Técnicas de Programação A

Luiz Fernando Carvalho

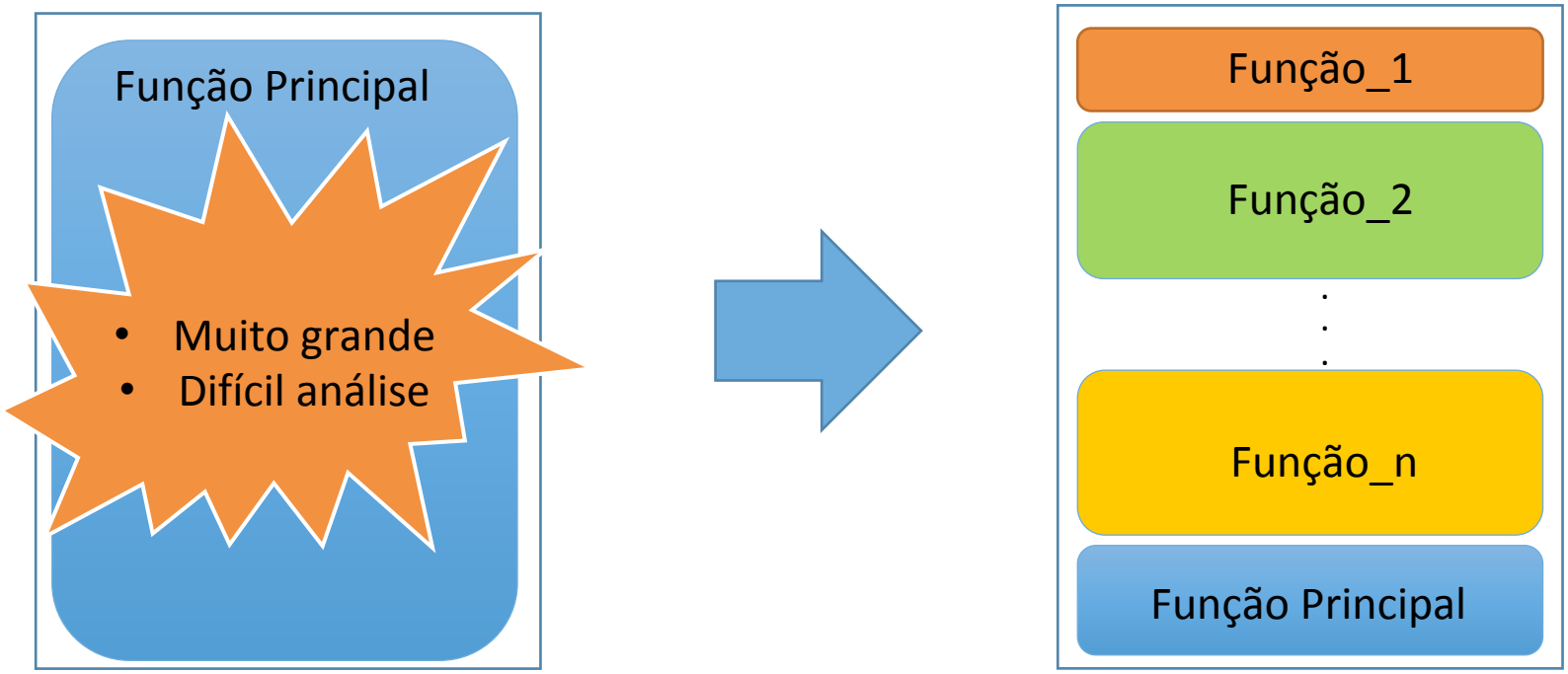
luizfcarvalhoo@gmail.com

Função

- Em C, uma **função** é um pedaço de código, dentro de um programa maior, que realiza uma certa tarefa com uma certa independência do resto do programa;
- Funções podem ser executadas várias vezes, e uma grande vantagem disso é a *reutilização de código*
  - Em vez de repetir várias vezes o código para executar certa tarefa, podemos simplesmente chamar várias vezes a função que executa essa tarefa;
- Além de economizar linhas de código, isso permite que você mude facilmente o código associado a essa tarefa;
- Ao organizarmos o código em várias funções, podemos focar cada parte do código em uma só tarefa, deixando o programa mais claro e limpo.

Função

- Blocos de construção da linguagem C
  - Divisão dos programas em partes menores;
  - Segmentar uma tarefa grande de computação em várias tarefas menores.



Todo programa possui ao menos uma função: *main*

Função

- *Em C, uma função deve ter as seguintes características:*
  - **Um *nome* pela qual ela possa ser chamada**. Os nomes possíveis seguem as mesmas restrições que os nomes de variáveis;
  - **Valores de *entrada* ou *parâmetros***. São os valores sobre os quais a função deve operar.
    - Os parâmetros das funções atuam de maneira análoga às *variáveis* das funções matemáticas;
    - Também é possível criar funções sem argumento;
  - Um valor de **saída**. Corresponde ao resultado da função
    - Também é possível criar funções que não devolvem nenhum valor de saída

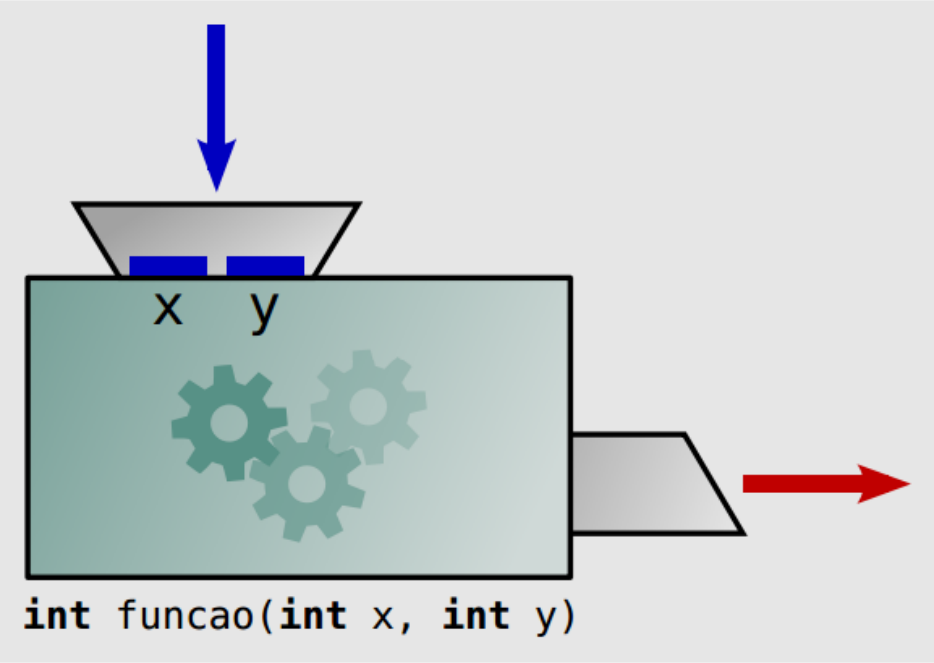
Função

- Para definir uma função, usamos a seguinte estrutura:

```
tipo_saída nome_da_função(tipo parâmetro_1, ..., tipo parâmetro_n)
{
        conteúdo da função;

}
```

- O **tipo_saída** pode ser qualquer um dos tipos usados para variáveis;
- No caso em que não há valor de saída, é possível usar no lugar do ***tipo*** a palavra ***void*** (*vazio*, em inglês)
  - Ela não é um tipo de variável; ela apenas indica a ausência de um valor;
- O conjunto dessas três definições (tipo, nome, parâmetros) é chamado de *cabeçalho/assinatura* da função.

Função



Exemplo

- Uma função que calcula a soma dos divisores de um número inteiro *n*:

```
int soma_divisores(int n)
```

- Como entrada, tem-se obviamente o número *n*, que será uma variável do tipo **`int`**;

- Como saída, teremos outro valor do tipo **`int`**, que corresponderá a soma dos divisores de *n*

Exemplo

- Uma função que recebe dois inteiros, **a** e **b**, devolve o valor da potência $\mathbf{a^b}$;

```
int potencia(int a, int b)
```

- Novamente, todos os valores envolvidos são do tipo **int**

```
int potencia(int a, b)
```

→ **Errado: O tipo de cada variável deve ser especificado**

Exemplo

- Uma função que recebe um mês e um ano e imprime o calendário desse mês;

```
void imprime_calendario(int mes, int ano)
```

- Nessa caso, não há nenhum valor de saída (os dados são enviados diretamente para a tela, com a função printf)

- Isso indica que a palavra **void** será usada no lugar do tipo da saída.

Chamada da função imprime_calendario:

```
imprime_calendario()
```


Errado: A função imprime_calendário exige 2 parâmetros do tipo int

Parâmetros da Função

- Uma vez definidos os parâmetros no cabeçalho da função, pode-se acessá-los como se fossem variáveis normais;
- Por exemplo, uma função que recebe dois inteiros e imprime sua soma na tela:

```
void imprime_soma(int a, int b)
{
    int soma;
    soma = a + b;
    printf("%d\n", soma);
}
```

```
void imprime_soma(int a, int b)
{
    printf("%d\n", a + b);
}
```

- A função não tem nenhum resultado a devolver para o programa
  - Portanto, usa-se a palavra **void** para o "tipo" de saída

O conteúdo de uma função deve ser construído como de costume dentro da função **main**

Retorno da Função

- Para devolver o valor de saída, usa-se o comando **return** seguido do valor de saída;
- O valor pode ser qualquer expressão que seja legítima de se colocar no lado direito de uma atribuição;
  - O valor de **uma** variável;
  - **Uma** constante numérica ou caractere;
  - **Uma** expressão aritmética;
  - **Um** vetor ou matriz;
  - Etc.

```
return 0;
return x;
return x * x;
return y + 1;
```

Retorna o resultado do cálculo

Não é possível ter mais de um retorno, isto é, somente um valor ou variável pode ser retornado

Retorno da Função

- A função a seguir devolve para o programa a soma dos dois números recebidos como parâmetros:

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

Retorno da Função

- É importante ressaltar que a instrução **return** também encerra a execução da função
  - O programador deve usar esse comando somente quando não houver mais nada a fazer dentro da função;
- Se o comando **return** for colocado no meio da função, ela devolverá o valor indicado e ignorará todo o resto da função.

```
float divisao(float a, int b)
{ //calcula a divisão a/b;
     if(b == 0)
        return 0;

     return a/b;
}
```

Se o valor de b for zero, a função é encerrada neste ponto

Se o **return** anterior não for executado, este encerrará a execução da função

Parâmetros da Função

- Vale também salientar que uma função **void** NÃO PODE devolver nenhum valor no **return**.
- **ATENÇÃO:** O comando **return NÃO É NECESSÁRIO** em funções com tipo de retorno **void**!
- No entanto, **PODE-SE** usar a instrução **return** (sem nenhum valor) para terminar uma função **void**.

```
void imprime_numero(int n)
{
      if(n < 0) {
         printf("Não quero imprimir números negativos!\n");
         return;
      }
      printf("%d\n", n);
}
```

Usando Funções

- Tendo em mãos o nome da função e a lista de valores que serão mandados como parâmetros de entrada, a "fórmula" para chamar uma função é simples:

```
nome_da_função(parâmetro_1, ..., parâmetro_n);
```

- Caso a função não tenha nenhum parâmetro, os parênteses devem ser escritos sem nada entre eles:

```
nome_da_função();
```

- Esses comandos são denominados de ***"chamada da função"***
  - fazem com que o computador pule para a função chamada, execute-a por inteiro e depois volte para o mesmo ponto de onde saiu.

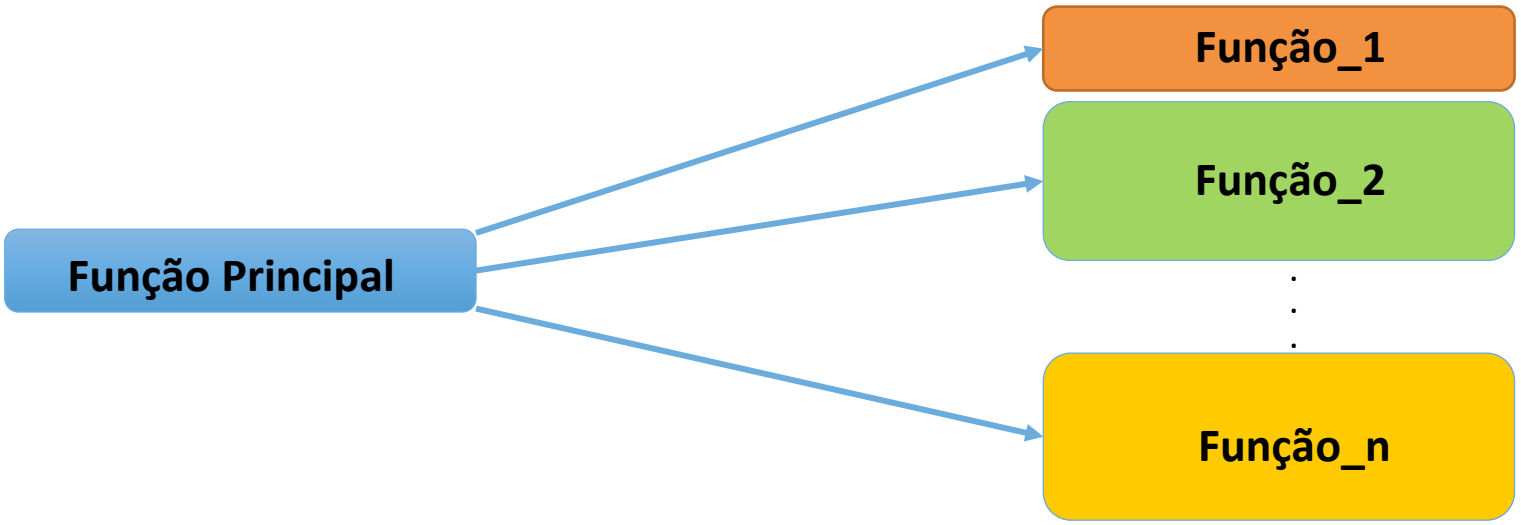
Usando Funções

- Se a função tiver um valor de saída, provavelmente o programador irá querer aproveitá-lo
  - Nesse caso, basta fazer uma variável receber o retorno da chamada da função;

```
int soma(int a, int b)
{
	return a + b;
}

int main()
{
	int resultado, resultado2, num1 = 3, num2 = 15;

	resultado = soma(5, 8);
	resultado2 = soma(num1, num2);

	return 0;
}
```

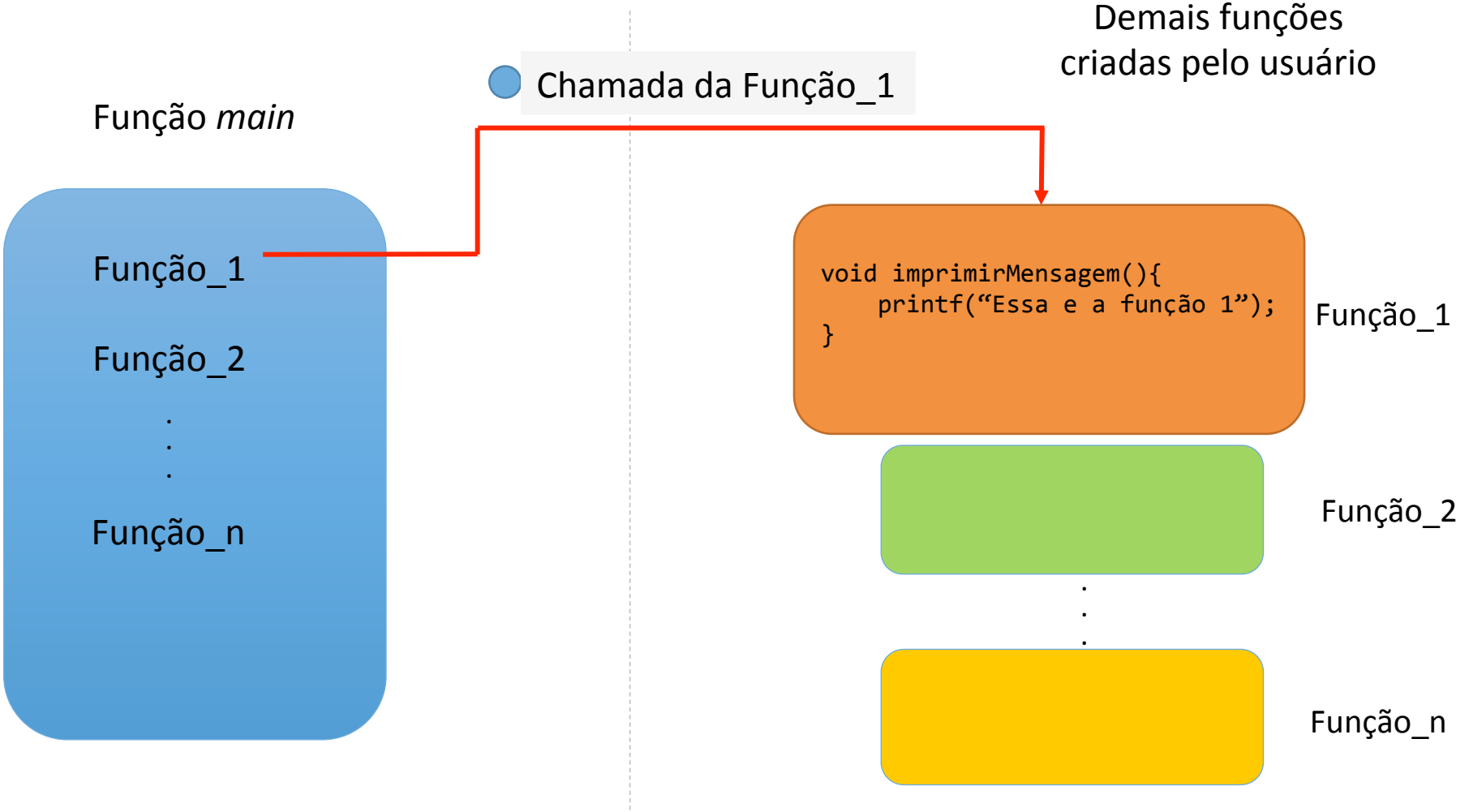
Usando Funções

- **Uma boa prática:** função *main* somente organiza as ***chamadas*** das demais funções



- Porque essa é uma boa prática?

Usando Funções



Usando Funções

Demais funções
criadas pelo usuário

Função main

Função_1

Função_2

.
.
.

Função_n

Execução dos
comandos da função

```
void imprimirMensagem(){
    printf("Essa e a função 1");
}
```

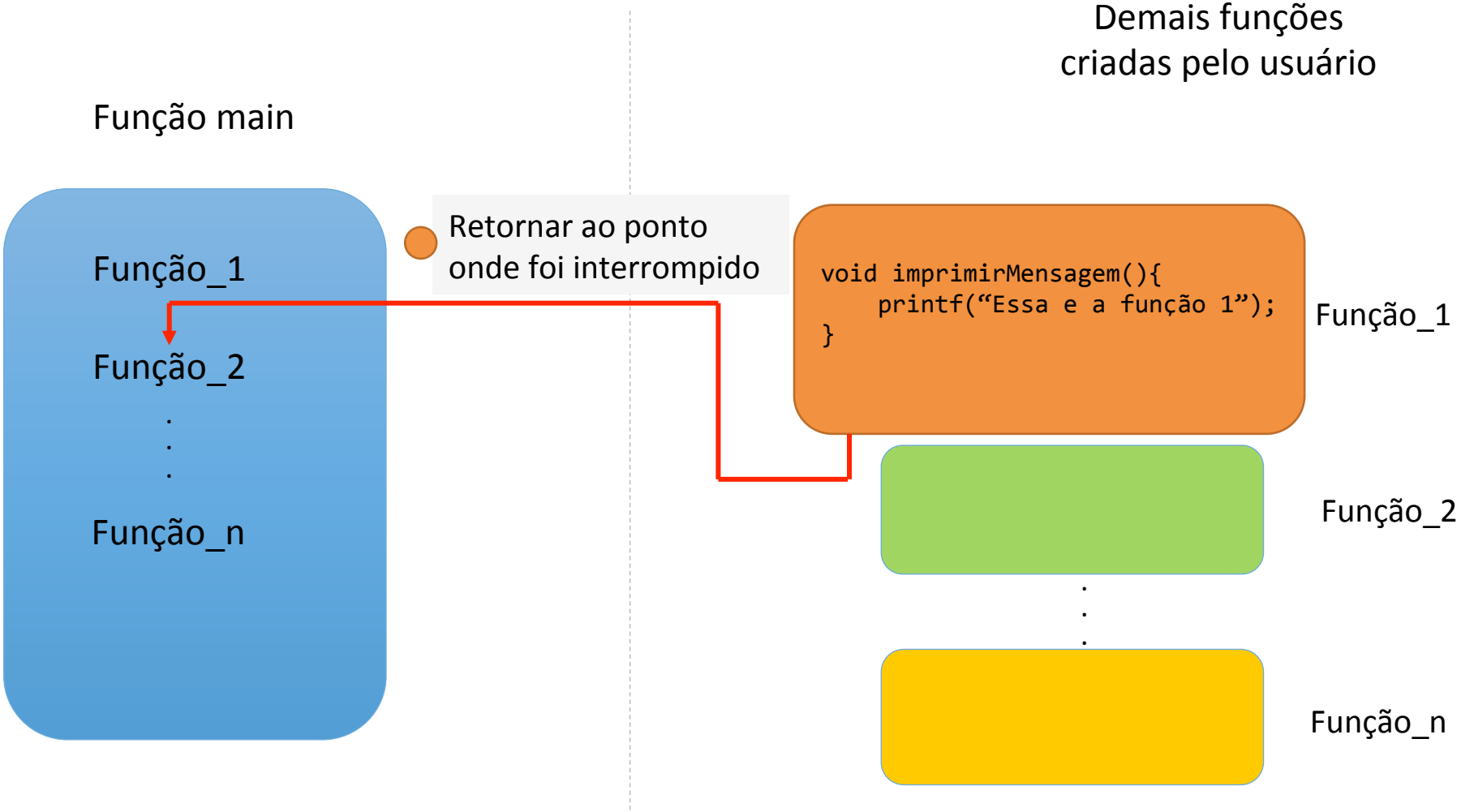
Função_1

Função_2

.
.
.

Função_n

Usando Funções



Definição de Funções

```
float divide(float dividendo, float divisor){

   return dividendo/divisor;
}
```

Definição da função

```
int main(){
   float dividendo = 10;
   float divisor = 3;
   float resultado;

   resultado = divide(dividendo, divisor);
   printf("%f", resultado);

   return 0;
}
```

Função *main*

Escopo de Variáveis

- Por **escopo de uma variável** entende-se o bloco de código onde esta variável é válida;
  - As variáveis valem no bloco que são definidas;
  - As variáveis definidas dentro de uma função recebem o nome de **variáveis locais**;
  - Os parâmetros de uma função valem também somente dentro da função **(também são variáveis locais)**;
  - Uma variável definida dentro de uma função não é acessível em outras funções, mesmo que essas variáveis tenham nomes idênticos;
  - Uma variável definida dentro de uma função “morre” quando a função termina a sua execução;

Escopo de Variáveis

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

void funcao1(variáveis locais de parâmetros)
{
   // declaração das variáveis locais da função1

}


int main()
{
  //declaração das variáveis locais da main()

   return 0;
}
```

Escopo de Variáveis

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

void funcao1(int a, int b)
{
    int soma;
    soma = a + b;
}

int main()
{
   int numero1, numero2;
   numero1 = 2;
   numero2 = 8;

   funcao1(numero1, numero2);
   printf("A soma e': %d", soma);

   return 0;
}
```

A variável **soma** só existe dentro da **funcao1**

Qual o valor será impresso?

Nenhum!!!

Esse comando resultará em um erro

Exercícios

1. Escreva um procedimento que recebe por parâmetro as 3 notas de um aluno e uma letra. Se a letra for **A,** o procedimento calcula a média aritmética das notas do aluno, se for **P**, a sua média ponderada (pesos: 5, 3 e 2) e se for **S**, a soma das notas. O valor calculado também deve ser retornado e "printado" na função main.
2. Faça uma função que recebe a média final de um aluno por parâmetro e retorna o seu conceito, conforme a tabela abaixo:

Nota	Conceito
De 0 a 4,9	D
De 5 a 6,9	C
De 7 a 8,9	B
De 9 a 10	A

Exercícios

3. Crie uma função que que receba o valor de um inteiro positivo N, calcule e retorne o fatorial desse número.

5. Escreva uma função que recebe por parâmetro um valor inteiro e positivo N e retorna o valor de S.

$$S=1+1/1! +1/2! +1/3! +...+1/N!$$

5. Você possui um pedaço retangular de papelão como mostra a Figura 1(a) e deve fazer uma caixa com ele. Para isso, vai cortar quadrados nos cantos (Figura 1(b)) e dobrar as laterais (Figura 1(c)), formando uma caixa (aberta em cima).

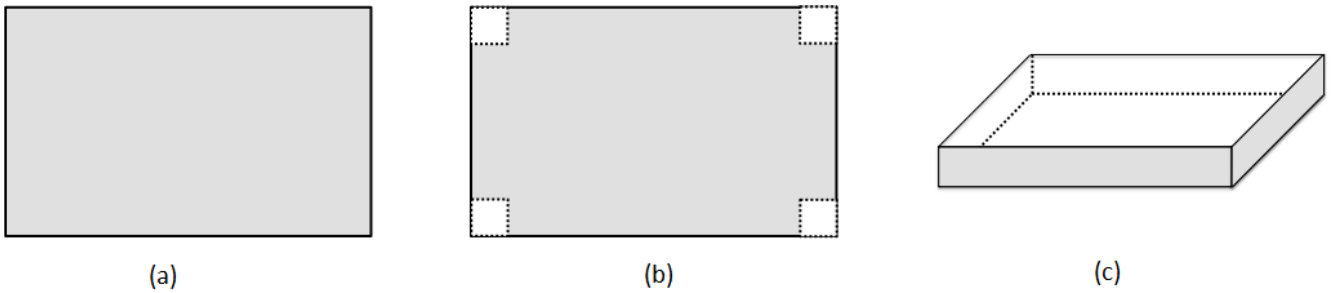


Figura 1

Os cantos cortados são quadrados do mesmo tamanho. A dúvida é: qual o tamanho dos lados desses cortes quadrados para que a caixa tenha o maior volume possível?

Por exemplo, com um pedaço de papelão de 25 x 40 cm (Figura 2(a)), o ideal é cortar quadrados de 5 cm (Figura 2-(b)), e assim obter uma caixa de volume 2250 cm^3 (Figura 2-(c)).

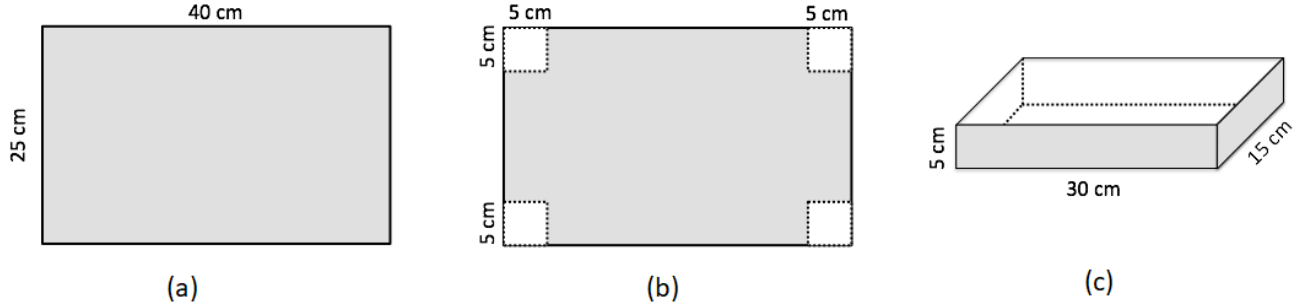


Figura 2