Algortimos e Programação
Aula 13 - Funções

Priscila Delabetha

Até agora, em todos os programas que criamos, codificamos uma única função: **main()**. Entretanto, em todos eles, diversas funções foram utilizadas: printf(), scanf(), getch(), putch(), etc.

Essas funções estão disponíveis no sistema através de bibliotecas que acompanham o compilador Turbo C, mas podemos CRIAR nossas próprias funções e utilizá-las da mesma maneira.

Mas por quê?

```
int a, b, c;
int main() {
  //modifica valores
  a = 1;
  b = 2;
  c = 3;
  //imprimir valores
  printf("a = %d\n", a);
  printf("b = %d\n", b);
  printf("c = %d,\n", c);
  //modifica valores
  a = 4;
  b = 5;
  c = 6;
  //imprimir valores
  printf("a = %d\n", a);
  printf("b = %d\n", b);
  printf("c = %d\n", c);
}
```


```
int a, b, c;
int main(){

  //modifica valores
  a = 1;
  b = 2;
  c = 3;

  //imprimir valores
  ImprimirValores();

  //modifica valores
  a = 4;
  b = 5;
  c = 6;

  //imprimir valores
  ImprimirValores();
}
```

Mas por quê?

- Evitar repetição de código
  - Código menor
  - Menos bugs
- Organização
  - O código fica mais fácil de ser compreendido por outra pessoa e até mesmo para o próprio programador

Declaração

Para definir uma função, empregamos a seguinte forma básica:

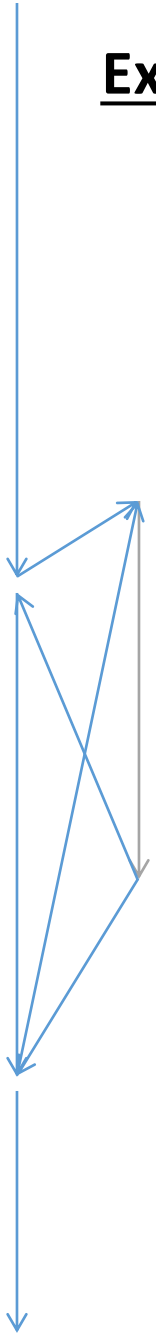
```
tipo nome(parâmetros){
     declarações
     comandos
   }
```

- tipo refere-se ao tipo de resposta que a função devolve e deve ser void (vazio) se a função não tem valor de resposta;
- nome é o identificador da função no resto do programa;
- parâmetros é uma lista de variáveis que representam valores de entrada para a função e deve ser void caso não haja valores de entrada;
- dentro do corpo da função, a primeira seção é destinada à declaração das variáveis e a segunda, aos comandos.

Execução Passo a Passo

Programa Principal

```
int a, b, c;
int main() {

  //modifica valores
  a = 1;
  b = 2;
  c = 3;

  //imprimir valores
  ImprimirValores();

  //modifica valores
  a = 4;
  b = 5;
  c = 6;

  //imprimir valores
  ImprimirValores();

  //fim
  printf("fim");
}
```



Subprograma ou função

```
void ImprimirValores(){
  printf("a = %d,\n",a);
  printf("b = %d,\n",b);
  printf("c = %d,\n",c);
}
```

Funções que não devolvem resposta

Nosso primeiro exemplo é uma função que não tem valor de resposta e não recebe argumentos ao ser chamada. Esta função tem como objetivo apenas exibir na tela o valor de algumas variáveis.

```
void ImprimirValores(){
  printf("a = %d,\n",a);
  printf("b = %d,\n",b);
  printf("c = %d,\n",c);
}
```

Funções que não devolvem resposta

Uma função do tipo void, quando executada, apenas produz um determinado efeito desejado, sem contudo devolver um valor de resposta. Sendo assim, o compilador não permite que funções void sejam usadas em expressões.

Funções que não devolvem resposta

```
int x;
x=ImprimirValores();
```

Assim, utilizar a função **ImprimirValores()** em uma expressão gera o seguinte erro:

```
error: void value not ignored as it ought to be
x=ImprimirValores();
```

Funções que não devolvem resposta

Isso acontece porque nossa função é **void**, quando declaramos ela no começo do programa, dissemos que essa função não retornaria valor algum!

Vamos ver um exemplo que você já está utilizando faz tempo!

Utilizando funções com retorno

Se uma função não é do tipo **void**, então ela deve, necessariamente, devolver um valor como resultado de sua execução.

Para isso, a função deve empregar o comando **return**. Esse comando, além de especificar a resposta da função, faz com que o controle retorne ao ponto onde ela foi chamada no programa, interrompendo imediatamente sua execução.

Funciona assim:

```
int somaValores(){
    return a+b+c;
}
```

Outras funções com retorno

```
int somaValores(){
      int x=a+b+c;
      return x;
}
```

```
char classifica(){
      char op;
      if(a > c){
            op='a';
      }else{
            op='c';
      }
return op;
}
```

O comando **return**

- A função termina sua execução quando se
  executado ou quando o comando return
- O comando return provoca a saída imedia ém
  e retorna a execução ao código chamado
- O comando return pode ser usado para d

```
int Vezes2(int a)
{
  a *= 2;
  printf("%d\n", a);

  return a;
}
```

A variável **a** recebe o valor retornado pela função Vezes2 →

```
int a = 1;
a = Vezes2(1);
printf("%d\n", a);
```

Output

```
2
2
```

Parâmetros de entrada

Mas nossas funções ainda são bem limitadas, você deve ter reparado que no código utilizado, tivemos que declarar as variáveis no início do programa (a, b e c) e nossas funções só trabalham com esses mesmos valores.

Existe então um recurso que nos possibilita utilizar a função para várias partes do programa, e diferentes variáveis: **parâmetros de entrada**.

Parâmetros de entrada

Parâmetros são variáveis que a função recebe. O número e tipo de parâmetros são preestabelecidos na declaração da função.

E essa declaração, do(s) tipo(s) e nome(s) da(s) variável(eis), acontece dentro dos parêntesis.

Por exemplo, posso criar uma função que some dois números, e posso receber eles por parâmetro:

```
void soma(int a, int b){
    //codigo
}
```

Parâmetros de entrada

- As funções pode receber dados de entrada

```
int a;
void ImprimirValor() {
    printf("%d", a);
}
```

Variável Global

Essa função só serve para imprimir o valor da variável **a**

Parâmetros de entrada

- As funções pode receber dados de entrada

```
void ImprimirValor(int a){
  printf("%d", a);
}
```

Variável **Local**, que recebeu o valor **passado** por **parâmetro**

Passagem de parâmetro por valor! Não altera o conteúdo da variável utilizada na chamada da função.

```
int main() {

  int Idade = 10;

  ImprimirValor(Idade);

  return 0;
}
```

Agora podemos usar essa função para imprimir qualquer valor passado por parâmetro

Parâmetros de entrada

Vamos utilizar parâmetros de entrada e valores de retorno em uma função para somar valores inteiros.

Variáveis Locais

- São declaradas dentro de uma função
- Só podem ser referenciadas por comandos que estão dentro do bloco(função, escopo) no qual elas foram declaradas, ou seja, não são reconhecidas fora da função onde foram declaradas
- Existem apenas enquanto o bloco de código em que foram declaradas está sendo executado

Programa Principal

```
int main() {

   int idade = 20;

   ImprimirVetor();

   printf("%d\n", idade);
}
```

Subprograma ou função

```
void ImprimirVetor(){
  int idade = 10;
  printf("%d\n", idade);
}
```

O que veremos na tela ao executarmos esse programa?

Variáveis Globais

- São declaradas fora das funções
  - No escopo global
- São reconhecidas pelo programa inteiro
- Podem ser usadas em qualquer trecho de código
- Existem durante toda a execução do programa
- Restringem a modularidade do programa

Programa Principal

```
int idade = 30;

int main() {

   idade = 20;

   ImprimirVetor();

   printf("%d\n", idade);
}
```

Subprograma ou função

```
void ImprimirVetor(){
  idade = 10;
  printf("%d\n", idade);
}
```

O que veremos na tela ao executarmos esse programa?

Protótipos

```
#include <stdio.h>
float square (float a);
```

Declaração

```
int main (){
  float num;
  printf ("Entre com um numero: ");
  scanf ("%f",&num);
  num=square(num);
  printf ("\n\nO seu quadrado vale: %f\n",num);
  return 0;
}

float square (float a){
  return (a*a);
}
```

Implementação

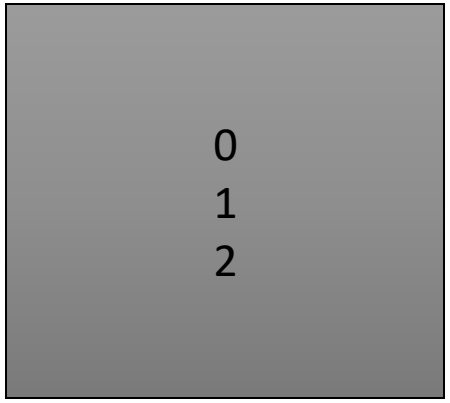
Programa Principal

```
int a;

int main() {

  a = 0;

  //imprimir a
  ImprimirVetor();

  a = 1;

  //imprimir a
  ImprimirVetor();

  a = 2;

  //imprimir a
  ImprimirVetor();
}
```

Subprograma ou função

```
void ImprimirVetor(){
    printf("%d\n", a);
}
```

Qual a saída do programa?



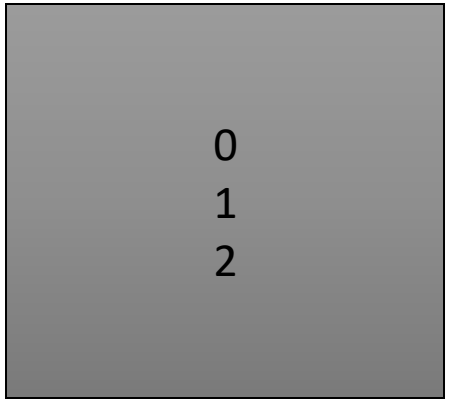
Programa Principal

```
int main() {

  int a;

  a = 0;

  //imprimir a
  ImprimirVetor(a);

  a = 1;

  //imprimir a
  ImprimirVetor(a);

  a = 2;

  //imprimir a
  ImprimirVetor(a);
}
```

Subprograma ou função

```
void ImprimirVetor(int a){


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

Qual a saída do programa?



Programa Principal

```
int main() {

  int a, b;

  a = 0;

  //imprimir a
  ImprimirVetor(a);

  b = 1;

  //imprimir a
  ImprimirVetor(b);

  a = 2;

  //imprimir a
  ImprimirVetor(a);
}
```

Subprograma ou função

```
void ImprimirVetor(int a){
  a = 3;

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

Qual a saída do programa?

```
3
3
3
```

Programa Principal

```
int main() {

  int a, b;

  a = 0;

  //imprimir a
  ImprimirVetor(a);

  b = 1;

  //imprimir a
  ImprimirVetor(b);

  a = 2;

  //imprimir a
  ImprimirVetor(a);

  //imprimir a
  printf("%d\n", a);
}
```

Subprograma ou função

```
void ImprimirVetor(int a){
  a = 3;

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

Qual a saída do programa?

```
3
3
3
2
```

Programa Principal

```
int main() {

  int a, b;

  a = 0;

  //imprimir a
  ImprimirVetor(a);

  b = 1;

  //imprimir a
  ImprimirVetor(b);

  a = 2;

  //imprimir a
  ImprimirVetor(a);

  //imprimir a
  printf("%d\n", a);
}
```

Subprograma ou função

```
void ImprimirVetor(int a){
  b = 3;
  printf("%d\n", a);
}
```

Qual a saída do programa?

Nenhuma! O programa não compila, a variável b não existe dentro da função

Programa Principal

```
int a, b;

int main() {

  a = b = 0;

  //imprimir a
  ImprimirVetor(a);

  //imprimir a
  printf("%d\n", a);

  b = 1;

  //imprimir a
  ImprimirVetor(b);

  //imprimir a
  printf("%d\n", a);

  a = 2;

  //imprimir a
  ImprimirVetor(a);

  //imprimir a
  printf("%d\n", a);
}
```

Subprograma ou função

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

  a = 9;
}
```

Variável local tem precedência!

Qual a saída do programa?

```
0
0
1
0
1
2
```

Programa Principal

```
int a, b;

int main() {

  a = b = 0;

  //imprimir a
  ImprimirVetor(a);

  //imprimir a
  printf("%d\n", a);

  b = 1;

  //imprimir a
  ImprimirVetor(b);

  //imprimir a
  printf("%d\n", a);

  a = 2;

  //imprimir a
  ImprimirVetor(a);

  //imprimir a
  printf("%d\n", a);
}
```

É o mesmo que deixar vazio entre os parênteses

Subprograma ou função

```
void ImprimirVetor(void){
  printf("%d\n", b);

  a = 9;
}
```

Qual a saída do programa?

Nenhuma! O programa não compila, a função não recebe parâmetro e no main ela está sendo chamada com parâmetro

Programa Principal

```
int a, b;

int main() {

  a = b = 0;

  //imprimir a
  ImprimirVetor();

  //imprimir a
  printf("%d\n", a);

  b = 1;

  //imprimir a
  ImprimirVetor();

  //imprimir a
  printf("%d\n", a);

  a = 2;

  //imprimir a
  ImprimirVetor();

  //imprimir a
  printf("%d\n", a);
}
```

Subprograma ou função

```
void ImprimirVetor(void){
  printf("%d\n", b);

  a = 9;
}
```

Qual a saída do programa?

```
0
9
1
9
1
9
```

Programa Principal

```
int a, b;

int main() {

  a = b = 0;

  //imprimir a
  ImprimirVetor();

  //imprimir a
  printf("%d\n", a);

  b = 1;

  //imprimir a
  ImprimirVetor();

  //imprimir a
  printf("%d\n", a);

  a = 2;

  //imprimir a
  ImprimirVetor();

  //imprimir a
  printf("%d\n", a);
}
```

Subprograma ou função

```
void ImprimirVetor(void){
  printf("%d\n", b);

}
```

Qual a saída do programa?

```
0
0
1
0
1
2
```

Programa Principal

```
int a, b;

int main() {

  a = b = 0;

  //imprimir a
  ImprimirVetor();

  //imprimir a
  printf("%d\n", b);

}
```

Subprograma ou função

```
void ImprimirVetor(void){
  printf("%d\n", b);

  for (int b = 0; b < 2; b++)
    printf("%d\n", b);
}
```

Variável local ao for!

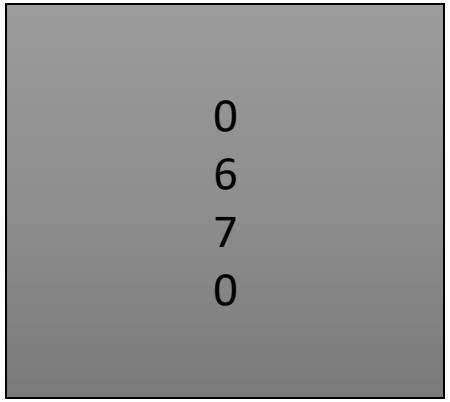
Qual a saída d

```
0
0
1
0
```

Programa Principal

```
int a, b;


int main() {

 a  = 0;

 //imprimir a
 printf("%d\n", a);

 if(1) {

  int a = 7;

  if(0) {

    int a = 9;
  }
   else {

     int a = 6;
     printf("%d\n", a);
   }

   printf("%d\n", a);
 }

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

Qual a saída do programa?



Passagem de vetores por parâmetro

- Vetores podem ser passados por parâmetro, a única diferença é que **vetores** são passados por **referência** e não por valor
- Passagem por referência **modifica** o conteúdo da **variável** usada na **chamada** da função

Declaração de função com vetores

Ao terminar a função pode retornar um valor e precisamos dizer de que tipo

Nome da função

<tipo de retorno> <identificad

<comandos separados por ;>

}

<tipo do dado> <identif

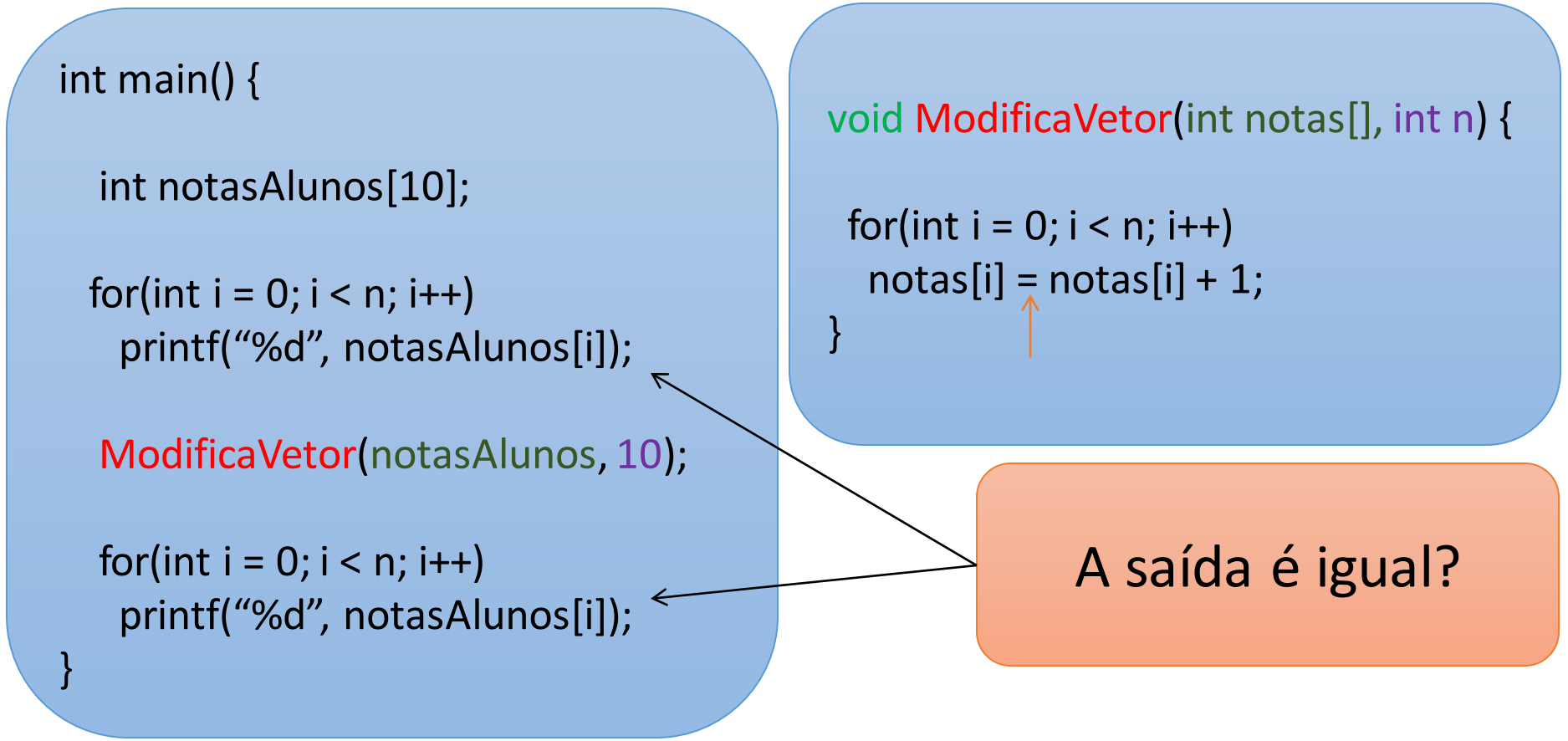
., n

Os [] dizem que a função recebe como parâmetro um vetor. Porém, não é possível saber o número de células no vetor sem que se passe uma variável que diga quantas células o vetor possui.

```
int main() {

   int notasAlunos[10];
   ImprimiVetor(notasAlunos, 10);
}
```

```
void ImprimiVetor(int notas[], int n) {

  for(int i = 0; i < n; i++)
    printf("%d", notas[i]);
}
```

Passagem por Referência



Declaração de função com matrizes

Ao terminar a função pode retornar um valor e precisamos dizer de que tipo

Nome da função

<tipo de retorno> <ident

<comandos separados p

}

<tipo do dado> <

..., n

Os [][] dizem que a função recebe como parâmetro uma matriz. Porém, não é possível saber o número de células na matriz sem que se passe duas variáveis que digam o número de células nas colunas e linhas.
Obs.: Para matrizes precisamos passar o tamanho da matriz, já em vetores não é necessário.

```
int main() {

   int notasAlunos[10][2];
   ImprimiMatriz(notasAlunos, 10, 2);
}
```

```
void ImprimiMatriz(int notas[10][2],
        int linhas, int colunas) {

  for(int i = 0; i < linhas; i++)
    for(int j = 0; j < colunas; j++)
      printf("%d", notas[i][j]);
}
```

Passagem por Referência

```
int main() {

   int notasAlunos[10][2];

  for(int i = 0; i < 10; i++)
    for(int j = 0; j < 2; j++)
      printf("%d", notasAlunos[i][j]);

   ModificaMtr(notasAlunos, 10, 2);

  for(int i = 0; i < 10; i++)
     for(int j = 0; j < 2; j++)
       printf("%d", notasAlunos[i][j]);
}
```

```
void ModificaMtr(int notas[10][2], int
linhas, int colunas) {

  for(int i = 0; i < linhas; i++)
    for(int j = 0; j < colunas; j++)
      notas[i][j] = notas[i][j] + 1;
}
```

A saída é igual?

Exercício

Menu

Faça um programa que imprime um menu na tela e para cada opção do menu crie uma função que faça algo relacionado a opção.

Seja muito criativo!