

Fundamentos da Programação

Funções

Conteúdo

- Funções e Procedimentos
- Escopo das variáveis

Funções

- Sequências de instruções que podem **receber** valores e podem **devolver** um resultado.
 - Definidas para a resolução de uma tarefa.
 - Facilitam o desenvolvimento dividindo um problema em **sub-problemas**.
 - O programa resolve o problema e as funções resolvem os sub-problemas.
- Facilitam a **reutilização**, **legibilidade** e **manutenção** de código.

Funções

- Não é necessário conhecer a sua implementação. Apenas é necessário:
 - Conhecer o seu comportamento (o que faz);
 - Fornecer informação adicional (caso seja necessário) através dos seus parâmetros;
 - Determinar o tipo de retorno da mesma (Caso seja necessário).
- Exemplos:
 - **printf("teste")** – função incluída na biblioteca **stdio.h** e que recebe como parâmetro uma string (e retorna um valor inteiro).
 - **int main()** – função que define o ponto inicial de execução.

Declaração

Sintaxe: **tipoRetorno nomeFunção(listaParâmetros) {**
 // Declaração de variáveis locais;
 // Primitivas executáveis;
 // retorno do resultado;
}

- **tipoRetorno**: tipos de dados do valor de retorno.
 - Uma função sem tipo de retorno declarado é, por defeito, do tipo **int**.
- **nomeFunção**: nome da função (segue as mesmas regras da declaração de variáveis).
 - É boa prática que represente uma **ação**, exemplo: **calculaMedia**
- **listaParâmetros**: são os parâmetros que a função recebe de “quem” a invocou.
 - Os **parâmetros** vão funcionar como **variáveis locais** da função.
 - É preciso indicar o tipo e nome de cada parâmetro. Quando se invoca uma função, o número, tipo e ordem dos argumentos enviados deve coincidir com os parâmetros na declaração da função.

Declaração

Sintaxe: `tipoRetorno nomeFunção(listaParâmetros) {`
 `// Declaração de variáveis locais;`
 `// Primitivas executáveis;`
 `// Retorno do resultado;`
`}`

- **Declaração de variáveis locais:** declaração das variáveis que serão usadas na execução da função.
- **Primitivas executáveis:** instruções responsáveis por resolver a tarefa.
- **Retorno do resultado:** termina a função e devolve o resultado a “quem” a invocou.
 - Representa-se através da instrução **return** seguido do valor ou expressão a devolver.
 - Uma função pode ter várias instruções de retorno mas apenas uma é executada em cada invocação da função.

Declaração

Sintaxe: `tipoRetorno nomeFunção(listaParâmetros) {`
 `// Declaração de variáveis locais;`
 `// Primitivas executáveis;`
 `// Retorno do resultado;`
`}`

- Podemos criar funções que não têm tipos de retorno, apenas executam as instruções.
 - Usualmente, apelidadas de procedimentos.
 - Usa-se o tipo de dados **void**.

```
...  
void imprimirLinha() {  
    puts("-----");  
}  
...
```

Declaração

Sintaxe: `tipoRetorno nomeFunção(listaParâmetros) {`
 `// Declaração de variáveis locais;`
 `// Primitivas executáveis;`
 `// Retorno do resultado;`
`}`

```
...  
long somar(int num1, int num2) {  
    long total;  
    total = num1 + num2;  
    return total;  
}  
...
```



```
...  
long somar(int num1, int num2) {  
    return num1 + num2;  
}  
...
```


Declaração

Sintaxe: `tipoRetorno nomeFunção(listaParâmetros) {`
 `// Declaração de variáveis locais;`
 `// Primitivas executáveis;`
 `// Retorno do resultado;`
`}`

```
...  
long somar(int num1, int num2) {  
    long total;  
    total = num1 + num2;  
    return total;  
}  
...
```



```
...  
long somar(int num1, int num2) {  
    return num1 + num2;  
}  
...
```

Invocação

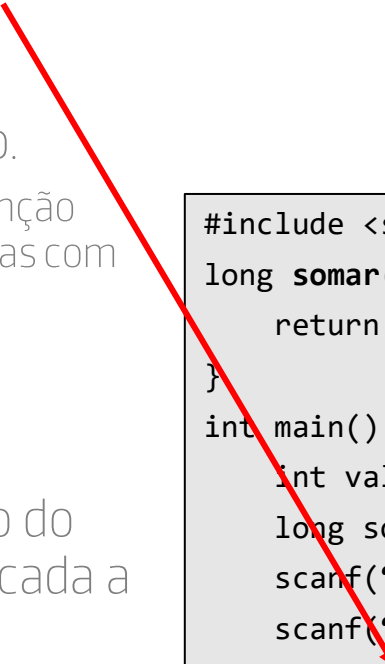
- Quando uma função é **invocada**, o bloco de código que a invoca é temporariamente “suspense”.
- Ao invocar uma função podem ser enviados argumentos, que serão recebidos pela função.
 - Estes serão armazenados em variáveis locais à função (parâmetros) que são automaticamente inicializadas com os argumentos enviados.
- De seguida são executadas as instruções da função.
- Quando esta **termina**, o controlo da execução do programa volta ao ponto onde tinha sido invocada a função.

```
#include <stdio.h>
long somar(int num1, int num2) {
    return num1 + num2;
}
int main() {
    int val1, val2;
    long soma;
    scanf("%d", &val1);
    scanf("%d", &val2);
    soma = somar(val1, val2);
    printf("%ld", soma);
    return 0;
}
```

Invocação

- Quando uma função é **invocada**, o bloco de código que a invoca é temporariamente “suspense”.
- Ao invocar uma função podem ser enviados argumentos, que serão recebidos pela função.
 - Estes serão armazenados em variáveis locais à função (parâmetros) que são automaticamente inicializadas com os argumentos enviados.
- De seguida são executadas as instruções da função.
- Quando esta **termina**, o controlo da execução do programa volta ao ponto onde tinha sido invocada a função.

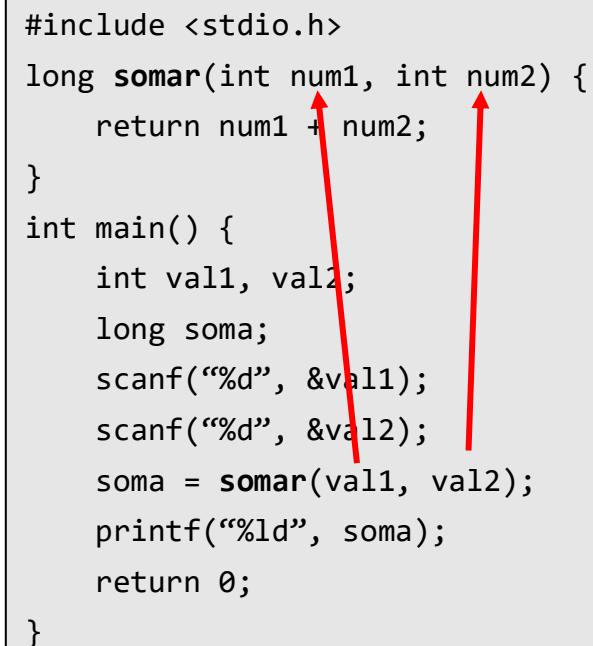
```
#include <stdio.h>
long somar(int num1, int num2) {
    return num1 + num2;
}
int main() {
    int val1, val2;
    long soma;
    scanf("%d", &val1);
    scanf("%d", &val2);
    soma = somar(val1, val2);
    printf("%ld", soma);
    return 0;
}
```



Invocação

- Quando uma função é **invocada**, o bloco de código que a invoca é temporariamente “suspense”.
- Ao invocar uma função podem ser enviados argumentos, que serão recebidos pela função.
 - Estes serão armazenados em variáveis locais à função (parâmetros) que são automaticamente inicializadas com os argumentos enviados.
- De seguida são executadas as instruções da função.
- Quando esta **termina**, o controlo da execução do programa volta ao ponto onde tinha sido invocada a função.


```
#include <stdio.h>
long somar(int num1, int num2) {
    return num1 + num2;
}
int main() {
    int val1, val2;
    long soma;
    scanf("%d", &val1);
    scanf("%d", &val2);
    soma = somar(val1, val2);
    printf("%ld", soma);
    return 0;
}
```

Two red arrows originate from the variables 'val1' and 'val2' in the 'scanf' statements within the 'main' function. They point upwards and to the right, terminating at the parameters 'num1' and 'num2' in the function signature of the 'somar' function, illustrating the passing of arguments.

Invocação

- Quando uma função é **invocada**, o bloco de código que a invoca é temporariamente “suspense”.
- Ao invocar uma função podem ser enviados argumentos, que serão recebidos pela função.
 - Estes serão armazenados em variáveis locais à função (parâmetros) que são automaticamente inicializadas com os argumentos enviados.
- De seguida são executadas as instruções da função.
- Quando esta **termina**, o controlo da execução do programa volta ao ponto onde tinha sido invocada a função.


```
#include <stdio.h>
long somar(int num1, int num2) {
    return num1 + num2;
}
int main() {
    int val1, val2;
    long soma;
    scanf("%d", &val1);
    scanf("%d", &val2);
    soma = somar(val1, val2);
    printf("%ld", soma);
    return 0;
}
```



Invocação

- Quando uma função é **invocada**, o bloco de código que a invoca é temporariamente “suspense”.
- Ao invocar uma função podem ser enviados argumentos, que serão recebidos pela função.
 - Estes serão armazenados em variáveis locais à função (parâmetros) que são automaticamente inicializadas com os argumentos enviados.
- De seguida são executadas as instruções da função.
- Quando esta **termina**, o controlo da execução do programa volta ao ponto onde tinha sido invocada a função.

```
#include <stdio.h>
long somar(int num1, int num2) {
    return num1 + num2;
}
int main() {
    int val1, val2;
    long soma;
    scanf("%d", &val1);
    scanf("%d", &val2);
    soma = somar(val1, val2);
    printf("%ld", soma);
    return 0;
}
```



Parâmetros vs. Argumentos

- Estes dois termos são por vezes usados de forma indiscriminada. Existe, no entanto, uma diferença:
 - **Parâmetros** surgem na definição das funções.
 - **Argumentos** aparecem na invocação das funções.

```
#include <stdio.h>
long somar(int num1, int num2) {
    return num1 + num2;
}
int main() {
    int val1, val2;
    long soma;
    scanf("%d", &val1);
    scanf("%d", &val2);
    soma = somar(val1, val2);
    printf("%ld", soma);
    return 0;
}
```

Onde implementar

- Uma função deve estar implementada antes da função que a irá invocar.
- Podemos invocar a função caso o **protótipo** da função esteja definida no topo.
 - Um protótipo consiste na declaração da função especificando o nome, parâmetros e tipo de retorno. Não contém o corpo da função.

Onde implementar

Problema

- Ao compilar o programa deparamo-nos com uma mensagem de erro

“...previous implicit declaration of ‘somar’ was here...”

```
#include <stdio.h>

int main(){
    printf("%ld", somar(1, 5));
    return 0;
}

long somar(int num1, int num2){
    return num1 + num2;
}
```

Onde implementar

Soluções #1

- Dar conhecimento que a função existe...
...definindo a função antes da sua invocação*.

* Em programas mais complexos, esta solução poderá tornar-se complexa de gerir senão mesmo impossível.

```
#include <stdio.h>

long somar(int num1, int num2){
    return num1 + num2;
}

int main() {
    printf("%ld", somar(1, 5));
    return 0;
}
```

Onde implementar

Soluções #2

- Dar conhecimento que a função existe...
...definindo o protótipo da função.

```
#include <stdio.h>

long somar(int num1, int num2);
// ou long somar(int, int);

int main(){
    printf("%ld", somar(1, 5));
    return 0;
}

long somar(int num1, int num2){
    return num1 + num2;
}
```

Resumo

- As funções **evitam a repetição** de código e **reduzem a complexidade**.
- Uma função deve efetuar **apenas uma tarefa**.
- Cada função tem que ter um **nome único** que servirá para que possa ser invocada.
- Ao terminar a execução pode **devolver um valor** ao bloco de código que a invocou.
- Devem ser o mais **independente** possível do restante programa.
 - Se possível, deve também ser o mais genérico possível para que a função possa ser reutilizada noutros programas.
- Uma função sem tipo de retorno é chamada de **procedimento**.

Arrays e funções

- É possível passar arrays para funções, no entanto, não pode retornar.
- Todas as dimensões são obrigatórias, menos a primeira.

```
...  
void escreverArray(int arr[], int tamanho){  
    int i = 0;  
    for (i=0; i < tamanho; ++i) {  
        printf("%d", arr[i]);  
    }  
}  
...
```

```
...  
void escreverMatriz(int mat[TAMLINS][TAMCOLS]){  
    int i = 0, j;  
    for (i=0; i < TAMLINS; ++i) {  
        for (j = 0; j < TAMCOLS; ++j)  
            printf("%d", mat[i][j]);  
        }  
    }  
}  
...
```

Escopo da variáveis

- “Região de influência” ou visibilidade de uma variável.

```
#include <stdio.h>
int a = 5, b;
void rotina(int num1) {
    printf("%d", num1);
    printf("%d", a);
    num1 = 10;
}
int main() {
    int num1, a;
    a = 8;
    num1 = 2;
    rotina(num1);
    printf("%d", num1);
    printf("%d", a);
    return 0;
}
```

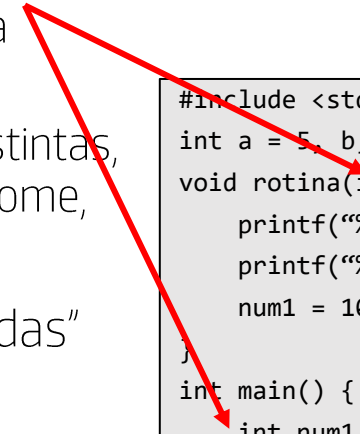
Escopo da variáveis

- As **variáveis globais** são declaradas logo no início do programa antes de qualquer função, sendo “visíveis” em todo o programa.
- Qualquer alteração aos seus valores repercute-se em todo o programa.

```
#include <stdio.h>
int a = 5, b;
void rotina(int num1) {
    printf("%d", num1);
    printf("%d", a);
    num1 = 10;
}
int main() {
    int num1, a;
    a = 8;
    num1 = 2;
    rotina(num1);
    printf("%d", num1);
    printf("%d", a);
    return 0;
}
```

Escopo da variáveis

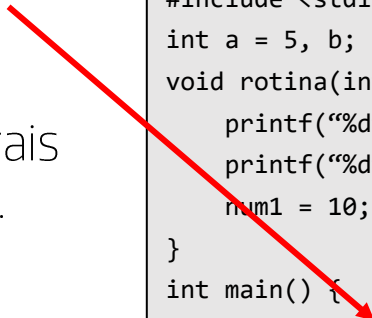
- As **variáveis locais** são declaradas nas funções.
 - Essas variáveis apenas são “visíveis” dentro da própria função.
 - Desde que estejam declaradas em funções distintas, podem existir variáveis locais com o mesmo nome, não existindo qualquer relação entre elas.
- Depois de terminada uma função, são “eliminadas” todas as suas variáveis locais.



```
#include <stdio.h>
int a = 5; b;
void rotina(int num1) {
    printf("%d", num1);
    printf("%d", a);
    num1 = 10;
}
int main() {
    int num1, a;
    a = 8;
    num1 = 2;
    rotina(num1);
    printf("%d", num1);
    printf("%d", a);
    return 0;
}
```

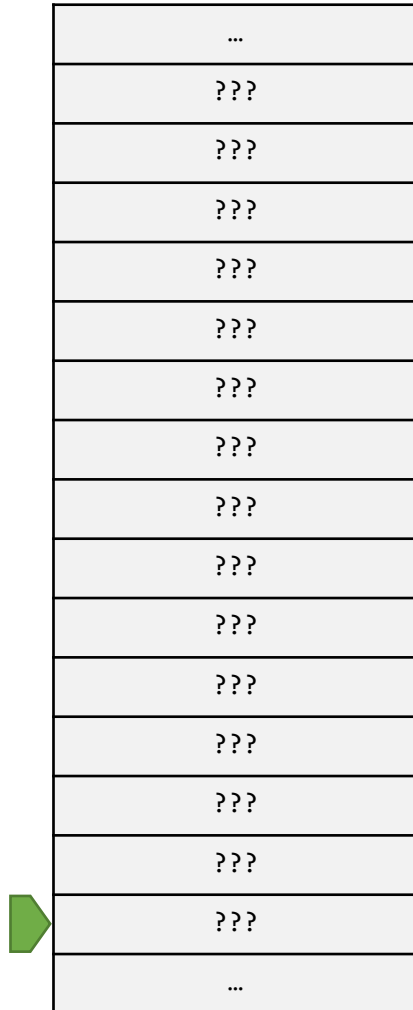

Escopo da variáveis

- Uma variável local com o mesmo nome de uma variável global tem primazia sobre esta última.
- Sempre que possível devem usar-se variáveis locais, evitando assim eventuais efeitos colaterais que ocorrem quando se usam variáveis globais.



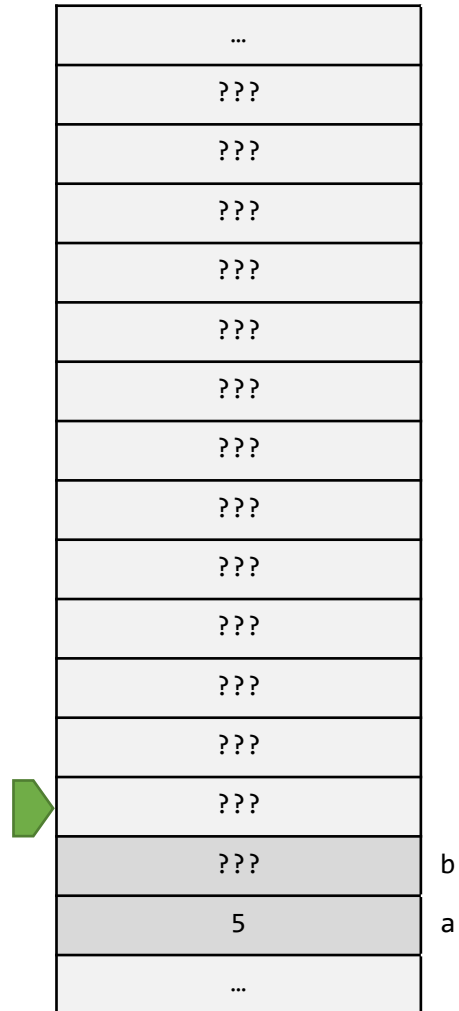
```
#include <stdio.h>
int a = 5, b;
void rotina(int num1) {
    printf("%d", num1);
    printf("%d", a);
    num1 = 10;
}
int main() {
    int num1, a;
    a = 8;
    num1 = 2;
    rotina(num1);
    printf("%d", num1);
    printf("%d", a);
    return 0;
}
```

Escopo da variáveis



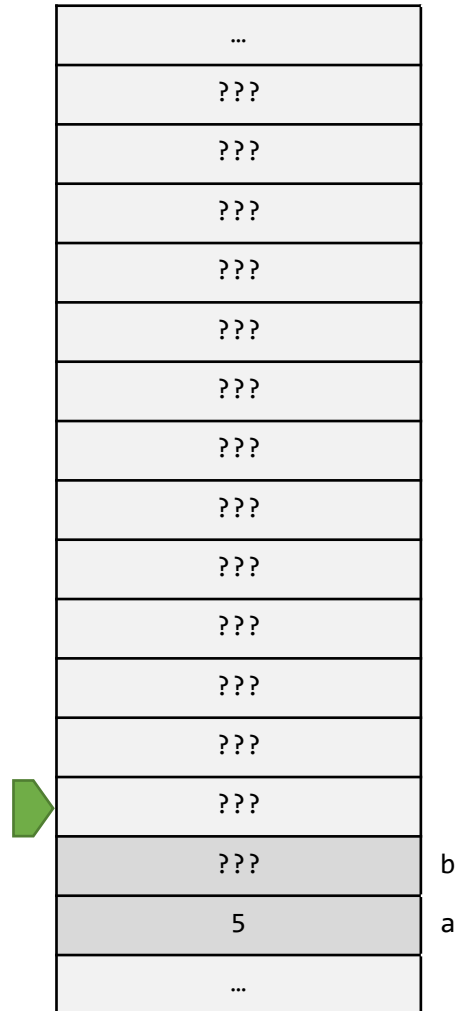
```
#include <stdio.h>
int a = 5, b;
void rotina(int num1) {
    printf("%d", num1);
    printf("%d", a);
    num1 = 10;
}
int main() {
    int num1, a;
    a = 8;
    num1 = 2;
    rotina(num1);
    printf("%d", num1);
    printf("%d", a);
    return 0;
}
```

Escopo da variáveis



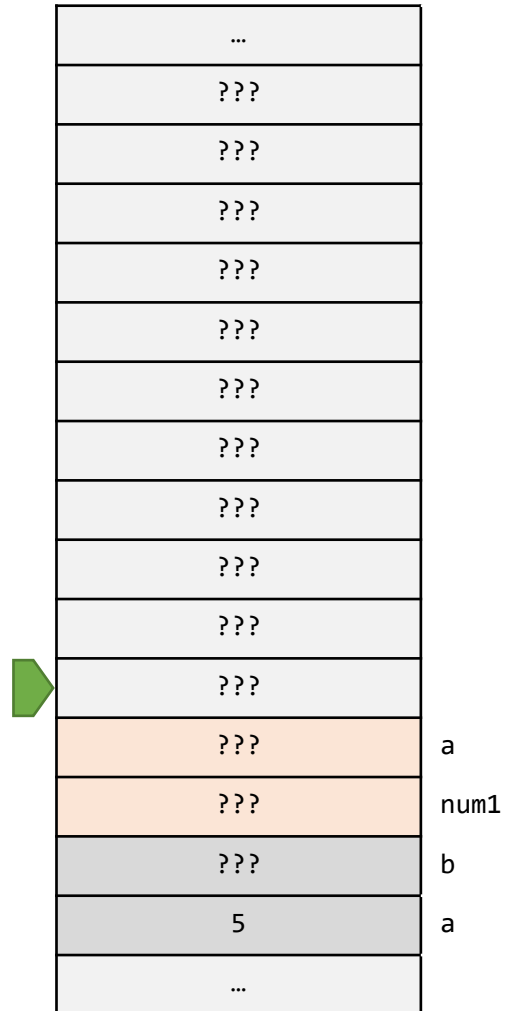
```
#include <stdio.h>
int a = 5, b;
void rotina(int num1) {
    printf("%d", num1);
    printf("%d", a);
    num1 = 10;
}
int main() {
    int num1, a;
    a = 8;
    num1 = 2;
    rotina(num1);
    printf("%d", num1);
    printf("%d", a);
    return 0;
}
```

Escopo da variáveis



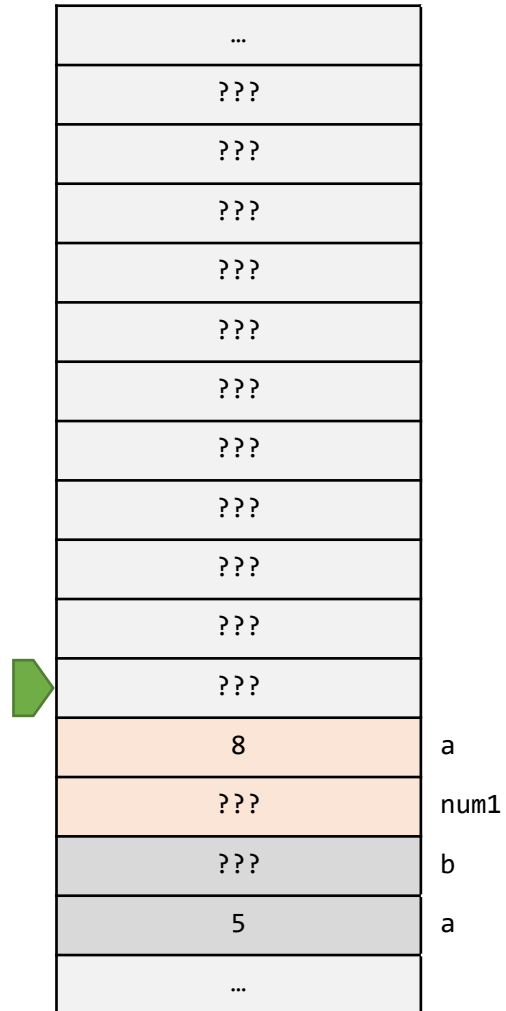
```
#include <stdio.h>
int a = 5, b;
void rotina(int num1) {
    printf("%d", num1);
    printf("%d", a);
    num1 = 10;
}
int main() {
    int num1, a;
    a = 8;
    num1 = 2;
    rotina(num1);
    printf("%d", num1);
    printf("%d", a);
    return 0;
}
```

Escopo da variáveis



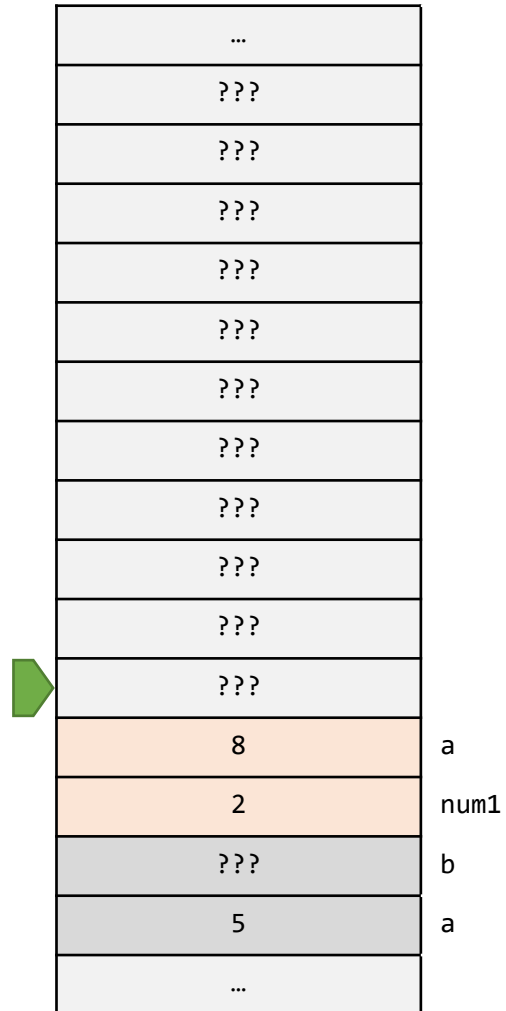
```
#include <stdio.h>
int a = 5, b;
void rotina(int num1) {
    printf("%d", num1);
    printf("%d", a);
    num1 = 10;
}
int main() {
    int num1, a;
    a = 8;
    num1 = 2;
    rotina(num1);
    printf("%d", num1);
    printf("%d", a);
    return 0;
}
```

Escopo da variáveis



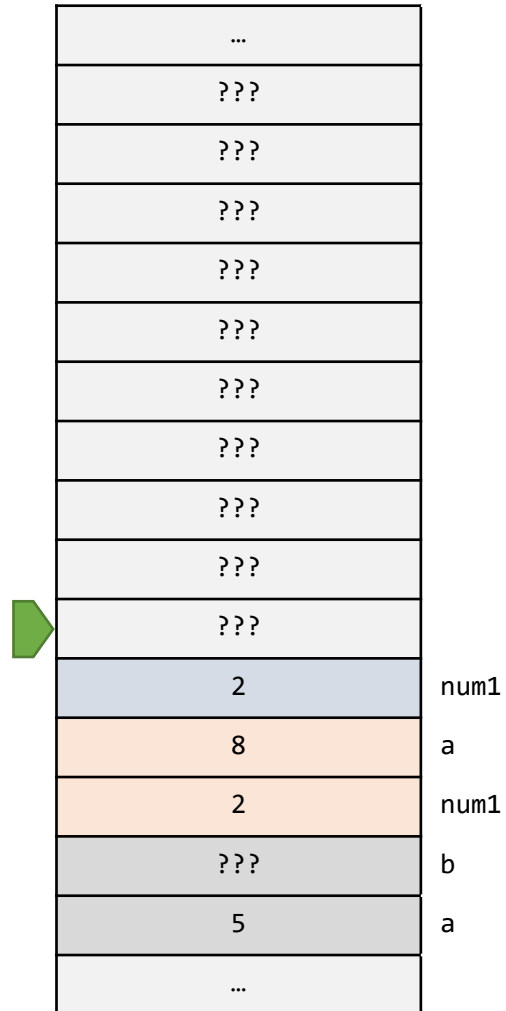
```
#include <stdio.h>
int a = 5, b;
void rotina(int num1) {
    printf("%d", num1);
    printf("%d", a);
    num1 = 10;
}
int main() {
    int num1, a;
    a = 8;
    num1 = 2;
    rotina(num1);
    printf("%d", num1);
    printf("%d", a);
    return 0;
}
```

Escopo da variáveis



```
#include <stdio.h>
int a = 5, b;
void rotina(int num1) {
    printf("%d", num1);
    printf("%d", a);
    num1 = 10;
}
int main() {
    int num1, a;
    a = 8;
    num1 = 2;
    rotina(num1);
    printf("%d", num1);
    printf("%d", a);
    return 0;
}
```

Escopo da variáveis

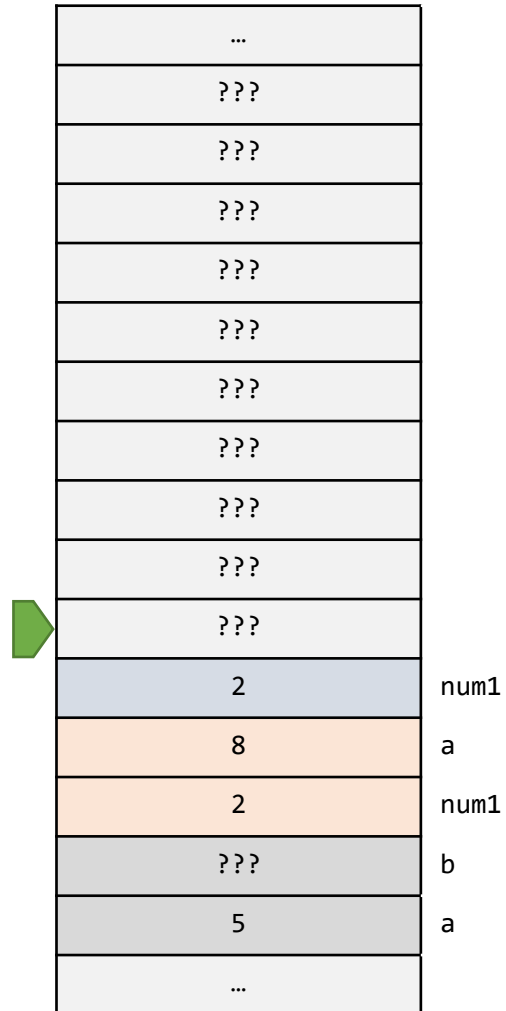


```
#include <stdio.h>
int a = 5, b;

void rotina(int num1) {
    printf("%d", num1);
    printf("%d", a);
    num1 = 10;
}

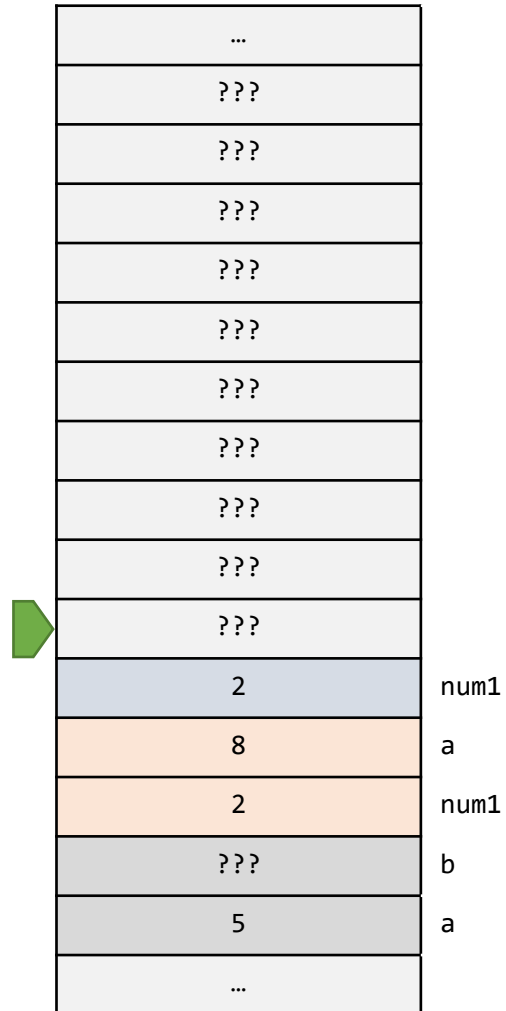
int main() {
    int num1, a;
    a = 8;
    num1 = 2;
    rotina(num1);
    printf("%d", num1);
    printf("%d", a);
    return 0;
}
```


Escopo da variáveis



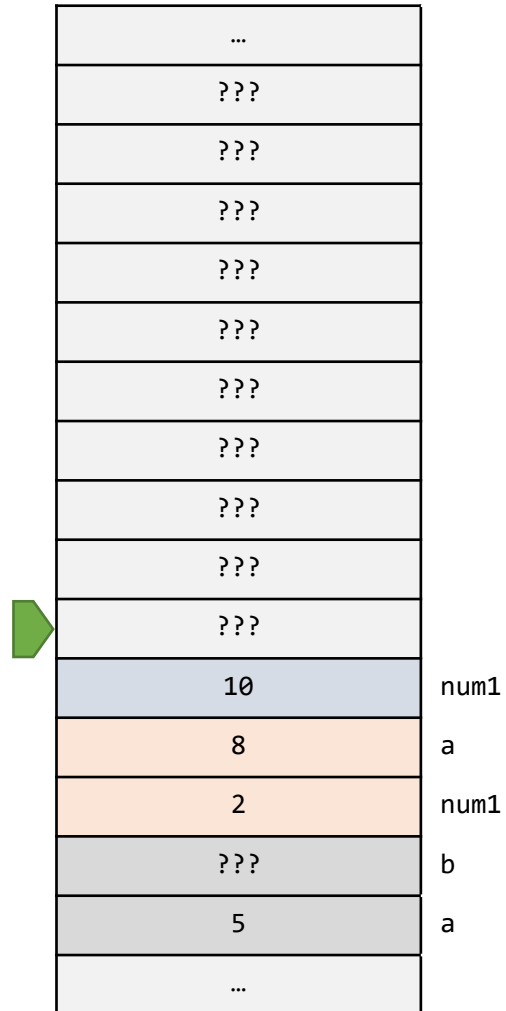
```
#include <stdio.h>
int a = 5, b;
void rotina(int num1) {
    printf("%d", num1);
    printf("%d", a);
    num1 = 10;
}
int main() {
    int num1, a;
    a = 8;
    num1 = 2;
    rotina(num1);
    printf("%d", num1);
    printf("%d", a);
    return 0;
}
```

Escopo da variáveis



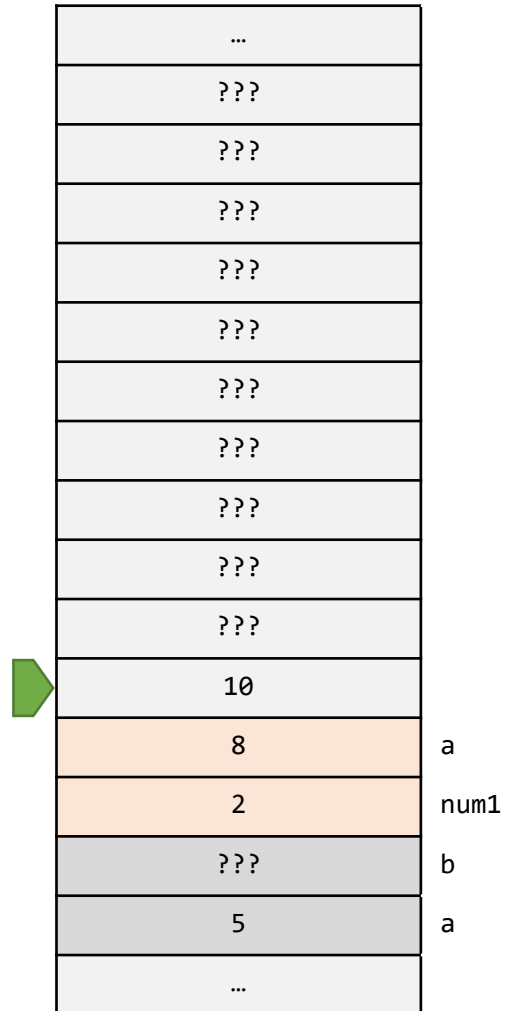
```
#include <stdio.h>
int a = 5, b;
void rotina(int num1) {
    printf("%d", num1);
    printf("%d", a);
    num1 = 10;
}
int main() {
    int num1, a;
    a = 8;
    num1 = 2;
    rotina(num1);
    printf("%d", num1);
    printf("%d", a);
    return 0;
}
```

Escopo da variáveis



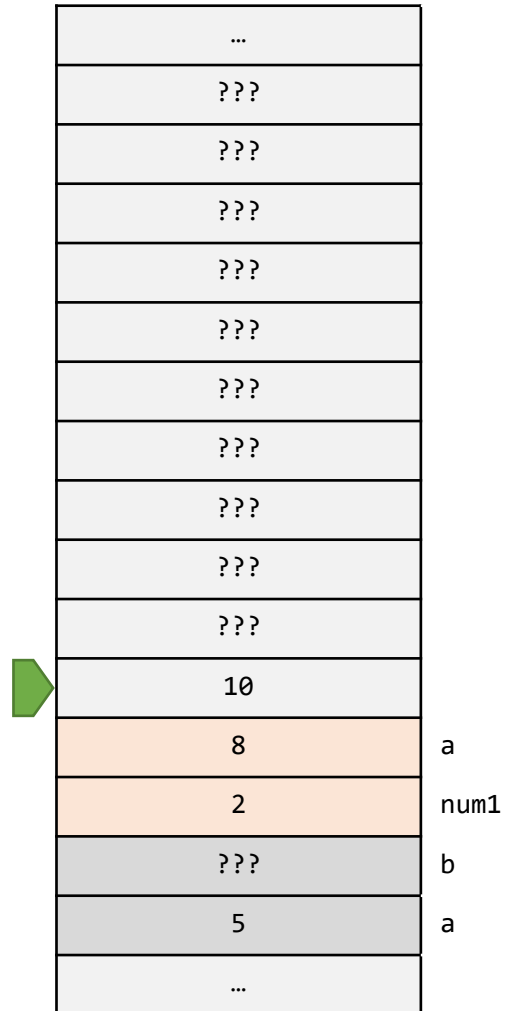
```
#include <stdio.h>
int a = 5, b;
void rotina(int num1) {
    printf("%d", num1);
    printf("%d", a);
    num1 = 10;
}
int main() {
    int num1, a;
    a = 8;
    num1 = 2;
    rotina(num1);
    printf("%d", num1);
    printf("%d", a);
    return 0;
}
```

Escopo da variáveis



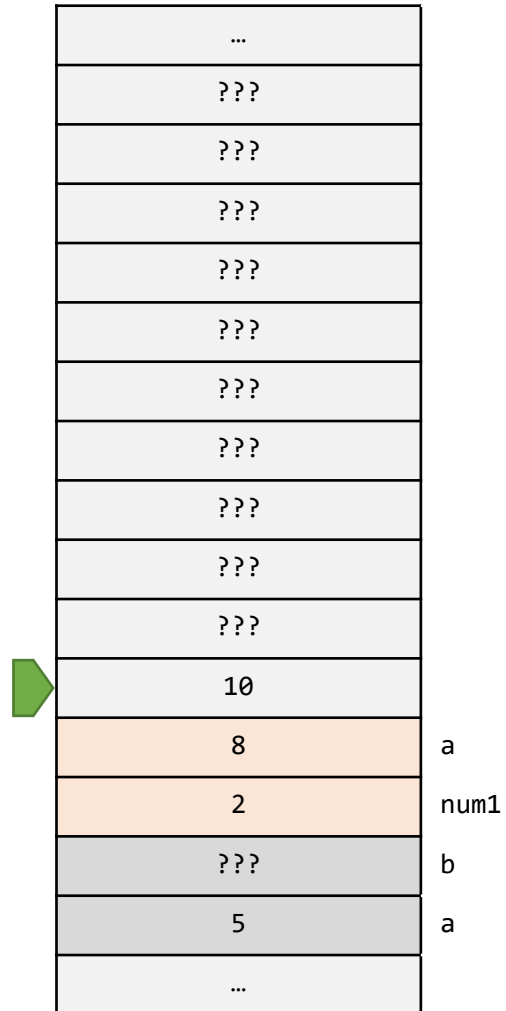
```
#include <stdio.h>
int a = 5, b;
void rotina(int num1) {
    printf("%d", num1);
    printf("%d", a);
    num1 = 10;
}
int main() {
    int num1, a;
    a = 8;
    num1 = 2;
    rotina(num1);
    printf("%d", num1);
    printf("%d", a);
    return 0;
}
```

Escopo da variáveis



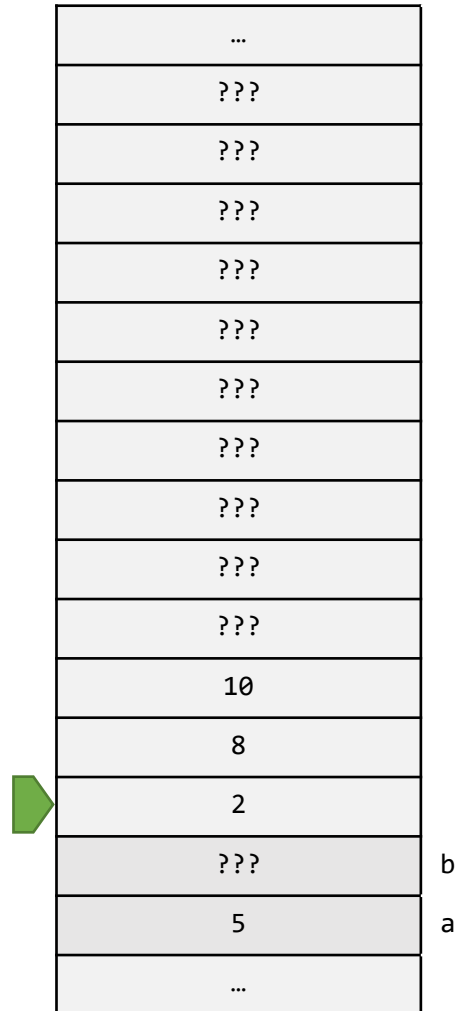
```
#include <stdio.h>
int a = 5, b;
void rotina(int num1) {
    printf("%d", num1);
    printf("%d", a);
    num1 = 10;
}
int main() {
    int num1, a;
    a = 8;
    num1 = 2;
    rotina(num1);
    printf("%d", num1);
    printf("%d", a);
    return 0;
}
```

Escopo da variáveis



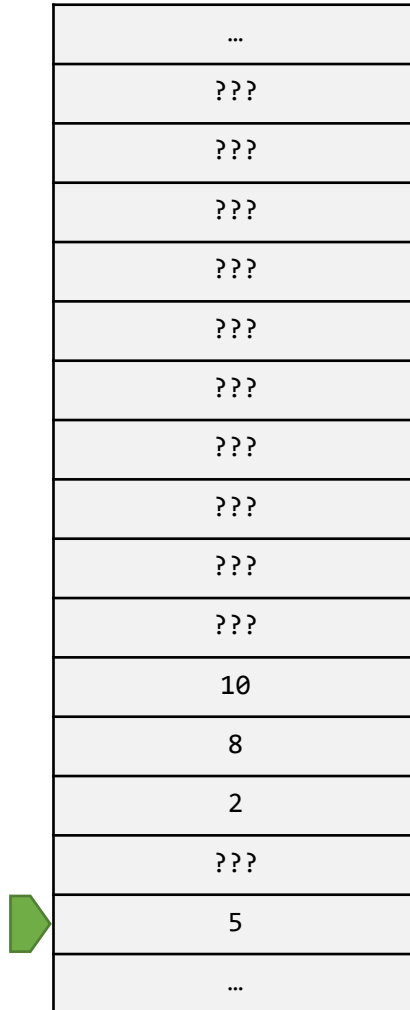
```
#include <stdio.h>
int a = 5, b;
void rotina(int num1) {
    printf("%d", num1);
    printf("%d", a);
    num1 = 10;
}
int main() {
    int num1, a;
    a = 8;
    num1 = 2;
    rotina(num1);
    printf("%d", num1);
    printf("%d", a);
    return 0;
}
```

Escopo da variáveis



```
#include <stdio.h>
int a = 5, b;
void rotina(int num1) {
    printf("%d", num1);
    printf("%d", a);
    num1 = 10;
}
int main() {
    int num1, a;
    a = 8;
    num1 = 2;
    rotina(num1);
    printf("%d", num1);
    printf("%d", a);
    return 0;
}
```

Escopo da variáveis



...
???
???
???
???
???
???
???
???
???
10
8
2
???
5
...

```
#include <stdio.h>
int a = 5, b;
void rotina(int num1) {
    printf("%d", num1);
    printf("%d", a);
    num1 = 10;
}
int main() {
    int num1, a;
    a = 8;
    num1 = 2;
    rotina(num1);
    printf("%d", num1);
    printf("%d", a);
    return 0;
}
```


Funções



Algoritmo: Calculadora

início

ler val1

ler val2

soma \leftarrow Soma val1, val2

Escrever soma

fim

Algoritmo: Soma

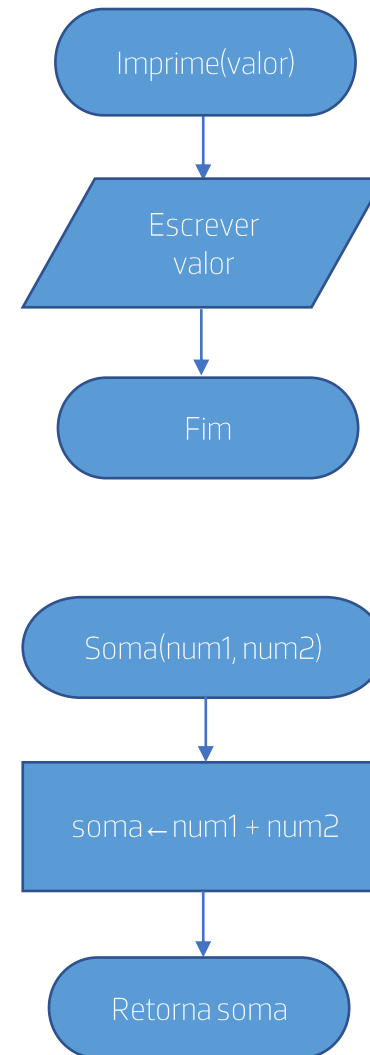
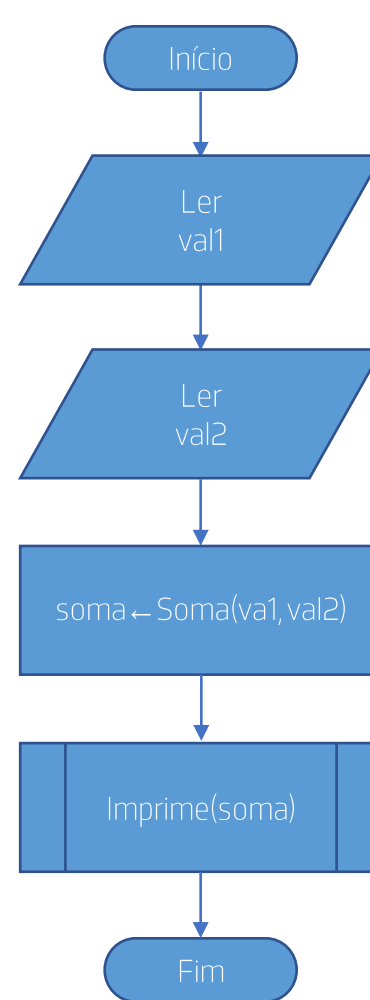
input: num1, num2

início

soma \leftarrow num1 + num2

retorna soma

fim



Leitura recomendada

- (Capítulo 5) Damas, L. Linguagem C; FCA – Editora de Informática, Lda, 1999; ISBN 9789727221561.

