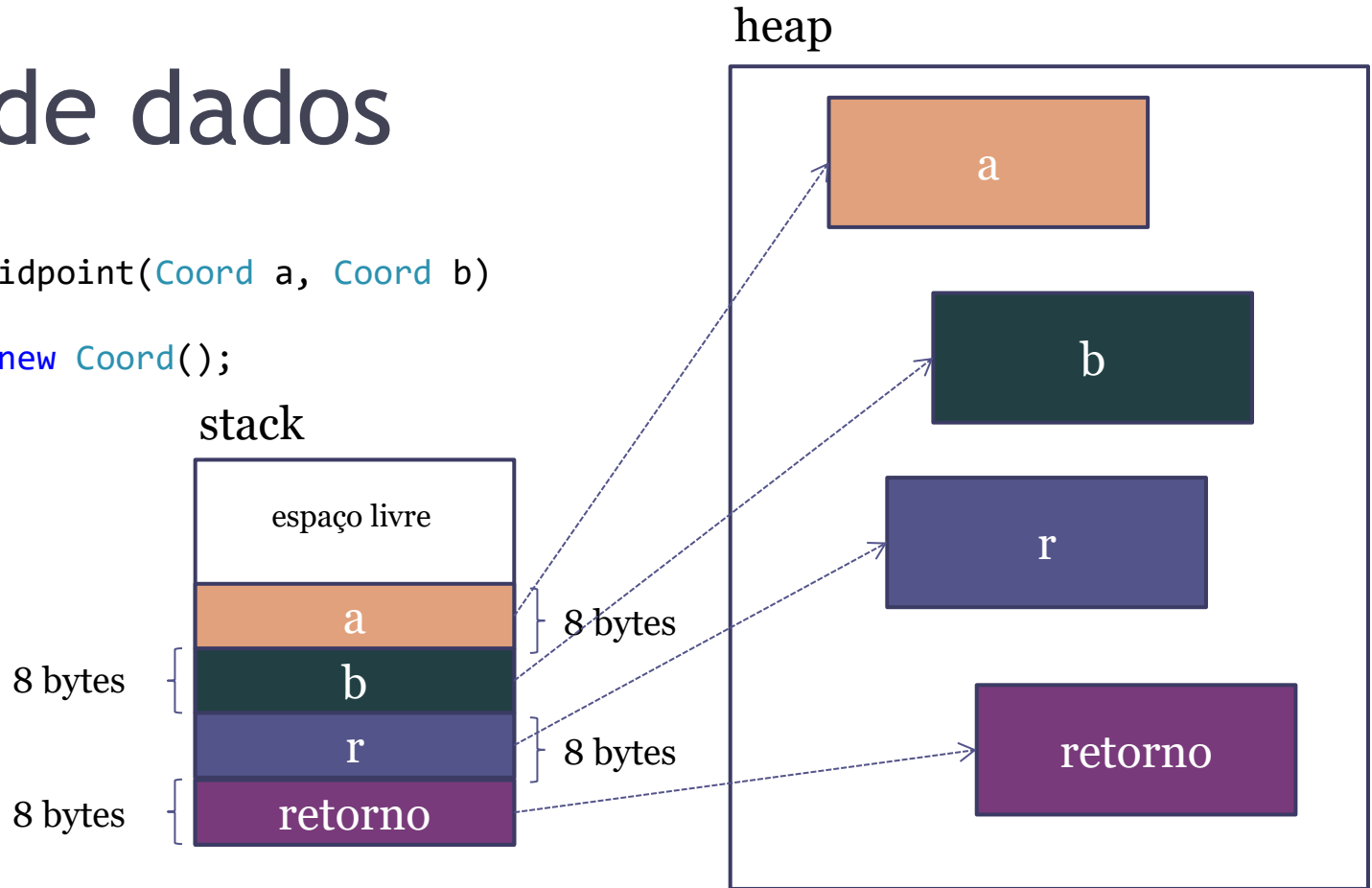
Tipos de dados

Considere uma nova versão do tipo *Coord*:

```
class Coord
{
    public double x;
    public double y;
}
```


Tipos de dados

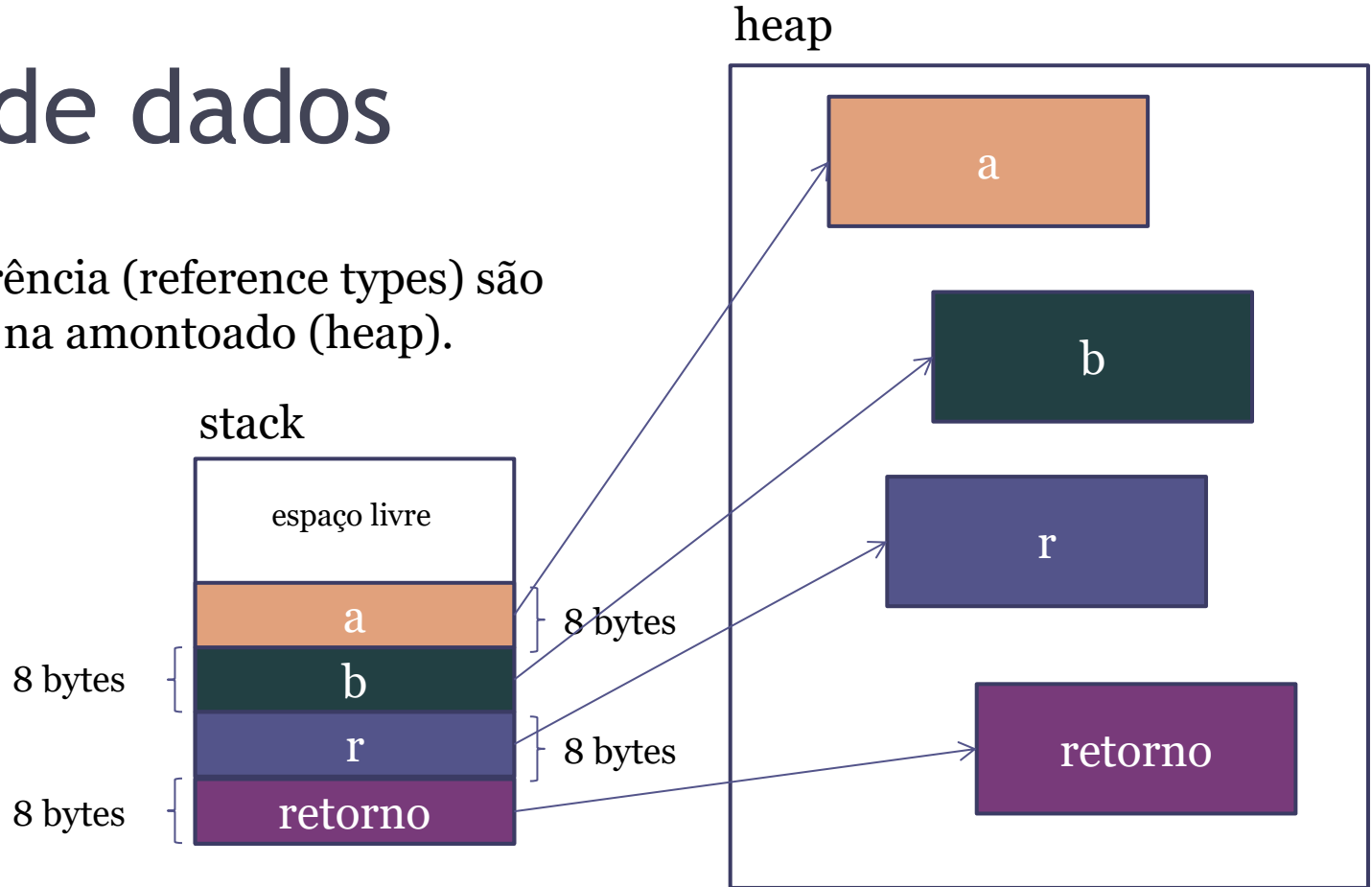
```
public Coord Midpoint(Coord a, Coord b)
{
    Coord r = new Coord();
    ...
    return r;
}
```



Observação: 8 bytes = tamanho do ponteiro

Tipos de dados

Tipos de referência (reference types) são armazenados na amontoado (heap).



Observação: 8 bytes = tamanho do ponteiro

Tipos de dados

```
struct Coord
{
    public double x;
    public double y;
}

public void Swap(Coord coord)
{
    double aux = coord.x;
    coord.x = coord.y;
    coord.y = aux;
}
```

```
Coord coord = new Coord();
coord.x = 10.0;
coord.y = 5.0;

Swap(coord);

Console.WriteLine($"x: {coord.x}");
Console.WriteLine($"y: {coord.y}");
```



Tipos de dados

```
class Coord
{
    public double x;
    public double y;
}

public void Swap(Coord coord)
{
    double aux = coord.x;
    coord.x = coord.y;
    coord.y = aux;
}
```

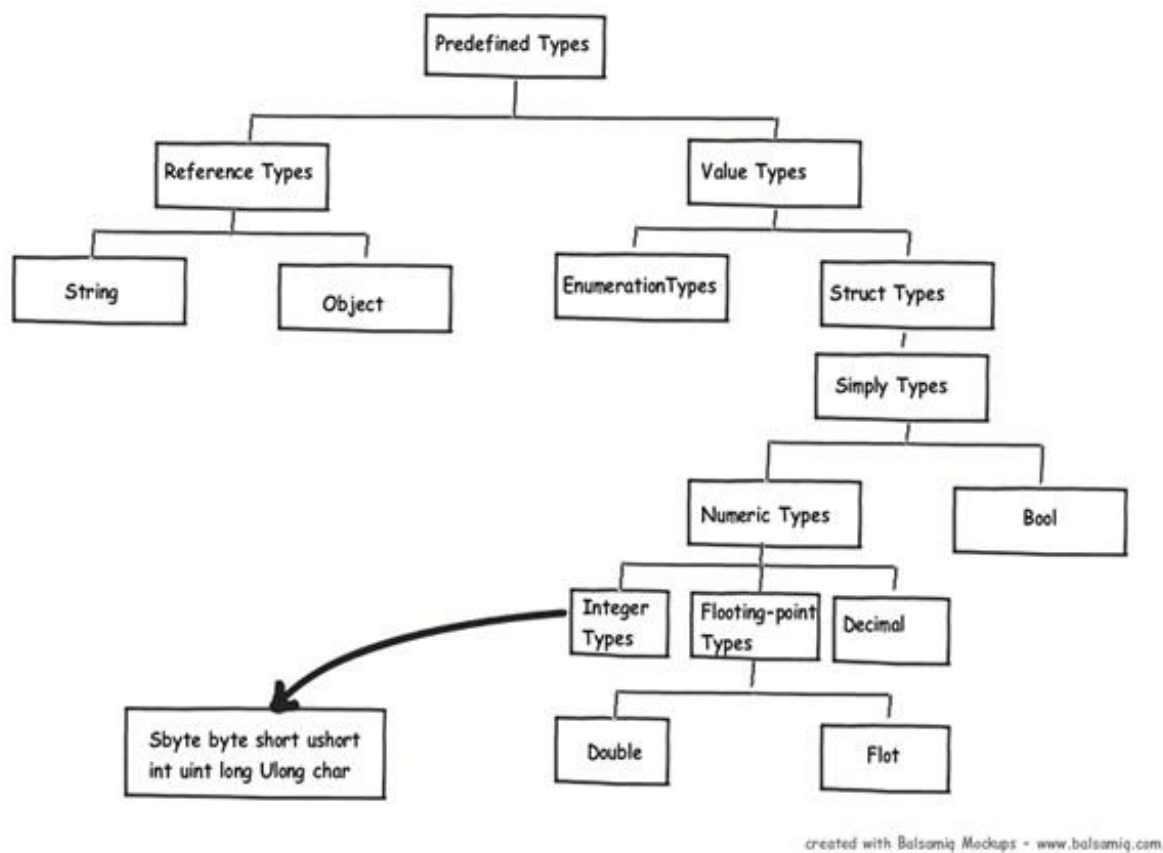
```
Coord coord = new Coord();
coord.x = 10.0;
coord.y = 5.0;

Swap(coord);

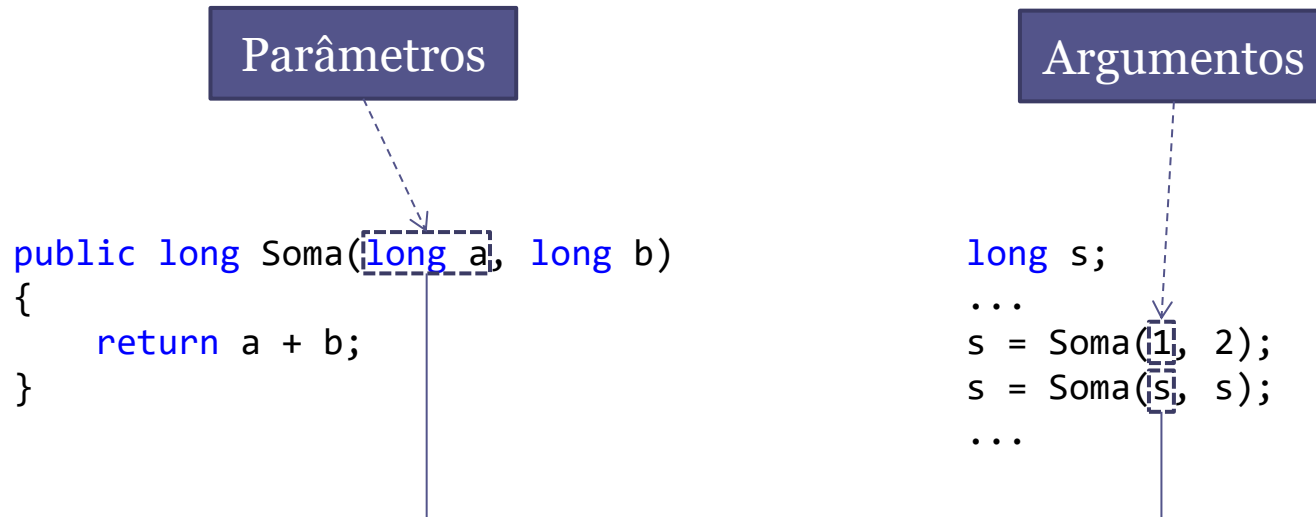
Console.WriteLine($"x: {coord.x}");
Console.WriteLine($"y: {coord.y}");
```



Tipos de dados



Parâmetro vs argumento



Passagem de parâmetros por cópia na entrada

```
public void Operacao(long a)
{
    a = 20;
}
```

```
long s;
s = 10;
Operacao(s);

s ?
```

Cópia



Passagem de parâmetros por referência

```
public void Operacao(ref long a)  
{  
    a = 20;  
}
```

```
long s;  
s = 10;  
Operacao(ref s);  
  
s ?
```

Referência

Passagem de parâmetros por cópia na entrada e na saída

```
public void Operacao(out long a)  
{  
    a = 20;  
}
```

```
long s;  
s = 10;  
Operacao(out s);  
  
s ?
```

Cópia na entrada e na saída

Passagem de parâmetros

out vs *ref*

Passagem de parâmetros *out* vs *ref*

```
public void Incrementa(ref long a)
{
    a = a + 1;
}
```

```
public void Incrementa(out long a)
{
    a = a + 1;
}
```

Passagem de parâmetros *out* vs *ref*

```
public void Incrementa(ref long a)
{
    a = a + 1;
}
```

```
public void Incrementa(out long a)
{
    a = a + 1;
}
```



Use of unassigned out parameter 'a'

Passagem de parâmetros *out* vs *ref*

```
public void Atribui(ref long a)
{
    ...
    a = 1;
    ...
    ...
}
```

```
long x;
Atribui(ref x);
```

x ?

```
public void Atribui(out long a)
{
    ...
    a = 1;
    ...
    ...
}
```

```
long x;
Atribui(out x);
```

x ?

Passagem de parâmetros *out* vs *ref*

```
public void Atribui(ref long a)
{
    ...
    a = 1;
    ...
    ...
}
---
long x;
Atribui(ref x);
```



Use of unassigned out parameter 'x'

```
public void Atribui(out long a)
{
    ...
    a = 1;
    ...
    ...
}
---
long x;
Atribui(out x);
```

x ?

Passagem de parâmetros *out* vs *ref*

```
public void Atribui(ref long a)
{
    ...
    a = 1;
    ...
    ...
}
```

```
long x = 0;
Atribui(ref x);
```

x ?

```
public void Atribui(out long a)
{
    ...
    a = 1;
    ...
    ...
}
```

```
long x = 0;
Atribui(out x);
```

x ?

Parâmetros Opcionais

```
public long SomaUmOuB(long a, long b = 1)
{
    return a + b;
}
```

```
long s;
...
s = Soma(1);
s = Soma(s, s);
```

s ?

Array de parâmetros

```
public long Somatorio(params long[] valores)
{
    long s = 0;

    foreach(long valor in valores)
    {
        s += valor; // equiv. a 's = s + valor'
    }

    return s;
}
```

```
long s;
...
s = Somatorio(1, 2, 3);

s ?

s = Somatorio(1, 2, 3, 4);

s ?

s = Somatorio();

s ?
```

Argumentos nomeados

```
public void Operacao(long a = 1, long b = 2, long c = 3, long d = 3)
{
    ...
}
```

Operacao();

Operacao(5, 6);

Operacao(d: 5);

Operacao(d: 5, c: 3);

Operacao(5, 6, d: 7);

Estruturas de desvio condicional (*if* e *else*)

...

```
if([booleano])
```

```
{
```

```
    [bloco de código]
```

```
}
```

...

Expressão booleana, exemplos:

- `true` *ou* `false`
- `a == b` *ou* `a != b`
- `!a` (se `a` for do tipo `bool`)
- `a && b || c`
- ...

Estruturas de desvio condicional (*if* e *else*)

```
...  
if([booleano])  
{  
    [bloco de código]  
}  
else  
{  
    [bloco de código]  
}  
...
```

Expressão booleana, exemplos:

- `true` *ou* `false`
- `a == b` *ou* `a != b`
- `!a` (se `a` for do tipo `bool`)
- `a && b` *ou* `a || b`
- ...

Estruturas de desvio condicional (*if* e *else*)

```
...  
if([booleano])  
{  
    [bloco de código]  
}  
else if([booleano])  
{  
    [bloco de código]  
}  
else  
{  
    [bloco de código]  
}  
...
```

Expressão booleana, exemplos:

- `true` ou `false`
- `a == b` ou `a != b`
- `!a` (se `a` for do tipo `bool`)
- `a && b` || `c`
- ...

Estruturas de desvio condicional (*if* e *else*)

```
if([booleano])
{
    [bloco de código]
}
else
{
    if([booleano])
    {
        [bloco de código]
    }
    else
    {
        [bloco de código]
    }
}
```

Expressão booleana, exemplos:

- `true` ou `false`
- `a == b` ou `a != b`
- `!a` (se `a` for do tipo `bool`)
- `a && b` || `c`
- ...

Estruturas de desvio condicional (*if* e *else*)

```
...  
if([booleano])  
    [instrução]  
else if([booleano])  
    [instrução]  
else  
    [instrução]  
...
```

Expressão booleana, exemplos:

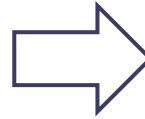
- `true` *ou* `false`
- `a == b` *ou* `a != b`
- `!a` (se `a` for do tipo `bool`)
- `a && b` *ou* `a || b`
- ...

Estruturas de desvio condicional (*if* e *else*)

```
long a = 1;  
long b = 2;  
long x;
```

```
x = a > b ? a : b ;
```

condição verdadeiro falso



```
if(a > b)  
{  
    x = a;  
}  
else  
{  
    x = b;  
}
```


Estruturas de desvio condicional (*switch/case*)

```
long s;  
long a = 1;  
long b = 2;  
  
switch (a)  
{  
    case 1:  
    case 2:  
        s = a + b;  
        break;  
    default:  
        s = 0;  
        break;  
}
```

Estruturas de repetição (while)

...

```
while([booleano])  
{  
    [bloco de código]  
}
```

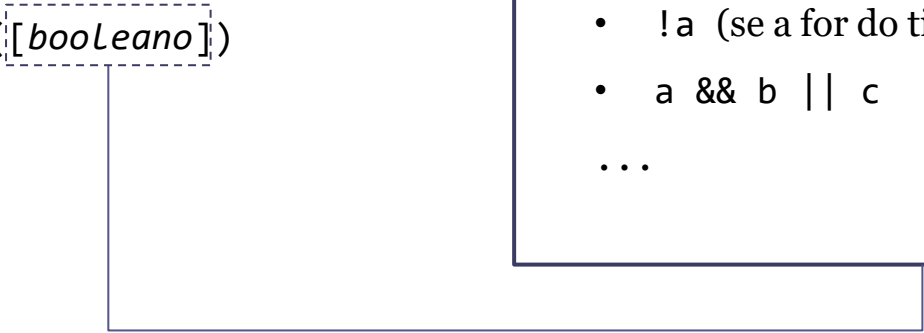
...

Condição de parada (booleano):

- `true` ou `false`
- `a == b` ou `a != b`
- `!a` (se `a` for do tipo `bool`)
- `a && b` || `c`
- ...

Estruturas de repetição (do ... while)

```
...  
  
do  
{  
    [bloco de código]  
}  
while([booleano])  
  
...
```

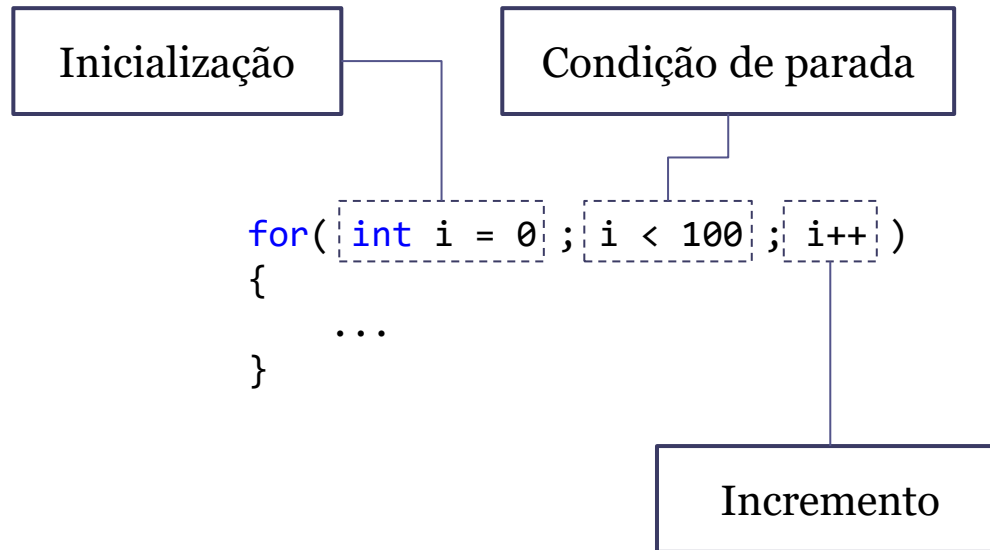


A dashed rectangular box is drawn around the condition `[booleano]` in the `while` statement. A thin blue line extends from the bottom of this box, goes down, then right, and finally up to point at the top-left corner of a larger box on the right side of the slide.

Condição de parada (booleano):

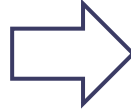
- `true` ou `false`
- `a == b` ou `a != b`
- `!a` (se `a` for do tipo `bool`)
- `a && b` || `c`
- ...

Estruturas de repetição (for)



Estruturas de repetição (for)

```
for(int i = 0; i < 100; i++)  
{  
    ...  
}
```



```
int i = 0;  
while(i < 100)  
{  
    ...  
    i++; // equivalente a 'i = i + 1'  
}
```

Estruturas de repetição (foreach)

```
long[] valores = { 1, 2, 3, 4};  
  
long s = 0;  
  
foreach(long valor in valores)  
{  
    s += valor; // equiv. a 's = s + valor'  
}  
  
s ?
```

Estruturas de repetição (foreach)

```
long[] valores = { 1, 2, 3, 4};
```

```
long s = 0;
```

```
foreach(long valor in valores)
{
    s += valor; // equiv. a 's = s + valor'
}
```

s ?

IEnumerable<long>

Delegates e eventos

```
public delegate void MyCustomDelegate ( long i, long j );
```


Delegates e eventos

```
public delegate void MyCustomDelegate ( long i, long j );
```

The diagram shows the following components of the delegate declaration:

- Tipo de retorno** (Return type): `void`
- Nome do delegate** (Delegate name): `MyCustomDelegate`
- Lista de parâmetros** (List of parameters): `(long i, long j)`

Delegates e eventos

```
public delegate void MyCustomDelegate(long i, long j);

public class MyClass
{
    private void UmaOperacaoQualquer(long i, long j)
    {
        // ...
    }

    private void SomaEExecutaUmaOperacao(long a, long b, MyCustomDelegate acao)
    {
        acao(a + b, 5);
    }

    public void Simular()
    {
        SomaEExecutaUmaOperacao(1, 2, UmaOperacaoQualquer);
    }
}
```

Delegates e eventos

```
public delegate void MyCustomDelegate(int i);

public class MyClass
{
    public event MyCustomDelegate MyEvent;

    private int hits = 0;

    public void MyAction()
    {
        hits++;

        if (MyEvent != null)
            MyEvent(hits);
    }
}
```

```
private void MyEvent_Fired(int i)
{
    Console.WriteLine($"Fired: {i}");
}
```

```
MyClass c = new MyClass();
```

```
[ c.MyEvent += MyEvent_Fired; ]
```

```
c.MyAction();
```

```
[ c.MyEvent -= MyEvent_Fired; ]
```

```
c.MyAction();
```

Tipos e métodos genéricos