

Funções

- Separação de etapas de um programa em módulos
- Organização do código
- Concisão de comandos
- A função *main* é obrigatória

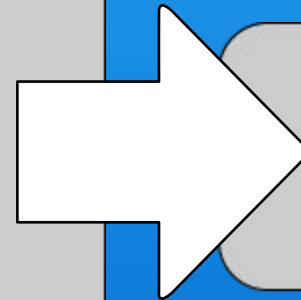
Funções

```
tipo0 NomeDaFuncao (tipo1 parametro1, tipo2  
parametro2, ..., tipoN parametroN) { instrucoes }
```

Funções

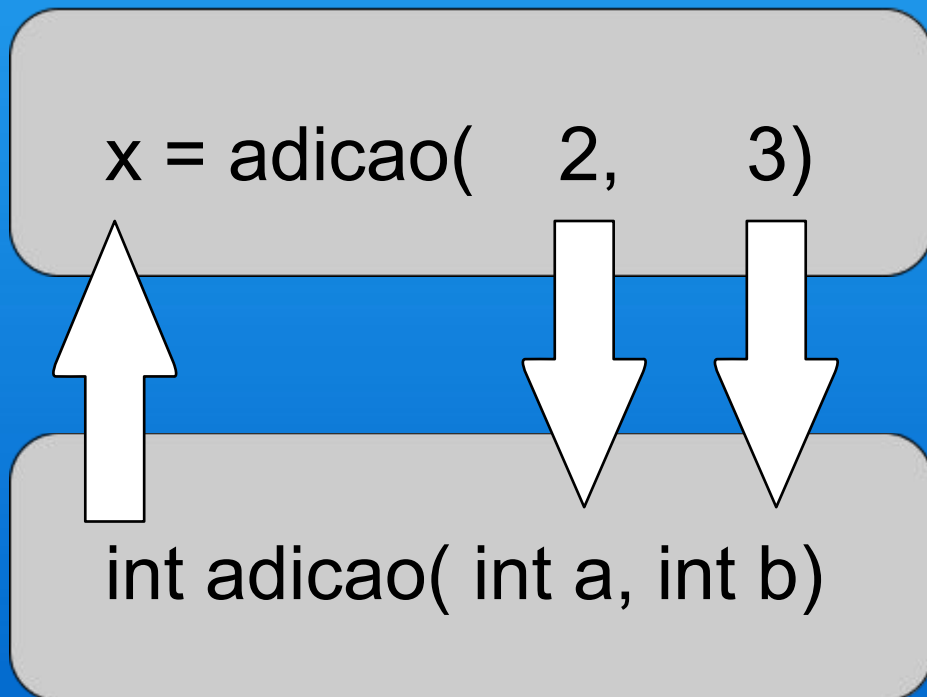
```
#include <stdio.h>
int adicao(int a, int b)
{
    int r;
    r = a+b;
    return r;
}

void main()
{
    int x, y;
    x = adicao(2,3);
    y = adicao(4,5);
    printf("x = %d, y = %d", x, y);
}
```



$x = 5, y = 9$

Funções

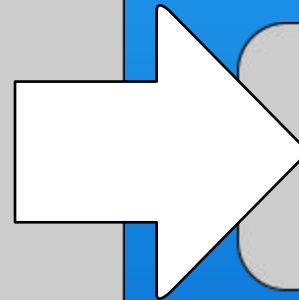


```
int adicao(int a, int b)
{
    int r;
    r = a+b;
    return r;
}
```

Funções

```
#include <stdio.h>
int adicao(int a, int b)
{
    int r;
    r = a+b;
    return r;
}

void main()
{
    int x, y;
    x = adicao(2,3);
    y = adicao(x,5);
    printf("x = %d, y = %d", x, y);
}
```



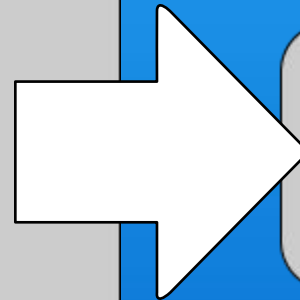
x = 5, y = 10

Funções

```
#include <stdio.h>
```

```
int adicao(int a, int b)
{
    int r;
    r = a+b;
    return r;
}
```

```
void main()
{
    int x, y;
    printf("2+3 = %d\n", adicao(2,3));
    printf("4+5 = %d\n", adicao(4,5));
    x = 10;
    y = 6 + adicao(x,9);
    printf("y = %d\n", y);
}
```



$2+3 = 5$

$4+5 = 9$

$y = 25$

Funções

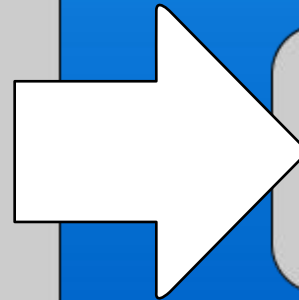
```
#include <stdio.h>

void msg( )
{
    printf("Mensagem!\n");
}

void escreve_valor(int a)
{
    printf("Valor = %d\n", a);
}

void main()
{
    int x = 10;
    msg();
    escreve_valor(0);
    escreve_valor(x);
}
```

Funções que não
recebem e nem
devolvem nenhum
parâmetro



Mensagem!
Valor = 0
Valor = 10

Funções

```
#include <stdio.h>
```

```
void msg(  
{
```

```
    printf(  
}  
}
```

```
void esc  
{
```

```
    printf(  
}  
}
```

```
void mai  
{
```

```
  
    int x = 10;  
    msg();  
    escreve_valor(0);  
    escreve_valor(x);  
}
```

**Repare que a chamada
à função exige os
parênteses. Sem eles, o
compilador entenderia uma
referência a uma variável**

msg.

Funções que não

recebem e nem
devolvem nenhum

Valor = 0

Valor = 10

Funções

- Escopo de variáveis
 - Dependendo de onde a variável for declarada, ela pode ou não ser acessada por outras funções.

Funções

```
#include <stdio.h>

int a;
void msg( )
{
    int b = a-10;
    if (b>10) b = 100;
    printf("a = %d, b = %d\n", a, b);
}

void main()
{
    int x = 10;
    a = x+5;
    msg();
    a *= 2;
    msg();
}
```

A variável *a* é global, podendo ser acessada em qualquer ponto do código após sua declaração

As variáveis *b* e *x* são locais, existindo somente nas funções em que foram declaradas

Funções

```
#include <stdio.h>
```

```
int a;
```

```
void msg( )
```

```
{
```

```
    int b = a-10;
```

```
    if (b>10) b = 100;
```

```
    printf("a = %d, b = %d\n", a, b);
```

```
}
```

```
void main()
```

```
{
```

```
    int x = 10;
```

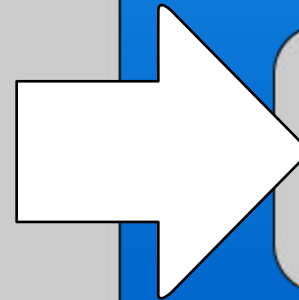
```
    a = x+5;
```

```
    msg();
```

```
    a *= 2;
```

```
    msg();
```

```
}
```



a = 15, b = 5
a = 30, b = 100

Funções

```
#include <stdio.h>
```

```
int a;
```

```
void msg( )
```

```
{
```

```
    int b = a-10;
```

```
    if (b>10) b = 100;
```

```
    printf("a = %d, b = %d\n", a, b);
```

```
}
```

```
void main()
```

```
{
```

```
    int b = 10;
```

```
    a = b+5;
```

```
    msg();
```

```
    a *= 2;
```

```
    msg();
```

```
}
```

As funções main() e msg() possuem variáveis locais com o mesmo nome. Não há conflito, porque cada uma existe dentro de uma função diferente

Funções

```
#include <stdio.h>
```

```
int a;
```

```
void msg( )
```

```
{
```

```
    int b = a-10;
```

```
    if (b>10) b = 100;
```

```
    printf("a = %d, b = %d\n", a, b);
```

```
}
```

```
void main()
```

```
{
```

```
    int b = 10;
```

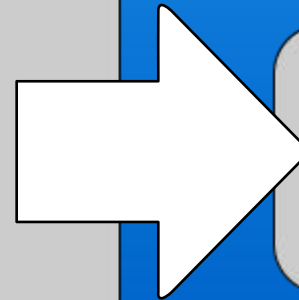
```
    a = b+5;
```

```
    msg();
```

```
    a *= 2;
```

```
    msg();
```

```
}
```



a = 15, b = 5
a = 30, b = 100

Funções

```
#include <stdio.h>
```

```
int adicao(int a, int b)
```

```
{
```

```
    int r;
```

```
    r = a+b;
```

```
    a *=100;
```

```
    return r;
```

```
}
```

```
void main()
```

```
{
```

```
    int x, y=2, z=4;
```

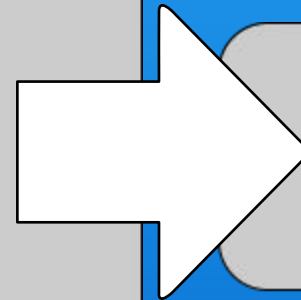
```
    x = adicao(y,z);
```

```
    printf("x = %d, y = %d, z = %d",
```

```
        x, y, z);
```

```
}
```

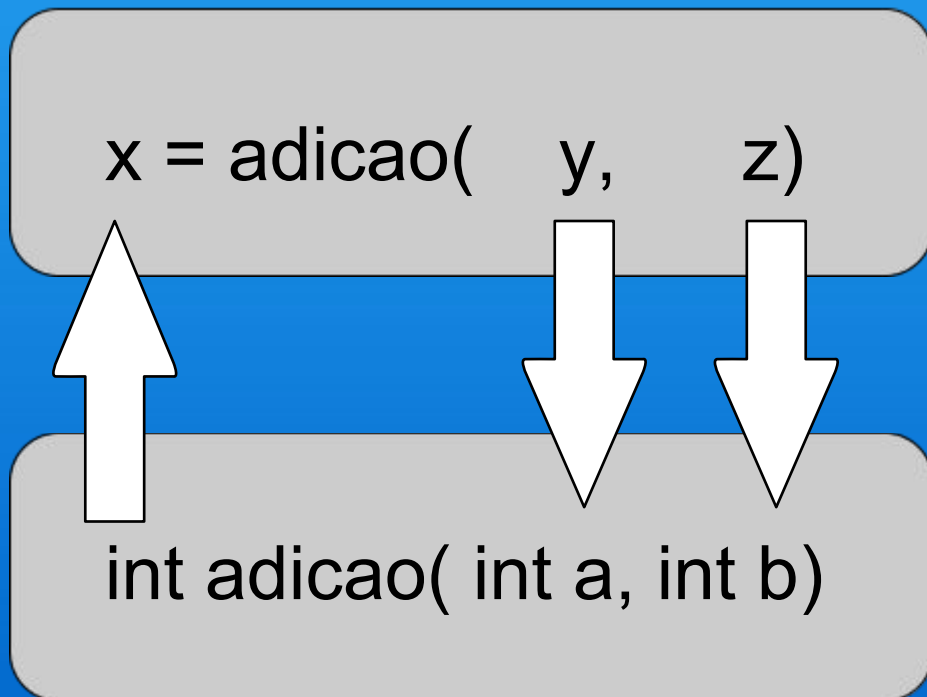
Argumentos passados
por valor ou por
referência



$x = 6, y = 2, z = 4$

Funções

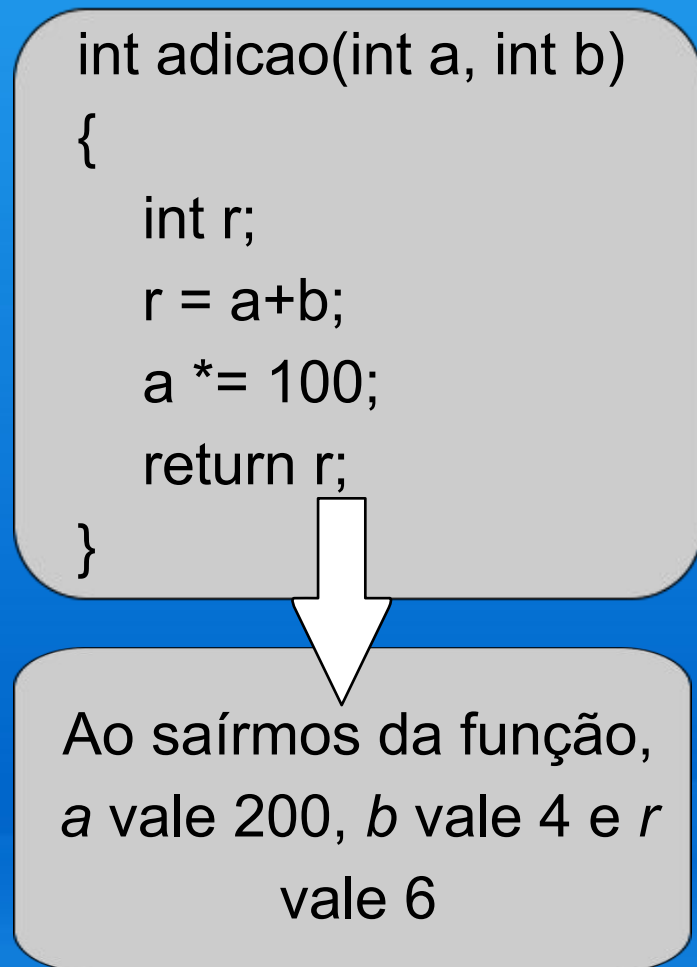
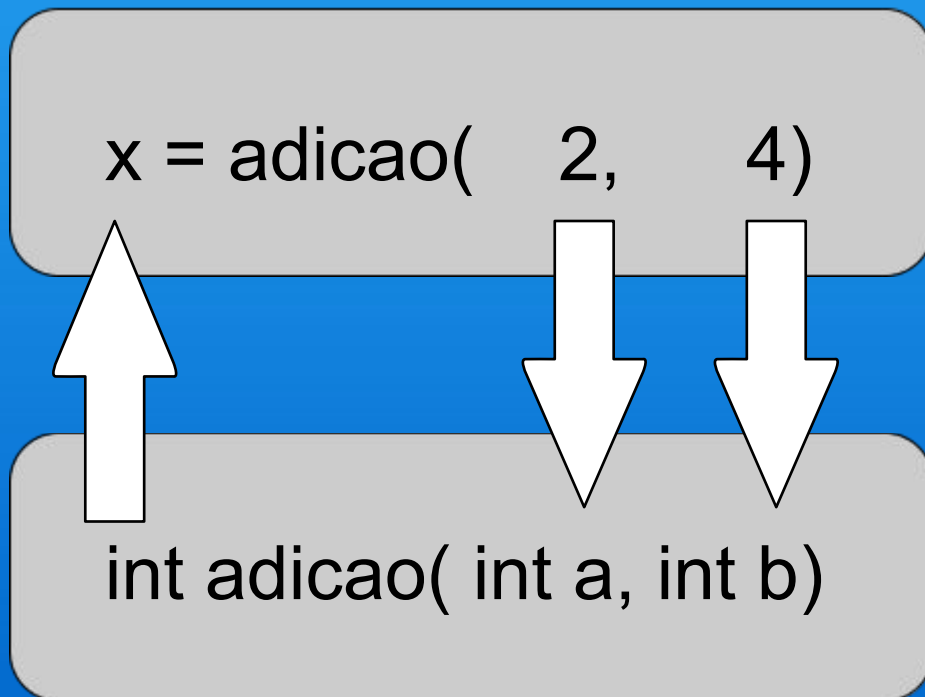
Argumentos passados por valor:



```
int adicao(int a, int b)
{
    int r;
    r = a+b;
    a *= 100;
    return r;
}
```

Funções

Argumentos passados por valor:



Funções

```
#include <stdio.h>
```

```
int adicao(int *a, int b)
```

```
{
```

```
    int r;
```

```
    r = (*a)+b;
```

```
    (*a) *=100;
```

```
    return r;
```

```
}
```

```
void main()
```

```
{
```

```
    int x, y=2, z=4;
```

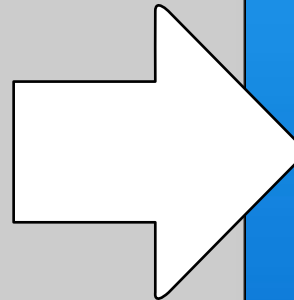
```
    x = adicao(&y,z);
```

```
    printf("x = %d, y = %d, z = %d",
```

```
        x, y, z);
```

```
}
```

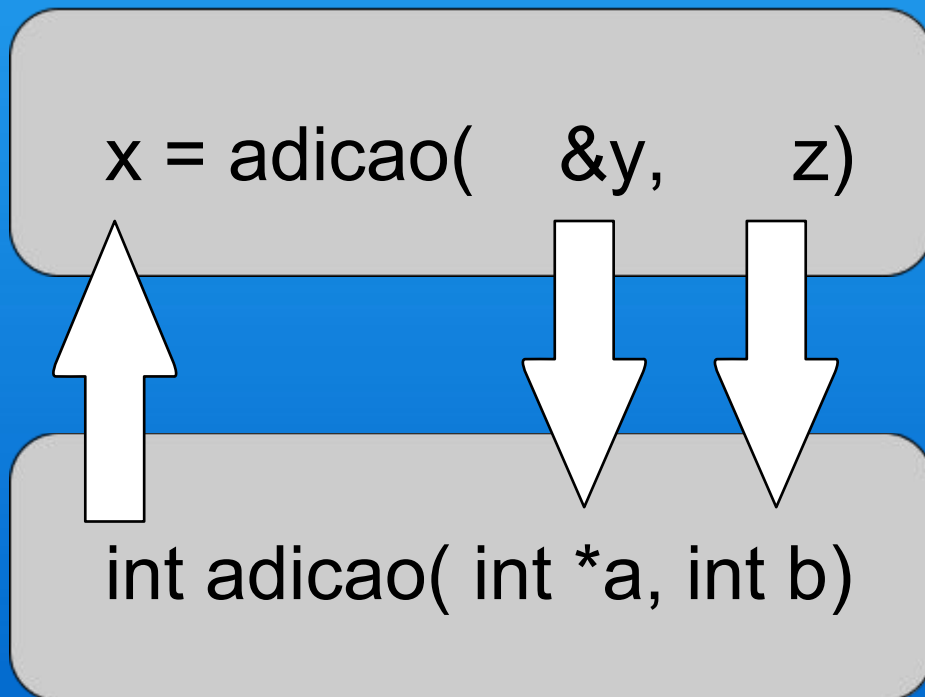
Argumentos passados
por valor ou por
referência



x = 6, y = 200, z = 4

Funções

Argumentos passados por referência:



```
int adicao(int *a, int b)
{
    int r;
    r = (*a)+b;
    (*a) *= 100;
    return r;
}
```

Funções

Valor hipotético para
o endereço da variável *y* *passada por referência:*

`x = adicao(1796, 4)`

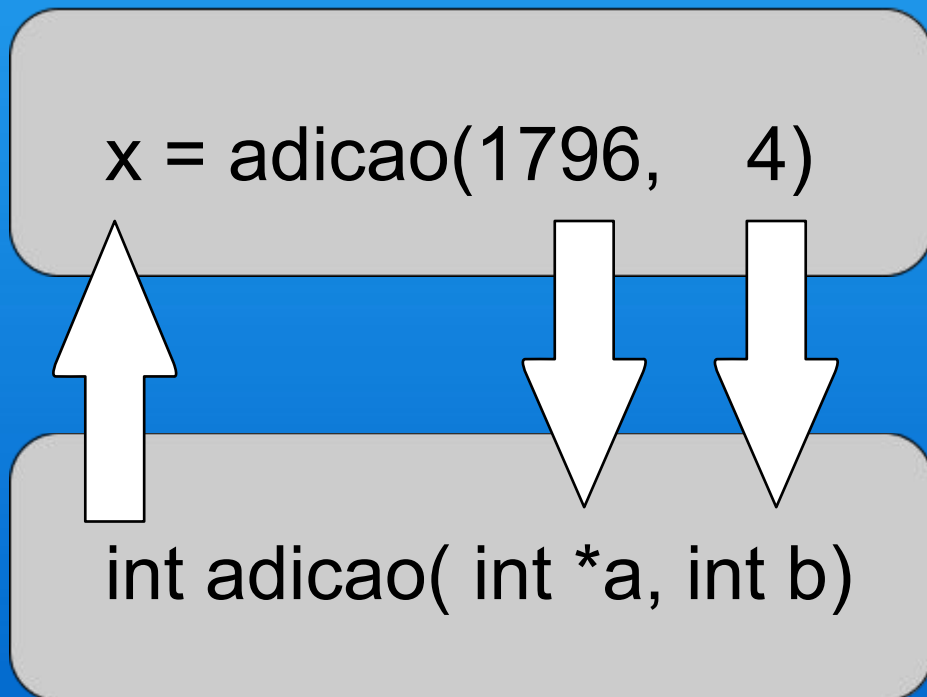
`int adicao(int *a, int b)`

```
int adicao(int *a, int b)
{
    int r;
    r = (*a)+b;
    (*a) *= 100;
    return r;
}
```

Valor **APONTADO** por *a* =
Valor guardado na posição 1796 =
Valor da variável *y*

Funções

Argumentos passados por referência:



```
int adicao(int *a, int b)
{
    int r;
    r = (*a)+b;
    (*a) *= 100;
    return r;
}
```

Ao sairmos da função, a
variável **APONTADA**
por *a* vale 200, *b* vale 4
e *r* vale 6

Funções

```
#include <stdio.h>
```

```
void duplicar(int *a)
{
    (*a) *= 2;
}
```

```
void main()
{
    int x=2;
    int *y;
    y = &x;
    printf("x = %d\n", x);
    duplicar(y);
    printf("x = %d\n", x);
    duplicar(&x);
    printf("x = %d\n", x);
}
```

Funções

```
#include <stdio.h>
```

```
void duplicar(int *a)
{
    (*a) *= 2;
}
```

```
void main()
{
    int x=2;
    int *y;
    y = &x;
    printf("x = %d\n", x);
    duplicar(y);
    printf("x = %d\n", x);
    duplicar(&x);
    printf("x = %d\n", x);
}
```



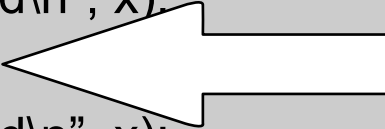
y aponta para x

Funções

```
#include <stdio.h>
```

```
void duplicar(int *a)
{
    (*a) *= 2;
}
```

```
void main()
{
    int x=2;
    int *y;
    y = &x;
    printf("x = %d\n", x);
    duplicar(y);
    printf("x = %d\n", x);
    duplicar(&x);
    printf("x = %d\n", x);
}
```



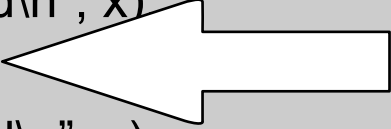
Comando para duplicar
a variável apontada por
y

Funções

```
#include <stdio.h>
```

```
void duplicar(int *a)
{
    (*a) *= 2;
}
```

```
void main()
{
    int x=2;
    int *y;
    y = &x;
    printf("x = %d\n", x);
    duplicar(y);
    printf("x = %d\n", x);
    duplicar(&x);
    printf("x = %d\n", x);
}
```



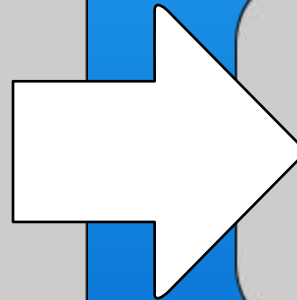
Comando para duplicar
a variável no
endereço de x =
a própria variável x

Funções

```
#include <stdio.h>
```

```
void duplicar(int *a)
{
    (*a) *= 2;
}
```

```
void main()
{
    int x=2;
    int *y;
    y = &x;
    printf("x = %d\n", x);
    duplicar(y);
    printf("x = %d\n", x);
    duplicar(&x);
    printf("x = %d\n", x);
}
```



```
x = 2
x = 4
x = 8
```

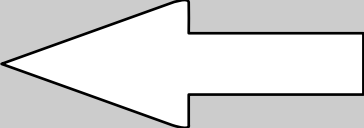
Funções

```
#include <stdio.h>
long fatorial1(long a)
{
    long cont, fatr=1;
    for(cont=1;cont<=a;cont++)
        fatr *= cont;
    return fatr;
}
long fatorial2(long a)
{
    if(a>1)
        return(a*fatorial2(a-1));
    else
        return(1);
}
void main()
{
    printf("!%d = %d\n", 10, fatorial1(10));
    printf("!%d = %d\n", 10, fatorial2(10));
}
```

Recursividade
de funções:
funções que fazem
chamadas a elas
mesmas

Funções

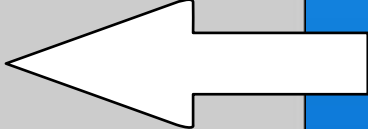
```
#include <stdio.h>
long fatorial1(long a)
{
    long cont, fatr=1;
    for(cont=1;cont<=a;cont++)
        fatr *= cont;
    return fatr;
}
long fatorial2(long a)
{
    if(a>1)
        return(a*fatorial2(a-1));
    else
        return(1);
}
void main()
{
    printf("!%d = %d\n", 10, fatorial1(10));
    printf("!%d = %d\n", 10, fatorial2(10));
}
```



Implementação do
cálculo do fatorial de
um número sem
utilizar recursividade

Funções

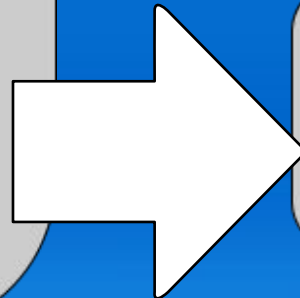
```
#include <stdio.h>
long fatorial1(long a)
{
    long cont, fatr=1;
    for(cont=1;cont<=a;cont++)
        fatr *= cont;
    return fatr;
}
long fatorial2(long a)
{
    if(a>1)
        return(a*fatorial2(a-1));
    else
        return(1);
}
void main()
{
    printf("!%d = %d\n", 10, fatorial1(10));
    printf("!%d = %d\n", 10, fatorial2(10));
}
```



Implementação
usando
recursividade

Funções

```
#include <stdio.h>
long fatorial1(long a)
{
    long cont, fatr=1;
    for(cont=1;cont<=a;cont++)
        fatr *= cont;
    return fatr;
}
long fatorial2(long a)
{
    if(a>1)
        return(a*fatorial2(a-1));
    else
        return(1);
}
void main()
{
    printf("!%d = %d\n", 10, fatorial1(10));
    printf("!%d = %d\n", 10, fatorial2(10));
}
```



!10 = 3628800

!10 = 3628800