UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Funções

QXD0001 - Fundamentos de Programação

Prof. Bruno Góis Mateus (brunomateus@ufc.br)

Agenda

- Introdução
- Funções
- Escopo de variáveis
- Pilha de execução
- Modularização
- Erros mais comuns

Introdução

Introdução

```go
package main

import "fmt"

func main() {
	fmt.Println("Bem-vindo ao sistema")
	fmt.Println("Bem-vindo ao sistema")
	fmt.Println("Bem-vindo ao sistema")
}
```

Introdução

Repetição

```go
package main

import "fmt"

func main() {
    fmt.Println("-----------")
    fmt.Println("Sistema")
    fmt.Println("-----------")

    fmt.Println("-----------")
    fmt.Println("Cadastro")
    fmt.Println("-----------")
}
```

Introdução

Modularização

- Programas grandes e complexos são difíceis de ler, testar e consertar.
- A Solução:
    - Dividir para Conquistar
    - Quebrar o problema em partes menores (módulos)

Funções permitem:

- Reutilizar código
- Organizar o programa
- Evitar repetição
- Melhorar leitura

Introdução

Funções

- Existem dois tipos de funções
- Built-in
  - São as funções providas como parte da linguagem
    - Printf, Scan …
- Funções
  - São aquelas que nós mesmos definimos

Funções

Funções

O que são?

- Definição: Um bloco de código nomeado que executa uma tarefa específica.
- A Caixa Preta:
  - Recebe Entradas (Parâmetros)
  - Faz um Processamento
  - Devolve uma Saída (Retorno)
- Exemplo do mundo real:
  - “somar”, "calcular média”, ”mostrar menu"

Funções

Criando a sua primeira função

```
func saudacao() {
    fmt.Println("Bem-vindo ao sistema")
}
```

Definindo uma função

```
package main

import "fmt"

func main() {
    saudacao()
}
```

Chamando uma função

Funções

Chamando várias vezes

```go
package main

import "fmt"

func saudacao() {
    fmt.Println("Bem-vindo ao sistema")
}

func main() {
    saudacao()
    saudacao()
    saudacao()
}
```

Chamando uma função

Funções

Anatomia de uma função

Nome da função (camelCase)

Parâmetros e seus tipos

```
func somar(a int, b int) int {
    return a + b
}
```

Tipo do retorno

Palavra chave que define o retorno da função

Palavra chave que indica que uma função está sendo declarada

Funções

Funções com parâmetros

```
package main

import "fmt"

func saudacao(nome string) {
    fmt.Println("Olá", nome)
}

func main() {
    saudacao("João")
    saudacao("Maria")
    saudacao("Pedro")
}
```

Argumento

Funções

Funções com retorno

```
package main

import "fmt"

func soma(a int, b int) int {
    return a + b
}

func main() {
    var resultado int = soma(10, 20)
    fmt.Println("Resultado:", resultado)
}
```

Armazenando o retorno da função

Funções

Funções com múltiplos retornos

```go
func dividir(a, b float64) (float64, error) {
    if b == 0 {
        return 0, errors.New("divisão por zero!")
    }
    return a / b, nil
}
```

- Diferente de C ou Java, Go permite devolver vários valores de uma vez.
- Utilidade: Muito usado para retornar o resultado e um erro.

Escopo de variáveis

Escopo de variáveis

Introdução

```go
func main() {
    var idade  int = 20
}

func imprimir() {
    fmt.Println(idade)
}
```

Qual o resultado da execução deste programa ?

Escopo:

A região do programa onde uma variável pode ser acessada.

Escopo de variáveis

Escopo local ou de bloco

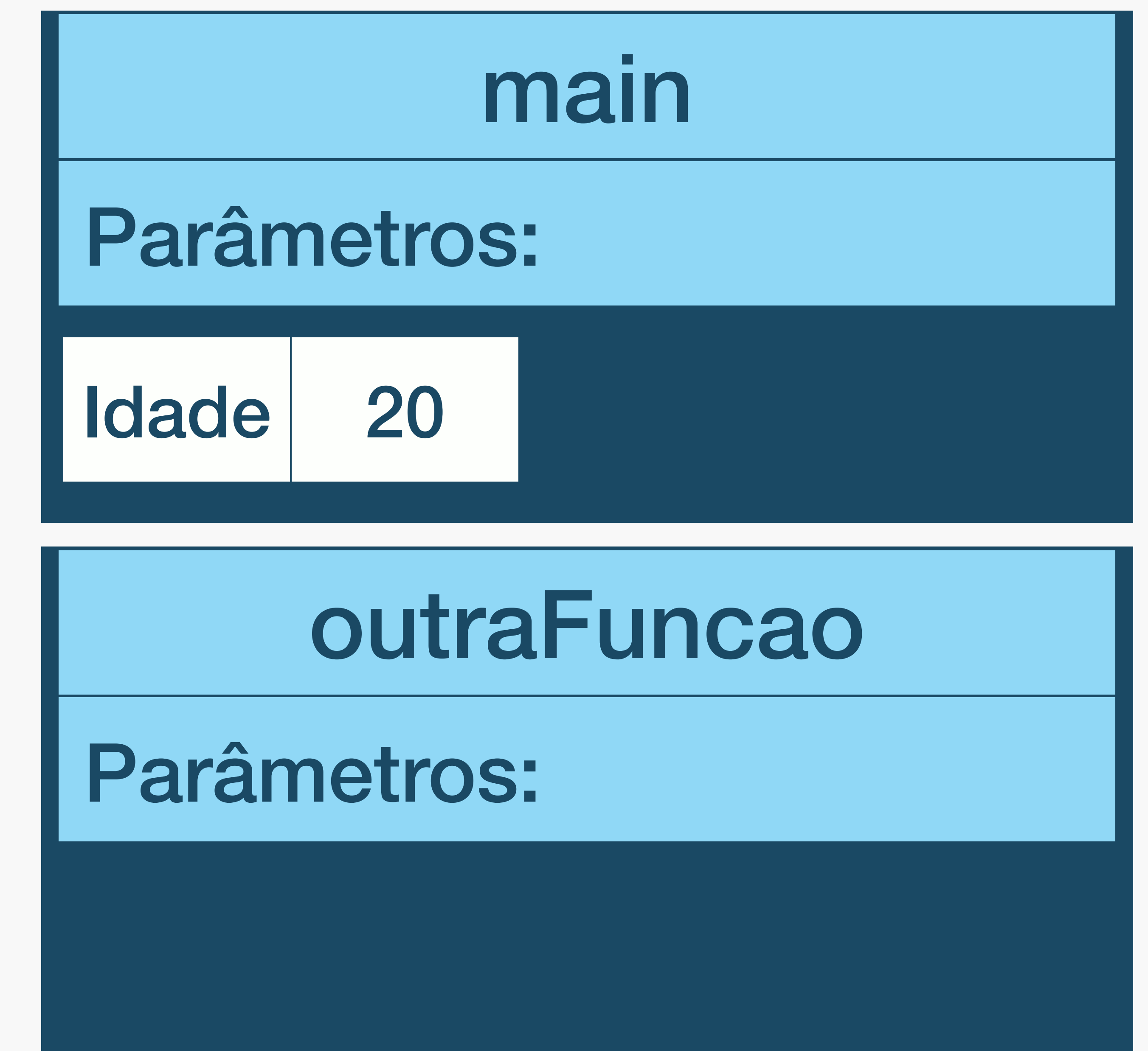
```go
package main

import "fmt"

func main() {
    idade := 20
    fmt.Println(idade)
}

func outraFuncao() {
    fmt.Println(idade)
}
```

A variável idade não é visível na outraFuncao

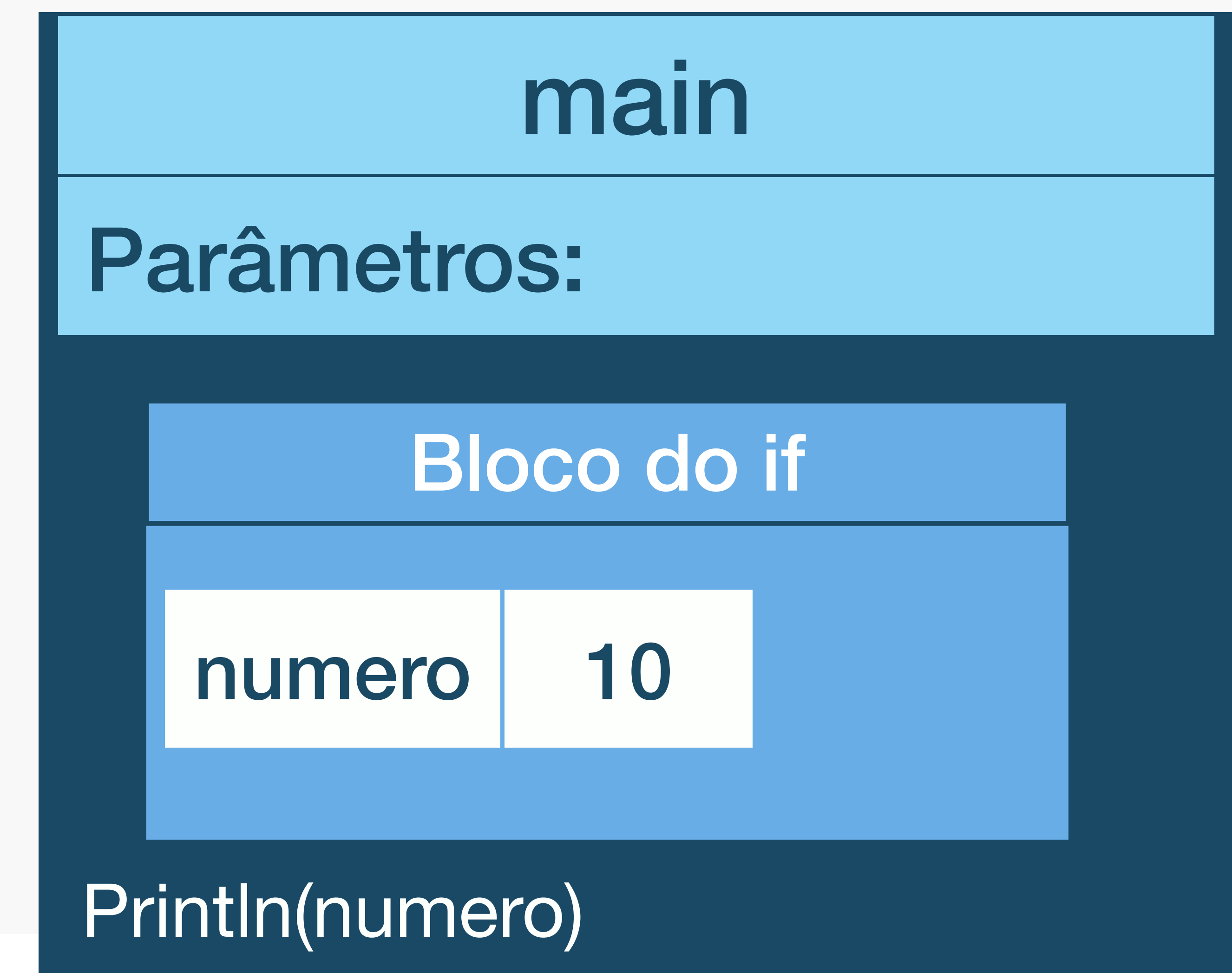


Escopo de variáveis

Escopo local ou de bloco

```go
func main() {
    if true {
        numero := 10
        fmt.Println(numero)
    }

    fmt.Println(numero)
}
```



A variável número existe apenas dentro do if

Escopo de variáveis

Escopo local ou de bloco

- Uma variável só existe dentro do bloco onde foi criada
  - Dentro de uma função ou de uma estrutura de controle (como if ou for)
  - São visíveis apenas dentro daquele par de chaves { }
  - A variável "nasce" quando o programa entra no bloco e "morre" (é liberada da memória) assim que o bloco termina
- Vantagens:
  - Evita conflitos de nomes
  - Você pode ter uma variável com mesmo nome em funções diferentes

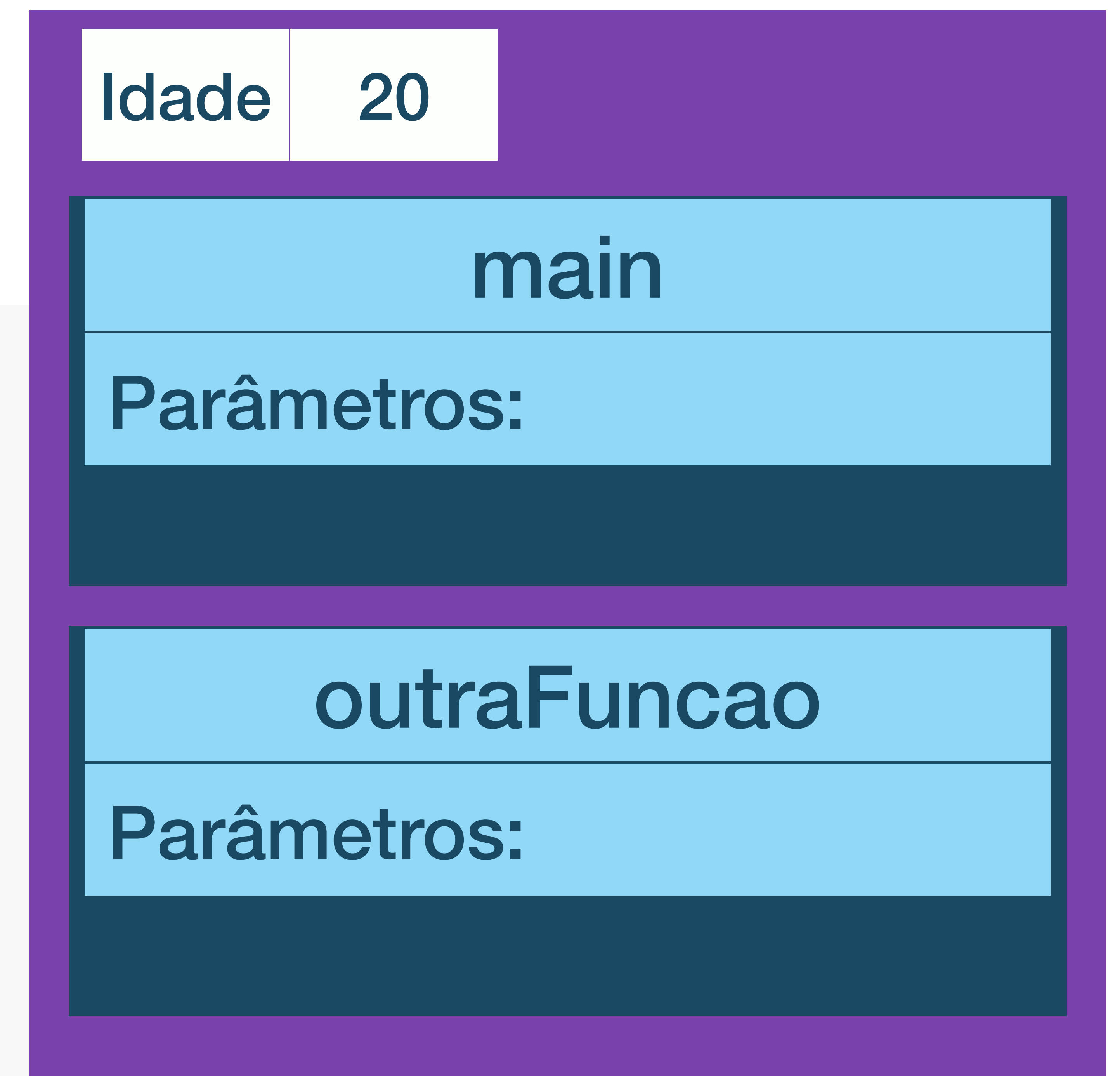
Escopo de variáveis

Escopo global

```
var idade int = soma(10, 20)

func main() {
    fmt.Println(idade)
}

func outra() {
    fmt.Println(idade)
}
```



A variável idade é visível em todo o arquivo (pacote)

Escopo de variáveis

Escopo global

- Uma variável declarada fora de qualquer função (no topo do arquivo)
  - Pode ser acessada por qualquer função dentro daquele mesmo pacote
- Uso:
  - Geralmente usada para configurações ou variáveis que precisam ser compartilhadas
  - Deve-se ter cuidado para não criar “variáveis globais” desnecessárias
    - Dificulta o rastreamento de erros

Escopo de variáveis

⚠️ Sombreamento

```go
var x = 10

func main() {
    x := 5
    fmt.Println(x)
}
```

Qual o resultado da execução deste programa ?

Se você declarar uma variável com o mesmo nome de uma externa dentro de um bloco interno, a interna "esconde" a externa

Escopo de variáveis

⚠️ Sombreamento

```go
var x = 10

func main() {
    x := 5
    fmt.Println(x)
}
```

Qual o resultado da execução deste programa ?

Se você declarar uma variável com o mesmo nome de uma externa dentro de um bloco interno, a interna "esconde" a externa

Pilha de execução

Pilha de execução

- De forma geral, construímos programas com funções que chamam funções
  - Composição de funções
- As execuções das funções são controladas pela pilha de execução
  - Funciona como a estrutura de dados Pilha
    - último que entra → primeiro que sai (LIFO)

Escopo de variáveis

Escopo local ou de bloco

Escopo de variáveis

Escopo local ou de bloco

```go
func soma(a, b int) int {
  resultado := a + b
  return resultado
}
```

Escopo de variáveis

Escopo local ou de bloco

```
func soma(a, b int) int {
  resultado := a + b
  return resultado
}

func subtrai(a, b int) int {
  resultado := soma(a, -b)
  return resultado
}
```

Escopo de variáveis

Escopo local ou de bloco

```
func soma(a, b int) int {
  resultado := a + b
  return resultado
}

func subtrai(a, b int) int {
  resultado := soma(a, -b)
  return resultado
}

func main() {
  valor := subtrai(10, 2)
  fmt.Println(valor)
}
```

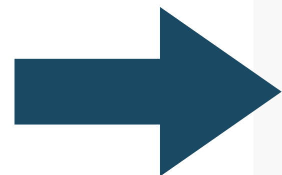
Escopo de variáveis

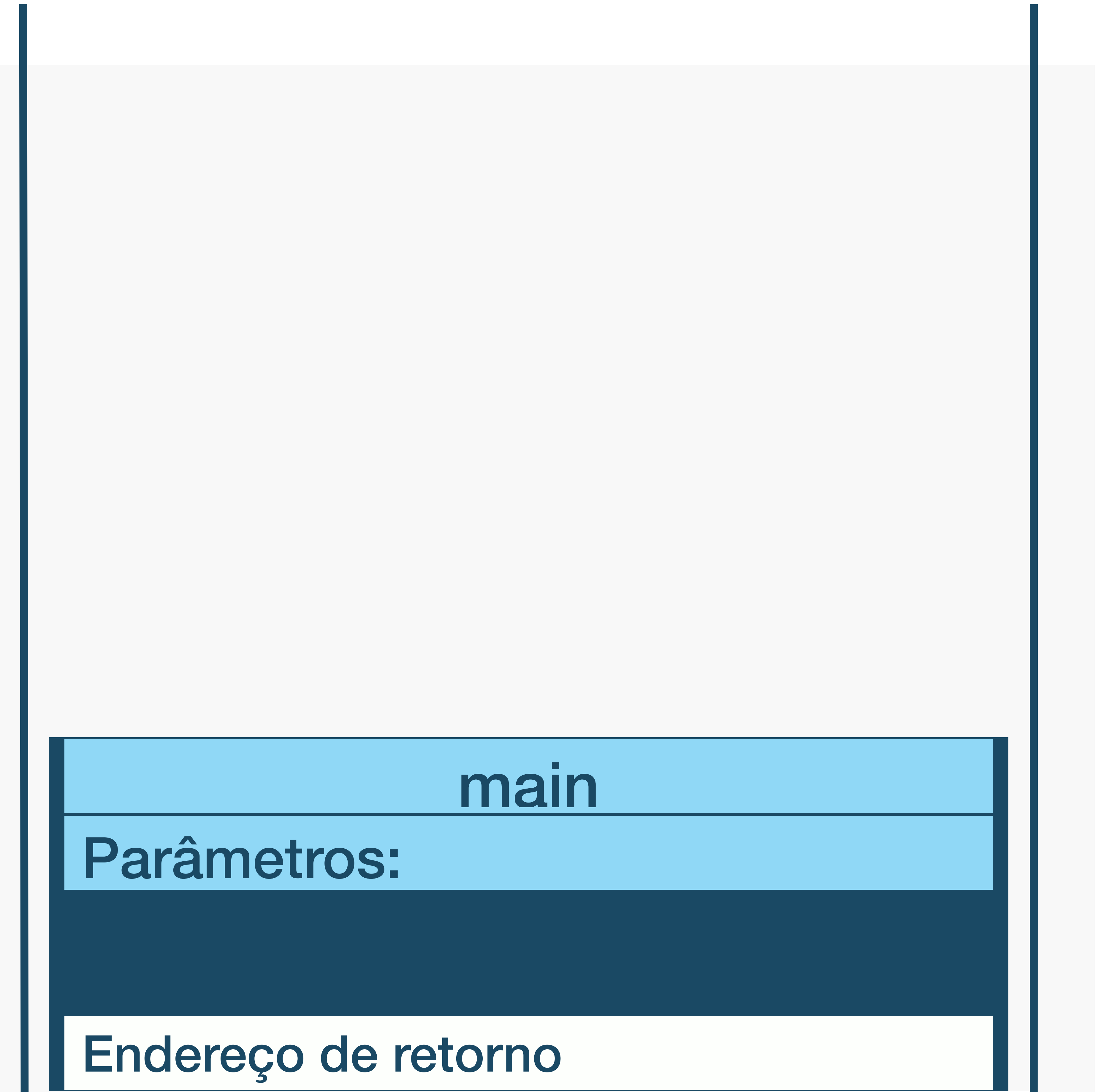
Escopo local ou de bloco

```
func soma(a, b int) int {
  resultado := a + b
  return resultado
}

func subtrai(a, b int) int {
  resultado := soma(a, -b)
  return resultado
}

func main() {
  valor := subtrai(10, 2)
  fmt.Println(valor)
}
```





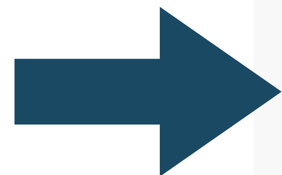
Escopo de variáveis

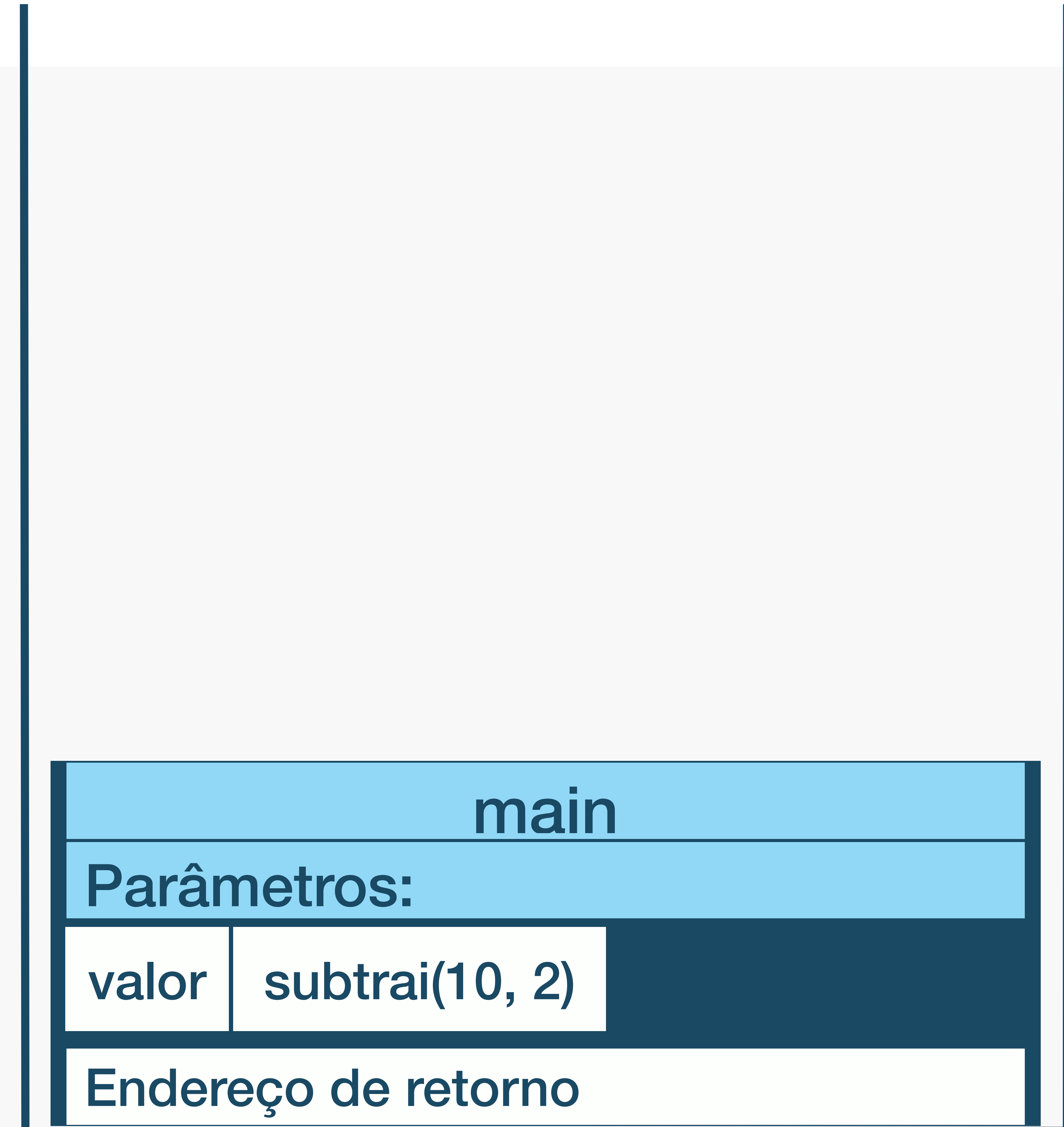
Escopo local ou de bloco

```go
func soma(a, b int) int {
  resultado := a + b
  return resultado
}

func subtrai(a, b int) int {
  resultado := soma(a, -b)
  return resultado
}

func main() {
  valor := subtrai(10, 2)
  fmt.Println(valor)
}
```





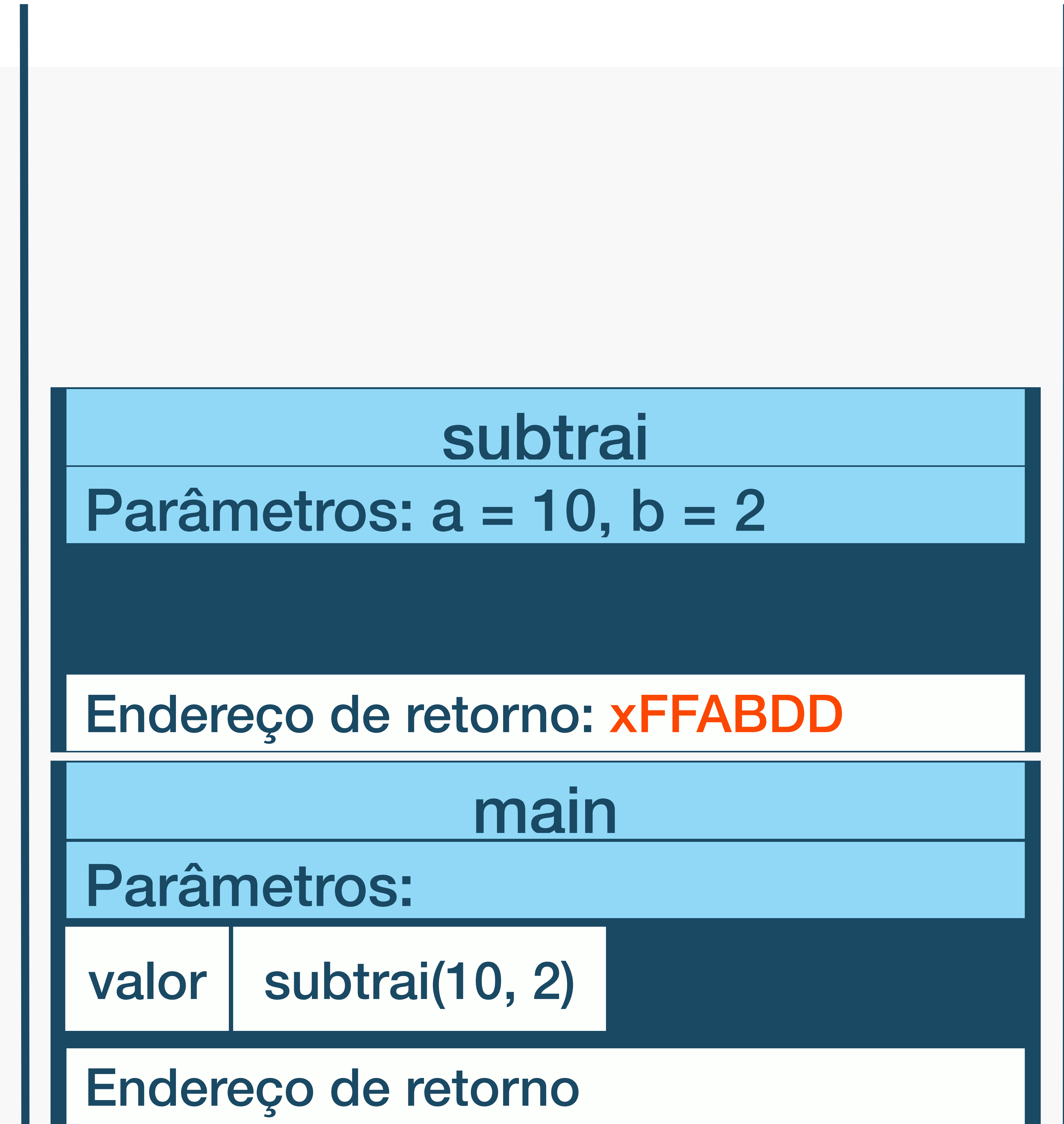
Escopo de variáveis

Escopo local ou de bloco

```go
func soma(a, b int) int {
  resultado := a + b
  return resultado
}

func subtrai(a, b int) int {
  resultado := soma(a, -b)
  return resultado
}

func main() {
  valor := subtrai(10, 2)
  fmt.Println(valor)
}
```



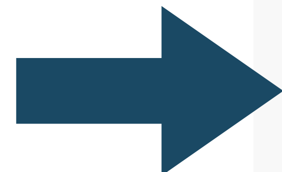
Escopo de variáveis

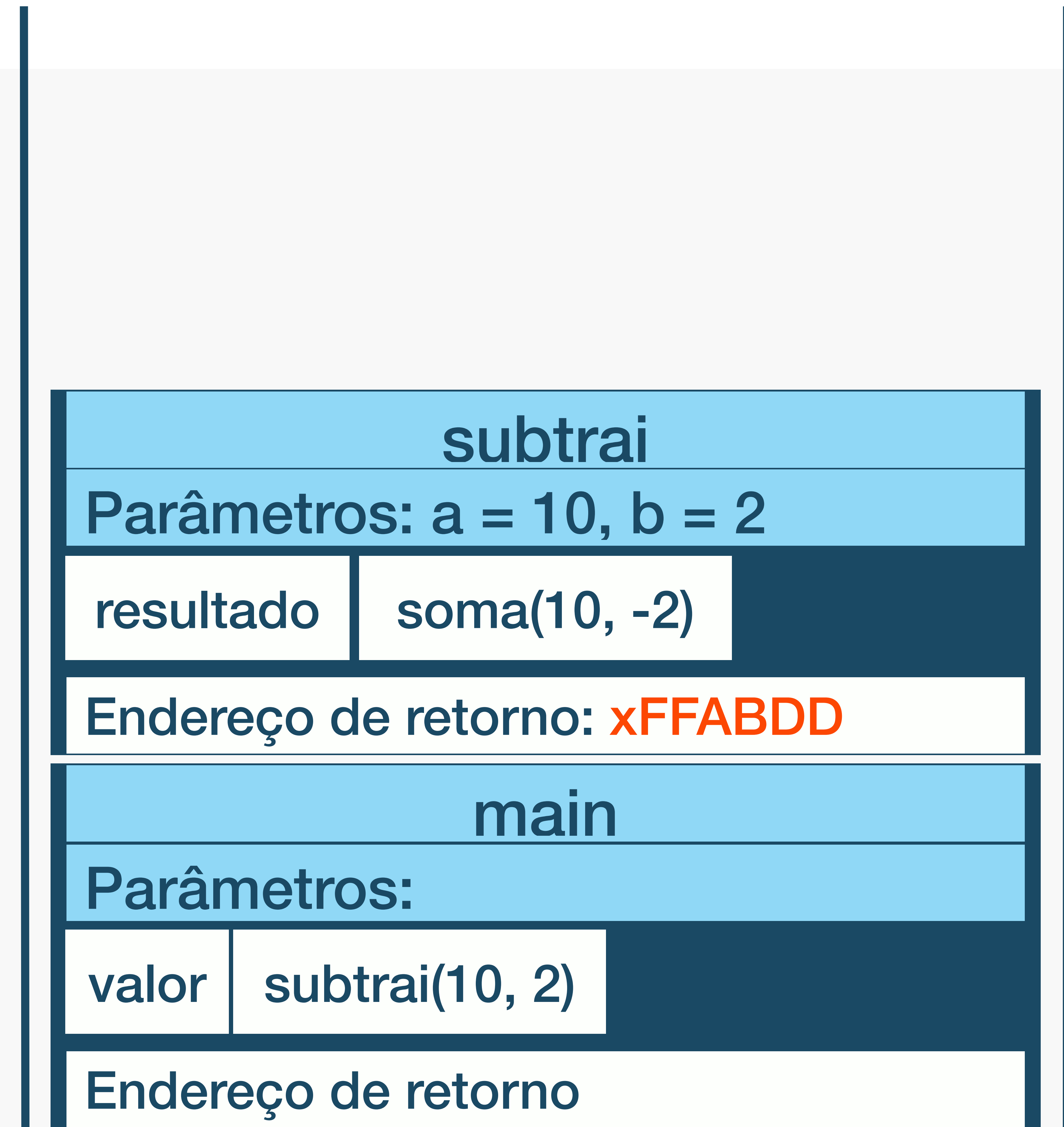
Escopo local ou de bloco

```
func soma(a, b int) int {
  resultado := a + b
  return resultado
}

func subtrai(a, b int) int {
  resultado := soma(a, -b)
  return resultado
}

func main() {
  valor := subtrai(10, 2)
  fmt.Println(valor)
}
```





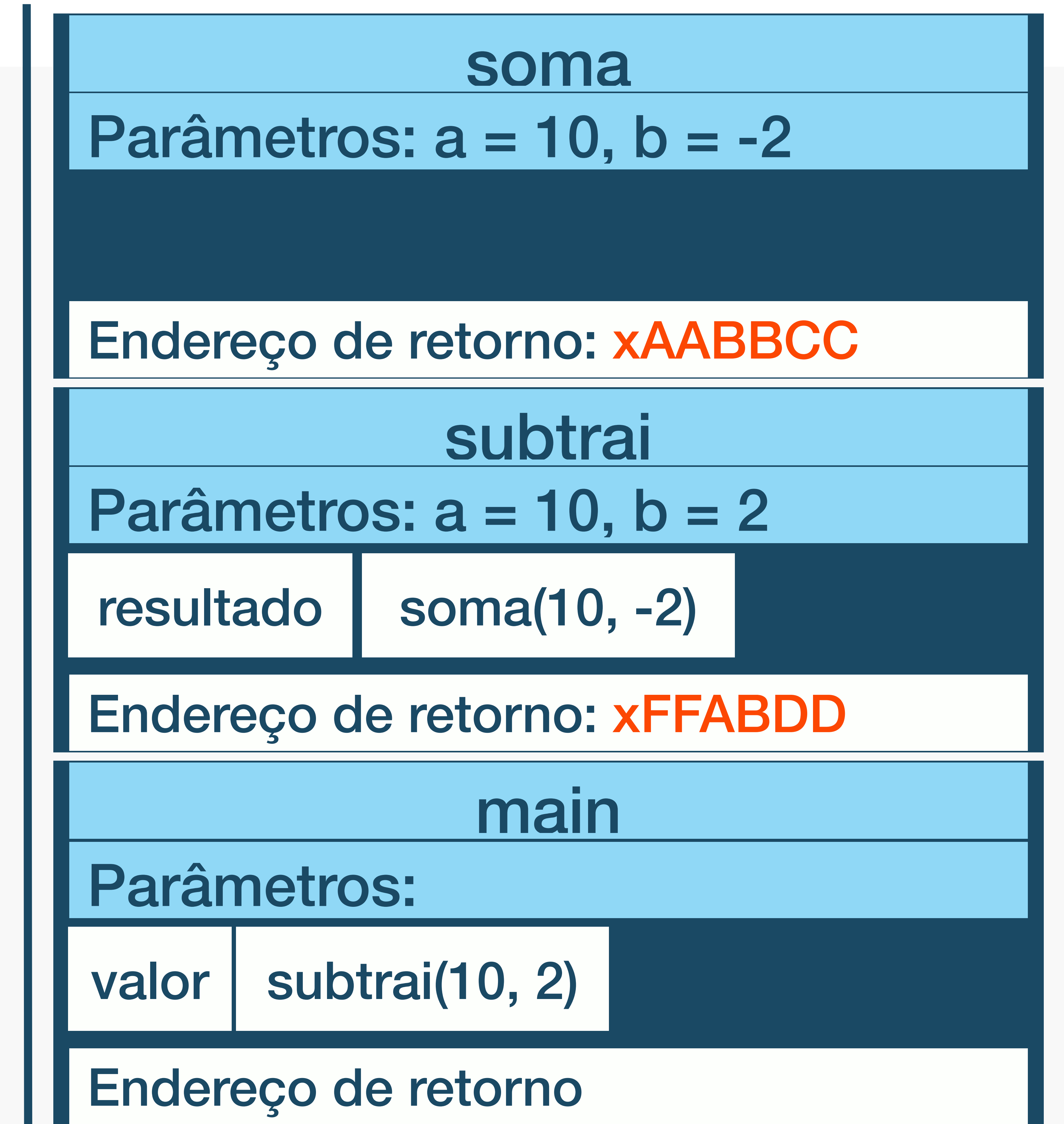
Escopo de variáveis

Escopo local ou de bloco

```
func soma(a, b int) int {
  resultado := a + b
  return resultado
}

func subtrai(a, b int) int {
  resultado := soma(a, -b)
  return resultado
}

func main() {
  valor := subtrai(10, 2)
  fmt.Println(valor)
}
```



Escopo de variáveis

Escopo local ou de bloco

```
func soma(a, b int) int {
  resultado := a + b
  return resultado
}

func subtrai(a, b int) int {
  resultado := soma(a, -b)
  return resultado
}

func main() {
  valor := subtrai(10, 2)
  fmt.Println(valor)
}
```

soma	
Parâmetros: a = 10, b = -2	
resultado	10 - 2
Endereço de retorno: xAABBCC	

subtrai	
Parâmetros: a = 10, b = 2	
resultado	soma(10, -2)
Endereço de retorno: xFFABDD	

main	
Parâmetros:	
valor	subtrai(10, 2)
Endereço de retorno	

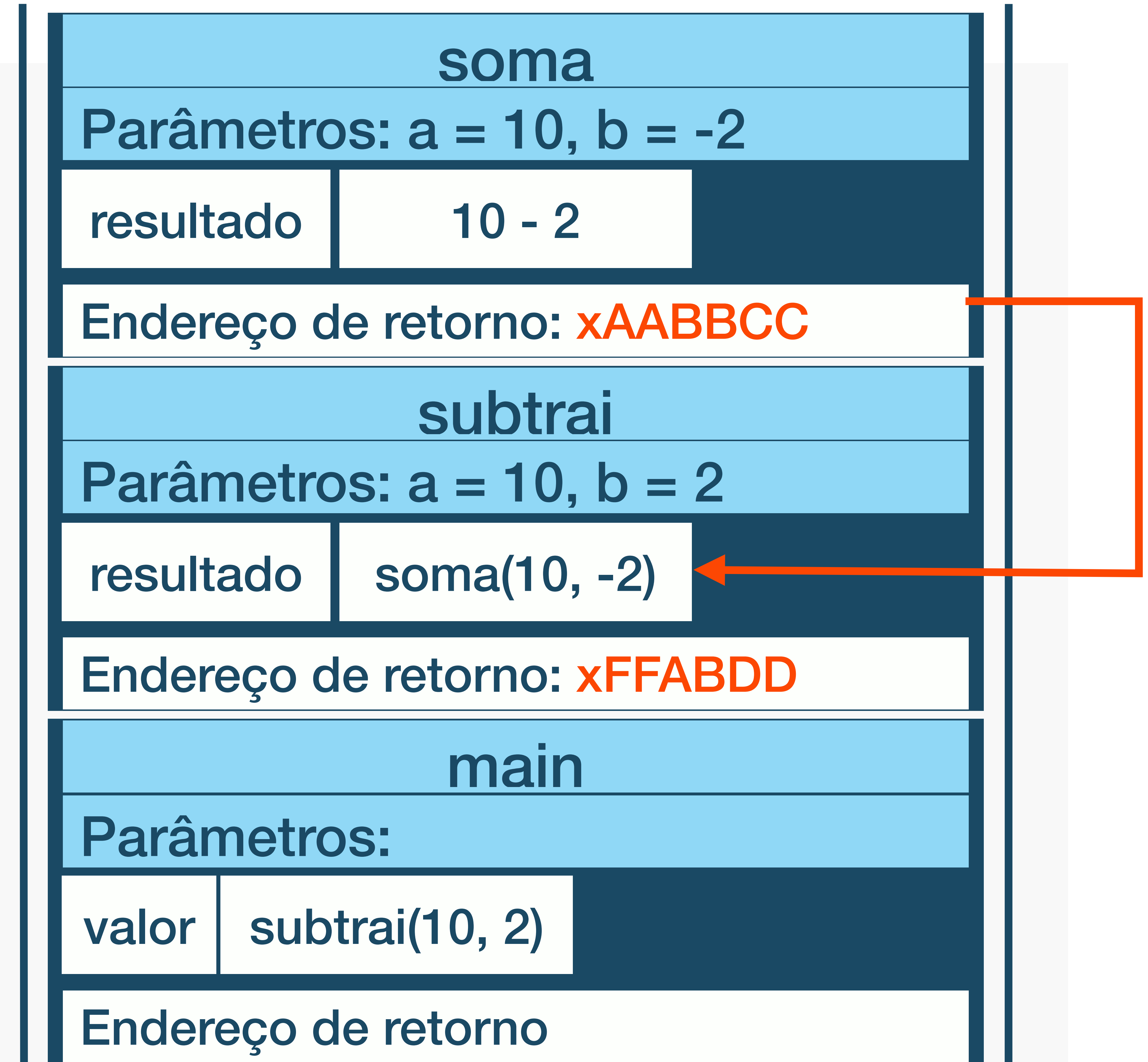
Escopo de variáveis

Escopo local ou de bloco

```go
func soma(a, b int) int {
  resultado := a + b
  return resultado
}

func subtrai(a, b int) int {
  resultado := soma(a, -b)
  return resultado
}

func main() {
  valor := subtrai(10, 2)
  fmt.Println(valor)
}
```



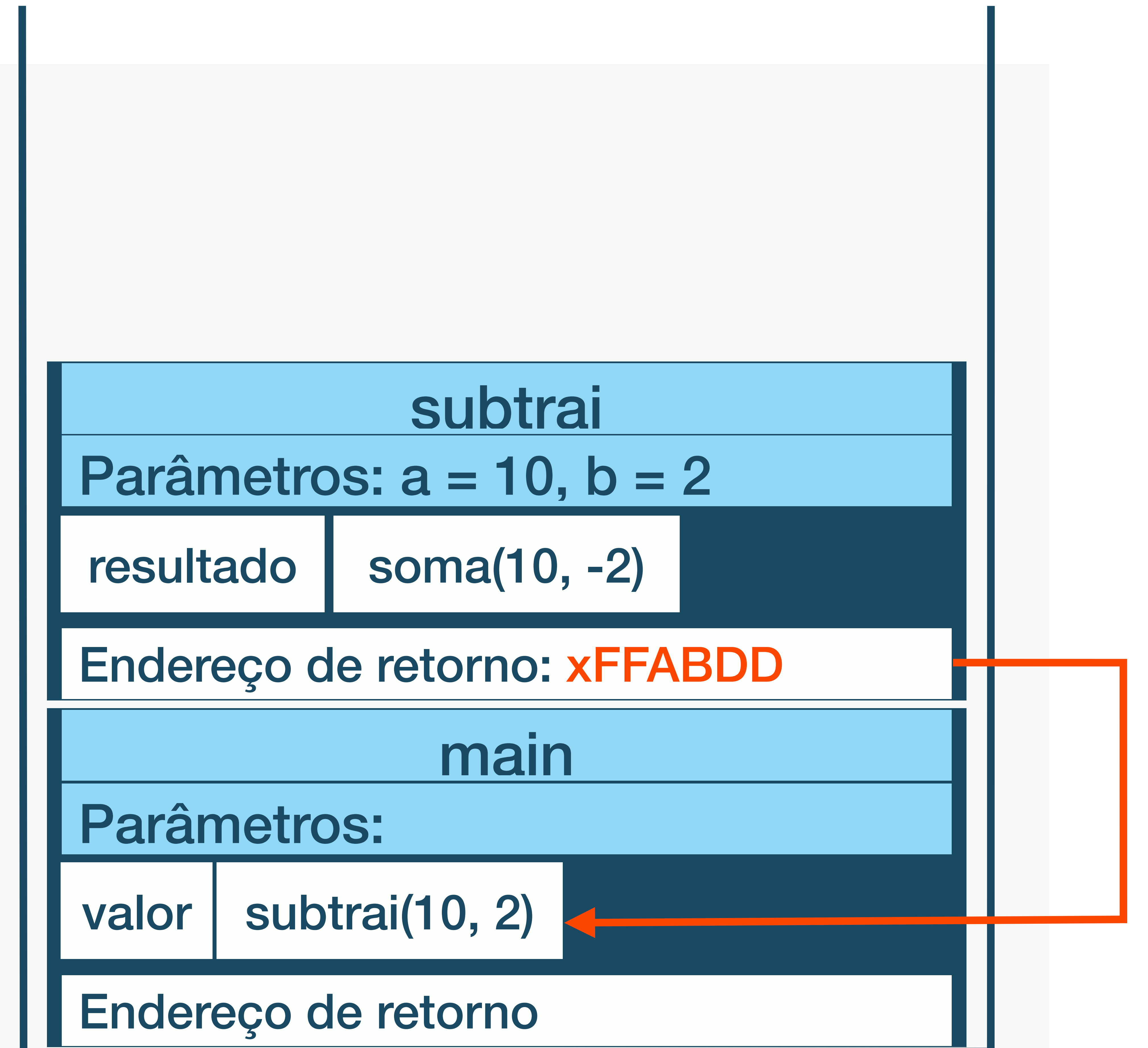
Escopo de variáveis

Escopo local ou de bloco

```go
func soma(a, b int) int {
  resultado := a + b
  return resultado
}

func subtrai(a, b int) int {
  resultado := soma(a, -b)
  return resultado
}

func main() {
  valor := subtrai(10, 2)
  fmt.Println(valor)
}
```



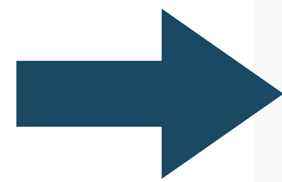
Escopo de variáveis

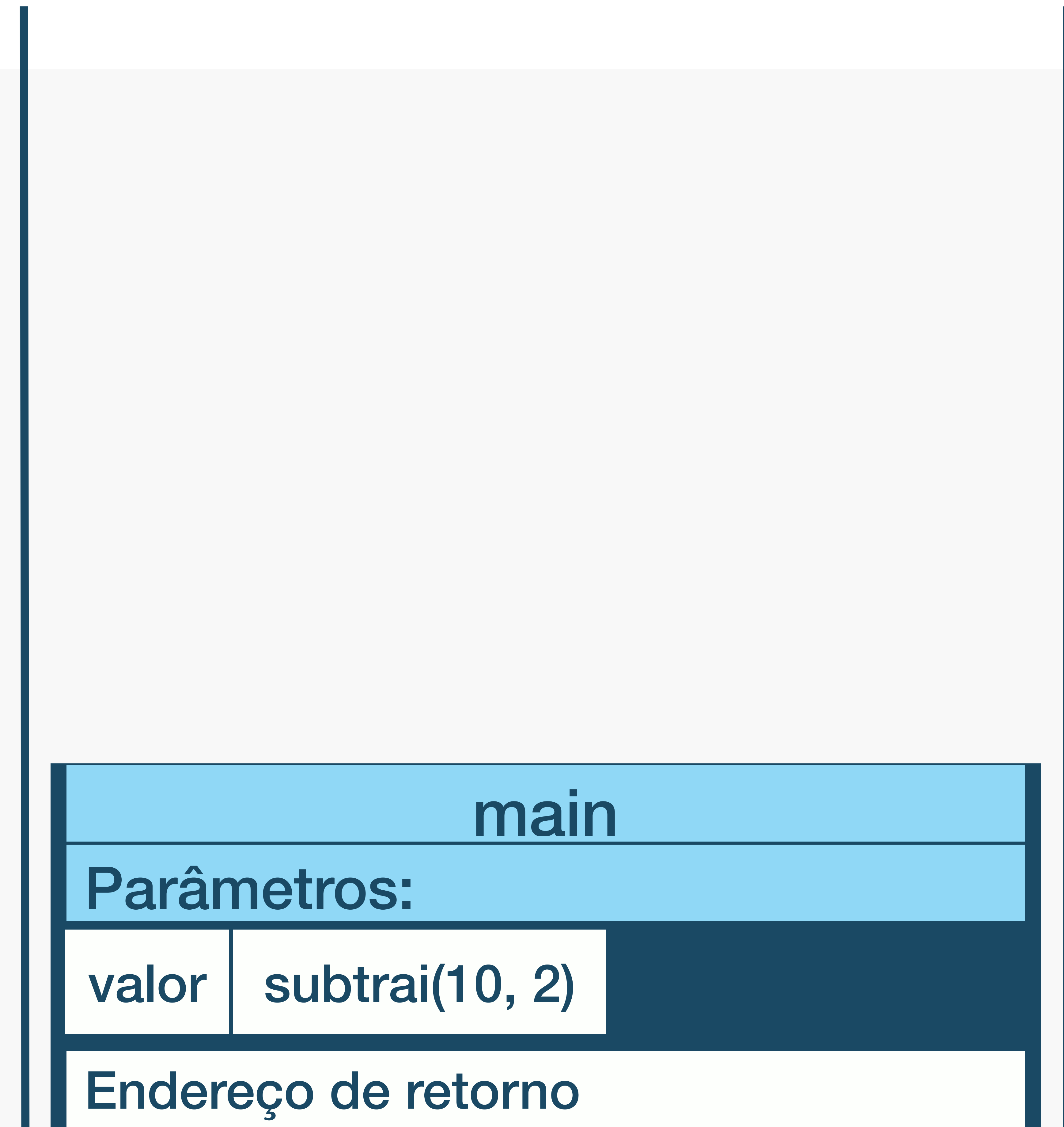
Escopo local ou de bloco

```go
func soma(a, b int) int {
  resultado := a + b
  return resultado
}

func subtrai(a, b int) int {
  resultado := soma(a, -b)
  return resultado
}

func main() {
  valor := subtrai(10, 2)
  fmt.Println(valor)
}
```





Modularização

Modularização

Princípio do menor privilégio

Isolamento

Uma função não deve ter permissão para mexer em variáveis que não são dela. Passe o dado por parâmetro!

Segurança

Se a variável *saldo* é global, um erro no *imprimirRecibo* pode zerar a conta do cliente. Mantenha local.

Memória

Variáveis locais são destruídas após o uso. Menos lixo, mais velocidade no sistema.

Clareza

Se a variável existe em 5 linhas, você só precisa entender aquelas 5 linhas para corrigir o código.

Modularização

Refatorando

```go
func main() {
  var consumo int
  fmt.Scan(&consumo)
  valor := consumo * 5.50
  if consumo > 100 { fmt.Println("Alerta: Alto Consumo") }
  fmt.Printf("Fatura: R$ %.2f", valor)
}
```

Modularização

Refatorando

```
func calcularPreco(m3 float64) float64 {
  return m3 * 5.50
}

func obterStatus(m3 float64) string {
  if m3 > 100 {
    return "ALTO CONSUMO"
  }
  return "NORMAL"
}

func main() {
  var consumo int
  fmt.Scan(&consumo)
  valor := calcularPreco(consumo)
  status := obterStatus(consumo)
  exibirRelatorio(valor, status)
}
```

Erros comuns

Erros comuns

❌ Erro 1 — Achar que função altera variável original

```
func dobrar(x int) {
    x = x * 2
}
```

Erros comuns

❌ Erro 2 — Esquecer o retorno

```go
func soma(a int, b int) int {
    a + b // erro
}
```

Erros comuns

❌ Erro 3 — Não usar o retorno

```
soma(2, 3) // correto variavel =  soma(2, 3)
```

Erros comuns

❌ Erro 4 — Nome de variável igual em escopos diferentes

```
func main() {
    var x int = 10

    if true {
        var x int = 20
        fmt.Println(x)
    }

    fmt.Println(x)
}
```

Erros comuns

❌ Erro 5— Variáveis fora do escopo

```go
if true {
    var x int = 10
}

fmt.Println(x)
```

Erros comuns

❌ Erro 6 — Parâmetros errados

```
//func soma(a int, b int)
soma(10)
```

Erros comuns

❌ Erro 7 — Tipos de parâmetros incompatíveis

```
//func soma(a int, b int)
soma(10, 2.5)
```

Referências

- Go 101