



CEFET – RJ / *Campus* Maria da Graça  
Centro Federal de Educação Tecnológica  
Celso Suckow da Fonseca – Rio de Janeiro



Prof. Cristiano Fuschilo

cristiano.fuschilo@cefet-rj.br

---

## Estrutura de Dados



Bacharelado em  
Sistemas de  
Informação



COORDENAÇÃO DE  
**Automação  
Industrial**  
Ensino de Qualidade



# Aula

---

Funções





# Funções



- Funções são as estruturas que permitem ao usuário separar seus programas em blocos. Se não as tivéssemos, os programas teriam que ser curtos e de pequena complexidade. Para fazermos programas grandes e complexos temos de construí-los bloco a bloco.
- Uma função no C tem a seguinte forma geral:

```
tipo_de_retorno nome_da_função (declaração_de_parâmetros){  
    corpo_da_função  
}
```

- O tipo-de-retorno é o tipo de variável que a função vai retornar. O default é o tipo int, ou seja, uma função para qual não declaramos o tipo de retorno é considerada como retornando um inteiro. A declaração de parâmetros é uma lista com a seguinte forma geral:

```
tipo nome1, tipo nome2, ... , tipo nomeN
```



# Return



- O comando return tem a seguinte forma geral:
  - `return valor_de_retorno;` ou
  - `return;`
- Digamos que uma função está sendo executada. Quando se chega a uma declaração return a função é encerrada imediatamente e, se o valor de retorno é informado, a função retorna este valor.
- É importante lembrar que o valor de retorno fornecido tem que ser compatível com o tipo de retorno declarado para a função.
- Uma função pode ter mais de uma declaração return. Isto se torna claro quando pensamos que a função é terminada quando o programa chega à primeira declaração return.





# Exemplos de uso do return

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

int main(int argc, char *argv[])
{
    int num, sr, flag;

    if (argc != 2) return 1;
    num = atoi(argv[1]);
    sr = (int)sqrt(num);
    if (num < 0)
        flag = 0;
    else
```



```
1  #include <stdio.h>
2
3  int Square (int a) {
4      return (a*a);
5  }
6
7  int main (){
8      int num;
9      printf ("Entre com um numero: ");
10     scanf ("%d",&num);
11     num=Square(num);
12     printf ("\n\n0 seu quadrado vale: %d\n",num);
13     return 0;
14 }
```



# Exemplos de uso do return

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

int main(int argc, char *argv[])
{
    int num, nr, flag;

    if (argc != 2) return 1;
    num = atoi(argv[1]);
    nr = (int)sqrt(num);
    if (num < 2)
        flag = 0;
    else
```



```
1  #include <stdio.h>
2
3  int EPar (int a) {
4      if (a%2)          /* Verifica se a e divisivel por dois */
5          return 0;     /* Retorna 0 se nao for divisivel */
6      else
7          return 1;     /* Retorna 1 se for divisivel */
8  }
9
10 int main () {
11     int num;
12     printf ("Entre com numero: ");
13     scanf ("%d",&num);
14     if (EPar(num))
15         printf ("\n\n0 numero e par.\n");
16     else
17         printf ("\n\n0 numero e impar.\n");
18     return 0;
19 }
```



# Retorno de valores



- É importante notar que, como as funções retornam valores, podemos aproveitá-los para fazer atribuições, ou mesmo para que estes valores participem de expressões. Mas não podemos fazer:

```
func(a,b)=x; /* Errado! */
```

- No segundo exemplo vemos o uso de mais de um return em uma função.
- Fato importante: se uma função retorna um valor você não precisa aproveitar este valor. Se você não fizer nada com o valor de retorno de uma função ele será descartado. Por exemplo, a função printf() retorna um inteiro que nós nunca usamos para nada. Ele é descartado.





# Protótipos de Funções



- Até agora, nos exemplos apresentados, escrevemos as funções antes de escrevermos a função `main()`. Isto é, as funções estão fisicamente antes da função `main()`. Isto foi feito por uma razão.
- Imagine-se na pele do compilador. Se você fosse compilar a função `main()`, onde são chamadas as funções, você teria que saber com antecedência quais são os tipos de retorno e quais são os parâmetros das funções para que você pudesse gerar o código corretamente.
- Foi por isto as funções foram colocadas antes da função `main()`: quando o compilador chegasse à função `main()` ele já teria compilado as funções e já saberia seus formatos.





# Protótipos de Funções



- Mas, muitas vezes, não poderemos nos dar ao luxo de escrever nesta ordem. Muitas vezes teremos o nosso programa espalhado por vários arquivos. Ou seja, estaremos chamando funções em um arquivo que serão compiladas em outro arquivo. Como manter a coerência?
- Em C++ uma função só pode ser usada se esta já foi declarada. Em C, o uso de uma função não declarada geralmente causava uma warning do compilador, mas não um erro. Em C++ isto é um erro.



# Protótipos de funções



- Para usar uma função que não tenha sido definida antes da chamada - tipicamente chamada de funções entre módulos - é necessário usar protótipos.
- Os protótipos de C++ incluem não só o tipo de retorno da função, mas também os tipos dos parâmetros:  
`void f (int a, float b); // protótipo da função f`
- Uma tentativa de utilizar uma função não declarada gera um erro de símbolo desconhecido.







# Como manter a coerência?

---

- A solução são os protótipos de funções. Protótipos são nada mais, nada menos, que declarações de funções. Isto é, você declara uma função que irá usar. O compilador toma então conhecimento do formato daquela função antes de compilá-la. O código correto será então gerado. Um protótipo tem o seguinte formato:

`tipo_de_retorno nome_da_função (declaração_de_parâmetros);`



# Como manter a coerência?

---



- Onde o tipo-de-retorno, o nome-da-função e a declaração-de-parâmetros são os mesmos que você pretende usar quando realmente escrever a função.
- Repare que os protótipos têm uma nítida semelhança com as declarações de variáveis.





# Exemplo

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

int main(int argc, char *argv[])
{
    int num, nr, flag;
    if (argc != 2) return 1;
    num = atoi(argv[1]);
    nr = (int)exp(sqrt(num));
    if (num < 0)
        flag = 0;
    else
```



```
1  #include <stdio.h>
2
3  float Square (float a);
4
5  int main (){
6      float num;
7      printf ("Entre com um numero: ");
8      scanf ("%f",&num);
9      num=Square(num);
10     printf ("\n\n0 seu quadrado vale: %f\n",num);
11     return 0;
12 }
13
14 float Square (float a){
15     return (a*a);
16 }
```



# Explicando

---



- Observe que a função `Square()` está colocada depois de `main()`, mas o seu protótipo está antes.
- Sem isto este programa não funcionaria corretamente.





# Protótipos



- Usando protótipos você pode construir funções que retornam quaisquer tipos de variáveis.
- É bom ressaltar que funções podem também retornar ponteiros sem qualquer problema.
- Os protótipos não só ajudam o compilador.
- Eles ajudam a você também: usando protótipos, o compilador evita erros, não deixando que o programador use funções com os parâmetros errados e com o tipo de retorno errado, o que é uma grande ajuda!





# O Tipo void

---

- Agora vamos ver o único tipo da linguagem C que não detalhamos ainda: o void. Em inglês, void quer dizer vazio e é isto mesmo que o void é. Ele nos permite fazer funções que não retornam nada e funções que não têm parâmetros! Podemos agora escrever o protótipo de uma função que não retorna nada:  
`void nome_da_função (declaração_de_parâmetros);`
- Numa função, como a acima, não temos valor de retorno na declaração **return**. Aliás, neste caso, o comando **return** não é necessário na função.





# void

---



- Podemos, também, fazer funções que não têm parâmetros:

```
tipo_de_retorno nome_da_função (void);
```

- ou, ainda, que não tem parâmetros e não retornam nada:

```
void nome_da_função (void);
```



# Exemplo - funções tipo void

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

int main(int argc, char *argv[])
{
    int num, nr, flag;

    if (argc != 2) return 1;
    num = atoi(argv[1]);
    nr = (int)exp(num);
    if (num < 0)
        flag = 0;
    else
```



```
1  #include <stdio.h>
2
3  void Mensagem (void);
4
5  int main (){
6      Mensagem();
7      printf ("\tDiga de novo:\n");
8      Mensagem();
9      return 0;
10 }
11
12 void Mensagem (void){
13     printf ("Ola! Eu estou vivo.\n");
14 }
```





# Funções que não recebem parâmetros



- Em C puro, um protótipo pode especificar apenas o tipo de retorno de uma função, sem dizer nada sobre seus parâmetros. Por exemplo,

```
float f(); // em C, não diz nada sobre os parâmetros de f é um protótipo incompleto da função f.
```

- Na realidade, esta é uma das diferenças entre C e C++. Um compilador de C++ interpretará a linha acima como o protótipo de uma função que retorna um float e não recebe nenhum parâmetro. Ou seja, é exatamente equivalente a uma função (void):



```
float f(); // em C++ é o mesmo que float f(void);
```



# Arquivos-Cabeçalhos



- São aqueles que temos mandado o compilador incluir no início de nossos exemplos e que sempre terminam em .h.
- A extensão .h vem de header (cabeçalho em inglês).
- Estes arquivos, na verdade, não possuem os códigos completos das funções. Eles só contêm protótipos de funções. É o que basta.
- O compilador lê estes protótipos e, baseado nas informações lá contidas, gera o código correto.





# Arquivos-Cabeçalhos



- O corpo das funções cujos protótipos estão no arquivo-cabeçalho, no caso das funções do próprio C, já estão compiladas e normalmente são incluídas no programa no instante da "linkagem".
- Este é o instante em que todas as referências a funções cujos códigos não estão nos nossos arquivos fontes são resolvidas, buscando este código nos arquivos de bibliotecas.
- Se você criar algumas funções que queira aproveitar em vários programas futuros, ou módulos de programas, você pode escrever arquivos-cabeçalhos e incluí-los também.





# Exemplo

---

- Suponha que a função 'int EPar(int a)', seja importante em vários programas, e desejemos declará-la num módulo separado.
- No arquivo de cabeçalho chamado por exemplo de "24\_funcao.h" teremos a seguinte declaração:

```
int EPar(int a);
```





# Exemplo



- O código da função será escrito num arquivo a parte. Vamos chamá-lo de "24\_funcao.c".
- Neste arquivo teremos a definição da função:

```
int EPar (int a){  
    if (a%2) // Verifica se a é divisível por dois  
        return 0;  
    else  
        return 1;  
}
```



# Programa Principal

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

int main(int argc, char *argv[])
{
    int num, nr, flag, i;

    if (argc != 2) return 1;
    num = atoi(argv[1]);
    nr = (int)sqrt(num);
    if (num < 2)
        flag = 0;
    else
```



```
1  #include <stdio.h>
2  #include "24_funcao.h"
3
4  int main(){
5      int num;
6      printf ("Entre com numero: ");
7      scanf ("%d",&num);
8      if (EPar(num))
9          printf ("\n\n0 numero e par.\n");
10     else
11         printf ("\n\n0 numero e impar.\n");
12 }
```





# Sobrecarga de Funções

---



- Em C++ é possível definir duas funções com o mesmo nome desde que a quantidade ou o tipo de parâmetros sejam diferentes.
- Isto é, podemos dar o mesmo nome a duas ou mais funções desde que estas possuam um número diferente de parâmetros ou parâmetros de tipos diferentes.
- Esta característica é designada por sobrecarga de funções (function overloading).



# Sobrecarga de Funções - Ex

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

int main(int argc, char *argv[])
{
    int num, nr, flag, i;

    if (argc != 2) return 1;
    num = atoi(argv[1]);
    nr = (int)exp((num));
    if (num < 0)
        flag = 0;
    else
```



```
1  #include <stdio.h>
2
3  int opera(int a, int b){
4      return (a * b);
5  }
6
7  float opera(float a, float b){
8      return (a*b);
9  }
10
11 int main(){
12     int x=5, y=2;
13     float n=5.0, m=2.0;
14     printf("Multiplica inteiro: %dx%d=%d \n\nMultiplica Real: %fx%f=%f",x,y,opera(x, y),n,m,opera(n, m) );
15     return 0;
16 }
```





# Explicando



- No exemplo anterior definimos duas funções com o mesmo nome, **opera**, mas uma delas aceita dois parâmetros do tipo int e a outra dois parâmetros do tipo float.
- O compilador sabe qual a função que pretendemos invocar analisando o tipo dos argumentos utilizados quando chamamos a função.



# Explicando



- Se for chamada com dois inteiros, utiliza a função que possui dois inteiros na sua definição.
- Se for chamada com dois reais, utiliza a função que possui dois reais na sua definição.
- Note por fim que uma função não pode ser sobrecarregada apenas à custa do tipo de retorno.
- Isto é, o compilador não permite que duas funções difiram apenas no tipo de retorno.





# Funções Recursivas

---



- A recursão é uma técnica que define um problema em termos de uma ou mais versões menores deste mesmo problema.
- A recursão pode ser utilizada sempre que for possível expressar a solução de um problema em função do próprio problema.
- Uma função é dita recursiva quando dentro do seu código existe uma chamada para si mesma.





# Exemplo Funções Recursivas

- Calcular o Fatorial de um número N inteiro qualquer. Se formos analisar a forma de cálculo temos:

$$\text{fat}(n) = \begin{cases} 1, & \text{se } n = 0 \text{ (solução trivial)} \\ n \times \text{fat}(n-1), & \text{se } n > 0 \text{ (solução recursiva)} \end{cases}$$

- Logo, temos que:

$$\text{fat}(5) = 5 \times \text{fat}(4)$$

$$\text{fat}(4) = 4 \times \text{fat}(3)$$

$$\text{fat}(3) = 3 \times \text{fat}(2)$$

$$\text{fat}(2) = 2 \times \text{fat}(1)$$

$$\text{fat}(1) = 1 \times \text{fat}(0)$$

$$\text{fat}(0) = 1$$





# Fatorial – Não Recursiva e Recursiva



```
1  #include <stdio.h>
2
3  int fatorial(int num){
4      int f, i;
5      if(num==0)
6          return 1;
7      else{
8          f=1
9          for(i=num;i>1;i++)
10             f *= i;
11      }
12      return f;
13  }
14
15  int main(){
16      int num=5;
17      printf("Fatorial de %d = %d",num,fatorial(num));
18      return 0;
19  }
```

NÃO Recursiva



# Fatorial – Não Recursiva e Recursiva



```
1  #include <stdio.h>
2
3  int fatorialRec(int num){
4      if(num==0)
5          return 1;
6      else
7          return num * fatorialRec(num-1);
8  }
9
10 int main(){
11     int num=5;
12     printf("Fatorial de %d = %d",num,fatorialRec(num));
13     return 0;
14 }
```

Recursiva





# Exercício



- Faça um programa em C para calcular a soma dos  $n$  primeiros números dados pelo usuário na entrada. Criar duas funções soma (uma recursiva e a outra não recursiva) que recebe como parâmetro de entrada o número lido.
- Lembre-se:

$$\text{somarec}(n) = \begin{cases} 1, & \text{se } n = 1 \text{ (solução trivial)} \\ n + \text{somarec}(n - 1), & \text{se } n > 1 \text{ (solução recursiva)} \end{cases}$$



