Modularização (Funções)

- Uma função corresponde a uma unidade autônoma de código do programa
- Soluciona uma tarefa em particular – "Dividir para conquistar"
- Sintaxe:

```
<tipo retorno> <nome da função>(<parâmetros>)
{
    <comandos>
}
```

- Exemplo (programa que calcula o máximo divisor comum entre dois números):

```
#include <stdio.h>

int x, y;  // variáveis globais do programa

int mdc( int a, int b )
{
   int menor, mdc, divisor_temp;  // variáveis locais da função mdc

   mdc = 1;
   if (a < b)
      menor = a;
   else
      menor = b;
   for (divisor_temp = 2; divisor_temp <= menor; divisor_temp++) {
      if (((a % divisor_temp) == 0) && ((b % divisor_temp) == 0))
         mdc = divisor_temp;
   }

   return mdc;
}

int main()
{
   scanf("%d %d", &x, &y);
   printf("O MDC entre %d e %d é %d", x, y, mdc(x,y));

   return 0;
}
```

- Como se pode notar, a função `main()` fica bem mais simples e legível quando fazemos uso de funções. Isto porque o programador considera que já possui uma função `mdc(int a, int b)` que, dados dois números (parâmetros *a* e *b* da função), calcula o máximo divisor comum (MDC) entre eles.
- Os próprios comandos `scanf` e `printf` são funções da biblioteca `stdio.h`. Sem elas, seria necessário implementar na função `main` a interação entre usuário e computador pelos dispositivos de entrada/saída.
- Note que a função `mdc(int a, int b)` declara três variáveis: `menor`, `mdc` e `divisor_temp`. Estas são *variáveis locais* que só existem para o programa durante a execução da chamada à função `mdc`.
- O programa usa duas *variáveis globais*: `x` e `y`. Estas são conhecidas por todo o programa e podem ser usadas em qualquer parte do código. *Não é aconselhável o seu uso por dificultar a identificação de erros.*

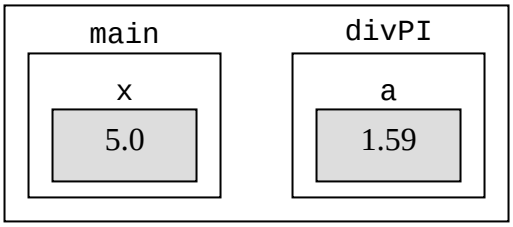
Passagem de Parâmetros

- Os parâmetros de uma função correspondem às ferramentas necessárias para a sua execução.
- Existem duas formas para se passar uma variável como parâmetro de uma função:
  - *Passagem por valor* – A função faz uma cópia do valor passado e qualquer alteração na cópia não modifica o valor original da variável passada.
  - *Passagem por referência* – A função usa uma referência (ponteiro) para a própria variável passada fazendo com que qualquer alteração nesta variável seja percebida fora da função.

usando Passagem por Valor

```
#include <stdio.h>

double divPI(double a) {
   a = a / 3.14;
   return a;
}

int main() {
   double x = 5.0;
   printf("%lf", divPI(x));
   return 0;
}
```

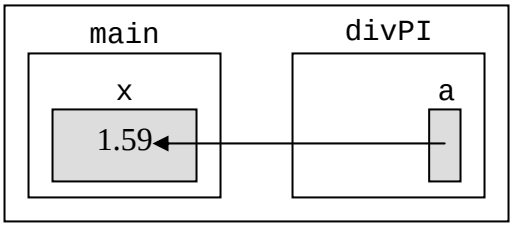
Memória Reservada para o Programa



usando Passagem por Referência

```
#include <stdio.h>

double divPI(double *a) {
   *a = *a / 3.14;
   return *a;
}

int main() {
   double x = 5.0;
   printf("%lf ", divPI(&x));
   return 0;
}
```

Memória Reservada para o Programa



- Em geral, a passagem por referência é mais eficiente do que a passagem por valor por não precisar realizar uma cópia da variável.
- No caso de funções que não precisam retornar nenhum valor, tem-se as chamadas *funções sem retorno* ou *procedimentos*. Neste caso, seu retorno é `void`.

Usando aritmética de ponteiros na referência

```
#include<stdio.h>

void imprimeVetor(int *vet, int tam) {
   for (int i = 0; i < tam; i++)
      printf("%d ", *(vet + i));
}

int main() {
   int vet[3] = {10, 20, 30};
   imprimeVetor( &vet[0], 3 );
}
```

Usando índice de vetores na referência

```
#include<stdio.h>

void imprimeVetor(int *vet, int tam){
   for (int i = 0; i < tam; i++)
      printf("%d ", vet[i]);
}

int main() {
   int vet[3] = {10, 20, 30};
   imprimeVetor( &vet[0], 3 );
}
```

Exercícios:

a) *Função Média* – Escreva um programa que receba do usuário dois vetores de 10 números cada e calcule a média de cada um deles. Para isso, crie uma função `média` que receba como parâmetro um vetor e que retorne a média de seus valores.

b) *Refinando a função Média* – Crie uma função `somatório` cujo objetivo é calcular o somatório dos valores de um vetor de 10 elementos. Esta função deve pertencer ao programa construído no item anterior. Faça com que a função `média` chame a função `somatório`.

c) Altere o programa anterior para aceitar vetores de qualquer tamanho (estipule um tamanho máximo de 100 elementos para a lista de números);

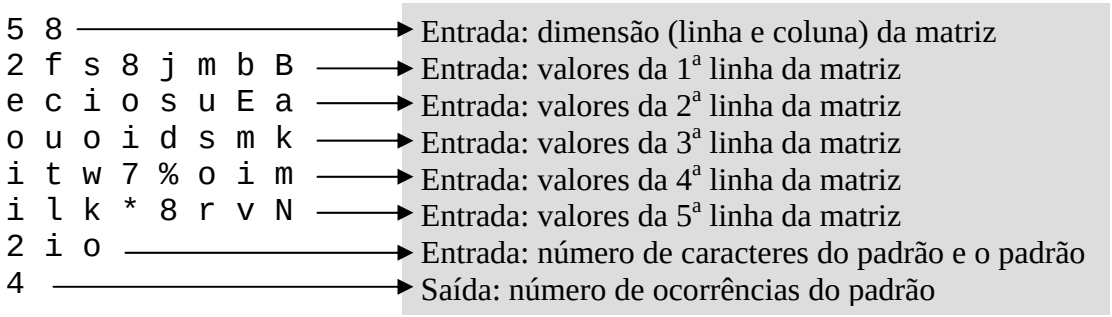
d) Qual a saída do seguinte programa em linguagem C:

```
#include <stdio.h>

void f1 (int a, int *b, int *c) {
   *b += a;
   a++;
   *c = a + *b;
   b = &a;
}

int main() {
   int x = 2;
   int y = 3;
   int z = -1;

   while (y < 10) {
      f1(x, &y, &z);
      printf("\n%d %d %d", x, y, z);
   }
}
```

e) *Caça-palavras* – Faça um programa que receba uma matriz de caracteres com *n* linhas por *m* colunas, tais que *n* e *m* não podem ser maiores do que 100. Em seguida, receba um padrão de *t* caracteres (onde $t \leq n$ e $t \leq m$) e procure-o na matriz nas direções horizontal e vertical em todos os sentidos. Ao término da busca, imprima na tela o número de ocorrências encontradas. No exemplo a seguir, o padrão `io` tem 4 ocorrências:

	0	1	2	3	4	5	6	7
0	2	f	s	8	j	m	b	B
1	e	c	i	o	s	u	E	a
2	o	u	o	#	d	s	m	k
3	i	t	w	7	%	o	i	m
4	i	l	k	*	8	r	v	N

A seguir tem-se as entradas fornecidas para o exemplo anterior e o resultado a ser apresentado na tela:



Observações Importantes:

- Se logo no início, quando o usuário informar as dimensões da matriz e estas forem maiores do que 100 por 100, interrompa o procedimento e apresente uma saída com uma dupla de zeros espaçados (“0 0”).

- É ***obrigado*** o uso de uma função chamada pela função `main()` que faça a busca na matriz e retorne o número de ocorrências do padrão nela. Esta função tem como parâmetros:
  - um ponteiro para o primeiro caractere da matriz (***obrigado*** usá-lo para buscar na matriz);
  - o tamanho dessa matriz;
  - um ponteiro para o primeiro caractere do padrão a ser procurado;
  - o tamanho deste padrão;

- Use outros parâmetros para a função se julgar necessário ou até mesmo crie novas funções se precisar.