

Programação de Computadores I

Prof. Rafael Nunes

A Linguagem C

Parte 2

Vamos *Recordar...*

Programa 1

Alomundo.c

Progama1 – Alomundo.c

```
01  /* Meu Primeiro Programa */
02  #include <stdio.h>
03  int main (void)
04  {
05      //Imprime a seguinte mensagem na tela
06      printf ("Ola! Eu nasci!\n");
07
08      return 0;
09
10  //Fim
```

Vamos avançar um pouco

Meu segundo Programa

Progama2 – Conv_dia_ano.c

```
01  /* Meu Segundo Programa */
02  #include <stdio.h>
03  #include <stdlib.h>
04  int main (void)
05  {
06      int dias;
07      float anos;
08      setbuf(stdout, NULL); /*ATENCAO!!!*/
09      printf ("Digite o número de dias: ");
10      fflush(stdin);
11      scanf ("%d",&dias);
12      anos=dias/365.25;
13      printf ("\n%d dias equivalem a %f anos.\n",dias,anos);
14      return 0;
15  }//Fim
```

Entendendo o Programa2

- São declaradas duas variáveis chamadas **dias** e **anos**.
- A primeira é um **int** (inteiro) e a segunda um **float** (ponto flutuante).
 - **int** = apenas **valores inteiros**
 - **float** = apenas valores do conjunto **real** (ponto flutuante)
- É feita então uma chamada à função **printf()**, que coloca uma mensagem na tela.

Entendendo o Programa2

- Queremos agora *ler um dado* que será *fornecido pelo usuário* e colocá-lo na variável dias.
- Devemos utilizar a função scanf()

```
08     printf ("Digite o número de dias: ");  
09     scanf ("%d",&dias);
```

A função scanf()...

A função scanf()

- Leitura de dados da entrada padrão
 - A string “%d” diz à função que iremos ler um inteiro.
- O *segundo parâmetro* passado à função diz que o dado lido deverá ser *armazenado* na variável *dias*.

```
08    printf ("Digite o número de dias: ");  
09    scanf ("%d",&dias);
```

Tá bom Prof...

*... mas e esse '&' antes
da variável dias?*

A função scanf()

- É importante ressaltar a necessidade de se colocar um **&** antes do nome da variável a ser lida quando se usa a função scanf().
- O motivo disto só ficará claro mais tarde.
- Observe que, no C, quando temos **mais de um parâmetro** para uma função, eles serão **separados por vírgula**.

Continuando o entendimento
do *Programa2*...

Entendendo o Programa2

- Temos então uma *expressão matemática* simples que atribui a **anos** o valor de **dias** dividido por 365.25.

```
10      anos=dias/365.25;
```

- Como **anos** é uma variável float o compilador fará uma *conversão automática* entre os tipos das variáveis
 - Veremos isto com detalhes mais tarde...

Entendendo o Programa2

- A segunda chamada à função printf() tem três argumentos:

```
printf ("\n%d dias equivalem a %f anos.\n",dias,anos);
```

1. A string "\n%d dias equivalem a %f anos.\n" diz à função para dar um **retorno de carro** (passar para a próxima linha)
2. colocar um **inteiro** na tela;
3. colocar a mensagem " dias equivalem a ";
4. colocar um valor **float** na tela;
5. colocar a mensagem " anos.";
6. realizar mais um **retorno de carro**.

Entendendo o Programa2

- Os parâmetros restantes são as variáveis das quais deverão ser lidos os valores do *inteiro* e do *float*, respectivamente.



```
printf ("\n%d dias equivalem a %f anos.\n", dias, anos);
```

Exercício de Fixação

O que faz o seguinte programa?

```
01  /* xxxxxxxxxxxxxxxxxxxxxx */
02  #include <stdio.h>
03  int main (void)
04  {
05      int x;
06      scanf("%d", &x);
07      printf("%d", x);
08      return 0;
09  }//Fim
```

Introdução a *Funções*

O que é uma *função*?

Funções

- Uma função é um ***bloco de código*** de programa que *pode ser usado diversas vezes* em sua execução.
- O uso de funções permite que o programa fique ***mais legível***, mais bem *estruturado*.
- Um programa em C consiste, no fundo, de ***várias funções colocadas juntas***.

Vejamos um exemplo...

Exemplo de Função

```
#include <stdio.h>
/* Funcao simples: Só imprime Ola! */
void mensagem ()
{
    printf ("Ola! ");
}
/* Funcao principal */
int main ( )
{
    mensagem();
    printf ("Tudo Bem?\n");
    return 0;
}
```


Argumentos

Argumentos

- Argumentos são **as entradas** que a função recebe.
- É através dos argumentos que passamos **parâmetros** para a função.
- Ex: As funções printf() e scanf() são funções que **recebem argumentos**.

Argumentos

Exemplo de função definida pelo usuário e que utiliza um argumento....

Exemplo de função com argumento

```
#include <stdio.h>
#include <stdlib.h>
/* Calcula o quadrado de x */
void square (int x)
{
    printf ("O quadrado e %d",(x*x));
}
```

```
int main ()
{
    setbuf(stdout, NULL);
    int num;
    printf ("Entre com um numero: ");
    fflush(stdin);
    scanf ("%d",&num);
    printf ("\n\n");
    square(num);
    return 0;
}
```

Argumentos - Vamos entender...

- **Na definição** da função `square()` dizemos que a função receberá um argumento inteiro *x*.



```
void square (int x)
```

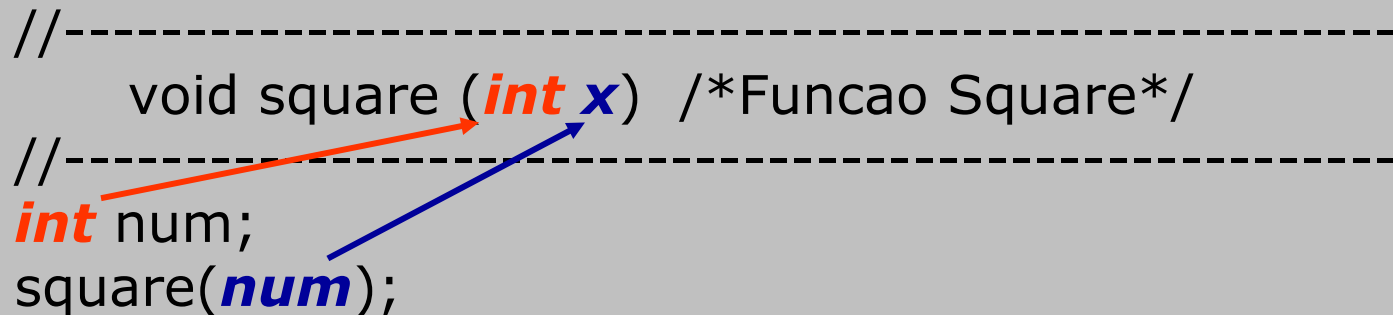
- Quando fazemos a **chamada** à função, o inteiro *num* é passado como argumento.

```
square (num)
```

Argumentos - Observações

- Em primeiro lugar temos de satisfazer aos requisitos da função **quanto ao tipo** e à **quantidade de argumentos** quando a chamamos.

```
//-----  
void square (int x) /*Funcao Square*/  
//-----  
int num;  
square(num);
```



The diagram illustrates the requirement for argument types in C. It shows a function definition `void square (int x) /*Funcao Square*/` and a function call `square(num);` where `num` is declared as `int`. An orange arrow points from the `int` type in the variable declaration to the `int` type in the function parameter list. A blue arrow points from the `num` variable in the function call to the `x` parameter in the function definition, demonstrating that the argument's type matches the parameter's type.

- Apesar de existirem algumas **conversões de tipo**, que o C faz automaticamente, é importante ficar atento.

Argumentos - Observações

- Em segundo lugar, **não** é importante o **nome da variável** que se passa como argumento, ou seja, a variável num, ao ser passada como argumento para square() é **copiada** para a variável x.

```
//-----  
void square (int x) /*Funcao Square*/  
//-----  
int num;  
square(num);
```

- Dentro de square() trabalha-se apenas com x. Se **mudarmos** o valor de x dentro de square() o valor de num na **função main()** **permanece inalterado**.

E quando precisamos passar
mais do que um argumento para
a função?

Argumentos – Mais do que um...

```
#include <stdio.h>
/* Multiplica 3 numeros */
void mult (float a, float b,float c)
{
    printf ("%b",a*b*c);
}
```

```
int main ()
{
    float x,y;
    x=23.5;
    y=12.9;
    mult (x,y,3.87);
    return 0;
}
```

Argumentos – Mais do que um...

- Repare que, neste caso, os argumentos são **separados por vírgula** e que deve-se explicitar **o tipo de cada um dos argumentos**, um a um.

```
void mult (float a, float b, float c)
```

- Note também que os argumentos passados para a função **não necessitam** ser todos **variáveis** porque mesmo sendo **constantes** serão copiados para a variável de entrada da função

```
mult (x,y,3.87);
```

Retornando Valores

*Para que eu preciso retornar
valores após realizar uma
função?*

Retornando Valores

- Muitas vezes é necessário fazer com que uma função **retorne um valor**.
 - As funções que vimos até aqui **não retornam nada**, pois especificamos um retorno void.
- Podemos especificar um **tipo de retorno** indicando-o antes do nome da função.
 - Mas para dizer ao C o *que* vamos retornar precisamos da palavra reservada **return**.

Sabendo disso fica fácil fazer
uma função para *multiplicar dois*
inteiros e que *retorna o*
resultado da multiplicação

Retornando Valores

```
#include <stdio.h>
int prod (int x,int y)
{
    return (x*y);
}

int main ()
{
    int saida;
    saida = prod(12,7);
    printf ("A saida e: %d\n",saida);
    return 0;
}
```

Retornando Valores

- Veja que como `prod` retorna o valor de 12 multiplicado por 7, este valor pode ser usado em uma expressão qualquer.
- No programa fizemos a atribuição deste resultado à variável ***saida***, que posteriormente foi impressa usando o `printf()`.

```
saida = prod (12,7);  
printf ("A saida e: %d\n",saida);
```

Retornando Valores

- Uma observação adicional:
 - Se não especificarmos o tipo de retorno de uma função, o compilador C automaticamente suporá que este ***tipo é inteiro***.
 - Porém, não é uma boa prática não se especificar o valor de retorno
 - Nesta disciplina, este valor será ***sempre deverá ser especificado***.

```
int prod (int x,int y)
```


Retornando Valores

*Já vimos que devemos retornar
algo para a função principal
(main)...*

Vamos ver mais um exemplo...

- A função agora recebe **dois floats** e também **retorna** um **float**.
- Repare que no exemplo a seguir especificamos um **valor de retorno para a função main** (int) e retornamos zero.
- Normalmente é isto que fazemos com a função main:
 - retorna zero quando é executada **sem** qualquer tipo de erro

Retornando Valores

```
#include <stdio.h>
float prod (float x,float y)
{
    return (x*y);
}

int main ()
{
    float saida;
    saida=prod (45.2,0.0067);
    printf ("A saida e: %f\n",saida);
    return 0;
}
```

Qual é então a forma geral de
uma função...

Sua estrutura...

Forma Geral

```
tipo_de_retorno nome_da_função (lista_de_argumentos)  
{  
    código_da_função  
}
```

Exercício

Exercício

- Escreva uma função que some dois inteiros e retorne o valor da soma
- Adicione ao mesmo programa uma função para subtrair dois inteiros e retornar o valor da subtração
- Adicione também ao programa uma função para multiplicar dois floats e retornar o valor da multiplicação
- Adicione ao programa uma função para dividir dois números...

Exercício 02

- Melhore seu programa...
- Conhecendo os conceito das funções `printf()` e `scanf()` ...
- Implemente a interação com o usuário...
Peça-o para entrar com os dados...

Até a próxima...