

Gets vs fgets

| Qual a diferença?

```
void main()
{
    int i;
    char teste[10]="1234567890";
    fgets(teste,12,stdin);
    for(i=0;i<10;i++)
        printf("%c",teste[i]);
    printf("***")
}
```

```
void main()
{
    int i;
    char teste[10]="1234567890";
    gets(teste);
    for(i=0;i<10;i++)
        printf("%c",teste[i]);
    printf("***")
}
```

```
void main()
{
    int i,pos=0, casos=0, tamanhoFrase;
    char fraseErrada[102], fraseCerta[102];
    scanf("%d", &casos);
    getchar();
    while(casos>0){
        fgets(fraseErrada, 102, stdin);
        tamanhoFrase=strlen(fraseErrada);

        for (i=(tamanhoFrase/2)-1;i>=0;i--)
            fraseCerta[pos++]=fraseErrada[i];
        for (i=tamanhoFrase-2;i>=tamanhoFrase/2;i--)
            fraseCerta[pos++]=fraseErrada[i];

        fraseCerta[tamanhoFrase-1]='\0';
        puts(fraseCerta);
        casos--;
        pos=0;
    }
}
```

Scanf vs gets

```
int main(){  
    char string[100];  
    int valor;  
    scanf("%i",&valor);  
    gets(string);  
}
```

Introdução

- Trabalhamos com diversas funções em nossos programas
 - `printf()`, `scanf()`, `strlen()`, `sqrt()`, `pow()`, ...
- Estas funções fazem partes de bibliotecas padrão
 - `stdio.h`, `stdlib.h`, `string.h`, `math.h`, ...
- É indispensável que um programador saiba como escrever suas próprias funções

Exemplo de repetição de código

```
#include<stdio.h>
#include <stdlib.h>
int main( ) {
    int i;
    for (i=1;i<20;i++){
        printf("*");
    }
    printf("\n");
    printf("Numeros de 1 a 5\n");
    for (i=1;i<20;i++){
        printf("*");
    }
    printf("\n");
    for (i=1;i<=5;i++) {
        printf("%d\n",i);
    }
    for (i=1;i<20;i++) {
        printf("*");
    }
    printf("\n");
    return 0;
}
```

Execução

```
*****
Numeros entre 1 e 5
*****
1
2
3
4
5
*****
Pressione qualquer tecla pa
```

Função: básico

- Funções podem ser vistas como pequenos sub-programas
- Funções precisam ser **declaradas e escritas**
- Funções são compostas por alguns elementos básicos
 - Um **cabeçalho**, definindo nome, entradas e saídas da função
 - Entrada \Rightarrow Processamento \Rightarrow Saída
 - Um **corpo**, código que implementa a função
- Após declaradas e escritas, funções podem ser **chamadas** em outras partes do programa

Solução com funções

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

```
void escreveLinha(){
    int k;
    for (k=0;k<20;k++){
        printf("*");
    }
    printf("\n");
}
```

```
int main(){
    int i;

    escreveLinha();

    printf("Numeros entre 1 e 5\n");

    escreveLinha();

    for (i=1;i<=5;i++) {
        printf("%d\n",i);
    }

    escreveLinha();

    return 0;
}
```


Outros aspectos de funções

- Devem executar uma tarefa específica
- Um programa C pode conter diversas funções
 - Além da função principal `main()`, que é obrigatória
- Após a execução, o fluxo retorna ao ponto imediatamente após o da chamada da função

Tipos de Funções

	Com Retorno Tipadas	Sem Retorno (void)
Com parâmetros	strlen(), sin(), cos(), pow(), sqrt()	printf()
Sem parâmetros	getche()	escreveLinha()

Tipos de Funções

- **Funções void:** não retornam valor associado à função
 - Em linguagem C, *void* é um tipo que indica ausência de tipos.
 - Funções void são também chamadas de *procedimentos*
- **Funções com retorno ou tipadas:** devolvem um valor
 - Valor produzido pela execução da função
 - Tipo do valor de retorno é o tipo da função
 - Associado a execução da função, usando o comando `return;`

Função void sem parâmetros

```
void nomeDaFuncao() {  
    <declaração de variáveis locais>  
    comando1;  
    comando2;  
    ...  
}
```

Cabeçalho da Função

Corpo da Função
(entre chaves)

Elemento	Significado
void	Palavra chave indicando que a função não retornará valor.
nomeDaFuncao	Nome da função.
()	Indicando que a função não receberá argumentos.
{	Indica início do corpo_da_função.
}	Indica fim do corpo_da_função.

Ex: função void sem parâmetros

```
void nomeDaFunção() {  
    <declaração de variáveis locais>  
    com ando1;  
    com ando2;  
    ...  
}
```

```
void escreveLinha() {  
    int i;  
    for (i=1;i<20;i++){  
        printf("*");  
    }  
    printf("\n");  
}
```

Função void com parâmetros

```
void nomeDaFuncao(<lista-de-parâmetros>) {  
    <declaração de variáveis locais>  
    comando1;  
    comando2;  
    ...  
}
```

- **Lista de parâmetros** são as entradas do procedimento
 - Devem ter o tipo especificado e estar separados por vírgulas
 - Se omitidos, significa que não há argumentos (caso anterior)
- Sintaxe da lista de parâmetros:

```
<tipo> <nome-arg1>, <tipo> <nome-arg2>, ...
```



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

```
void escreveLinha (char caract, int numVezez){
    int k;
    for (k=0; k < numVezez; k++) {
        printf("%c", caract);
    }
    printf("\n");
}
```

Esse código funciona??

```
int main()
    int k;

    EscreveLinha(20, '#');

    printf("Numeros entre 1 e 5\n");

    escreveLinha('+',5);

    printf("\n");
    for (i=k; k<=5; k++) {
        printf("%d\n",k);
        escreveLinha('*',k);
    }

    escreveLinha('#', 20);

    return 0;
}
```



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

```
void escreveLinha (char caract, int numVezez){
    int k;
    for (k=0; k < numVezez; k++) {
        printf("%c", caract);
    }
    printf("\n");
}
```

```
int main(){
    int i;

    escreveLinha('#',20);

    printf("Numeros entre 1 e 5\n");

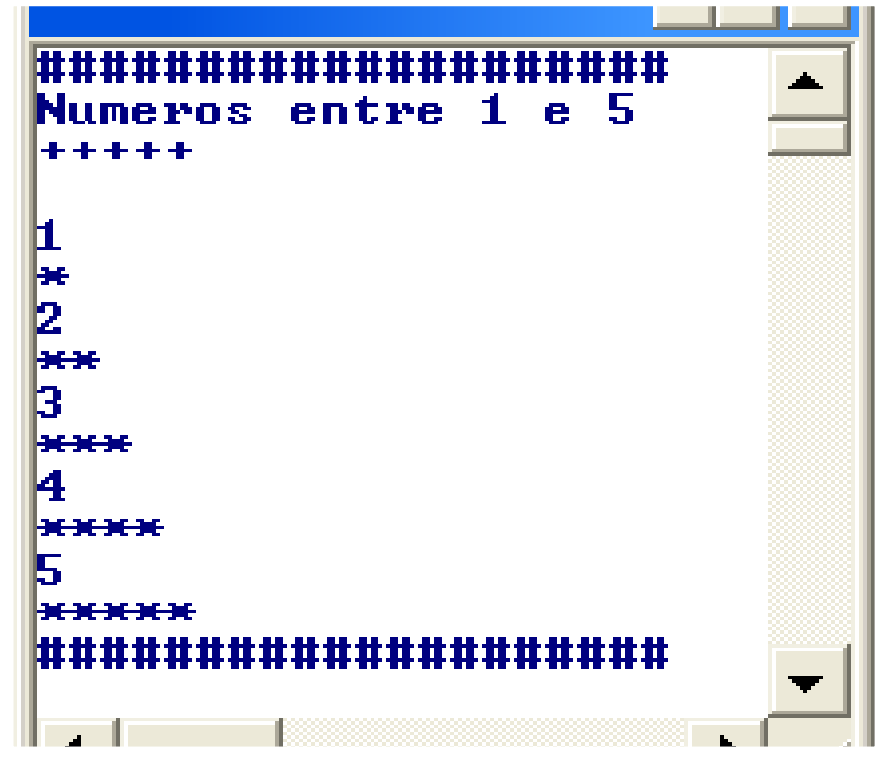
    escreveLinha('+',5);

    printf("\n");
    for (i=1; i<=5; i++) {
        printf("%d\n",i);
        escreveLinha('*',i);
    }

    escreveLinha('#', 20);

    return 0;
}
```

Execução



```
#####
Numeros entre 1 e 5
+++++

1
*
2
**
3
***
4
****
5
*****
#####
```

Função tipada

```
< tipo> nom eDaFuncao(< lista-de-parâmetros> ) {  
    < declaração de variáveis locais>  
    com ando1;  
    com ando2;  
    ...  
    return < valor> ;  
}
```

Elemento	Significado
< tipo>	Tipo do valor retornado pela função (int, char, float, ...).
nom eDaFunção	Nome da função.
(< lista-de-parâmetros>)	Um ou mais parâmetros, a serem passados à função como argumentos, quando da sua execução. Devem estar separados por vírgulas. Se omitido, significa que não há.
return < valor> ;	Comando que retorna o valor calculado pela função. <valor> pode ser tanto um valor constante ou uma variável. <i>Este valor deve ser do mesmo tipo da função.</i>

Ex: função boba-sem-sentido ^{^^}

//Função com retorno que soma reais:

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

//Soma os valores de a e b

```
float soma(float a, float b) {
    float res;
    res = a + b;
    return res;
}
```

```
int main(){
```

```
    float v1, v2, somadois;
```

```
    printf("valores a serem somados");
```

```
    scanf("%f%f",&v1,&v2);
```

```
    somadois = soma(v1, v2); // usa na atribuição
```

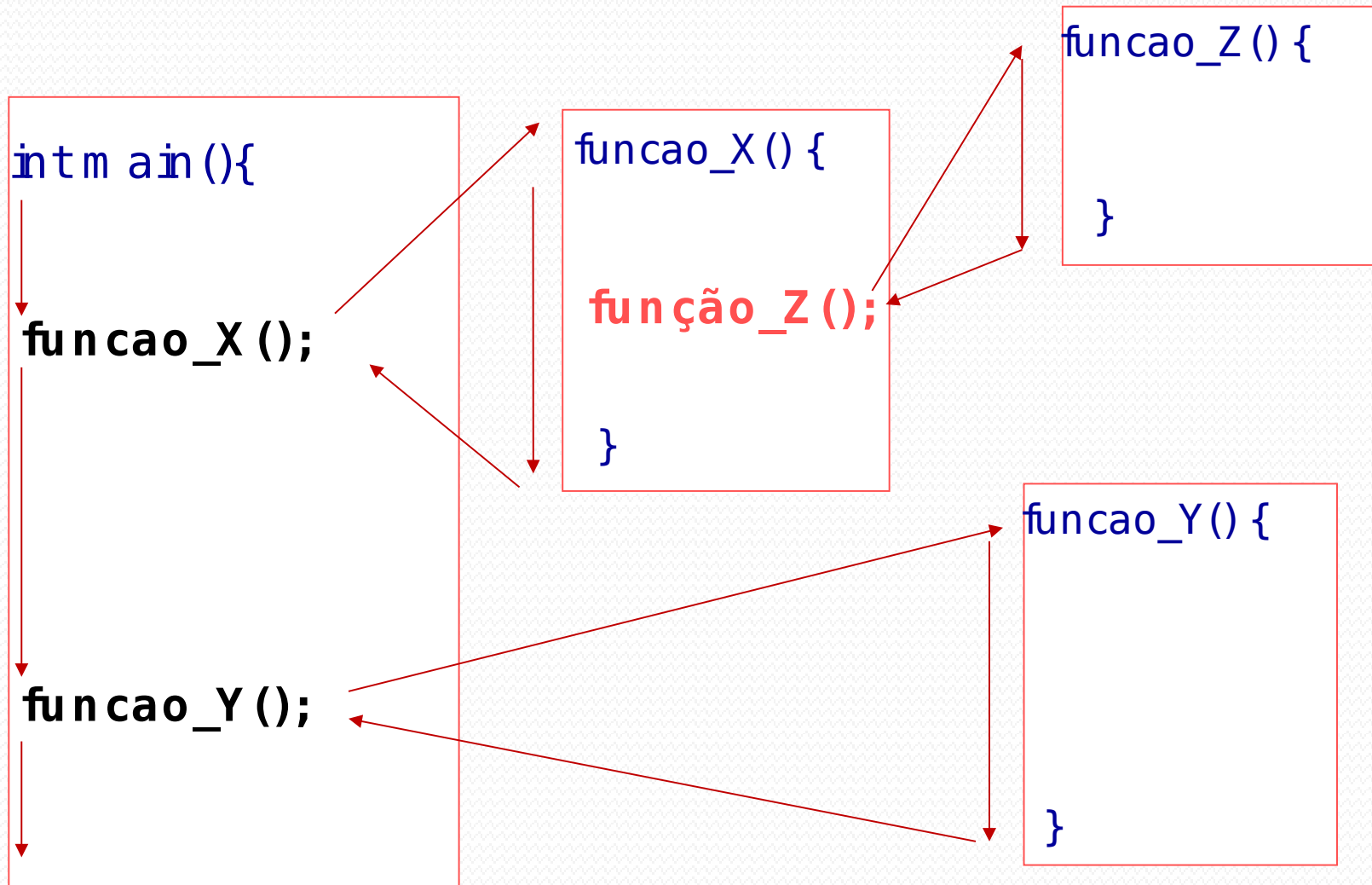
```
    printf("Soma= %f\n",somadois);
```

```
    printf("Outra vez: %f\n", soma (v1, v2)); // direto na impressão
```

```
    return 0;
```

```
}
```

Vários Níveis de Chamada



Exercício

- Faça um programa que leia os dois catetos de um triângulo retângulo e calcule a hipotenusa. A hipotenusa deve ser escrita como uma função tipada de dois parâmetros,
- Faça uma função que recebe três valores inteiros de entrada e retorne a média desses três valores.

Escopo de variáveis

- Um {bloco de comandos} define um escopo de variáveis
- Variáveis declaradas dentro de um bloco pertencem somente aquele bloco
 - Variáveis declaradas dentro de uma função f1 não podem ser referenciadas dentro de uma função f2
 - Mesmo que f1 chame f2.

Variáveis Locais e Funções

- **Variáveis locais**

- Declaradas dentro de uma função
- Só podem ser referenciadas por comandos que estão dentro do bloco (função) no qual elas foram declaradas
- “Existem” apenas enquanto o bloco de código em que foram declaradas está sendo executado
- Variáveis locais só podem ser referenciadas dentro do bloco (função) no qual elas foram declaradas
- **Variáveis de mesmo escopo devem ter sempre identificadores diferentes**
 - Contudo, diferentes variáveis de diferentes escopos podem ter o mesmo identificador

Exemplo

```
void funcao1(){
    int a = 11;
    int b = 12;
    printf("Dentro da funcao1(): a=%d; b=%d\n", a, b);
}
```

```
void funcao2(){
    int a = 21;
    int b = 22;
    printf("Dentro da funcao2(): a=%d; b=%d\n", a, b);
}
```

```
int main() {
    int a = 01;
    int b = 02;
    printf("Dentro da main: a=%d; b=%d\n", a, b);
    funcao1();
    printf("Dentro da main depois de f1: a=%d; b=%d\n", a, b);
    funcao2();
    printf("Dentro da main depois de f2: a=%d; b=%d\n", a, b);
}
```


Variáveis Globais

- Variáveis declaradas fora de qualquer bloco (função)
- Valem em todo o programa
- **Devem ser evitadas!**
- Mas eventualmente são úteis;

Exercício

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

int dobro(int x) {
    return x * 2;
}

int main() {
    int a, c;

    for (c=1; c<=3; c++) {
        printf("\n\ndigite um numero: ");
        scanf("%d", &a);
        printf("dobro do numero eh: %d",
dobro(a));
    }
    printf("\n");

    return 0;
}
```

1. Faça a simulação dos valores que vão ser impressos após a execução do programa dado.
2. Simule novamente a execução do programa dado, considerando que seja incluída a seguinte linha antes do *return* da função dobro:
int a=5;
3. Simule novamente a execução do programa dado, considerando que seja incluída a seguinte linha antes do *return* da função dobro:
x=5;

Exercício

Fazer um programa para obter três notas e imprimir a menor delas usando uma função *mínimo()*.

Fazer uma função que calcula o fatorial de um número. Construir um programa que teste a função.

Fazer uma função que receba um caractere como parâmetro e retorne verdadeiro caso o caractere seja uma vogal, e falso caso contrário.

Solução com funções

```
#include<stdio.h>
#include <stdlib.h>
void escreve_linha()
{
    int i; // variável local
    for (i=1;i<20;i++)
        printf("*");
    printf("\n");
}

int main( )
{
    int i;
    escreve_linha();
    printf("Numeros entre 1 e 5\n");
    escreve_linha();
    for (i=1;i<=5;i++)
        printf("%d\n",i);
    escreve_linha();
    system("pause");
    return 0;
}
```

Variáveis locais só existem durante a execução da função

Não existe relação entre as variáveis

São independentes

Variáveis globais existem durante a execução de todo o programa

Solução com funções

```
#include<stdio.h>
#include <stdlib.h>
void escreve_linha()
{
    int i; // variável local
    for (i=1;i<20;i++)
        printf("*");
    printf("\n");
}

int main( )
{
    int i;
    escreve_linha();
    printf("Numeros entre 1 e 5\n");
    escreve_linha();
    for (i=1;i<=5;i++)
        printf("%d\n",i);
    escreve_linha();
    system("pause");
    return 0;
}
```

Execução

```
*****
Numeros entre 1 e 5
*****
1
2
3
4
5
*****
Pressione qualquer tecla y
```