




Algoritmos e Estrutura de Dados
Videoaula 01 (Parte 1/3)


Prof. Rafael Nunes – M.Sc. em Engenharia de Computação pela UFMG



Revisando a última aula!

Algoritmos e Estrutura de Dados

Prof. Rafael Nunes – M.Sc. em Engenharia de Computação pela UFMG



Primeiro Programa

Prof. Rafael Nunes – M.Sc. em Engenharia de Computação pela UFMG



Programa 01 – alomundo.c

Abram a ferramenta e após criar um novo projeto digitem o seguinte programa abaixo:

```
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
```



Tipos e Modificadores



Tipos da Linguagem C

❑ O C possui 5 tipos básicos:

- ❑ char
- ❑ int
- ❑ float
- ❑ void
- ❑ double



Modificadores de Tipo

- ❑ Para os tipos de variáveis também podemos aplicar os **modificadores de tipo**.
- ❑ Os modificadores de tipo do C **são quatro**:
 - ❑ signed,
 - ❑ unsigned,
 - ❑ long
 - ❑ short.



Operadores



Operadores Aritméticos

- ❑ Os operadores aritméticos são usados para desenvolver operações matemáticas.
- ❑ A seguir apresentamos a lista dos operadores aritméticos do C:

Operador	Ação
+	Soma (inteira e ponto flutuante)
-	Subtração ou Troca de sinal (inteira e ponto flutuante)
*	Multiplicação (inteira e ponto flutuante)
/	Divisão (inteira e ponto flutuante)
%	Resto de divisão (de inteiros)
++	Incremento (inteiro e ponto flutuante)
--	Decremento (inteiro e ponto flutuante)



Operadores Relacionais

❑ Os operadores relacionais do C **realizam comparações** entre variáveis.

❑ São eles:

== (**igual**),
 != (**diferente de**),
 > (**maior que**),
 < (**menor que**),
 >= (**maior ou igual**),
 <= (**menor ou igual**).

❑ Os operadores relacionais retornam verdadeiro (1) ou falso (0)

Prof. Rafael Nunes – M.Sc. em Engenharia de Computação pela UFMG

10



Operadores Lógicos

❑ Para fazer **operações com valores lógicos** (verdadeiro e falso) temos os operadores lógicos:

&& (**and (e)**),
 || (**or (ou)**),
 ! (**not (não)**).

Prof. Rafael Nunes – M.Sc. em Engenharia de Computação pela UFMG

11

Objetivos de Aprendizagem

- ❑ Controle de **Fluxo**
 - ❑ Estrutura de decisão
 - ❑ O comando **switch**
 - ❑ Estrutura de repetição
 - ❑ Comandos break e continue

❑ Funções



A Linguagem C

Conceitos Básicos Segundo Programa

Abra a ferramenta **Eclipse**
e após criar um **novo projeto**
digite o programa a seguir!

Programa 02 – convDiaAno.c

```
1  /* Meu Segundo Programa */
2  #include <stdio.h>
3  #include <stdlib.h>
4  int main (void)
5  {
6      int dias;
7      float anos;
8      setbuf(stdout, NULL); /*ATENCAO!!!*/
9      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
```

Analizando meu Segundo Programa em C

Analizando o Segundo Programa

- ☐ 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.

Analizando o Segundo Programa

- ☐ 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 fflush(stdin);
10 scanf("%d",&dias);
```

A função `scanf()`

Linha 10
do programa "convDiaAno.c"

A Função `scanf()`

- ☐ Leitura de dados da entrada padrão
 - ☐ A tag `"%d"` diz à função que iremos ler um inteiro.
- ☐ O *segundo parâmetro* passado à função diz que o dado capturado deverá ser *armazenado* na variável *dias*.

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

A função `scanf()`

& → Atenção



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 a Análise do Segundo Programa ...



Analisando o Segundo Programa

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

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

- ❑ Como **anos** é uma variável float o compilador fará uma **conversão automática** entre os tipos das variáveis



Analisando o Segundo Programa

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

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

- ❑ A string `"\n%d dias equivalem a %f anos.\n"` diz à função para *saltar para a próxima linha*
 - ❑ colocar um *inteiro* na tela;
 - ❑ colocar a *mensagem* " dias equivalem a ";
 - ❑ colocar um valor *float* na tela;
 - ❑ colocar a *mensagem* " anos.";
 - ❑ realizar mais um *retorno de carro*.



Analisando o Segundo Programa

- ❑ 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

```

Prof. Rafael Nunes – M.Sc. em Engenharia de Computação pela UFMG

28

No próximo bloco...



- ☐ Continuaremos nossos estudos sobre a Linguagem C, agora sobre os *comandos de controle de fluxo*:
 - ☐ Estruturas de decisão;
 - ☐ Comando switch;
 - ☐ Estruturas de repetição.

Bons estudos!

Prof. Rafael Nunes – M.Sc. em Engenharia de Computação pela UFMG

29



Algoritmos e Estrutura de Dados Videoaula 01 (Parte 2/3)

Prof. Rafael Nunes – M.Sc. em Engenharia de Computação pela UFMG

A Linguagem C

Controle de Fluxo

Estruturas de *Decisão*

O comando *if*



O comando if

- ❑ O comando if representa uma **tomada de decisão** do tipo "SE *isto* ENTÃO aquilo".
- ❑ A sua forma geral é:
if (*condição*) declaração;



O comando if

- ❑ A condição do **comando if** é uma expressão que **será avaliada**.
- ❑ Se o resultado for **zero** a declaração **não será executada**.
- ❑ Se o resultado for **qualquer coisa diferente de zero** a declaração **será executada**.



Vamos **entender...**



O comando if

```
teste = 1;
```

1. if (teste == 0)
2. printf("ALO!") *//NÃO IRÁ EXECUTAR*
- //-----
1. if (teste != 0)
2. printf("ALO!") *//IRÁ EXECUTAR*



Vejamos agora um *exemplo...*



O comando if

```
#include <stdio.h>
int main ()
{
    setbuf(stdout,NULL);
    int num;
    printf ("Digite um numero: ");
    scanf ("%d",&num);
    if (num>10)
        printf ("\n\nO numero e maior que 10");
    if (num==10)
    {
        printf ("\n\nVoce acertou!\n");
        printf ("O numero e igual a 10.");
    }
    if (num<10)
        printf ("\n\nO numero e menor que 10");
    return 0;
}
```



O comando if

- ❑ No programa acima **a expressão $num > 10$** é avaliada e retorna um **valor diferente de zero, se verdadeira, e zero, se falsa.**
- ❑ Repare que quando queremos testar igualdades usamos o operador **$==$** e não **$=$**
- ❑ Isto é porque o operador **$=$** representa apenas uma **atribuição**.



O comando **if-else**



O comando if-else

- ❑ Podemos pensar no comando else como sendo um **complemento do comando if**.
- ❑ O **comando if** completo tem a seguinte forma geral:
 - if (**condição**) declaração_1;
 - else** declaração_2;



O comando if-else

- ❑ A *expressão da condição* será avaliada
- ❑ Se ela for *diferente de zero* (verdadeiro) a *declaração 1* será executada.
 - ❑ if (*condição!=0*) executa *declaração_1*
- ❑ Se for *zero* (falso) a *declaração 2* será executada, ou seja, o bloco do *else*
 - ❑ if (*condição==0*) executa *declaração_2*



O comando if-else

- ❑ É importante nunca esquecer que, quando usamos a estrutura *if-else*, estamos garantindo que uma das duas declarações *sempre será executada*
- ❑ Nunca serão executadas *as duas* ou *nenhuma delas*



Vejamos um *exemplo...*

O comando if-else



```
#include <stdio.h>
int main ()
{
    int num;
    printf ("Digite um numero: ");
    scanf ("%d",&num);
    if (num==10)
    {
        printf ("\n\nVoce acertou!\n");
        printf ("O numero e igual a 10.\n");
    }
    else
    {
        printf ("\n\nVoce errou!\n");
        printf ("O numero e diferente de 10.\n");
    }
    return 0;
}
```

Bloco Verdadeiro
= if

Bloco falso
= else

Prof. Rafael Nunes – M.Sc. em Engenharia de Computação pela UFMG

46

O comando *if-else-if*



Prof. Rafael Nunes – M.Sc. em Engenharia de Computação pela UFMG

47

O comando if-else-if



- ☐ A estrutura *if-else-if* é apenas uma extensão da estrutura if-else.
- ☐ Sua forma geral pode ser escrita como sendo:

```
if (condição_1) declaração_1;
else if (condição_2) declaração_2;
else if (condição_3) declaração_3;
...
else if (condição_n) declaração_n;
else declaração_default; //Opcional
```

Prof. Rafael Nunes – M.Sc. em Engenharia de Computação pela UFMG

48

O comando if-else-if



⏏ Atenção!

A última declaração (**default**) é a que será executada no caso de todas as condições falharem, ou seja, serem falsas (zero)

Ela **é opcional !**

Vejamos um **exemplo...**



O comando if-else-if



```
#include <stdio.h>
int main ()
{
    int num;
    printf ("Digite um numero: ");
    scanf ("%d",&num);
    if (num>10)
        printf ("\n\nO numero e maior que 10");
    else if (num==10)
    {
        printf ("\n\nVoce acertou!\n");
        printf ("O numero e igual a 10.");
    }
    else if (num<10)
        printf ("\n\nO numero e menor que 10");
    return(0);
}
```

Vamos refletir um pouco sobre *a expressão condicional*

A expressão condicional

- ☐ Quando o compilador avalia uma condição, ele espera um **valor de retorno** para poder tomar sua decisão.
- ☐ Mas esta expressão não precisa ser uma expressão *no sentido convencional*.
- ☐ Uma **variável sozinha** pode avaliada como "expressão"
 - ☐ Ela apenas retorna seu próprio valor.

A expressão condicional

- ☐ Isto quer dizer que teremos as seguintes expressões:


```
int num;
if (num!=0) ....
if (num==0) ....
for (i = 0; string[i] != '\0'; i++)
```
- ☐ equivalem a

```
int num;
if (num) ....
if (!num) ....
for (i = 0; string[i]; i++)
```
- ☐ ... e que podemos simplificar essas expressões.

if's *aninhados*

if's aninhados

- ❑ O if aninhado é simplesmente um comando *if* declarado dentro de um *outro comando if*.
- ❑ Devemos ter cuidado na organização dos diversos comandos e saber exatamente qual *else* pertence a qual *if*

if's aninhados

```
...
if (num==10)
{
    printf ("\n\nVoce acertou!\n");
    printf ("O numero e igual a 10.\n");
}
else
{
    if (num>10)
    {
        printf ("O numero e maior que 10.");
    }
    else
    {
        printf ("O numero e menor que 10.");
    }
}
}
```

O comando *switch*

O comando switch

- ☐ O comando *if-else* e o comando *switch* são os dois comandos de *tomada de decisão*.
- ☐ Sem dúvida alguma o mais importante dos dois é o *if*, mas o comando *switch* tem aplicações valiosas.
- ☐ O comando *switch* é próprio para se *testar* uma variável em relação a diversos valores pré-estabelecidos.

O comando switch

- ☐ Forma geral:

```
switch(<expressão>){
  case <const1>: <instruções1>
  case <const2>: <instruções2>
  ...
  case <constN>: <instruçõesN>

  default: <instruções>
}
```

O comando switch - Exemplo



```
switch(dia){
    case 0: printf("Domingo\n"); break;
    case 1: printf("Segunda\n"); break;
    case 2: printf("Terça\n"); break;
    case 3: printf("Quarta\n"); break;
    case 4: printf("Quinta\n"); break;
    case 5: printf("Sexta\n"); break;
    case 6: printf("Sabado\n"); break;

    default: printf("Opção Inválida\n"); break;
}
```

O comando switch



- ☐ Podemos fazer uma analogia entre o switch e a estrutura if-else-if apresentada anteriormente:
 - ☐ A diferença fundamental é que a estrutura switch **não aceita expressões**.
 - ☐ Somente aceita **constantes**.
- ☐ O switch testa a variável e executa a declaração cujo case corresponda ao valor atual da variável.
- ☐ A declaração default é opcional

O comando switch



- ☐ O comando **break**, faz com que o **switch** seja interrompido assim que uma das declarações seja executada.
 - ☐ Mas ele **não** é essencial ao comando switch.
- ☐ Se após a execução da declaração não houver um break, o programa continuará executando normalmente. (**Cuidado!**)

Vejamos um *exemplo* completo...

```
#include <stdio.h>
int main ()
{
    int num;
    printf ("Digite um numero: ");
    scanf ("%d",&num);
    switch (num) {
        case 9:
            printf ("\nO numero eh igual a 9.\n");
            break;
        case 10:
            printf ("\nO numero eh igual a 10.\n");
            break;
        case 11:
            printf ("\nO numero eh igual a 11.\n");
            break;
        default:
            printf ("\nO numero não é nem 9 nem 10 nem 11.\n");
    }
    return 0;
}
```

Estruturas de *Repetição*

O comando *for*

O comando for

- ☐ O loop (laço) *for* é utilizado para *repetir um comando*, ou bloco de comandos, *diversas vezes*;
- ☐ Sua forma geral é:
for (*inicialização*; *condição*; *incremento*)
declaração;

O comando for

- ☐ A declaração no comando for também pode ser *um bloco* ({ }) e neste caso o *;* é omitido.
- ☐ O melhor modo de se entender o loop for é observando a *maneira como ele funciona* "internamente".

O comando for



- ❑ O **loop for** é equivalente a seguinte estrutura:

```
inicialização;
if (condição)
{
    declaração;
    incremento;
    "Volte para o comando if"
}
```

O comando for



- ❑ Podemos ver então que o **for** executa a **inicialização** incondicionalmente e testa a condição.
 - ❑ Se a **condição** for falsa ele não faz mais nada. Sai do loop.
 - ❑ Se a **condição** for verdadeira ele executa a declaração (apenas uma vez), o **incremento** e volta a testar a condição novamente.
- ❑ Ele repete estas operações até que a condição se torne **falsa**.

Vejamos um **exemplo...**



O comando for



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

```
int main () {
    int count;
    for (count=1; count<=100; count=count+1) printf ("%d ",count);
    return 0;
}
```

O comando **for**

Loop infinito



O comando for – Loop Infinito



- ☐ O loop infinito tem a forma

for (inicialização; <vazio>; incremento) declaração;

- ☐ Este loop chama-se loop infinito porque **será executado para sempre**, a não ser que ele seja interrompido
 - ☐ Para interromper um loop como este usamos o comando break.
 - ☐ O comando break vai quebrar o loop infinito e o programa continuará sua execução normalmente.



O comando for – Loop Infinito

- ❑ O programa que faz a leitura de uma tecla e sua impressão na tela, até que o usuário aperte uma tecla sinalizadora de final (um FLAG). O nosso FLAG será a letra 'X'.

```
#include <stdio.h>
#include <stdlib.h>
int main () {
    setbuf(stdout, NULL);
    int Count;
    char ch;
    for (Count=1; ;Count++){
        ch = getchar();
        if (ch == 'X') break;
        printf("\nLetra: %c",ch);
    }
    return 0;
}
```



O comando *while*



O comando while

- ❑ O comando while tem a seguinte forma geral:

while (*condição*) declaração;



O comando while

- ❑ Assim como fizemos para o comando **for**, vamos tentar mostrar como o comando **while** funciona fazendo uma analogia.
- ❑ Então o **while** seria equivalente a:

```

if (condição)
{
    declaração;
    "Volte para o comando if"
}
  
```



O comando while

- ❑ Podemos ver que o comando **while** testa uma **condição**.
- ❑ Se esta **condição** for verdadeira a declaração é executada e faz-se o teste novamente, e assim por diante.



O comando while

- ❑ Assim como no caso do **for**, podemos fazer um loop infinito.
 - ❑ Para tanto basta colocar uma expressão eternamente verdadeira na condição.
- ❑ Pode-se também **omitir a declaração** e fazer um **loop sem conteúdo**



O comando while

- ❑ O programa abaixo espera o usuário digitar a tecla 'q' e só depois finaliza:

```
#include <stdio.h>

int main () {
    setbuf(stdout, NULL);
    char Ch;
    Ch='\0';
    while (Ch!='q')
    {
        Ch = getchar();
    }
    return 0;
}
```



O comando **do-while**



O comando do-while

- ❑ Forma geral:

```
do
{
    declaração;
} while (condição);
```

- ❑ Mesmo que a declaração seja apenas um comando é uma boa prática deixar as chaves.
 - ❑ O ponto-e-vírgula final é **obrigatório**.



O comando do-while

- ❑ Vemos pela análise do bloco anterior que a estrutura do-while **executa a declaração**, **testa a condição** e, se esta for verdadeira, volta para a declaração.
- ❑ A grande novidade no comando do-while é que ele, ao contrário do for e do while, garante que a declaração **será executada pelo menos uma vez**.

Prof. Rafael Nunes – M.Sc. em Engenharia de Computação pela UFMG

85

```
int main () {
    int i;
    do {
        printf ("\n\nEscolha a fruta pelo numero:\n\n");
        printf ("%i)...Mamão\n");
        printf ("%i)...Abacaxi\n");
        printf ("%i)...Laranja\n");
        scanf ("%d", &i);
    } while ((i<1)||i>3);

    switch (i) {
        case 1:
            printf ("%i\tVoce escolheu Mamão.\n");
            break;
        case 2:
            printf ("%i\tVoce escolheu Abacaxi.\n");
            break;
        case 3:
            printf ("%i\tVoce escolheu Laranja.\n");
            break;
    }
    return 0;
}
```

Prof. Rafael Nunes – M.Sc. em Engenharia de Computação pela UFMG

86




Exercício de Fixação

1. Escreva um programa que peça dois inteiros, correspondentes a horas e minutos (Ex: 23:59).
2. Peça os números até conseguir valores que estejam na faixa correta (horas entre 0 e 23 e minutos entre 0 e 59).
3. Imprima a hora digitada pelo usuário no formato "xx:xx"
4. Comente seu programa

Prof. Rafael Nunes – M.Sc. em Engenharia de Computação pela UFMG

87

 UNIVERSIDADE FUMEC

No próximo bloco...

- ❑ Continuaremos nossos estudos sobre a Linguagem C, agora sobre:
 - ❑ Comando break e continue
 - ❑ Funções


Bons estudos!

Prof. Rafael Nunes – M.Sc. em Engenharia de Computação pela UFMG 88

 UNIVERSIDADE FUMEC

 **Algoritmos e Estrutura de Dados**
Videoaula 01 (Parte 3/3)

Prof. Rafael Nunes – M.Sc. em Engenharia de Computação pela UFMG

 UNIVERSIDADE FUMEC

O comando *break*

Prof. Rafael Nunes – M.Sc. em Engenharia de Computação pela UFMG 90



O comando break

- ☐ Nós já vimos dois usos para o comando break:
 - ☐ interrompendo os comandos switch e for.
- ☐ Na verdade, estes são os dois usos do comando break:
 - ☐ ele pode quebrar a execução de um comando (como no caso do switch) ou ...
 - ☐ interromper a execução de qualquer loop (como no caso do for, do while ou do do while).



O comando break

- ☐ O break faz com que a execução do programa continue na primeira linha seguinte ao loop ou bloco que está sendo interrompido.



O comando *continue*



O comando continue

- ❑ O comando **continue** pode ser visto como sendo o oposto do break.
 - ❑ Ele só funciona dentro de um loop.
- ❑ Quando o comando **continue** é encontrado, o loop pula para a próxima iteração, sem o abandono do loop
- ❑ Ao contrário do que acontecia no comando break.

Prof. Rafael Nunes – M.Sc. em Engenharia de Computação pela UFMG

94

```
int main() {
    setbuf(stdout, NULL);
    int opcao;
    while (opcao != 5) {
        printf("\n\n Escolha uma opcao entre 1 e 5: ");
        fflush(stdin);
        scanf("%d", &opcao);
        /* Opcao invalida: volta ao inicio do loop */
        if ((opcao > 5)|| (opcao < 1)) continue;
        switch (opcao) {
            case 1: printf("\n --> Primeira opcao.."); break;
            case 2: printf("\n --> Segunda opcao.."); break;
            case 3: printf("\n --> Terceira opcao.."); break;
            case 4: printf("\n --> Quarta opcao.."); break;
            case 5: printf("\n --> Abandonando.."); break;
        }
    }
    return 0;
}
```

Prof. Rafael Nunes – M.Sc. em Engenharia de Computação pela UFMG

95



A Linguagem C

Funções

Prof. Rafael Nunes – M.Sc. em Engenharia de Computação pela UFMG

96

O que é uma *função*?

O que é uma função?

- ☐ 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**.

Estrutura de uma *função*...

Estrutura Geral de uma Função



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

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;
}
```

Funções

Argumentos

Argumentos de Funções

- ☐ 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**.

Vejamos um **exemplo...**

Argumentos de Funções



```
#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;
}
```

Prof. Rafael Nunes – M.Sc. em Engenharia de Computação pela UFMG

106

Entendendo os *argumentos...*



Prof. Rafael Nunes – M.Sc. em Engenharia de Computação pela UFMG

107

Argumentos de Funções



- 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)`

Prof. Rafael Nunes – M.Sc. em Engenharia de Computação pela UFMG

108



Argumentos de Funções

- 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);
```

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



Argumentos de Funções

- 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**.



Múltiplos Argumentos

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

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

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

Múltiplos Argumentos

- 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 **retorne 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 **deixar de** especificar o valor de retorno
 - ❑ Nesta disciplina, este valor será **sempre deverá ser especificado**.

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



Retornando Valores – Outro 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;
}
```



Exercício de Fixação

1. Implemente uma mini calculadora com as quatro funções básicas:
 - Soma
 - Subtração
 - Divisão
 - Multiplicação (Você irá encontrar nos exercícios apresentados!)
2. Obs.: Você deverá escolher os tipos de dados (int ou float) para as operações solicitadas



Para finalizar...

Faça os exercícios propostos
e não deixe de acessar o
ambiente virtual

Bons estudos e até a nossa
próxima videoaula!

Referências

Waldemar Celes, Renato Cerqueira, José Lucas Rangel,. Introdução a Estruturas de Dados, Editora Campus. (2004)
