Algoritmos e Programação Estruturada

Procedimentos e funções

Você sabia que seu material didático é interativo e multimídia? Isso significa que você pode interagir com o conteúdo de diversas formas, a qualquer hora e lugar. Na versão impressa, porém, alguns conteúdos interativos ficam desabilitados. Por essa razão, fique atento: sempre que possível, opte pela versão digital. Bons estudos!

Um software deve ser construído de forma organizada onde cada funcionalidade deve ser colocada em um “local” com uma respectiva identificação, para que o requisitante possa encontrá-la. Uma das técnicas de programação utilizada para construir programas dessa forma é a construção de funções. Assim, nesta webaula vamos ver a criação de funções na linguagem C bem como seu tipo de retorno.

Funções da linguagem C

`printf()e scanf()` são exemplos de funções que fazem parte das bibliotecas da linguagem C.

Na imagem a seguir, o comando na linha 2, `int main()` especifica uma função que chama “`main`” e que irá devolver para quem a requisitou um valor inteiro, nesse caso, zero.

Código Hello World

```
1  #include<stdio.h>
2  int main(){
3      printf("Hello World!");
4      return 0;
5  }
```

Fonte: elaborado pelo autor.

A ideia de criar programas com blocos de funcionalidades vêm de uma técnica de projeto de algoritmos chamada *dividir para conquistar* (MANZANO; MATOS; LOURENÇO, 2015). A ideia é simples, dado um problema, este deve ser dividido em problemas menores, que facilitem a resolução e organização. A técnica consiste em três passos:

Dividir	Conquistar	Combinar
Quebrar um problema em outros subproblemas menores.	Usar uma sequência de instruções separada, para resolver cada subproblema.	Juntar a solução de cada subproblema para alcançar a solução completa do problema original.

Função

Uma função é um trecho de código escrito para solucionar um subproblema. Esses blocos são escritos tanto para dividir a complexidade de um problema maior, quanto para evitar a repetição de códigos. Essa técnica também pode ser chamada de modularização, ou seja, um problema será resolvido em diferentes módulos

Para criar uma função utiliza-se a sintaxe a seguir.

```
<tipo de retorno> <nome> (<parâmetros>){
   <Comandos da função>
   <Retorno> (não obrigatório)
}
```

Em cada declaração da função alguns parâmetros são obrigatórios e outros opcionais. Veja-os a seguir:

`<tipo de retorno>`

Obrigatório. Esse parâmetro indica qual o tipo de valor a função irá retornar. Pode ser um valor inteiro (`int`), decimal (`float` ou `double`), caractere (`char`), etc. Quando a subrotina faz um processamento e não retorna nenhum valor, usa-se o parâmetro void e, nesse caso, é chamado de **procedimento** (MANZANO, 2015).

`<nome>`

Obrigatório. Parâmetro que especifica o nome que identificará a função. É como o nome de uma pessoa, para você convidá-la para sair você precisa “chamá-la pelo nome”. O nome não pode ter acento, nem caractere especial e nem ser nome composto (mesmas regras para nomes de variáveis).

`<parênteses depois do nome>`

Obrigatório. Toda função ou procedimento, sempre terá o nome acompanhado de parênteses. Por exemplo, `main(), printf(), somar()`, etc.

`<parâmetros>`

Opcional. Estudaremos mais a adiante.

`<comandos da função>`

Obrigatório. Só faz sentido criar uma função se ela tiver um conjunto de comandos para realizar.

`<retorno>`

Quando o tipo de retorno for `void` esse parâmetro não precisa ser usado, porém, quando não for `void` é obrigatório. O valor a ser retornado tem que ser compatível com o tipo de retorno, senão o problema dará um erro de compilação em algumas linguagens, em outras retornará um valor errôneo. Na linguagem C, irá ser retornado um valor de acordo com o tipo.

Local da função

Em qual parte do código a função deve ser programada?

Na linguagem C, vamos adotar sempre criar as funções (subrotinas) antes da função `main()`, por uma questão de praticidade e conveniência.

Veja a seguir, um exemplo de programa que utiliza uma função para calcular a soma entre dois números.

Explicação

Outra característica da utilização de funções é que estas “quebram” a linearidade de execução, pois a execução pode “dar saltos” quando uma função é invocada (SOFFNER, 2013).

Para entender melhor como funciona esse mecanismo, veja a seguir uma função que solicita um número para o usuário, calcula o quadrado desse número e retorna o resultado.

Explicação

O uso de funções com ponteiros

Uma função pode retornar um número inteiro, um real e um caractere, assim como também pode retornar um vetor. Para isso, devemos utilizar ponteiros (ou apontador). A única forma de retornar um vetor é por meio de um ponteiro, pois não é possível criar funções como por exemplo, `int[10] calcular()`, onde `int[10]` quer dizer que a função retorna um vetor com 10 posições. (MANZANO, 2015).

A seguir veja um exemplo de uso desse recurso através de uma função, que cria um vetor de dez posições e os preenche com valores aleatórios, imprime os valores, e posteriormente passa esse vetor para “quem” chamar a função.

Explicação

Nesta webaula vimos como criar funções que após um determinado conjunto de instruções retorna um valor para “quem” chamou a subrotina. Esse conhecimento permitirá criar programas mais organizados e também evitar repetição de códigos.