Funções

Roteiro

Função

- Programas que fazem uma tarefa específica;
- Que podem ser utilizadas por qualquer outra função;
- Não precisar reimplementar uma funcionalidade já implementada;
- C é uma linguagem orientada a procedimentos
- Exemplos:
  - printf: imprime na saída padrão (tela)
  - scanf: lê da entrada padrão (teclado)
  - strlen: calcula o tamanho da string
  - strcmp: compara duas strings
  - toupper: converte caractere para maiúsculo
  - sqrt: raiz quadrada

Roteiro

Definição: implementação de uma função

- Componentes:

```
1 tipo_retorno nome_funcao()
2 {
3     ...
4     tipo_retorno a;
5     ...
6     return a;
7 }
```

  - **tipo_retorno**: o que a função devolve para o código que chamou:
    - ★ int, double, float, char;
    - ★ void (vazio, nada);
  - **nome_funcao**: o nome utilizado para chamar a função;
  - **return**: palavra reservada que indica o que será retornado;
- A função principal (**main**) também pode retornar um valor:
  - igual a 0: sucesso
  - diferente de 0: erro

```
1 int main() {
2     ...
3     return 0;
4 }
```

Exemplos

```c
int quadrado()
{
    int n;
    scanf("%d", &n);
    return n*n;
}

void imprime()
{
    printf("Ola mundo!\n");
}

float area_triangulo()
{
    float base, altura;
    scanf("%f", &base, &altura);
    return (base*altura)/2;
}
```

Retorno

- A função **scanf** retorna:
  - man scanf
  - **Número de itens** de entrada **combinados e atribuído** com sucesso;
  - O valor **EOF** é retornado se o final da entrada é alcançado antes da primeira leitura ou falha de correspondência ou erro de leitura.

```
1 while(scanf("%d", &d) != EOF) { } //ctrl+d = end of file
2 while(scanf("%d", &d) == 1)   { }
3
```

```
1 int somas() {
2     int a, s=0;
3     while(scanf("%d", &a) != EOF) {
4         s=s+a;
5     }
6     return s;
7 }
```

Retorno

- A função **printf** retorna:
  - man 3 printf
  - Sucesso: **número de caracteres impressos**
  - Erro: **número negativo**

```c
int a = printf("alo\n");
printf("%d\n", a);
/*
Saida
alo
4
*/
```

Exemplos - return

```c
int procura_v1() {
    int v[10], x, i;
    for(i=0; i<10; i++) {
        scanf("%d", &v[i]);
    }

    scanf("%d", &x);
    for(i=0; i<10 && v[i]!=x; i++);
    return i;
}

int procura_v2() {
    int v[10], x, i;
    for(i=0; i<10; i++) {
        scanf("%d", &v[i]);
    }

    scanf("%d", &x);
    for(i=0; i<10; i++) {
        if(v[i]==x) return i;
    }
    return i;
}
```

Exemplos - return

```c
int repete() {
    int v[10];
    for(int i=0; i<10; i++) {
        scanf("%d", &v[i]);
    }
    for(int i=0; i<10; i++) {
        for(int j=i+1; j<10; j++) {
            if(v[i]==v[j]) return 1;
        }
    }
    return 0;
}
```

Exemplos - return vazio

```c
void f1() {
    int x;
    while(scanf("%d", &x)==1) {
        if(x<0) return;
        printf("%d\n", x);
    }
    printf("fim\n");
}

void f2() {
    int x;
    while(scanf("%d", &x)==1) {
        if(x<0) break;
        printf("%d\n", x);
    }
    printf("fim\n");
}

void f3() {
    int x;
    while(scanf("%d", &x)==1 && x>=0) {
        printf("%d\n", x);
    }
    printf("fim\n");
}
```

- Entrada: 1 2 3 -1
- Saídas:
  - f1: ??

Exemplos - return vazio

```c
void f1() {
    int x;
    while(scanf("%d", &x)==1) {
        if(x<0) return;
        printf("%d\n", x);
    }
    printf("fim\n");
}

void f2() {
    int x;
    while(scanf("%d", &x)==1) {
        if(x<0) break;
        printf("%d\n", x);
    }
    printf("fim\n");
}

void f3() {
    int x;
    while(scanf("%d", &x)==1 && x>=0) {
        printf("%d\n", x);
    }
    printf("fim\n");
}
```

- Entrada: 1 2 3 -1
- Saídas:
  - f1: 1 2 3
  - f2: ??

Exemplos - return vazio

```c
void f1() {
    int x;
    while(scanf("%d", &x)==1) {
        if(x<0) return;
        printf("%d\n", x);
    }
    printf("fim\n");
}

void f2() {
    int x;
    while(scanf("%d", &x)==1) {
        if(x<0) break;
        printf("%d\n", x);
    }
    printf("fim\n");
}

void f3() {
    int x;
    while(scanf("%d", &x)==1 && x>=0) {
        printf("%d\n", x);
    }
    printf("fim\n");
}
```

- Entrada: 1 2 3 -1
- Saídas:
  - f1: 1 2 3
  - f2: 1 2 3 fim
  - f3: ??

Exemplos - return vazio

```c
void f1() {
    int x;
    while(scanf("%d", &x)==1) {
        if(x<0) return;
        printf("%d\n", x);
    }
    printf("fim\n");
}

void f2() {
    int x;
    while(scanf("%d", &x)==1) {
        if(x<0) break;
        printf("%d\n", x);
    }
    printf("fim\n");
}

void f3() {
    int x;
    while(scanf("%d", &x)==1 && x>=0) {
        printf("%d\n", x);
    }
    printf("fim\n");
}
```

- Entrada: 1 2 3 -1
- Saídas:
  - f1: 1 2 3
  - f2: 1 2 3 fim
  - f3: 1 2 3 fim

Roteiro

Definição/Criação da função - ANTES do uso

```c
#include <stdio.h>

int par() {
    int a;
    while(scanf("%d", &a)==1 && a%2!=0);
    return a;
}

int soma_pares() {
    int a = par() + par();
    return a;
}

int main() {
    printf("%d\n", soma_pares());
    return 0;
}
```

Definição/Criação da função - DEPOIS do uso

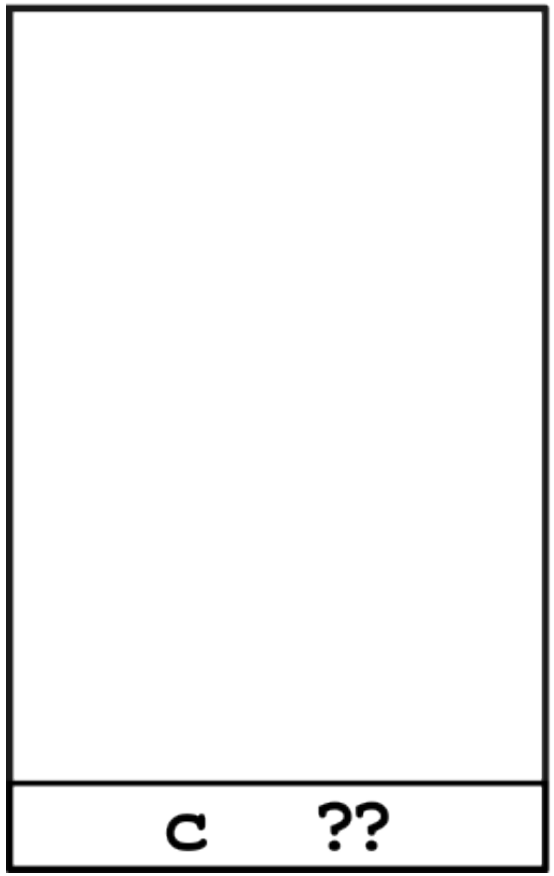
- Protótipo: assinatura da função

```c
#include <stdio.h>

//tipo_retorno nome_funcao();
//ponto e virgula no final
int soma_pares();
int par();

int main() {
    printf("%f\n", soma_pares());
    return 0;
}

int par() {
    int a;
    while(scanf("%f", &a)==1 && a%2!=0);
    return a;
}

int soma_pares() {
    int a = par() + par();
    return a;
}
```

Roteiro

Invocação/Chamada da função

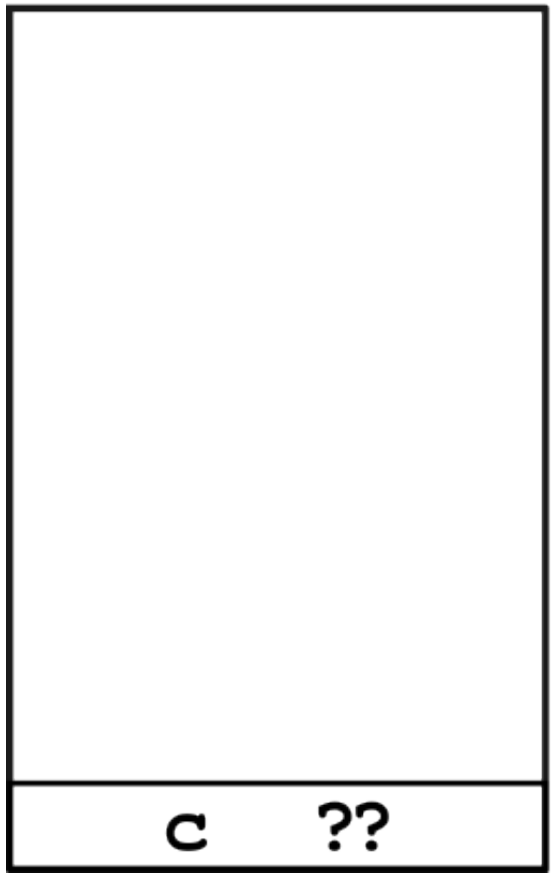
- Quando a função é chamada, o fluxo de execução do programa é desviado
- A função chamada inicia seu fluxo de execução
- Quando a função invocada termina, o fluxo volta para a função invocadora




```
int soma( ) {
    int a = 5;
    int b = 10;
    int c = a+b;
    return c;
}

int main(){
    int c;
    c = soma();
    return 0;
}
```

Invocação/Chamada da função

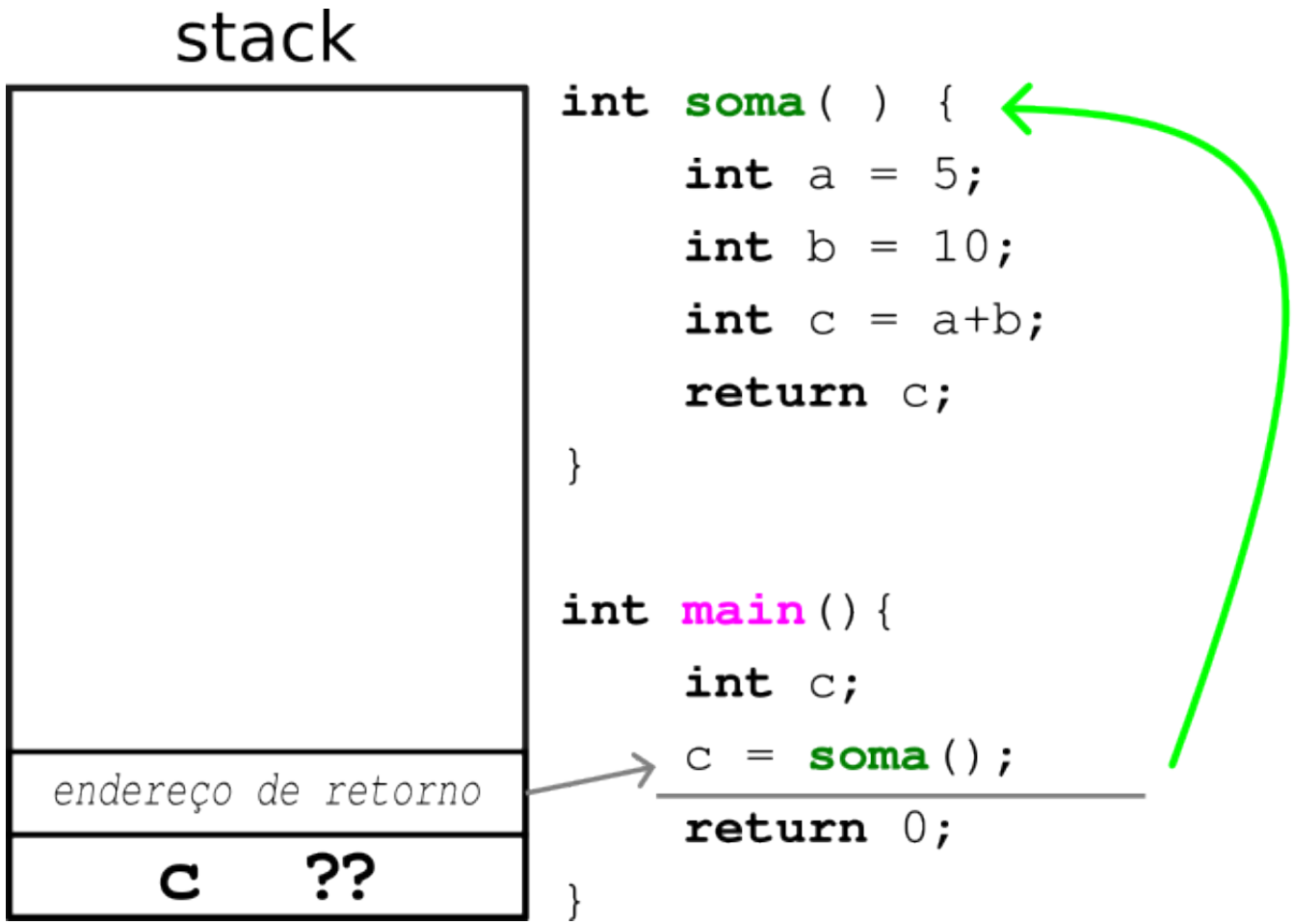
- Quando a função é chamada, o fluxo de execução do programa é desviado
- A função chamada inicia seu fluxo de execução
- Quando a função invocada termina, o fluxo volta para a função invocadora




```
int soma( ) {
    int a = 5;
    int b = 10;
    int c = a+b;
    return c;
}

int main(){
    int c;
    c = soma();
    return 0;
}
```

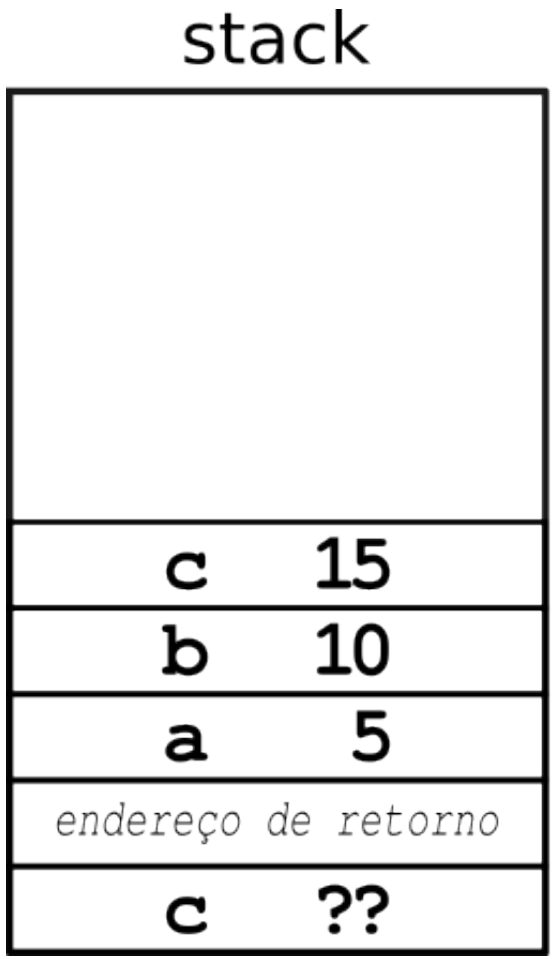
Invocação/Chamada da função

- Quando a função é chamada, o fluxo de execução do programa é desviado
- A função chamada inicia seu fluxo de execução
- Quando a função invocada termina, o fluxo volta para a função invocadora



Invocação/Chamada da função

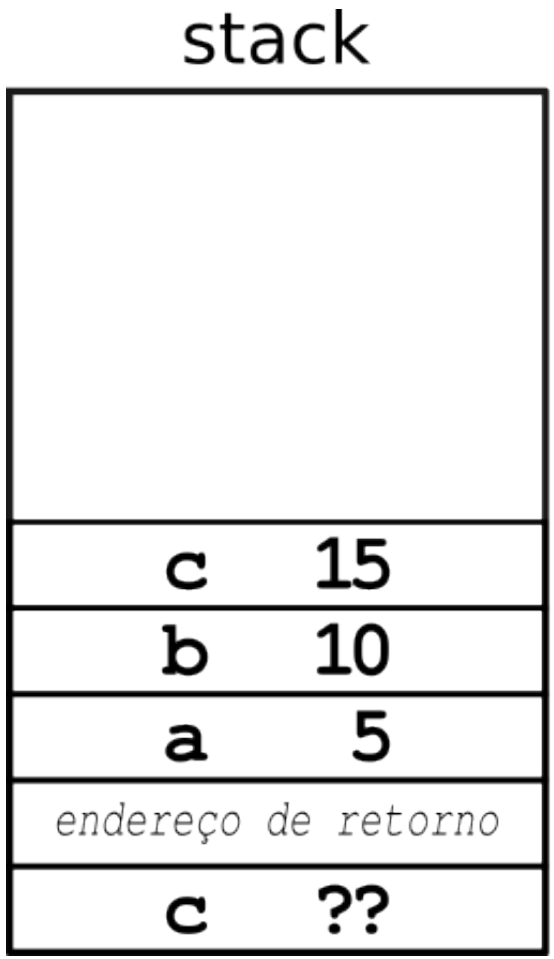
- Quando a função é chamada, o fluxo de execução do programa é desviado
- A função chamada inicia seu fluxo de execução
- Quando a função invocada termina, o fluxo volta para a função invocadora




```
int soma( ) {
    int a = 5;
    int b = 10;
    int c = a+b;
    return c;
}

int main(){
    int c;
    c = soma();
    return 0;
}
```

Invocação/Chamada da função

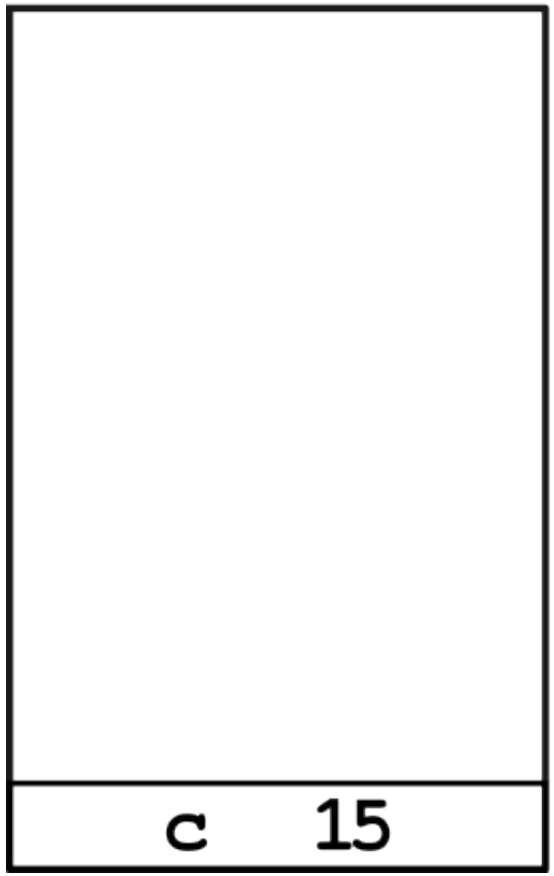
- Quando a função é chamada, o fluxo de execução do programa é desviado
- A função chamada inicia seu fluxo de execução
- Quando a função invocada termina, o fluxo volta para a função invocadora




```
int soma( ) {
    int a = 5;
    int b = 10;
    int c = a+b;
    return c;
}

int main(){
    int c;
    c = soma();
    return 0;
}
```

Invocação/Chamada da função

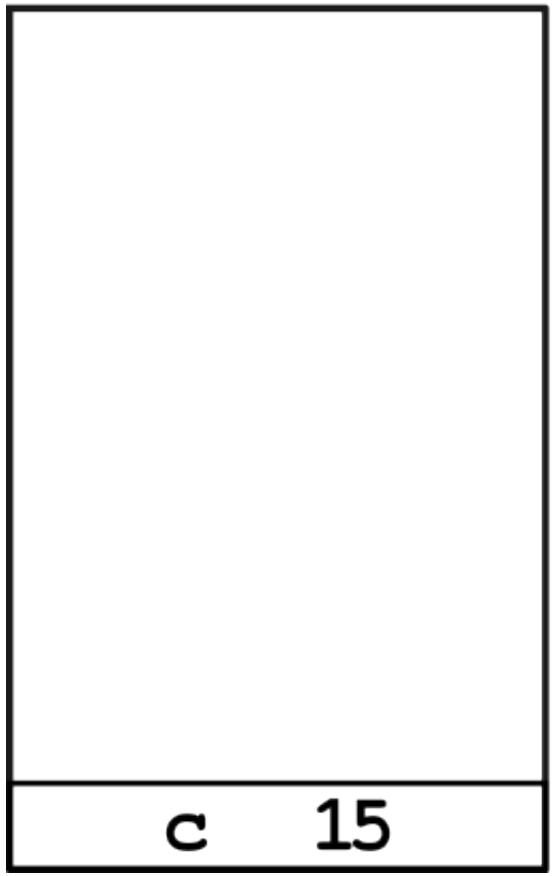
- Quando a função é chamada, o fluxo de execução do programa é desviado
- A função chamada inicia seu fluxo de execução
- Quando a função invocada termina, o fluxo volta para a função invocadora




```
int soma( ) {
    int a = 5;
    int b = 10;
    int c = a+b;
    return c;
}

int main(){
    int c;
    c = soma();
    return 0;
}
```

Invocação/Chamada da função

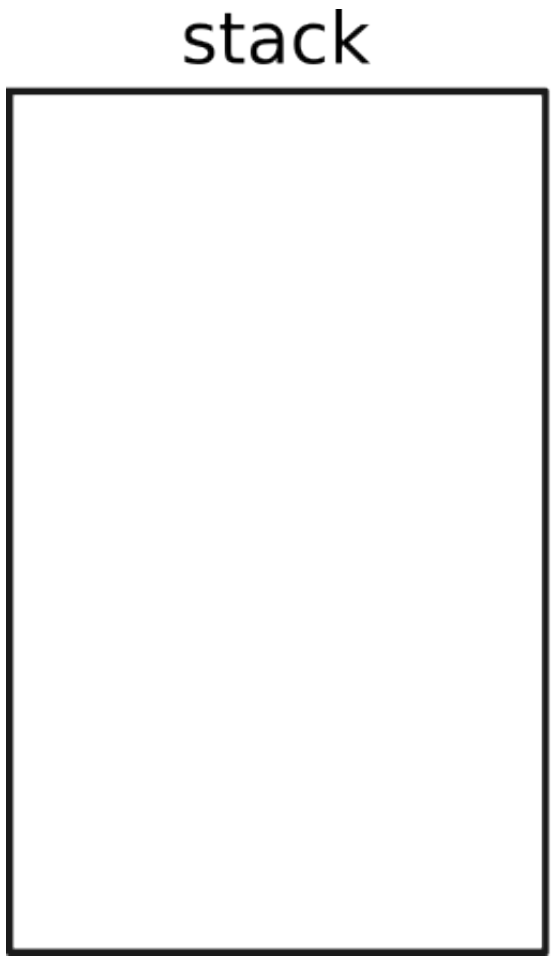
- Quando a função é chamada, o fluxo de execução do programa é desviado
- A função chamada inicia seu fluxo de execução
- Quando a função invocada termina, o fluxo volta para a função invocadora




```
int soma( ) {
    int a = 5;
    int b = 10;
    int c = a+b;
    return c;
}

int main(){
    int c;
    c = soma();
    return 0;
}
```

Invocação/Chamada da função

- Quando a função é chamada, o fluxo de execução do programa é desviado
- A função chamada inicia seu fluxo de execução
- Quando a função invocada termina, o fluxo volta para a função invocadora




```c
int soma( ) {
    int a = 5;
    int b = 10;
    int c = a+b;
    return c;
}

int main(){
    int c;
    c = soma();
    return 0;
}
```

Vamos praticar

Funções

Faça uma função que leia, repetidamente até EOF, a quantidade de pessoas que entra e sai, respectivamente, de um elevador e imprima quantas pessoas restaram.

A função não retorna valor.

Defina abaixo da main, portanto com declaração do protótipo conforme exemplo: **void** elevador();

Chame a função elevador() na main.

```
1 while ( scanf(...) != EOF )
2 {
3
4 }
```

Funções

```c
#include <stdio.h>
//declaracao do prototipo
void elevador();

//funcao principal
int main() {
    //chamada
    elevador();

    return 0;
}

//definicao da funcao elevador
void elevador()
{
    int entra, sai, e=0;
    while( scanf("%d%d", &entra, &sai) != EOF )
    {
        e = e + entra - sai;
    }
    printf("%d\n", e);
}
```

Vamos praticar

Funções

Faça uma função que leia a capacidade máxima de um elevador, seguido de leituras repetidas até EOF da quantidade de pessoas que entra e sai, respectivamente, do elevador. A função deve calcular quantas vezes o limite foi

ultrapassado e retornar esse valor.

Defina abaixo da main, portanto com declaração do protótipo conforme exemplo: **int** elevador();

Chame a função elevador() na main e imprima o valor retornado.

```
1 while ( scanf(...) != EOF )
2 {
3
4 }
5
```

Funções

```c
#include<stdio.h>
//declaracao do prototipo
int elevador();

//funcao principal
int main() {
    //chamada
    int i = elevador();
    printf("%d\n", i);

    return 0;
}

//definicao da funcao elevador
int elevador() {
    int limite, entra, sai, e=0, ultra=0;
    scanf("%d", &limite);
    while ( scanf("%d%d", &entra, &sai)!=EOF ) {
        e = e + entra - sai;
        if (e > limite)
            ultra++;
    }
    return ultra;
}
```

Vamos praticar

Funções

Faça uma função que leia a capacidade máxima de um elevador, seguido de leituras repetidas até EOF da quantidade de pessoas que entra e sai, respectivamente, do elevador.

A função deve retornar 'S' se o limite foi ultrapassado em algum momento ou 'N' caso contrário.

Defina abaixo da main, portanto com declaração do protótipo.

Funções

```c
//prototipo
char elevador();

int main() {
    //chamada
    printf("%c\n", elevador());

    return 0;
}

char elevador() {
    int limite, entra, sai, e=0;
    char u='N';

    scanf("%d", &limite);
    while ( scanf("%d%d", &entra, &sai)!=EOF ) {
        e = e + entra - sai;
        if (e > limite) {
            u='S';
        }
    }
    return u;
}
```

Roteiro

Parâmetros: passando valores

- **Passagem de argumentos**: **a função pode receber valores externos**;

```
1 tipo_retorno nome_funcao ( lista de parametros ) {
2
3 }
4
```

- **Lista de parâmetros**:
  - **Lista de tipos e nomes de variáveis** separados por vírgulas que recebem os valores (**argumentos**) quando a função é chamada;
    - ★ Pode ser vazia;
  - **Os parâmetros são variáveis locais da função**;

```
1  void imprime() {
2      printf("Ola mundo!\n");
3  }
4
5  void soma(int a, int b) {
6      int c=a+b;
7      printf("%d\n", c);
8  }
9
10 int quad(int n) {
11     return n*n;
12 }
```

Parâmetros: declarando o protótipo

```
tipo_retorno nome_funcao ( lista de parametros );
```

```
//prototipos
void imprime();           //procedimentos
void soma(int a, int b); //ou void soma(int, int);
int quad(int n);          //ou int quad(int);

int main(){
    int a=1, b=2, n=3;
    imprime();
    soma(a, b);
    quad(n);
    return 0;
}
int quad(int n) {
    return n*n;
}
void imprime() {
    printf("Ola mundo!\n");
}
void soma(int a, int b) {
    int c=a+b;
    printf("%d\n", c);
}
```

Roteiro

Passagem por valor

- Passagem somente de um valor
- Cópia do valor original
- Valor que inicializa o parâmetro (variável local) da função
- Não altera a variável original que forneceu o valor

```c
void teste (char k) {
    printf("%c\n", k); //podemos utilizar o valor de k
    k = 'a';           //podemos alterar o valor de k
    printf("%c\n", k); //a
}

int main() {
    char k = 'b';

    teste(k); //passando uma copia do valor de k
    printf("%c\n", k); //b

    return 0;
}
```

Passagem por valor

- Variável da função $\neq$ variável original;
- Portanto, variável original pode ter ou não o mesmo nome da usada na função.

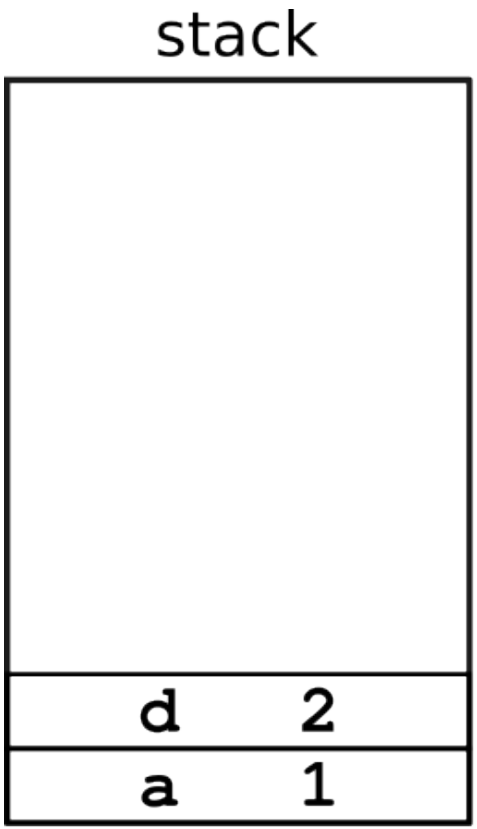
```
1  void teste (char k) {
2      printf("%c\n", k); //podemos utilizar o valor de k
3      k = 'a';           //podemos alterar o valor de k
4      printf("%c\n", k);
5  }
6
7  int main() {
8      char a = 'b';
9
10     printf("%c\n", a); //b
11     teste(a);          //passando uma copia do valor de a
12     printf("%c\n", a); //??
13
14     return 0;
15 }
```

Passagem por valor

- Variável da função $\neq$ variável original;
- Valores podem ser passados direto, sem o uso de variáveis.

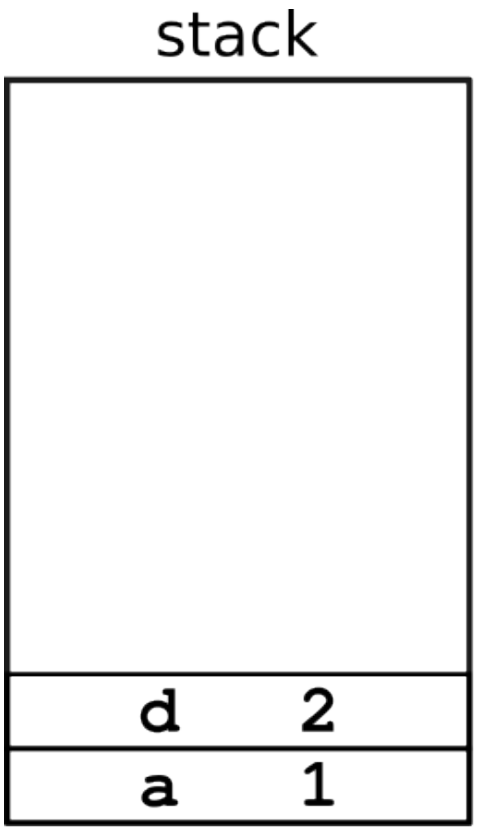
```
1  char teste (char k) {
2      printf("%c\n", k); //podemos utilizar o valor de k
3      k='a';             //podemos alterar o valor de k
4      printf("%c\n", k);
5      return k;
6  }
7
8  int main() {
9      char k = 'b', a;
10
11     a = teste(k); //passando uma copia do valor de k
12     printf("%c %c\n", a, k); //????
13
14     k = teste('c');     //passando o valor 'c'
15     printf("%c\n", k); //????
16
17     return 0;
18 }
19
```

Passagem por valor



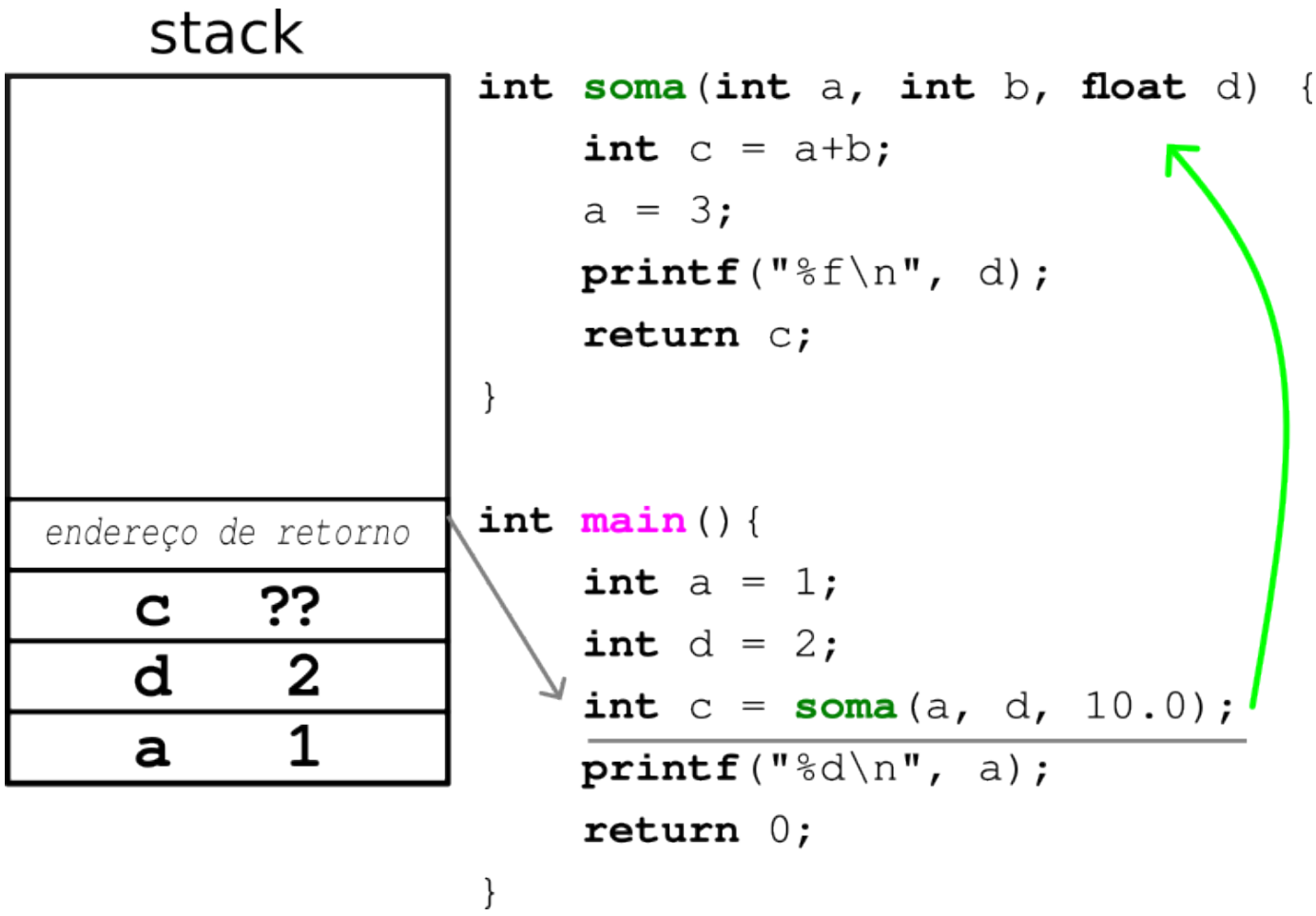

```
int soma(int a, int b, float d) {
    int c = a+b;
    a = 3;
    printf("%f\n", d);
    return c;
}

int main(){
    int a = 1;
    int d = 2;
    int c = soma(a, d, 10.0);
    printf("%d\n", a);
    return 0;
}
```

Passagem por valor

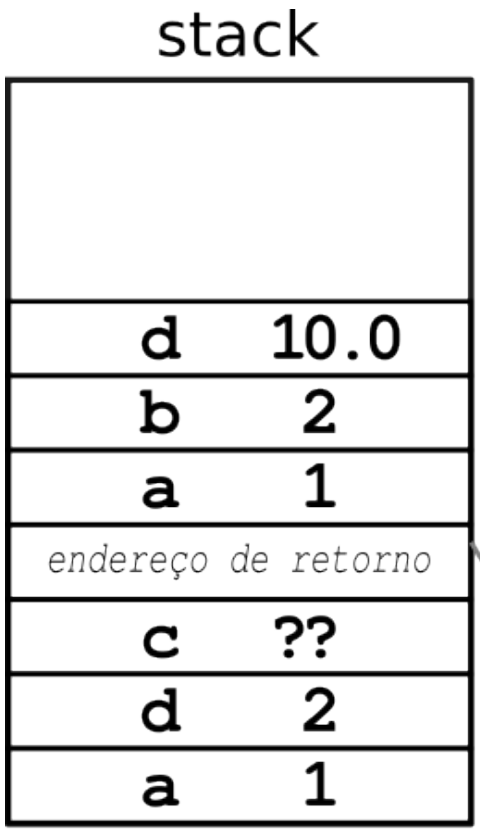



```
int soma(int a, int b, float d) {
    int c = a+b;
    a = 3;
    printf("%f\n", d);
    return c;
}

int main(){
    int a = 1;
    int d = 2;
    int c = soma(a, d, 10.0);
    printf("%d\n", a);
    return 0;
}
```

Passagem por valor

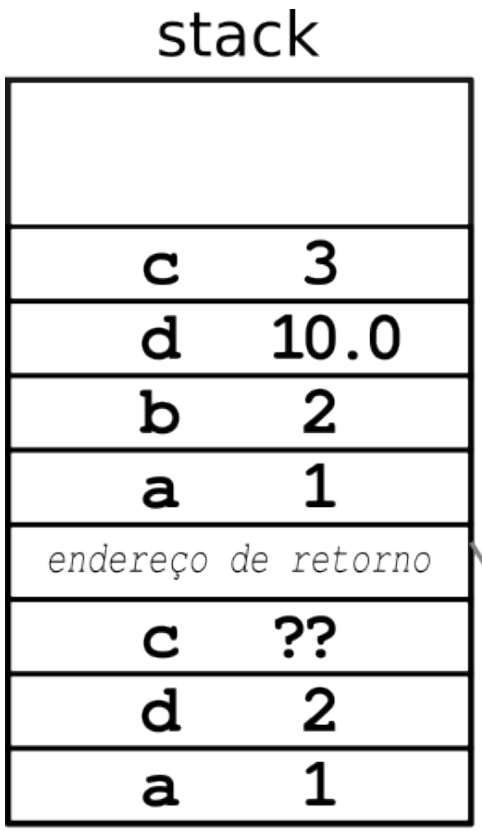


Passagem por valor



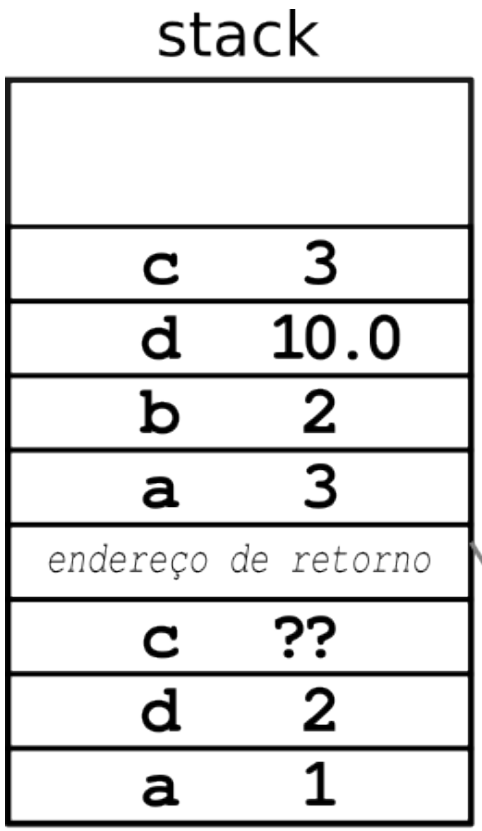

```
int soma(int a, int b, float d) {
    int c = a+b;
    a = 3;
    printf("%f\n", d);
    return c;
}

int main(){
    int a = 1;
    int d = 2;
    int c = soma(a, d, 10.0);
    printf("%d\n", a);
    return 0;
}
```

Passagem por valor



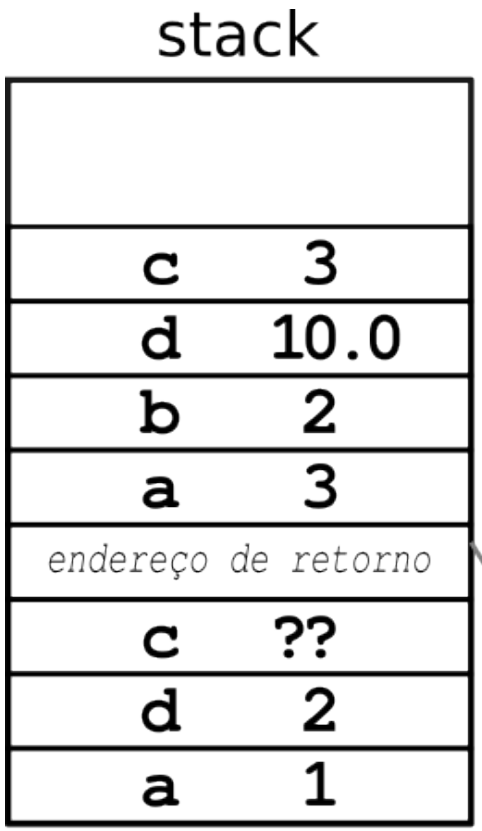

```
int soma(int a, int b, float d) {
    int c = a+b;
    a = 3;
    printf("%f\n", d);
    return c;
}

int main(){
    int a = 1;
    int d = 2;
    int c = soma(a, d, 10.0);
    printf("%d\n", a);
    return 0;
}
```

Passagem por valor



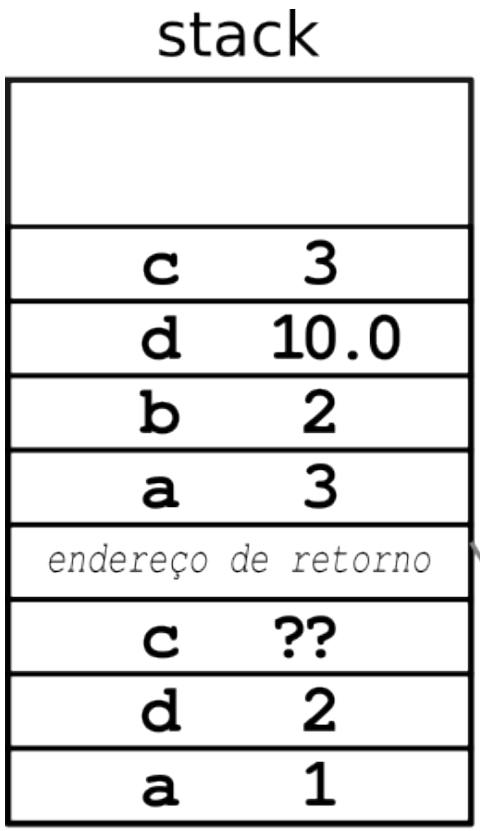

```
int soma(int a, int b, float d) {
    int c = a+b;
    a = 3;
    printf("%f\n", d);
    return c;
}

int main(){
    int a = 1;
    int d = 2;
    int c = soma(a, d, 10.0);
    printf("%d\n", a);
    return 0;
}
```

Passagem por valor



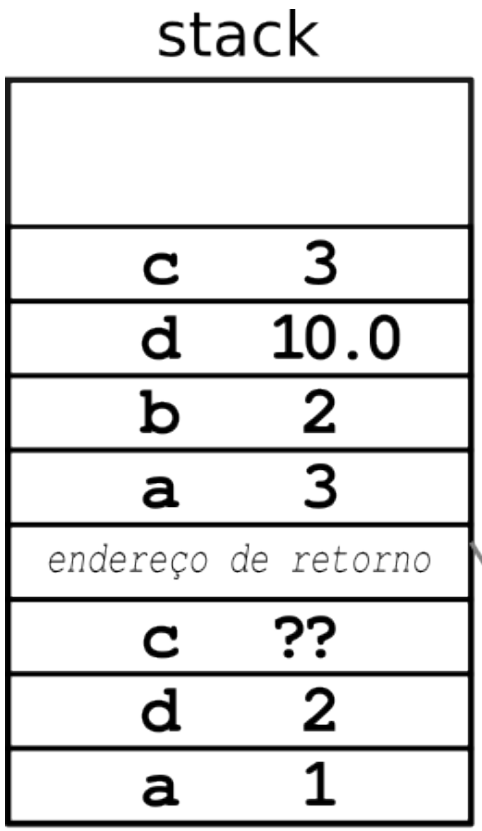

```
int soma(int a, int b, float d) {
    int c = a+b;
    a = 3;
    printf("%f\n", d);
    return c;
}

int main(){
    int a = 1;
    int d = 2;
    int c = soma(a, d, 10.0);
    printf("%d\n", a);
    return 0;
}
```

Passagem por valor



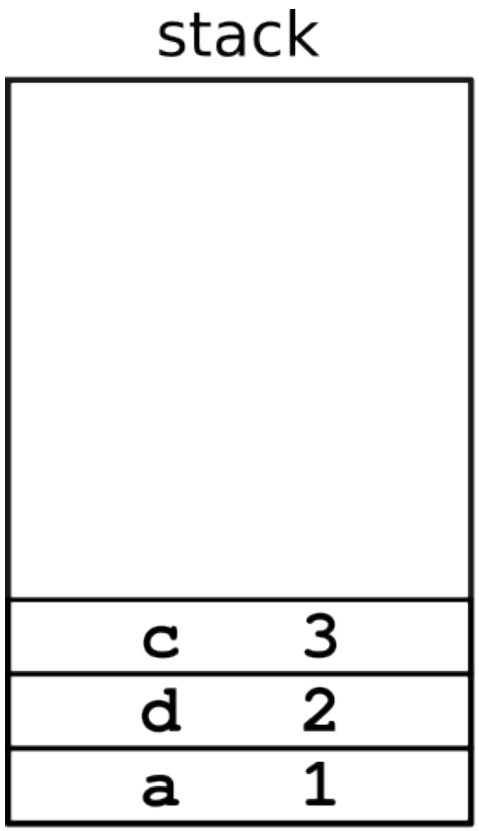

```
int soma(int a, int b, float d) {
    int c = a+b;
    a = 3;
    printf("%f\n", d);
    return c;
}

int main(){
    int a = 1;
    int d = 2;
    int c = soma(a, d, 10.0);
    printf("%d\n", a);
    return 0;
}
```

Passagem por valor



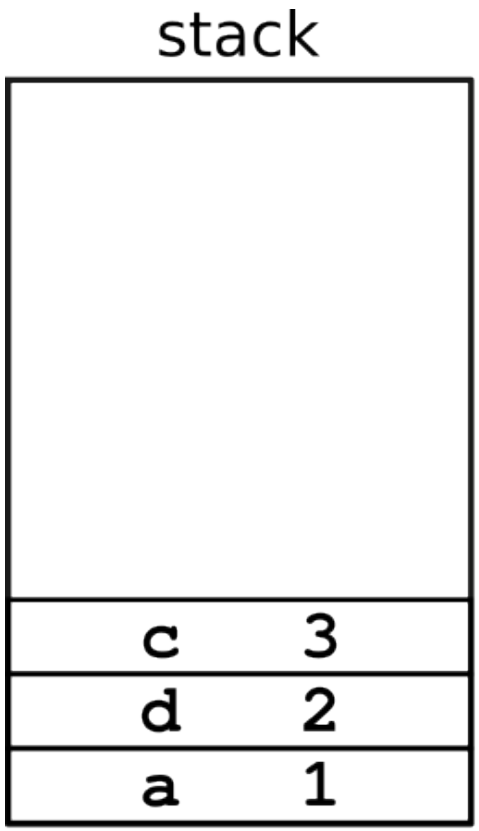

```
int soma(int a, int b, float d) {
    int c = a+b;
    a = 3;
    printf("%f\n", d);
    return c;
}

int main(){
    int a = 1;
    int d = 2;
    int c = soma(a, d, 10.0);
    printf("%d\n", a);
    return 0;
}
```

Passagem por valor



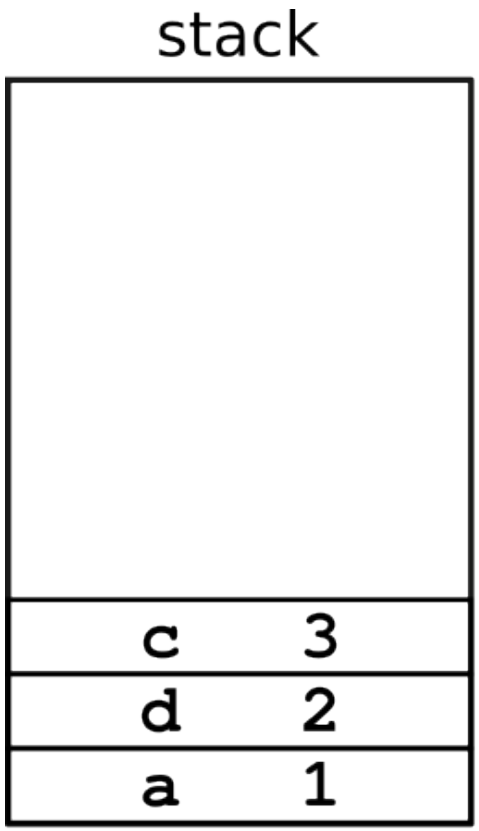

```
int soma(int a, int b, float d) {
    int c = a+b;
    a = 3;
    printf("%f\n", d);
    return c;
}

int main(){
    int a = 1;
    int d = 2;
    int c = soma(a, d, 10.0);
    printf("%d\n", a);
    return 0;
}
```

Passagem por valor



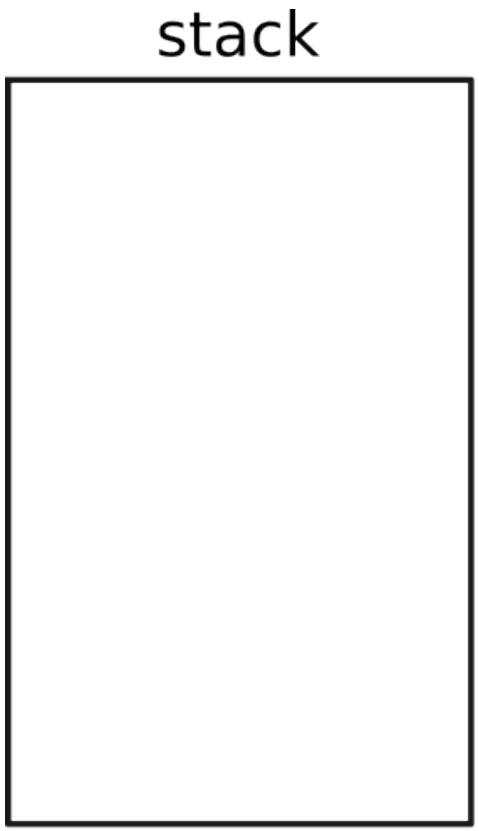

```c
int soma(int a, int b, float d) {
    int c = a+b;
    a = 3;
    printf("%f\n", d);
    return c;
}

int main(){
    int a = 1;
    int d = 2;
    int c = soma(a, d, 10.0);
    printf("%d\n", a);
    return 0;
}
```

Passagem por valor




```
int soma(int a, int b, float d) {
    int c = a+b;
    a = 3;
    printf("%f\n", d);
    return c;
}

int main(){
    int a = 1;
    int d = 2;
    int c = soma(a, d, 10.0);
    printf("%d\n", a);
    return 0;
}
```

Passagem por valor




```
int soma(int a, int b, float d) {
    int c = a+b;
    a = 3;
    printf("%f\n", d);
    return c;
}

int main(){
    int a = 1;
    int d = 2;
    int c = soma(a, d, 10.0);
    printf("%d\n", a);
    return 0;
}
```

Vamos praticar

Funções

Faça uma função que receba a quantidade de pessoas dentro, que entra e que sai de um elevador.

A função deve calcular e retornar a quantidade de pessoas restaram no elevador.

Defina abaixo da main, portanto com declaração do protótipo conforme exemplo: **int** elevador(int d, int e, int s);

Chame a função elevador() na main e imprima o valor retornado.

```c
//prototipo
int elevador(int d, int e, int s);
//ou int elevador(int, int, int);

int main() {
    int atual=10, entra=2, sai=1;
    int sobra;

    sobra = elevador(atual, entra, sai);
    printf("%d\n", sobra); //??

    sobra = elevador(50, entra, sai);
    printf("%d\n", sobra); //??

    scanf("%d%d", &entra, &sai); //1 2
    sobra = elevador(sobra, entra, sai);
    printf("%d\n", sobra); //??

    return 0;
}

int elevador(int d, int e, int s) {
    d = d + e - s;
    return d;
}
```

Roteiro

Parâmetros da função

- O que a função faz?

```
1  void funcao(char, char);
2
3  int main() {
4      char a = 'A';
5      char b = 'P';
6      funcao(a, b);
7      printf("%c %c\n", a, b);//?
8
9      return 0;
10 }
11
12 void funcao(char a, char b) {
13     char c = a;
14     a = b;
15     b = c;
16 }
17
```

	Variáveis na memória				
	da main		da funcao		
linha	a	b	a	b	c
4					

Parâmetros da função

- O que a função faz?

```
1  void funcao(char, char);
2
3  int main() {
4      char a = 'A';
5      char b = 'P';
6      funcao(a, b);
7      printf("%c %c\n", a, b);//?
8
9      return 0;
10 }
11
12 void funcao(char a, char b) {
13     char c = a;
14     a = b;
15     b = c;
16 }
17
```

	Variáveis na memória				
	da main		da funcao		
linha	a	b	a	b	c
4	'A'				

Parâmetros da função

- O que a função faz?

```
1  void funcao(char, char);
2
3  int main() {
4      char a = 'A';
5      char b = 'P';
6      funcao(a, b);
7      printf("%c %c\n", a, b);//?
8
9      return 0;
10 }
11
12 void funcao(char a, char b) {
13     char c = a;
14     a = b;
15     b = c;
16 }
17
```

	Variáveis na memória				
	da main		da funcao		
linha	a	b	a	b	c
4	'A'	X			

Parâmetros da função

- O que a função faz?

```
1  void funcao(char, char);
2
3  int main() {
4      char a = 'A';
5      char b = 'P';
6      funcao(a, b);
7      printf("%c %c\n", a, b);//?
8
9      return 0;
10 }
11
12 void funcao(char a, char b) {
13     char c = a;
14     a = b;
15     b = c;
16 }
17
```

	Variáveis na memória				
	da main		da funcao		
linha	a	b	a	b	c
4	'A'	X	X		

Parâmetros da função

- O que a função faz?

```
1  void funcao(char, char);
2
3  int main() {
4      char a = 'A';
5      char b = 'P';
6      funcao(a, b);
7      printf("%c %c\n", a, b);//?
8
9      return 0;
10 }
11
12 void funcao(char a, char b) {
13     char c = a;
14     a = b;
15     b = c;
16 }
17
```

	Variáveis na memória				
	da main		da funcao		
linha	a	b	a	b	c
4	'A'	X	X	X	

Parâmetros da função

- O que a função faz?

```
1  void funcao(char, char);
2
3  int main() {
4      char a = 'A';
5      char b = 'P';
6      funcao(a, b);
7      printf("%c %c\n", a, b);//?
8
9      return 0;
10 }
11
12 void funcao(char a, char b) {
13     char c = a;
14     a = b;
15     b = c;
16 }
17
```

	Variáveis na memória				
	da main		da funcao		
linha	a	b	a	b	c
4	'A'	X	X	X	X
5					

Parâmetros da função

- O que a função faz?

```
1  void funcao(char, char);
2
3  int main() {
4      char a = 'A';
5      char b = 'P';
6      funcao(a, b);
7      printf("%c %c\n", a, b);//?
8
9      return 0;
10 }
11
12 void funcao(char a, char b) {
13     char c = a;
14     a = b;
15     b = c;
16 }
17
```

	Variáveis na memória				
	da main		da funcao		
linha	a	b	a	b	c
4	'A'	X	X	X	X
5	'A'				

Parâmetros da função

- O que a função faz?

```
1  void funcao(char, char);
2
3  int main() {
4      char a = 'A';
5      char b = 'P';
6      funcao(a, b);
7      printf("%c %c\n", a, b);//?
8
9      return 0;
10 }
11
12 void funcao(char a, char b) {
13     char c = a;
14     a = b;
15     b = c;
16 }
17
```

	Variáveis na memória				
	da main		da funcao		
linha	a	b	a	b	c
4	'A'	X	X	X	X
5	'A'	'P'			

Parâmetros da função

- O que a função faz?

```
1  void funcao(char, char);
2
3  int main() {
4      char a = 'A';
5      char b = 'P';
6      funcao(a, b);
7      printf("%c %c\n", a, b);//?
8
9      return 0;
10 }
11
12 void funcao(char a, char b) {
13     char c = a;
14     a = b;
15     b = c;
16 }
17
```

	Variáveis na memória				
	da main		da funcao		
linha	a	b	a	b	c
4	'A'	X	X	X	X
5	'A'	'P'	X		

Parâmetros da função

- O que a função faz?

```
1  void funcao(char, char);
2
3  int main() {
4      char a = 'A';
5      char b = 'P';
6      funcao(a, b);
7      printf("%c %c\n", a, b);//?
8
9      return 0;
10 }
11
12 void funcao(char a, char b) {
13     char c = a;
14     a = b;
15     b = c;
16 }
17
```

	Variáveis na memória				
	da main		da funcao		
linha	a	b	a	b	c
4	'A'	X	X	X	X
5	'A'	'P'	X	X	

Parâmetros da função

- O que a função faz?

```
1  void funcao(char, char);
2
3  int main() {
4      char a = 'A';
5      char b = 'P';
6      funcao(a, b);
7      printf("%c %c\n", a, b);//?
8
9      return 0;
10 }
11
12 void funcao(char a, char b) {
13     char c = a;
14     a = b;
15     b = c;
16 }
17
```

	Variáveis na memória				
	da main		da funcao		
linha	a	b	a	b	c
4	'A'	X	X	X	X
5	'A'	'P'	X	X	X
12					

Parâmetros da função

- O que a função faz?

```
1  void funcao(char, char);
2
3  int main() {
4      char a = 'A';
5      char b = 'P';
6      funcao(a, b);
7      printf("%c %c\n", a, b);//?
8
9      return 0;
10 }
11
12 void funcao(char a, char b) {
13     char c = a;
14     a = b;
15     b = c;
16 }
17
```

	Variáveis na memória				
	da main		da funcao		
linha	a	b	a	b	c
4	'A'	X	X	X	X
5	'A'	'P'	X	X	X
12	'A'				

Parâmetros da função

- O que a função faz?

```
1  void funcao(char, char);
2
3  int main() {
4      char a = 'A';
5      char b = 'P';
6      funcao(a, b);
7      printf("%c %c\n", a, b);//?
8
9      return 0;
10 }
11
12 void funcao(char a, char b) {
13     char c = a;
14     a = b;
15     b = c;
16 }
17
```

	Variáveis na memória				
	da main		da funcao		
linha	a	b	a	b	c
4	'A'	X	X	X	X
5	'A'	'P'	X	X	X
12	'A'	'P'			

Parâmetros da função

- O que a função faz?

```
1  void funcao(char, char);
2
3  int main() {
4      char a = 'A';
5      char b = 'P';
6      funcao(a, b);
7      printf("%c %c\n", a, b);//?
8
9      return 0;
10 }
11
12 void funcao(char a, char b) {
13     char c = a;
14     a = b;
15     b = c;
16 }
17
```

	Variáveis na memória				
	da main		da funcao		
linha	a	b	a	b	c
4	'A'	X	X	X	X
5	'A'	'P'	X	X	X
12	'A'	'P'	'A'		

Parâmetros da função

- O que a função faz?

```
1  void funcao(char, char);
2
3  int main() {
4      char a = 'A';
5      char b = 'P';
6      funcao(a, b);
7      printf("%c %c\n", a, b);//?
8
9      return 0;
10 }
11
12 void funcao(char a, char b) {
13     char c = a;
14     a = b;
15     b = c;
16 }
17
```

	Variáveis na memória				
	da main		da funcao		
linha	a	b	a	b	c
4	'A'	X	X	X	X
5	'A'	'P'	X	X	X
12	'A'	'P'	'A'	'P'	

Parâmetros da função

- O que a função faz?

```
1  void funcao(char, char);
2
3  int main() {
4      char a = 'A';
5      char b = 'P';
6      funcao(a, b);
7      printf("%c %c\n", a, b);//?
8
9      return 0;
10 }
11
12 void funcao(char a, char b) {
13     char c = a;
14     a = b;
15     b = c;
16 }
17
```

	Variáveis na memória				
	da main		da funcao		
linha	a	b	a	b	c
4	'A'	X	X	X	X
5	'A'	'P'	X	X	X
12	'A'	'P'	'A'	'P'	X
13					

Parâmetros da função

- O que a função faz?

```
1  void funcao(char, char);
2
3  int main() {
4      char a = 'A';
5      char b = 'P';
6      funcao(a, b);
7      printf("%c %c\n", a, b);//?
8
9      return 0;
10 }
11
12 void funcao(char a, char b) {
13     char c = a;
14     a = b;
15     b = c;
16 }
17
```

	Variáveis na memória				
	da main		da funcao		
linha	a	b	a	b	c
4	'A'	X	X	X	X
5	'A'	'P'	X	X	X
12	'A'	'P'	'A'	'P'	X
13	'A'				

Parâmetros da função

- O que a função faz?

```
1  void funcao(char, char);
2
3  int main() {
4      char a = 'A';
5      char b = 'P';
6      funcao(a, b);
7      printf("%c %c\n", a, b);//?
8
9      return 0;
10 }
11
12 void funcao(char a, char b) {
13     char c = a;
14     a = b;
15     b = c;
16 }
17
```

	Variáveis na memória				
	da main		da funcao		
linha	a	b	a	b	c
4	'A'	X	X	X	X
5	'A'	'P'	X	X	X
12	'A'	'P'	'A'	'P'	X
13	'A'	'P'			

Parâmetros da função

- O que a função faz?

```
1  void funcao(char, char);
2
3  int main() {
4      char a = 'A';
5      char b = 'P';
6      funcao(a, b);
7      printf("%c %c\n", a, b);//?
8
9      return 0;
10 }
11
12 void funcao(char a, char b) {
13     char c = a;
14     a = b;
15     b = c;
16 }
17
```

	Variáveis na memória				
	da main		da funcao		
linha	a	b	a	b	c
4	'A'	X	X	X	X
5	'A'	'P'	X	X	X
12	'A'	'P'	'A'	'P'	X
13	'A'	'P'	'A'		

Parâmetros da função

- O que a função faz?

```
1  void funcao(char, char);
2
3  int main() {
4      char a = 'A';
5      char b = 'P';
6      funcao(a, b);
7      printf("%c %c\n", a, b);//?
8
9      return 0;
10 }
11
12 void funcao(char a, char b) {
13     char c = a;
14     a = b;
15     b = c;
16 }
17
```

	Variáveis na memória				
	da main		da funcao		
linha	a	b	a	b	c
4	'A'	X	X	X	X
5	'A'	'P'	X	X	X
12	'A'	'P'	'A'	'P'	X
13	'A'	'P'	'A'	'P'	

Parâmetros da função

- O que a função faz?

```
1  void funcao(char, char);
2
3  int main() {
4      char a = 'A';
5      char b = 'P';
6      funcao(a, b);
7      printf("%c %c\n", a, b);//?
8
9      return 0;
10 }
11
12 void funcao(char a, char b) {
13     char c = a;
14     a = b;
15     b = c;
16 }
17
```

	Variáveis na memória				
	da main		da funcao		
linha	a	b	a	b	c
4	'A'	X	X	X	X
5	'A'	'P'	X	X	X
12	'A'	'P'	'A'	'P'	X
13	'A'	'P'	'A'	'P'	'A'
14					

Parâmetros da função

- O que a função faz?

```
1  void funcao(char, char);
2
3  int main() {
4      char a = 'A';
5      char b = 'P';
6      funcao(a, b);
7      printf("%c %c\n", a, b);//?
8
9      return 0;
10 }
11
12 void funcao(char a, char b) {
13     char c = a;
14     a = b;
15     b = c;
16 }
17
```

	Variáveis na memória				
	da main		da funcao		
linha	a	b	a	b	c
4	'A'	X	X	X	X
5	'A'	'P'	X	X	X
12	'A'	'P'	'A'	'P'	X
13	'A'	'P'	'A'	'P'	'A'
14	'A'				

Parâmetros da função

- O que a função faz?

```
1  void funcao(char, char);
2
3  int main() {
4      char a = 'A';
5      char b = 'P';
6      funcao(a, b);
7      printf("%c %c\n", a, b);//?
8
9      return 0;
10 }
11
12 void funcao(char a, char b) {
13     char c = a;
14     a = b;
15     b = c;
16 }
17
```

	Variáveis na memória				
	da main		da funcao		
linha	a	b	a	b	c
4	'A'	X	X	X	X
5	'A'	'P'	X	X	X
12	'A'	'P'	'A'	'P'	X
13	'A'	'P'	'A'	'P'	'A'
14	'A'	'P'			

Parâmetros da função

- O que a função faz?

```
1  void funcao(char, char);
2
3  int main() {
4      char a = 'A';
5      char b = 'P';
6      funcao(a, b);
7      printf("%c %c\n", a, b);//?
8
9      return 0;
10 }
11
12 void funcao(char a, char b) {
13     char c = a;
14     a = b;
15     b = c;
16 }
17
```

	Variáveis na memória				
	da main		da funcao		
linha	a	b	a	b	c
4	'A'	X	X	X	X
5	'A'	'P'	X	X	X
12	'A'	'P'	'A'	'P'	X
13	'A'	'P'	'A'	'P'	'A'
14	'A'	'P'	'P'		

Parâmetros da função

- O que a função faz?

```
1  void funcao(char, char);
2
3  int main() {
4      char a = 'A';
5      char b = 'P';
6      funcao(a, b);
7      printf("%c %c\n", a, b);//?
8
9      return 0;
10 }
11
12 void funcao(char a, char b) {
13     char c = a;
14     a = b;
15     b = c;
16 }
17
```

	Variáveis na memória				
	da main		da funcao		
linha	a	b	a	b	c
4	'A'	X	X	X	X
5	'A'	'P'	X	X	X
12	'A'	'P'	'A'	'P'	X
13	'A'	'P'	'A'	'P'	'A'
14	'A'	'P'	'P'	'P'	

Parâmetros da função

- O que a função faz?

```
1  void funcao(char, char);
2
3  int main() {
4      char a = 'A';
5      char b = 'P';
6      funcao(a, b);
7      printf("%c %c\n", a, b);//?
8
9      return 0;
10 }
11
12 void funcao(char a, char b) {
13     char c = a;
14     a = b;
15     b = c;
16 }
17
```

	Variáveis na memória				
	da main		da funcao		
linha	a	b	a	b	c
4	'A'	X	X	X	X
5	'A'	'P'	X	X	X
12	'A'	'P'	'A'	'P'	X
13	'A'	'P'	'A'	'P'	'A'
14	'A'	'P'	'P'	'P'	'A'
15					

Parâmetros da função

- O que a função faz?

```
1  void funcao(char, char);
2
3  int main() {
4      char a = 'A';
5      char b = 'P';
6      funcao(a, b);
7      printf("%c %c\n", a, b);//?
8
9      return 0;
10 }
11
12 void funcao(char a, char b) {
13     char c = a;
14     a = b;
15     b = c;
16 }
17
```

	Variáveis na memória				
	da main		da funcao		
linha	a	b	a	b	c
4	'A'	X	X	X	X
5	'A'	'P'	X	X	X
12	'A'	'P'	'A'	'P'	X
13	'A'	'P'	'A'	'P'	'A'
14	'A'	'P'	'P'	'P'	'A'
15	'A'				

Parâmetros da função

- O que a função faz?

```
1  void funcao(char, char);
2
3  int main() {
4      char a = 'A';
5      char b = 'P';
6      funcao(a, b);
7      printf("%c %c\n", a, b);//?
8
9      return 0;
10 }
11
12 void funcao(char a, char b) {
13     char c = a;
14     a = b;
15     b = c;
16 }
17
```

	Variáveis na memória				
	da main		da funcao		
linha	a	b	a	b	c
4	'A'	X	X	X	X
5	'A'	'P'	X	X	X
12	'A'	'P'	'A'	'P'	X
13	'A'	'P'	'A'	'P'	'A'
14	'A'	'P'	'P'	'P'	'A'
15	'A'	'P'			

Parâmetros da função

- O que a função faz?

```
1  void funcao(char, char);
2
3  int main() {
4      char a = 'A';
5      char b = 'P';
6      funcao(a, b);
7      printf("%c %c\n", a, b);//?
8
9      return 0;
10 }
11
12 void funcao(char a, char b) {
13     char c = a;
14     a = b;
15     b = c;
16 }
17
```

	Variáveis na memória				
	da main		da funcao		
linha	a	b	a	b	c
4	'A'	X	X	X	X
5	'A'	'P'	X	X	X
12	'A'	'P'	'A'	'P'	X
13	'A'	'P'	'A'	'P'	'A'
14	'A'	'P'	'P'	'P'	'A'
15	'A'	'P'	'P'		

Parâmetros da função

- O que a função faz?

```
1  void funcao(char, char);
2
3  int main() {
4      char a = 'A';
5      char b = 'P';
6      funcao(a, b);
7      printf("%c %c\n", a, b);//?
8
9      return 0;
10 }
11
12 void funcao(char a, char b) {
13     char c = a;
14     a = b;
15     b = c;
16 }
17
```

	Variáveis na memória				
	da main		da funcao		
linha	a	b	a	b	c
4	'A'	X	X	X	X
5	'A'	'P'	X	X	X
12	'A'	'P'	'A'	'P'	X
13	'A'	'P'	'A'	'P'	'A'
14	'A'	'P'	'P'	'P'	'A'
15	'A'	'P'	'P'	'A'	

Parâmetros da função

- O que a função faz?

```
1  void funcao(char, char);
2
3  int main() {
4      char a = 'A';
5      char b = 'P';
6      funcao(a, b);
7      printf("%c %c\n", a, b);//?
8
9      return 0;
10 }
11
12 void funcao(char a, char b) {
13     char c = a;
14     a = b;
15     b = c;
16 }
17
```

	Variáveis na memória				
	da main		da funcao		
linha	a	b	a	b	c
4	'A'	X	X	X	X
5	'A'	'P'	X	X	X
12	'A'	'P'	'A'	'P'	X
13	'A'	'P'	'A'	'P'	'A'
14	'A'	'P'	'P'	'P'	'A'
15	'A'	'P'	'P'	'A'	'A'
6					

Parâmetros da função

- O que a função faz?

```
1  void funcao(char, char);
2
3  int main() {
4      char a = 'A';
5      char b = 'P';
6      funcao(a, b);
7      printf("%c %c\n", a, b);//?
8
9      return 0;
10 }
11
12 void funcao(char a, char b) {
13     char c = a;
14     a = b;
15     b = c;
16 }
17
```

	Variáveis na memória				
	da main		da funcao		
linha	a	b	a	b	c
4	'A'	X	X	X	X
5	'A'	'P'	X	X	X
12	'A'	'P'	'A'	'P'	X
13	'A'	'P'	'A'	'P'	'A'
14	'A'	'P'	'P'	'P'	'A'
15	'A'	'P'	'P'	'A'	'A'
6	'A'				

Parâmetros da função

- O que a função faz?

```
1  void funcao(char, char);
2
3  int main() {
4      char a = 'A';
5      char b = 'P';
6      funcao(a, b);
7      printf("%c %c\n", a, b);//?
8
9      return 0;
10 }
11
12 void funcao(char a, char b) {
13     char c = a;
14     a = b;
15     b = c;
16 }
17
```

	Variáveis na memória				
	da main		da funcao		
linha	a	b	a	b	c
4	'A'	X	X	X	X
5	'A'	'P'	X	X	X
12	'A'	'P'	'A'	'P'	X
13	'A'	'P'	'A'	'P'	'A'
14	'A'	'P'	'P'	'P'	'A'
15	'A'	'P'	'P'	'A'	'A'
6	'A'	'P'			

Parâmetros da função

- O que a função faz?

```
1  void funcao(char, char);
2
3  int main() {
4      char a = 'A';
5      char b = 'P';
6      funcao(a, b);
7      printf("%c %c\n", a, b);//?
8
9      return 0;
10 }
11
12 void funcao(char a, char b) {
13     char c = a;
14     a = b;
15     b = c;
16 }
17
```

	Variáveis na memória				
	da main		da funcao		
linha	a	b	a	b	c
4	'A'	X	X	X	X
5	'A'	'P'	X	X	X
12	'A'	'P'	'A'	'P'	X
13	'A'	'P'	'A'	'P'	'A'
14	'A'	'P'	'P'	'P'	'A'
15	'A'	'P'	'P'	'A'	'A'
6	'A'	'P'	X		

Parâmetros da função

- O que a função faz?

```
1  void funcao(char, char);
2
3  int main() {
4      char a = 'A';
5      char b = 'P';
6      funcao(a, b);
7      printf("%c %c\n", a, b);//?
8
9      return 0;
10 }
11
12 void funcao(char a, char b) {
13     char c = a;
14     a = b;
15     b = c;
16 }
17
```

	Variáveis na memória				
	da main		da funcao		
linha	a	b	a	b	c
4	'A'	X	X	X	X
5	'A'	'P'	X	X	X
12	'A'	'P'	'A'	'P'	X
13	'A'	'P'	'A'	'P'	'A'
14	'A'	'P'	'P'	'P'	'A'
15	'A'	'P'	'P'	'A'	'A'
6	'A'	'P'	X	X	

Parâmetros da função

- O que a função faz?

```
1  void funcao(char, char);
2
3  int main() {
4      char a = 'A';
5      char b = 'P';
6      funcao(a, b);
7      printf("%c %c\n", a, b);//?
8
9      return 0;
10 }
11
12 void funcao(char a, char b) {
13     char c = a;
14     a = b;
15     b = c;
16 }
17
```

	Variáveis na memória				
	da main		da funcao		
linha	a	b	a	b	c
4	'A'	X	X	X	X
5	'A'	'P'	X	X	X
12	'A'	'P'	'A'	'P'	X
13	'A'	'P'	'A'	'P'	'A'
14	'A'	'P'	'P'	'P'	'A'
15	'A'	'P'	'P'	'A'	'A'
6	'A'	'P'	X	X	X
7					

Parâmetros da função

- O que a função faz?

```
1  void funcao(char, char);
2
3  int main() {
4      char a = 'A';
5      char b = 'P';
6      funcao(a, b);
7      printf("%c %c\n", a, b);//?
8
9      return 0;
10 }
11
12 void funcao(char a, char b) {
13     char c = a;
14     a = b;
15     b = c;
16 }
17
```

	Variáveis na memória				
	da main		da funcao		
linha	a	b	a	b	c
4	'A'	X	X	X	X
5	'A'	'P'	X	X	X
12	'A'	'P'	'A'	'P'	X
13	'A'	'P'	'A'	'P'	'A'
14	'A'	'P'	'P'	'P'	'A'
15	'A'	'P'	'P'	'A'	'A'
6	'A'	'P'	X	X	X
7	**'A'**	**'P'**	X	X	X

Passagem por referência

- E como fazer uma função que troque o valor de dois parâmetros?

Passagem por referência

- E como fazer uma função que troque o valor de dois parâmetros?
- Passando o endereço da variável original

Passagem por referência

- E como fazer uma função que troque o valor de dois parâmetros?
- Passando o endereço da variável original
- As alterações são feitas diretamente no endereço da variável original;

Passagem por referência - ponteiro

- Variáveis capazes de armazenar e manipular endereços de memória
- Indicado na declaração da variável pelo **símbolo ***
- Sintaxe: TIPO *ponteiro;
  - TIPO: indica o tipo de dados da variável que o ponteiro irá apontar
  - int, float, double, char, struct
- Tamanho dos ponteiros: fixo, depende da arquitetura
- Tipo dos ponteiros: utilizado para desreferenciar (conteúdo) e operações aritméticas
- Pode ser NULL: indica endereço inválido; valor é 0 (zero)

Passagem por referência - ponteiro

```c
int i;
int *p; //ponteiro p
p = &i; //conteúdo de p = endereço de i
        //   p aponta para i
        //   p referencia a variavel i

i = 5;

//*p = apontado por p, ou seja, i
printf("%d\n", *p); //*p é igual a i
```

	Variáveis na memória		
endereço	1000	1004	
linha	i	p	*p (apontado por p)
1			

Passagem por referência - ponteiro

```c
int i;
int *p; //ponteiro p
p = &i; //conteúdo de p = endereço de i
        //   p aponta para i
        //   p referencia a variavel i

i = 5;

//*p = apontado por p, ou seja, i
printf("%d\n", *p); //*p é igual a i
```

	Variáveis na memória		
endereço	1000	1004	
linha	i	p	*p (apontado por p)
1	??		

Passagem por referência - ponteiro

```c
int i;
int *p; //ponteiro p
p = &i; //conteúdo de p = endereço de i
        //   p aponta para i
        //   p referencia a variavel i

i = 5;

//*p = apontado por p, ou seja, i
printf("%d\n", *p); //*p é igual a i
```

	Variáveis na memória		
endereço	1000	1004	
linha	i	p	*p (apontado por p)
1	??	X	X
2			

Passagem por referência - ponteiro

```c
int i;
int *p; //ponteiro p
p = &i; //conteúdo de p = endereço de i
        //   p aponta para i
        //   p referencia a variavel i

i = 5;

//*p = apontado por p, ou seja, i
printf("%d\n", *p); //*p é igual a i
```

	Variáveis na memória		
endereço	1000	1004	
linha	i	p	*p (apontado por p)
1	??	X	X
2	??		

Passagem por referência - ponteiro

```c
int i;
int *p; //ponteiro p
p = &i; //conteúdo de p = endereço de i
        //   p aponta para i
        //   p referencia a variavel i

i = 5;

//*p = apontado por p, ou seja, i
printf("%d\n", *p); //*p é igual a i
```

	Variáveis na memória		
endereço	1000	1004	
linha	i	p	*p (apontado por p)
1	??	X	X
2	??	??	??
3			

Passagem por referência - ponteiro

```c
int i;
int *p; //ponteiro p
p = &i; //conteúdo de p = endereço de i
        //   p aponta para i
        //   p referencia a variavel i

i = 5;

//*p = apontado por p, ou seja, i
printf("%d\n", *p); //*p é igual a i
```

	Variáveis na memória		
endereço	1000	1004	
linha	i	p	*p (apontado por p)
1	??	X	X
2	??	??	??
3	??		

Passagem por referência - ponteiro

```c
int i;
int *p; //ponteiro p
p = &i; //conteúdo de p = endereço de i
        //   p aponta para i
        //   p referencia a variavel i

i = 5;

//*p = apontado por p, ou seja, i
printf("%d\n", *p); //*p é igual a i
```

	Variáveis na memória		
endereço	1000	1004	
linha	i	p	*p (apontado por p)
1	??	X	X
2	??	??	??
3	??	1000	i
7			

Passagem por referência - ponteiro

```c
int i;
int *p; //ponteiro p
p = &i; //conteúdo de p = endereço de i
        //   p aponta para i
        //   p referencia a variavel i

i = 5;

//*p = apontado por p, ou seja, i
printf("%d\n", *p); //*p é igual a i
```

	Variáveis na memória		
endereço	1000	1004	
linha	i	p	*p (apontado por p)
1	??	X	X
2	??	??	??
3	??	1000	i
7	5		

Passagem por referência - ponteiro

```c
int i;
int *p; //ponteiro p
p = &i; //conteúdo de p = endereço de i
        //   p aponta para i
        //   p referencia a variavel i

i = 5;

//*p = apontado por p, ou seja, i
printf("%d\n", *p); //*p é igual a i
```

	Variáveis na memória		
endereço	1000	1004	
linha	i	p	*p (apontado por p)
1	??	X	X
2	??	??	??
3	??	1000	i
7	5	1000	

Passagem por referência - ponteiro

```c
int i;
int *p; //ponteiro p
p = &i; //conteúdo de p = endereço de i
        //   p aponta para i
        //   p referencia a variavel i

i = 5;

//*p = apontado por p, ou seja, i
printf("%d\n", *p); //*p é igual a i
```

	Variáveis na memória		
endereço	1000	1004	
linha	i	p	*p (apontado por p)
1	??	X	X
2	??	??	??
3	??	1000	i
7	5	1000	i
10			

Passagem por referência - ponteiro

```c
int i;
int *p; //ponteiro p
p = &i; //conteúdo de p = endereço de i
        //   p aponta para i
        //   p referencia a variavel i

i = 5;

//*p = apontado por p, ou seja, i
printf("%d\n", *p); //*p é igual a i
```

	Variáveis na memória		
endereço	1000	1004	
linha	i	p	*p (apontado por p)
1	??	X	X
2	??	??	??
3	??	1000	i
7	5	1000	i
10	**5**	1000	**i**

Passagem por referência - ponteiro

```c
void troca (int *p, int *q) {
    //conteudo de t = conteudo do apontado por p
    int t = *p;

    //conteudo do apontado por p = conteudo do apontado por q
    *p = *q;

    //conteudo do apontado por q = conteudo de t
    *q = t;

}

int main() {
    int a = 3, b = 9;
    troca(&a, &b);
    printf("%d %d\n", a, b); //saida??
}
```

Passagem por referência - ponteiro

```c
void troca(int *p, int *q)
{
    int t = *p;
    *p = *q;
    *q = t;
}

int main()
{
    int a = 3, b = 9;
    troca(&a, &b);
    printf("%d %d\n", a, b);
}
```

	Variáveis na memória				
	da main		da troca		
end.	100	104	300	308	316
linha	a	b	p	q	t
10					

Passagem por referência - ponteiro

```c
void troca(int *p, int *q)
{
    int t = *p;
    *p = *q;
    *q = t;
}

int main()
{
    int a = 3, b = 9;
    troca(&a, &b);
    printf("%d %d\n", a, b);
}
```

	Variáveis na memória				
	da main		da troca		
end.	100	104	300	308	316
linha	a	b	p	q	t
10	3				

Passagem por referência - ponteiro

```c
void troca(int *p, int *q)
{
    int t = *p;
    *p = *q;
    *q = t;
}

int main()
{
    int a = 3, b = 9;
    troca(&a, &b);
    printf("%d %d\n", a, b);
}
```

	Variáveis na memória				
	da main		da troca		
end.	100	104	300	308	316
linha	a	b	p	q	t
10	3	9			

Passagem por referência - ponteiro

```c
void troca(int *p, int *q)
{
    int t = *p;
    *p = *q;
    *q = t;
}

int main()
{
    int a = 3, b = 9;
    troca(&a, &b);
    printf("%d %d\n", a, b);
}
```

	Variáveis na memória				
	da main		da troca		
end.	100	104	300	308	316
linha	a	b	p	q	t
10	3	9	X		

Passagem por referência - ponteiro

```c
void troca(int *p, int *q)
{
    int t = *p;
    *p = *q;
    *q = t;
}

int main()
{
    int a = 3, b = 9;
    troca(&a, &b);
    printf("%d %d\n", a, b);
}
```

	Variáveis na memória				
	da main		da troca		
end.	100	104	300	308	316
linha	a	b	p	q	t
10	3	9	X	X	

Passagem por referência - ponteiro

```c
void troca(int *p, int *q)
{
    int t = *p;
    *p = *q;
    *q = t;
}

int main()
{
    int a = 3, b = 9;
    troca(&a, &b);
    printf("%d %d\n", a, b);
}
```

	Variáveis na memória				
	da main		da troca		
end.	100	104	300	308	316
linha	a	b	p	q	t
10	3	9	X	X	X
11	3	9	X	X	X
1	3	9			

Passagem por referência - ponteiro

```c
void troca(int *p, int *q)
{
    int t = *p;
    *p = *q;
    *q = t;
}

int main()
{
    int a = 3, b = 9;
    troca(&a, &b);
    printf("%d %d\n", a, b);
}
```

	Variáveis na memória				
	da main		da troca		
end.	100	104	300	308	316
linha	a	b	p	q	t
10	3	9	X	X	X
11	3	9	X	X	X
1	3	9	100		

Passagem por referência - ponteiro

```c
void troca(int *p, int *q)
{
    int t = *p;
    *p = *q;
    *q = t;
}

int main()
{
    int a = 3, b = 9;
    troca(&a, &b);
    printf("%d %d\n", a, b);
}
```

	Variáveis na memória				
	da main		da troca		
end.	100	104	300	308	316
linha	a	b	p	q	t
10	3	9	X	X	X
11	3	9	X	X	X
1	3	9	100	104	

Passagem por referência - ponteiro

```c
void troca(int *p, int *q)
{
    int t = *p;
    *p = *q;
    *q = t;
}

int main()
{
    int a = 3, b = 9;
    troca(&a, &b);
    printf("%d %d\n", a, b);
}
```

	Variáveis na memória				
	da main		da troca		
end.	100	104	300	308	316
linha	a	b	p	q	t
10	3	9	X	X	X
11	3	9	X	X	X
1	3	9	100	104	X
3	3	9	100	104	

Passagem por referência - ponteiro

```c
void troca(int *p, int *q)
{
    int t = *p;
    *p = *q;
    *q = t;
}

int main()
{
    int a = 3, b = 9;
    troca(&a, &b);
    printf("%d %d\n", a, b);
}
```

	Variáveis na memória				
	da main		da troca		
end.	100	104	300	308	316
linha	a	b	p	q	t
10	3	9	X	X	X
11	3	9	X	X	X
1	3	9	100	104	X
3	3	9	100	104	3
4					

Passagem por referência - ponteiro

```c
void troca(int *p, int *q)
{
    int t = *p;
    *p = *q;
    *q = t;
}

int main()
{
    int a = 3, b = 9;
    troca(&a, &b);
    printf("%d %d\n", a, b);
}
```

	Variáveis na memória				
	da main		da troca		
end.	100	104	300	308	316
linha	a	b	p	q	t
10	3	9	X	X	X
11	3	9	X	X	X
1	3	9	100	104	X
3	3	9	100	104	3
4	9				

Passagem por referência - ponteiro

```c
void troca(int *p, int *q)
{
    int t = *p;
    *p = *q;
    *q = t;
}

int main()
{
    int a = 3, b = 9;
    troca(&a, &b);
    printf("%d %d\n", a, b);
}
```

	Variáveis na memória				
	da main		da troca		
end.	100	104	300	308	316
linha	a	b	p	q	t
10	3	9	X	X	X
11	3	9	X	X	X
1	3	9	100	104	X
3	3	9	100	104	3
4	9	9	100	104	3
5					

Passagem por referência - ponteiro

```c
void troca(int *p, int *q)
{
    int t = *p;
    *p = *q;
    *q = t;
}

int main()
{
    int a = 3, b = 9;
    troca(&a, &b);
    printf("%d %d\n", a, b);
}
```

	Variáveis na memória				
	da main		da troca		
end.	100	104	300	308	316
linha	a	b	p	q	t
10	3	9	X	X	X
11	3	9	X	X	X
1	3	9	100	104	X
3	3	9	100	104	3
4	9	9	100	104	3
5	9				

Passagem por referência - ponteiro

```c
void troca(int *p, int *q)
{
    int t = *p;
    *p = *q;
    *q = t;
}

int main()
{
    int a = 3, b = 9;
    troca(&a, &b);
    printf("%d %d\n", a, b);
}
```

	Variáveis na memória				
	da main		da troca		
end.	100	104	300	308	316
linha	a	b	p	q	t
10	3	9	X	X	X
11	3	9	X	X	X
1	3	9	100	104	X
3	3	9	100	104	3
4	9	9	100	104	3
5	9	3	100	104	3
12					

Passagem por referência - ponteiro

```c
void troca(int *p, int *q)
{
    int t = *p;
    *p = *q;
    *q = t;
}

int main()
{
    int a = 3, b = 9;
    troca(&a, &b);
    printf("%d %d\n", a, b);
}
```

	Variáveis na memória				
	da main		da troca		
end.	100	104	300	308	316
linha	a	b	p	q	t
10	3	9	X	X	X
11	3	9	X	X	X
1	3	9	100	104	X
3	3	9	100	104	3
4	9	9	100	104	3
5	9	3	100	104	3
12	**9**	**3**	X	X	X

Passagem por referência - ponteiro

```c
void troca (int *p, int *q) {
    int t = *p; //t = a da main = 3
    *p = *q;    //a da main = b
    *q = t;     //b da main = t = 3

    p = &t;     //conteúdo de p = endereço de t
    printf("%d\n", *p); //saida??
}

int main() {
    int a = 3, b = 9;
    troca(&a, &b);
    printf("%d %d\n", a, b); //saida??
}
```

Passagem por referência - ponteiro

- `float *p;` : declaração da variável ponteiro
- `p = &i;` : conteúdo de p é o endereço (referência) de i
- `*p = 6;` :
  - desreferenciando o apontado por p
  - acessando i através de p

Roteiro

Passagem por referência: array (vetor, matriz, string)

- Um **array** aponta para o **endereço da primeira posição**:
  - Índices são deslocamentos a partir da primeira posição.
  - Passar para uma função = **passar seu endereço**;
  - **Endereço original** = **Array original**.

```
1  void teste(int v[], int n) {
2      for(int i=0; i<n; i++)
3          v[i]=0;
4  }
5
6  int main() {
7      int v1[4], i;
8      v1[0] = -1;
9
10     teste(v1, 4);
11
12     for(i=0; i<4; i++)
13         printf("%d ", v1[i]);
14
15     printf("\n");
16
17     return 0;
18 }
```

- Protótipos

```c
void vetores(int [], int, char []);
void matriz(int [][4], int);
```

- Passagem de vetor e string

```c
void vetores(int v[], int tam, char s[]) {
    for(int i=0; i<tam; i++) scanf("%d", &v[i]);
    for(int t=0; s[t]!='\0'; t++) s[t] = 'a';
}
```

- Passagem de matriz:
  - Quantidade de colunas é obrigatória, pois,
  - Como a alocação de memória é sequencial, as colunas indicam quantos espacos devem ser "pulados" até a próxima linha

```c
void matriz(int a[][4], int i) {
    for(int t=0; t<i; t++)
        for(int l=0; l<i; l++)
            scanf("%d", &a[t][l]);
}
void matriz2(int i, int a[][i]) {
    for(int t=0; t<i; t++)
        for(int l=0; l<i; l++)
            scanf("%d", &a[t][l]);
}
```

Exemplo

```c
void inicializar_matriz(int n, int [], int [][n]);

int main() {
    int v[2], m[2][2]={{1,2},{3,4}}, x=2;

    inicializar_matriz(x, v, m);
    printf("%d %d %d\n", v[0], v[1], x); //??

    return 0;
}

void inicializar_matriz(int x, int a[], int b[][i]) {
    int l, c, s;

    for(l=0; l<x; l++){
        s=0;
        for(c=0; c<x; c++)
            s=s+b[l][c];
        a[l] = s;
    }
    x=10;
}
```

Vamos praticar

1. Faça uma função que receba como parâmetro um vetor, um inteiro indicando quantos elementos tem no vetor e um inteiro x. Deve retornar 1 se x está contido no vetor ou 0 caso contrário.

   ```
   int pertence(int [], int, int);
   ```

2. Faça uma função que leia uma matriz n x n passada como parâmetro da função.

   ```
   void matriz(int n, int [][n]);
   ```

3. Na função principal, leia um número N, leia e guarde em um vetor N números, chame a função "pertence" implementada e imprima 1 "pertence" se encontrar x.

4. Depois chame a função "matriz", passando o valor N e uma matriz N x N. Imprima a matriz modificada.

```c
int pertence(int [], int, int);

int main() {
    int n, z;
    scanf("%d", &n);
    int y[n];

    for(int i=0; i<n; i++)
        scanf("%d", &y[i]);

    scanf("%d", &z);
    n = pertence(y, z);
    if(n==1)
        printf("pertence\n");

    return 0;
}

int pertence(int v[], int n, int x) {
    int i, r=0;
    for(i=0; i<n && v[i]!=x; i++);

    if(i<n) r=1;
    return r;
}
```

```c
void matriz(int n, int [][n]);

int main() {
    int n;
    scanf("%d", &n);
    //...codigo anterior do pertence...

    int x[n][n], i, j;
    matriz(n,x);
    for(i=0; i<n; i++) {
        for(j=0; j<n; j++)
            printf("%d ", x[i][j]);
        printf("\n");
    }
    return 0;
}

void matriz(int n, int m[][n]) {
    int i, j;
    for(i=0; i<n; i++)
        for(j=0; j<n; j++)
            scanf("%d", &m[i][j]);
}
```

Roteiro

Passagem por valor: struct

```c
struct endereco { char rua[50]; int num; };

void imprimir_endereco(struct endereco);
struct endereco ler_endereco();

int main() {
    struct endereco e;
    e = ler_endereco();
    imprimir_endereco(e);
    return 0;
}
//retornar struct
struct endereco ler_endereco() {
    struct endereco ender;
    scanf("%s", ender.rua);
    scanf("%d", &ender.num);
    return ender;
}
//por valor = somente conteudo
void imprimir_endereco(struct endereco ender) {
    printf("%s\n", ender.rua);
    printf("%d\n", ender.num);
    ender.num = 20; //nao altera a original
}
```

Passagem por referência: struct

```c
struct endereco{ char rua[50]; int num; };

void imprimir_endereco(struct endereco *);
void ler_endereco(struct endereco *);

int main() {
    struct endereco e;
    ler_endereco(&e);
    imprimir_endereco(&e);
    return 0;
}

//por referencia = endereco original
void imprimir_endereco(struct endereco *ender) {
    //troca . por ->
    printf("%s\n", ender->rua);
    printf("%d\n", ender->num);
}

void ler_endereco(struct endereco *ender) {
    //troca . por ->
    scanf(" %[^\n]", ender->rua);
    scanf("%d", &ender->num);
}
```

Vamos praticar

1. Crie um registro do tipo aluno (campos nome e idade);
2. Faça uma função que receba um parâmetro do tipo aluno imprima seu conteúdo.
3. Faça uma função que declare e inicialize um registro do tipo aluno pela entrada padrão e devolva esse registro.
4. Na função principal, utilize as duas funções.

```
1  //criando o tipo aluno
2  struct aluno{
3      ...//campos
4  };
5
6  void imprimir_aluno (struct aluno);
7  struct aluno ler_aluno ();
8
9  //declarando uma variavel do tipo aluno
10 struct aluno a;
11
12 //acessando um campo
13 printf("%s\n", a.nome);
14 printf("%d\n", a.idade);
15
```

```c
struct aluno {
    char nome[100];
    int idade;
};

void imprimir_aluno(struct aluno);
struct aluno ler_aluno();

int main() {
    struct aluno a;
    a = ler_aluno();
    imprimir_aluno(a);
    return 0;
}

struct aluno ler_aluno() {
    struct aluno p;
    scanf("%[^\n]", p.nome);
    scanf("%d", &p.idade);
    return p;
}

void imprimir_aluno(struct aluno p) {
    printf("%s\n", p.nome);
    printf("%d\n", &p.idade);
}
```

Funções

Vantagens

- **Reduzir o trabalho** de desenvolvimento e manutenção de um programa grande;
- **Dividir** de uma tarefa complexa **em passos mais simples**;
- Aumentar a **qualidade e confiabilidade** (menor o código = maior precisão);
- Melhorar a rastreabilidade, a **depuração**;
- **Códigos específicos** reutilizáveis: reduz códigos duplicados dentro de um programa;
- **Dividir do projeto** entre os vários programadores ou fases de um projeto;
- **Esconder detalhes** de implementação de usuários:
  - Exemplos: printf(), scanf(), abs(), main();
  - printf(), scanf(): são funções de uma biblioteca padrão (do C);
  - Você não sabe como elas foram escritas, mas já viu como utilizá-las;
  - Ou seja, você sabe o nome das funções e quais informações específicas você deve fornecer a elas (valores que devem ser passados para as funções) para que a função produza os resultados esperados.

Funções

Desvantagens

- Sua chamada impõe alguma **sobrecarga computacional**;
  - Cada chamada de função: dados e endereços são armazenado em uma área da memória chamada stack (pilha)
  - Quando a função termina a execução, a área reservada para a função é desalocada.

Roteiro

1. Programação Estruturada - Função
   - Estrutura/Componente
   - Definição/Criação
   - Invocação/Chamada
   - Parâmetros: passando valores
     - Parâmetros: passagem por valor ou cópia
     - Parâmetros: passagem por referência
     - Passagem de array
     - Passagem de struct
2. Vamos praticar

Vamos praticar I

1. Faça uma função que receba como argumento um caractere e devolva se 1 se for maiuscula ou 0 se for minúscula
2. Faça uma função que receba como argumento um caractere e transforme em maiúscula
3. Faça uma função que receba como argumento um inteiro e devolva o valor absoluto (módulo)
4. Faça uma função que receba como argumento um inteiro n, leia n notas e n pesos e devolva a média ponderada
5. Faça uma função que receba como argumento uma string e devolva o tamanho
6. Faça uma função que receba como argumento duas strings e devolva 0 se forem iguais ou um número negativo se a primeira string for lexicograficamente menor que a segunda ou um número positivo se for maior
7. Faça uma função que receba como argumento um vetor e seu tamanho, e imprima seu conteúdo
8. Faça uma função que receba como argumento um vetor e seu tamanho, e devolva quantos elementos se repetem

Vamos praticar II

9. Faça uma função que receba como argumento um vetor e seu tamanho, e zere todos os elementos ímpares. Imprima na main através da função criada anteriormente
10. Faça uma função que receba como argumento um vetor e seu tamanho, e ordene crescentemente seus elementos
11. Faça uma função que receba como argumento uma matriz e seu tamanho (linha e coluna), e devolva a soma da maior coluna
12. Faça uma função que receba como argumento duas matrizes e seu tamanho (linha e coluna), e copie a primeira na segunda
13. Faça uma função que receba como argumento dois vetores e seu tamanho (ambos do mesmo tamanho), e copie o primeiro no segundo
14. Faça uma função que receba como argumento dois vetores e seu tamanho (ambos do mesmo tamanho), e retorne 'S' se os vetores forem iguais posição por posição (ex. 1 2 3 é diferente de 2 3 1 apesar de ter os mesmos elementos) e 'N', caso contrário
15. Faça uma função que receba como argumento duas strings e copie a primeira na segunda