Funções e Procedimentos

Módulo 6 Aula 0

Linguagem C, o Curso Definitivo WR Kits

Autor: Dr. Eng. Wagner Rambo

A Linguagem C é toda baseada em funções. Tanto é verdade, que até o presente momento estamos programando em C sem criar qualquer função, no entanto, já estamos utilizando várias delas. Todo código C sempre inicia a execução pela função principal (main()). Também já utilizamos inúmeras vezes funções da biblioteca padrão, tal como *printf*, *scanf*, *putchar*. O que torna a Linguagem C ainda mais linda, é o fato de podermos desenvolver nossas próprias funções, de forma rápida e também incrivelmente flexível.

Na fase básica do curso você já estudou os tipos de dados, estes também são relevantes quando falamos de funções. Na presente aula, entenderemos como funciona uma função, qual é o modo correto e profissional de inseri-la em seu código, além de criarmos nossas próprias funções que não retornam valor algum, também conhecidas como procedimentos. Outra grande vantagem das funções, é tornar nossos programas mais modulares.

Para iniciar os estudos em funções, considere o código do Box 1.

```
main()
{
  int i, j;

  for(i=0;i<3;i++)
  {
    for(j=0;j<30;j++)
     putchar('#');

    putchar('\n');

  } /* end for */

  printf("\nMenu principal do programa\n\n");

  for(i=0;i<3;i++)
  {
    for(j=0;j<30;j++)
     putchar('#');

    putchar('\n');

  } /* end for */

  printf("\nMenu tarefas a executar\n\n");

  for(i=0;i<3;i++)
  {
    for(j=0;j<30;j++)
     putchar('#');

    putchar('\n');

  } /* end for */

  printf("\nMenu Help\n\n");
```

```
} /* end main */
```

Box 1 - Apresentação de menus hipotéticos.

O código gera uma apresentação de menus hipotéticos, que apresentam 3 linhas de #s antes de cada título. Estas 3 linhas são construídas a partir de laços *for* e seus códigos sempre se repetem. Neste caso, é mais interessante encapsular o algoritmo que gera estas 3 linhas em uma função específica, tornando seu código muito mais "limpo". Confira no Box 2 o resultado.

```
void menu_lines();          /* protótipo da função */

main()
{

  menu_lines();

  printf("\nMenu principal do programa\n\n");

  menu_lines();

  printf("\nMenu tarefas a executar\n\n");

  menu_lines();

  printf("\nMenu Help\n\n");


} /* end main */


void menu_lines()           /* desenvolvimento da função */
{
  int i, j;

  for(i=0;i<3;i++)
  {
    for(j=0;j<30;j++)
     putchar('#');

    putchar('\n');

  } /* end for */

} /* end menu_lines */
```

Box 2 - Função para gerar as 3 linhas de #s.

Compare os códigos do Box 1 e Box 2, qual lhe parece mais profissional? Sem dúvida o código do Box 2 está muito mais elegante e apresentável, uma vez que utilizamos uma função e encapsulamos o algoritmo para gerar as 3 linhas de #s nela.

Quando a função é chamada, o código é desviado para outra região da memória, onde a função foi desenvolvida. Ao terminar de executar os algoritmos daquela função, o programa continua sua execução na instrução imediatamente após a chamada da função.

Vamos entender agora exatamente como formalizar as nossas funções em C.

A declaração do protótipo de uma função segue a sintaxe do Box 3.

```
tipo_de_dado nome_da_função(parâmetros);
```

Box 3 - Sintaxe para o protótipo de uma função em C.

Uma função em C tem os mesmos tipos de dados que já conhecemos. O tipo de dado determina também qual o tipo que esta função retornará. Quando uma função não retorna nenhum valor, utiliza-se o tipo *void*, que traduzindo para o português significa "vazio". O nome da função segue as mesmas regras de declaração de variáveis e os parâmetros podem ser os mais diversos tipos de dados, que são as entradas da função. Quando uma função não apresenta parâmetros, pode-se deixar os parênteses vazios ou então inserir a palavra *void* para deixar esse fato ainda mais evidente. No Box 4 temos o protótipo de uma função para mover um robô para frente.

```
void move_robo(void);
```

Box 4 - Protótipo de função para mover robô para frente.

Temos portanto uma função do tipo *void* (que não retorna nada), um nome seguindo as regras de declaração e não há parâmetros. Este tipo de função também recebe o nome de procedimento em C, uma vez que consiste em um desvio do fluxo natural do programa para a execução de algoritmos específicos.

Um código em C profissional deve apresentar a organização conforme o Box 5.

```
/* Protótipo das Funções */

main()
{

} /* end main*/

/* Desenvolvimento das Funções */
```

Box 5 - Organização de um código em C profissional.

Antes da função principal deve-se sempre declarar o protótipo de cada função presente em seu código, conforme Box 4, tipo de dado, nome da função e parâmetros seguidos de ponto-e-vírgula. O desenvolvimento das funções se dá após a função principal. A função do Box 4 pode ter o desenvolvimento como vemos no Box 5.

```
void move_robo(void)
{
  /* instruções da função */

} /* end move_robo */
```

Box 6 - Desenvolvimento de uma função.

A chamada das funções sempre ocorrerá dentro da função principal do seu código em C e também é importante destacar que suas funções poderão chamar outras funções. A função do Box 4 poderá ser chamada na função principal, conforme sintaxe do Box 7.

```
main()
{

  move_robo();


} /*end main */
```

Box 7 - Chamada da função move_robo().

Exercício resolvido: criar uma função/procedimento em C que imprima 10 asteriscos em sequência, sempre que o usuário digitar o número 7.

Exercício proposto: desenvolva uma função em C para imprimir na tela a mensagem do Box 8 quando a entrada do usuário for o número 3.

```
++++++++++++++++++
|                |
| N U M E R O 3  |
|                |
++++++++++++++++++
```

Box 8 - Saída para quando digitado o número 3.

Desenvolva uma segunda função que terá a mensagem do Box 9, sempre que o usuário entrar com o número 7.

```
++++++++++++++++++
|                |
| N U M E R O 7  |
|                |
++++++++++++++++++
```

Box 9 - Saída para quando digitado o número 7.

Bibliografia: DAMAS, Luís; Linguagem C, décima edição.

Disponível em: https://amzn.to/3nGdIbN