



LÓGICA DE PROGRAMAÇÃO E ALGORITMOS

AULA 6



Prof. Sandro de Araújo



CONVERSA INICIAL

Essa aula teve como base os livros: Lógica de programação e estruturas de dados, Fundamentos da Programação de Computadores e Treinamento em Linguagem C. Em caso de dúvidas ou aprofundamento consulte-os em nossa Biblioteca Virtual Pearson.

A aula apresenta a seguinte estrutura de conteúdo:

1. Procedimento;
2. Função;
3. Declaração de uma função;
4. Parâmetros;
5. Passagem de parâmetros.

O objetivo desta aula é conhecer os principais conceitos e aplicações de procedimentos e funções e uma introdução de como declará-los nas construções de algoritmos para resolver problemas computacionais.

TEMA 1 – PROCEDIMENTO

Procedimentos são estruturas que juntam um conjunto de comandos, que são executados no momento em que são chamados. Ex.: O envio de um *e-mail*, impressão de caracteres na tela do usuário, entre outros.

Um procedimento (*procedure*), também conhecido como *sub-rotina*, é o que realiza uma determinada tarefa. Um algoritmo de procedimento é criado da mesma maneira que outro algoritmo qualquer, devendo ser identificado e possuir variáveis, operações e até funções (Ascencio, 2012; Puga; Rissetti, 2016). O procedimento é identificado com o nome <identificador> acompanhado de parênteses () e pode possuir ou não parâmetros (Puga; Rissetti, 2016).

Quando o programa principal chama um procedimento, por meio do seu identificador, o controle do fluxo de execução do programa passa para o procedimento e, no momento em que o procedimento finalizar a tarefa, o controle do fluxo de execução retornará para o programa principal.

Para criar um procedimento em pseudocódigo, utiliza-se a seguinte estrutura:



```
procedimento <nome-de-procedimento> [(<declarações-de-parâmetros>)]  
var  
    // Seção de Declarações Internas  
Inicio  
    // Seção de Comandos  
fimprocedimento
```

É importante ressaltar que, na linguagem de programação C, não existe o chamado *procedimento* propriamente dito. Em vez disso, usa-se apenas a sintaxe de uma função sem retorno. Diferenciar procedimentos e funções na linguagem C é simples: se não tem *retorno*, é um procedimento; se tem, é uma *função*. Assim, a linguagem de programação C baseia-se em não ter ou ter um retorno (Mizrahi, 2008).

Nesse caso, usa-se *void* no tipo de retorno do cabeçalho da função e, se a função não for receber nenhum parâmetro, também colocamos o *void* no local da listagem dos parâmetros.

Para criar um procedimento em linguagem de programação C, utiliza-se a seguinte estrutura:

```
void nome<identificador>( )  
{  
    // Seção de Comandos  
    return; /* retorno de uma função void */  
}
```

Para procedimentos, ou seja, para uma função do tipo *void*, devemos usar o comando **return**; sem nenhum parâmetro, como último comando do bloco.

A seguir veremos um exemplo de procedimento em pseudocódigo e Linguagem C:

Exemplo:

Considere um algoritmo que vai somar dois números usando um procedimento, e vai mostrar o resultado no programa principal.



1.1 Pseudocódigo

```
algoritmo "CalculaSoma"

procedimento SOMA
var
    resultado,a,b:real
inicio
    escreval("ABAIXO A ROTINA DA SOMA")
    escreva("*** Digite o primeiro numero: ")
    leia(a)
    escreva("*** Digite o segundo numero: ")
    leia(b)
    resultado<-a+b
    escreval("")
    escreval("A soma dos dois valores é:",resultado)
    escreval("")
fimprocedimento

inicio //programa principal

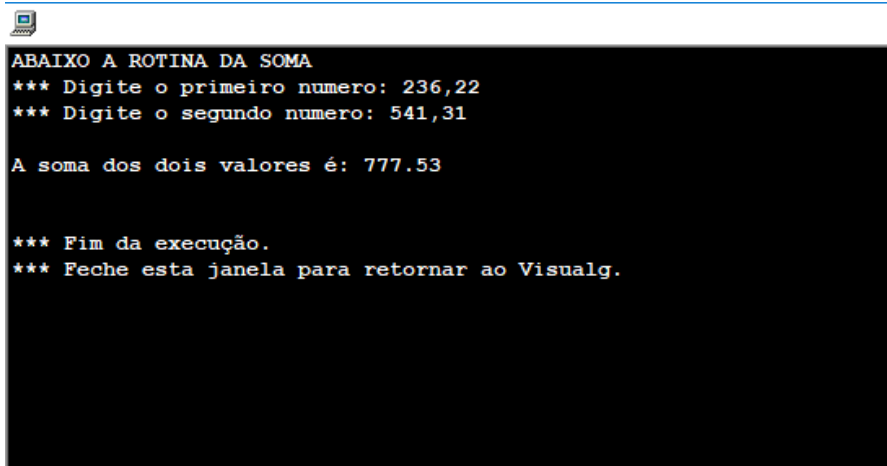
SOMA

finalgoritmo
```

No algoritmo em pseudocódigo acima, temos a chamada do procedimento por meio da identificação SOMA. Ao chamar SOMA, toda rotina dentro do algoritmo será executada.

A Figura 1 mostra a saída do procedimento acima após execução do algoritmo:

Figura 1 – Saída do algoritmo CalculaSoma



```
ABAIXO A ROTINA DA SOMA
*** Digite o primeiro numero: 236,22
*** Digite o segundo numero: 541,31

A soma dos dois valores é: 777.53

*** Fim da execução.
*** Feche esta janela para retornar ao Visualg.
```



Na linguagem de programação C, podemos dividir o algoritmo em dois arquivos. Um arquivo para o programa principal e o outro arquivo para o procedimento, conforme mostrado abaixo.

1.2 Linguagem C

1.2.1 Primeiro arquivo (main.c)

```
#include <stdio.h>

#include <stdlib.h>

int main() //programa principal
{
    SOMA();
    return 0;
}
```

1.2.2 Segundo arquivo (procedimento.c)

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

void SOMA()
{
    float resultado, a, b;

    printf("ABAIXO A ROTINA DA SOMA");
    printf("\n*** Digite o primeiro numero: ");
    scanf("%f", &a);
    printf("*** Digite o segundo numero: ");
    scanf("%f", &b);

    resultado = a + b;

    printf("\n A soma dos dois valores digitados eh: %.2f\n", resultado);

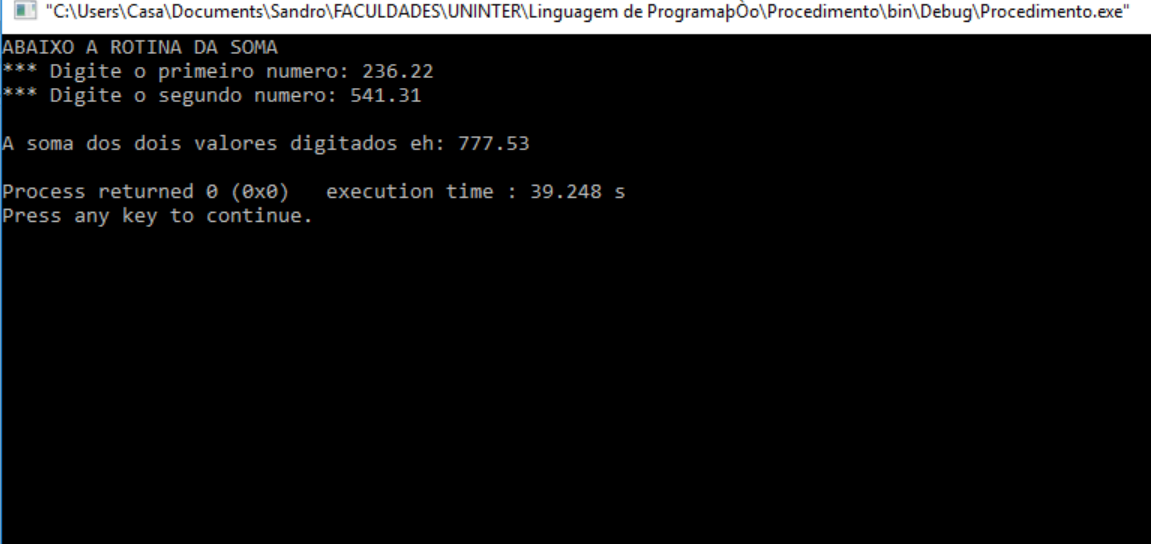
    return;
}
```



No algoritmo em linguagem C acima temos a chamada do procedimento através da identificação SOMA(). Quando o primeiro arquivo (programa principal) chamar o procedimento SOMA(), toda rotina dentro do segundo arquivo será executada.

A Figura 2 mostra a saída do procedimento acima após execução do algoritmo:

Figura 2 – Saída do algoritmo na linguagem C



```
"C:\Users\Casa\Documents\Sandro\FACULDADES\UNINTER\Linguagem de Programação\Procedimento\bin\Debug\Procedimento.exe"
ABAXO A ROTINA DA SOMA
*** Digite o primeiro numero: 236.22
*** Digite o segundo numero: 541.31

A soma dos dois valores digitados eh: 777.53

Process returned 0 (0x0)   execution time : 39.248 s
Press any key to continue.
```

TEMA 2 – FUNÇÃO

A função é um tipo especial de procedimento, também conhecida como *sub-rotina*. Trata-se de um conjunto de instruções construídas para cumprir uma tarefa específica e agrupadas numa unidade.

As grandes tarefas de computação são divididas em tarefas menores, pelas funções, que permitem que outros façam o reuso do código, em vez de partir do zero.

As funções podem frequentemente esconder detalhes de operações de partes do programa que não necessitam conhecê-las, esclarecendo o todo e facilitando mudanças (Mizrahi, 2008).

O uso de funções evita que o programador tenha de reescrever o mesmo código repetidas vezes. Por exemplo, em vários momentos no programa, que está sendo desenvolvido, teremos que calcular o quadrado de um número.

Um programa sem o uso de funções terá que reescrever o bloco de código sempre que precisar realizar um cálculo. Em vez disso, podemos saltar para um bloco que já faça o cálculo do quadrado de um número e voltar para a mesma



posição de execução. Dessa forma, um simples conjunto de instruções do código pode ser executado repetidas vezes no mesmo programa sem a necessidade de reescrever tudo novamente (Puga; Rissetti, 2016).

Para criar uma função em pseudocódigo, utiliza-se a seguinte estrutura:

```
funcao <nome-de-função> [(<declarações-de-parâmetros>)]: <tipo-de-  
dado>  
    // Seção de Declarações Internas  
  
    inicio  
  
        // Seção de Comandos  
  
        retorne valor_de_retorno  
  
    fimfuncao
```

Para criar uma função na linguagem de programação C, utiliza-se a seguinte estrutura:

```
tipo_da_funcao  nome<identificador> (<tipo>1º parâmetro, <tipo>2º  
parâmetro, ...)  
{  
    // Seção de Comandos  
    return valor_de_retorno;  
}
```

A seguir, veremos um exemplo de procedimento em pseudocódigo e Linguagem C:

Exemplo:

Considere um algoritmo que vai ter uma rotina da qual recebe um número do tipo inteiro e calcula o quadrado.



2.1 Pseudocódigo

algoritmo "CalculaQuadrado"

var

n1, a, resultado: inteiro

funcao calcula_quadrado(a : inteiro): inteiro

var retorna: inteiro

inicio

retorna <- a*a

retorne retorna

fimfuncao

inicio

escreval ("Digite um número inteiro: ")

leia (n1)

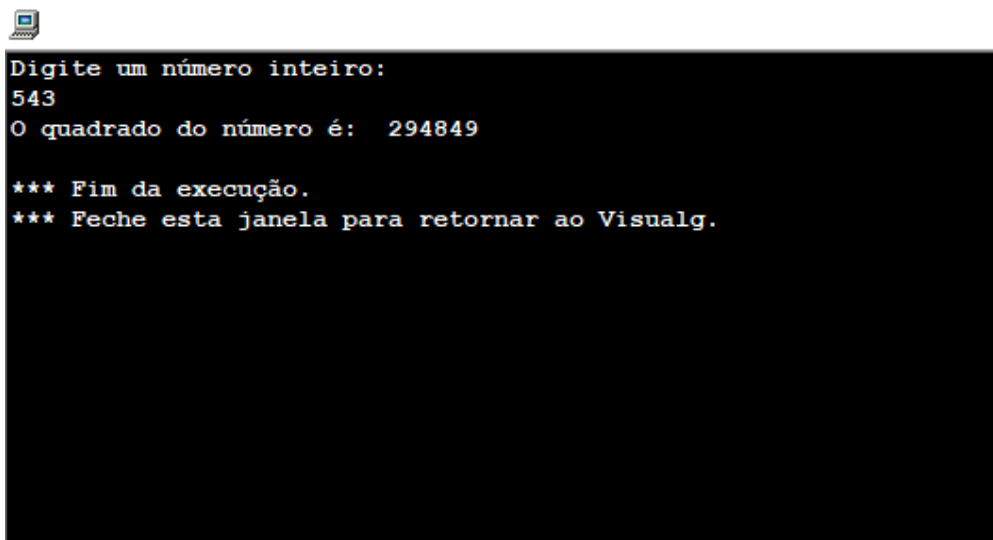
escreval ("O quadrado do número é: ", calcula_quadrado(n1))

fimalgoritmo

No pseudocódigo temos a função **calcula_quadrado()** que recebe a variável **a** do tipo inteiro e após o processamento retorna o resultado que será usado no código.

A Figura 3 mostra a saída da função acima após execução do algoritmo:

Figura 3 – Saída do algoritmo CalculaQuadrado



```
Digite um número inteiro:
543
O quadrado do número é: 294849

*** Fim da execução.
*** Feche esta janela para retornar ao Visualg.
```

Agora veremos a resolução do problema na linguagem de programação C. Conforme mostrado no exemplo abaixo.



2.2 Linguagem C

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

int quadrado(int n1)
{
    int resultado;
    resultado = n1 * n1;
    return resultado;
}

int main()
{
    int numero;

    printf("Digite um numero inteiro: \n");

    scanf("%d", &numero);

    printf("O quadrado do numero eh: %d", quadrado(numero));

    return 0;
}
```

A Figura 4 mostra a saída da função acima, na linguagem C, após execução do algoritmo:

Figura 4 – Saída do algoritmo CalculaQuadrado

Do mesmo modo, na linguagem de programação C, temos a função **calcula_quadrado()**, que recebe a variável **numero** do tipo inteiro e, após o processamento do dado digitado, retorna o seu resultado. Note que nesse



exemplo a função **calcula_quadrado()** teve uma passagem de parâmetro do tipo inteiro armazenada na variável **numero**.

Qualquer sequência de instruções que apareça mais de uma vez no programa é candidata a ser uma função. O código de uma função é agregado ao programa uma única vez e pode ser executado muitas vezes no decorrer do programa.

Esse conceito de poder executar o mesmo bloco de código muitas vezes no decorrer do programa fortalece um princípio de linguagens estruturadas, que é o de dividir um programa em funções para evitar que blocos de código fiquem grandes demais e sejam difíceis de ler e entender. Dividir o código ajuda também na organização do programa, pois permite o reaproveitamento do código que já foi escrito pelo programador em outras partes do programa (Puga; Rissetti, 2016).

TEMA 3 – DECLARAÇÃO DE UMA FUNÇÃO

Quando uma chamada de função é encontrada no momento em que o programa está sendo compilado, o compilador necessita que seja informado corretamente qual o tipo de retorno e os parâmetros da função, para que ele possa manipulá-los (Mizrahi, 2008). Para um melhor entendimento desse conceito, vejamos o exemplo abaixo:

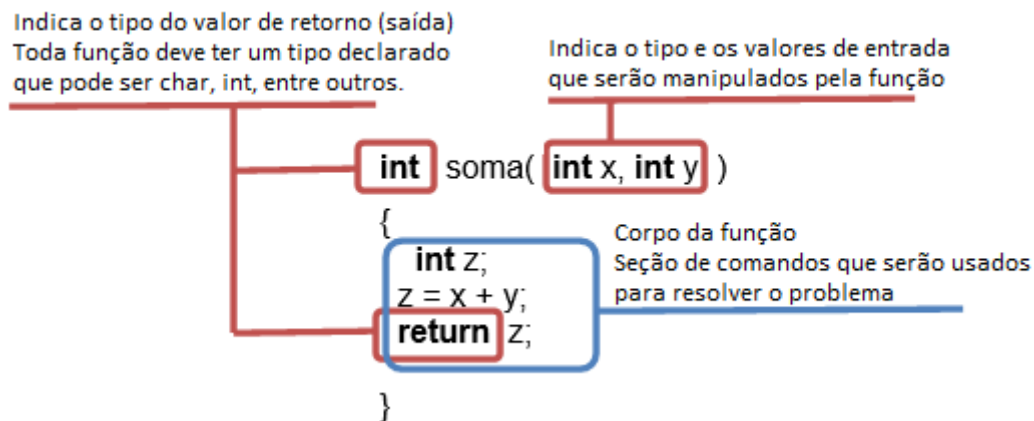
Exemplo de uma função:

```
int soma (int x, int y)
{
    int z;
    z = x + y;
    return z;
}
```

A Figura 5 mostra em partes como se declara uma função.



Figura 5 – Declaração de uma função



Em uma função declara-se o tipo de dado de entrada e o de saída, conforme mostrado na Figura 5. O valor de retorno da função se dá com o comando **return**. Esse comando é sempre o último a ser executado por uma função, finalizado o bloco de instruções.

O tipo *void* representa o “sem retorno”, ou seja, um retorno com conteúdo indeterminado. Conforme exemplo usado na página 5 na função `SOMA()`.

TEMA 4 – PARÂMETROS

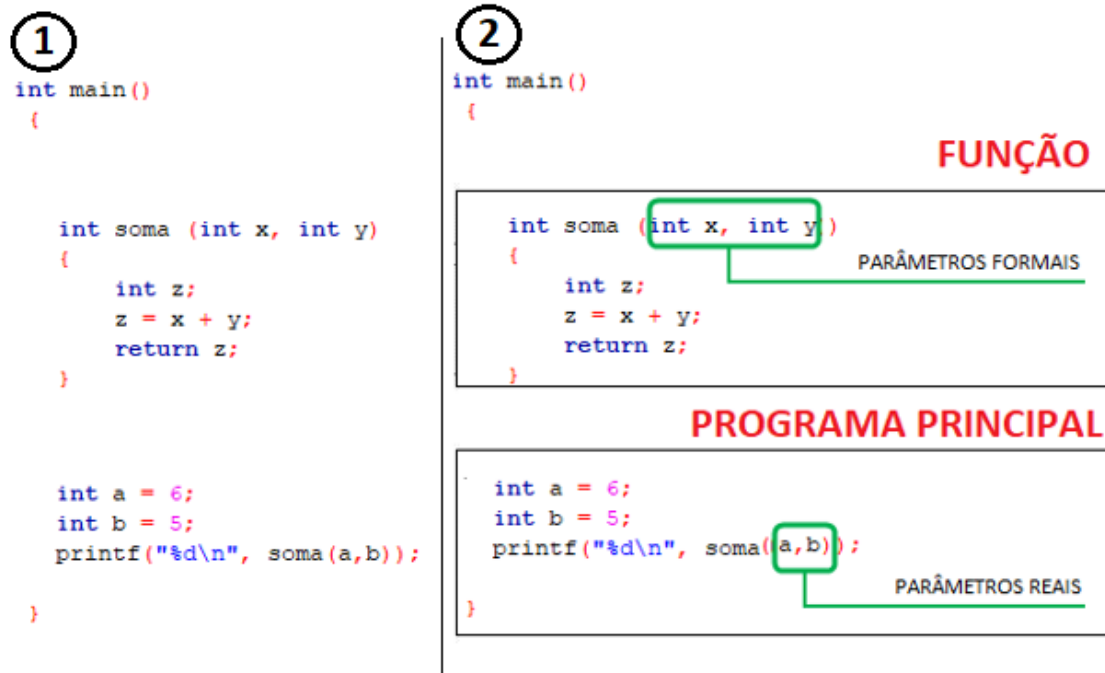
Parâmetros são valores recebidos e/ou retornados por uma função. Esses valores são fornecidos quando uma função é chamada. É comum chamar os parâmetros de *argumentos*, embora este termo esteja associado ao valor de um parâmetro.

Os parâmetros podem ser divididos em duas categorias:

1. Formais: que correspondem aos parâmetros utilizados na descrição da função;
2. Reais: que correspondem aos parâmetros especificados na instrução de chamada.

A Figura 6 mostra um código em linguagem C descrevendo a localidade dos parâmetros formais e reais.

Figura 6 – Parâmetros formais e reais



A Figura 6 divide o mesmo código em duas partes. Na primeira parte, o código está no seu formato original; na segunda parte, destacam-se os parâmetros formais e reais do algoritmo.

Os parâmetros formais só existem para o programa no momento da execução da função. Após sua execução, eles deixam de existir, diferentemente dos parâmetros reais, que podem ser usados tanto na chamada da função quanto em outros momentos no programa principal.

A seguir, um exemplo na linguagem de programação C para demonstrar os parâmetros formais e reais.

```
int main()
{
    int soma (int x, int y)    // Parâmetros Formais
    {
        int z;
        z = x + y;
        return z;
    }

    int a = 6;
    int b = 5;
    int c;

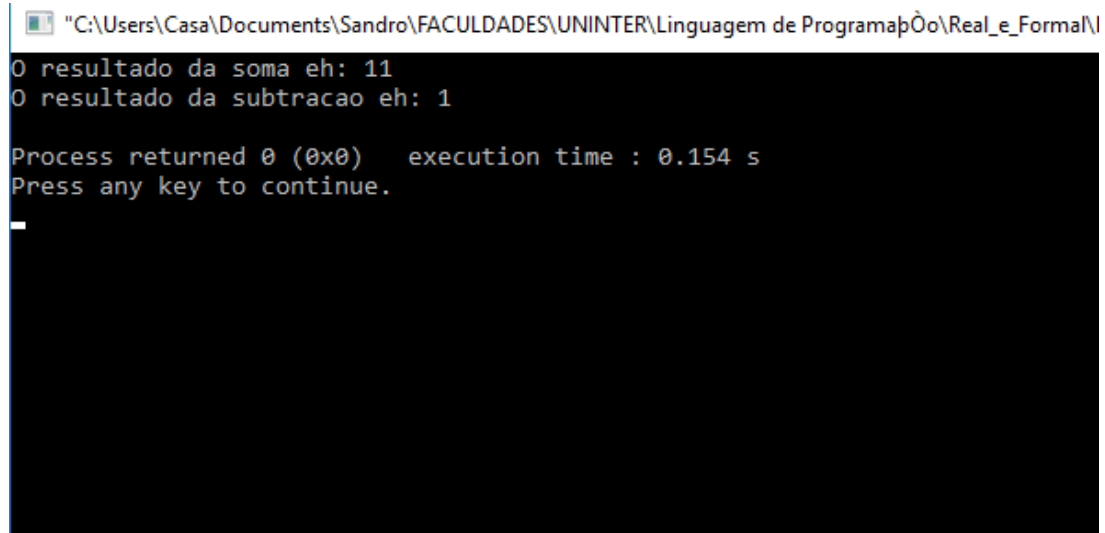
    c = a - b; // Usados em outro momento para subtração
}
```



```
printf("O resultado da soma eh: %d\n", soma(a,b)); // Dentro da função
printf("O resultado da subtracao eh: %d\n", c); // Fora da função
}
```

A Figura 7 mostra a saída após execução do algoritmo:

Figura 7 – Saída do algoritmo



```
"C:\Users\Casa\Documents\Sandro\FACULDADES\UNINTER\Linguagem de Programação\Real_e_Formal\I
O resultado da soma eh: 11
O resultado da subtracao eh: 1

Process returned 0 (0x0)   execution time : 0.154 s
Press any key to continue.
-
```

TEMA 5 – PASSAGEM DE PARÂMETROS

As linguagens de programação permitem que os parâmetros sejam passados para as funções de duas maneiras, conforme mostrado a seguir:

1. Por valor – uma cópia do parâmetro é feita – um valor da expressão é calculado, e o valor resultante é passado para a execução da função;
2. Por referência – o endereço de um parâmetro é passado na chamada da função. Com isso, a função pode modificar a variável diretamente, por exemplo, para a criação de funções que devolvem mais de um valor.

A passagem por valor pode ser usada e alterada dentro da função sem afetar a variável da qual ela foi gerada. Já na passagem por referência, passa-se o endereço onde esse parâmetro está localizado na memória, e é usada quando uma função precisa ser capaz de alterar os valores das variáveis usadas como argumentos (Mizrahi, 2008; Ascencio, 2012).



Exemplo de passagem de parâmetros por valor:

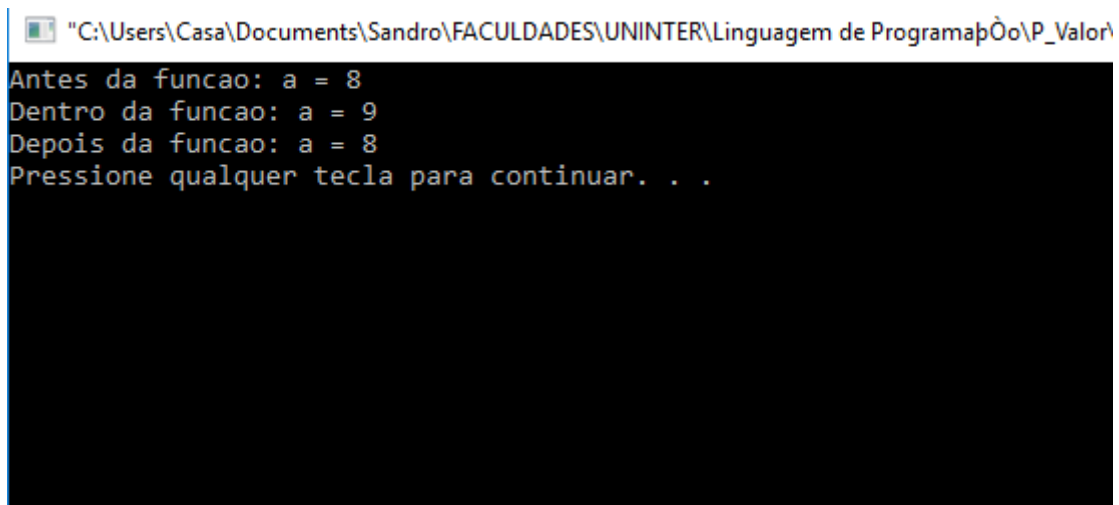
```
#include <stdio.h>
#include <stdlib.h>

void soma_mais_1(int num){ // A função faz uma cópia do dado de "a" em "num".
    num = num + 1;
    printf("Dentro da funcao: a = %d\n", num);
}

int main()
{
    int a = 8;
    printf("Antes da funcao: a = %d\n",a); //Impressão de "a" antes da função.
    soma_mais_1(a); //Chamada da função. A função recebe o parâmetro de "a".
    printf("Depois da funcao: a = %d\n",a); // Impressão de "a" depois da função.
    system("pause");
    return 0;
}
```

A Figura 8 mostra a saída após execução do algoritmo:

Figura 8 – Saída do algoritmo



```
"C:\Users\Casa\Documents\Sandro\FACULDADES\UNINTER\Linguagem de Programação\P_Valor"
Antes da funcao: a = 8
Dentro da funcao: a = 9
Depois da funcao: a = 8
Pressione qualquer tecla para continuar. . .
```

Na Figura 8, vemos que, na primeira linha, encontra-se a impressão do parâmetro 8 antes da execução da função; na segunda linha, o parâmetro foi modificado dentro da função pelo valor 9; e, na terceira linha, volta o dado que ainda está armazenado na memória na impressão do parâmetro depois da execução da função.

A função `scanf()` da linguagem de programação C, que usamos em exemplos anteriores nas nossas aulas, também faz passagem de parâmetros por referência. Veja o exemplo a seguir:



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

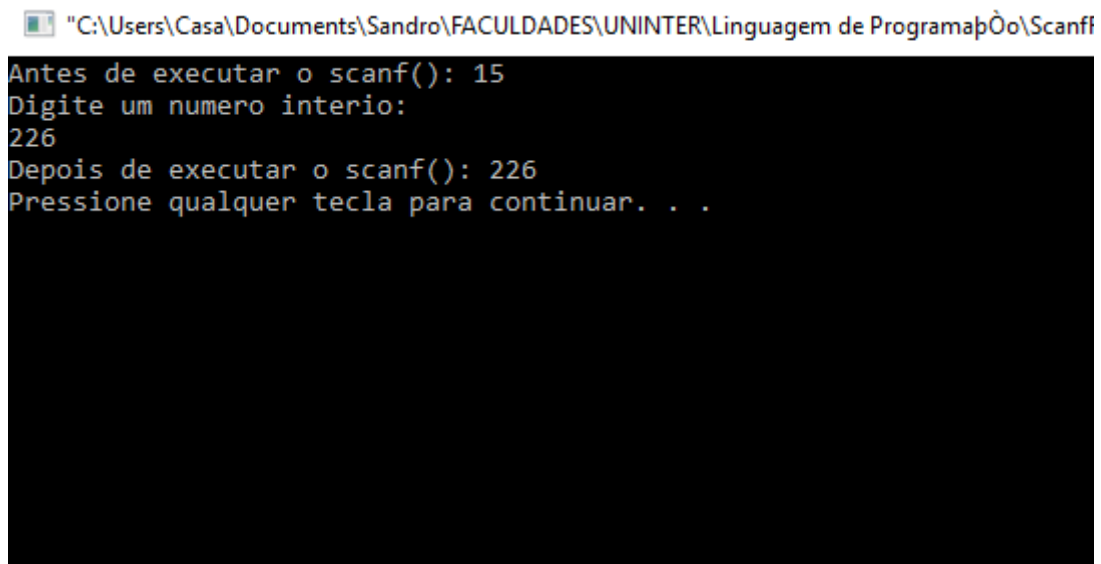
int main()
{
    int num = 15; //Parâmetro guardado na memória.

    printf("Antes de executar o scanf(): %d\n", num); //Parâmetro antes da função ser
    executada.
    printf("Digite um numero inteiro: \n");
    scanf("%d",&num); //Chamada da função scanf( ).
    printf("Depois de executar o scanf(): %d\n", num); //Parâmetro depois da função ser
    executada.

    system("pause");
    return 0;
}
```

A Figura 9 mostra a saída do algoritmo após execução da função scanf():

Figura 9 – Saída do algoritmo



```
"C:\Users\Casa\Documents\Sandro\FACULDADES\UNINTER\Linguagem de Programação\Scanf"
Antes de executar o scanf(): 15
Digite um numero inteiro:
226
Depois de executar o scanf(): 226
Pressione qualquer tecla para continuar. . .
```

Na Figura 9 vemos que na primeira linha aparece a impressão do parâmetro antes da execução da função; já na segunda linha aparece uma mensagem para informativa que pede para o usuário digitar um número inteiro; na terceira linha, é o momento que o usuário digita o número 226, que logo é capturado e armazenado na memória pela função scanf(); e na quarta linha temos a impressão do parâmetro que foi sobrescrito pela função.



Para que a passagem de um parâmetro seja por referência, basta colocar o símbolo “*” antes da sua definição dos parâmetros formais e o operador “&”, na chamada do parâmetro (MIZRAHI, 2008). O uso do asterisco indica que esses parâmetros podem ser modificados dentro da função, ou seja, as alterações dos parâmetros sofridas dentro da função também serão sentidas fora dela, esses efeitos não ocorrem quando os parâmetros são passados por valor (sem o uso do asterisco (*)).

Saiba mais

O operador unário & retorna o endereço na memória de seu operando (Mizrahi, 2008).

Veja a seguir o mesmo exemplo da Figura 7 com a implementação da passagem de parâmetros por referência.:

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

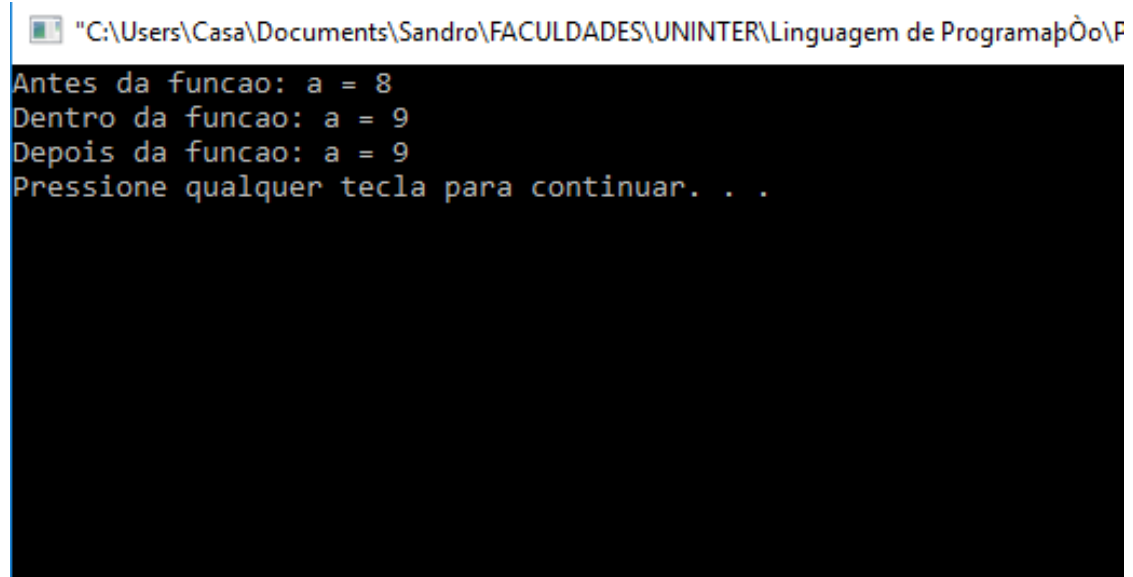
void soma_mais_1(int *num){ //Pega o endereço do Parâmetro "a".
    *num = *num + 1;
    printf("Dentro da funcao: a = %d\n", *num);
}

int main()
{
    int a = 8;
    printf("Antes da funcao: a = %d\n",a); //Impressão de "a" antes da função.
    soma_mais_1(&a); //Chamada da função. A função recebe o endereço de "a".
    printf("Depois da funcao: a = %d\n",a); //Impressão de "a" depois da função.

    system("pause");
    return 0;
}
```

A Figura 10 mostra a saída após execução do algoritmo:

Figura 10 – Saída do algoritmo



```
"C:\Users\Casa\Documents\Sandro\FACULDADES\UNINTER\Linguagem de Programação\F"
Antes da funcao: a = 8
Dentro da funcao: a = 9
Depois da funcao: a = 9
Pressione qualquer tecla para continuar. . .
```

Na Figura 10, vemos que na primeira linha encontra-se a impressão do parâmetro antes da execução da função; na segunda linha, o parâmetro foi modificado dentro da função pelo valor 9; e, após a execução da função, na terceira linha, o dado modificado está armazenado na memória e sua alteração aparece na impressão fora da função.

Portanto, na passagem por valor, o parâmetro formal comporta-se como uma variável local, e as alterações feitas nessa variável não terão efeito sobre o parâmetro real, que pertence ao programa que fez a chamada. Na passagem por referência, o parâmetro formal comporta-se como se fosse uma variável global, em que todas as alterações feitas nesta variável são feitas no parâmetro real, e pode ser alterada pelo subprograma que vai continuar com o valor alterado (Mizrahi, 2008).

FINALIZANDO

Nesta aula aprendemos os principais conceitos e aplicações de funções, procedimentos. Também tivemos uma introdução de como declará-los nas construções de algoritmos.



REFERÊNCIAS

ASCENCIO, A. F. G. **Fundamentos da programação de computadores:** algoritmos, Pascal, C/C++ (padrão ANSI) JAVA. 3. ed. São Paulo: Pearson, 2012.

MIZRAHI, V. V. **Treinamento em linguagem C.** 2. ed. São Paulo: Pearson, 2008.

PUGA, S.; RISSETTI, G. **Lógica de programação e estruturas de dados.** São Paulo: Pearson Educación, 2016.