Introdução ao uso de funções em C: organização do código

Nesta seção apresentamos as questões essenciais para o uso de *funções* (ou *procedimentos* - funções sem valores devolvidos): usar funções para **não** repetir códigos e declarar as funções de modo bem organizado.
Experimente alterar os exemplos e criar outros: grave-os, compile-os e os *rode*. **Para aprender programação é preciso programar!**

1. Sintaxe de funções em *C*

Uma *função matemática* deve ter o *domínio* e o *contra-domínio*, uma função em *Python* segue o mesmo padrão, mas com uma sintaxe bem específica. O *domínio* é definido pelo tipo de dado dos *parâmetros* da *função* e o *contra-domínio* é definido pelo tipo de dado de devolução, por meio do comando `return`. Além disso, em *C*, na declaração de função o seu "tipo" (*contra-domínio*) deve ser explicitado. A sintaxe de uma *função* em *C* está ilustrada no código 1.

```
tipo nome_funcao (lista_parametros_com_tipos) {
  // o codigo (atuando sobre a "lista_parametros_com_tipos" e
devolvendo algo do "tipo"
  // geralmente deve-se terminar a funcao com comando "return
'expressao_simples'
  }
```

Cód. 1. Estrutura de uma função em C.

Um exemplo em devemos utilizar o conceito de *função* é se precisarmos computar a combinação de *N, k-a-k*, ou seja, considerando *N* elementos diferentes, quantos são os conjuntos distintos com exatamente *k* elementos que é possível formar (vide aqui uma explicação, com exemplos das loterias). Isso está ilustrado no código 2, lançando mão de definir uma função fatorial.

C

```
1  int fat (int n) { // define funcao com 1 parametro
2    int ft = 1, i=1;
3    while (i < n) {
4      i = i + 1; // poderiamos tambem usar "i++"
5      ft = ft * i; // poderiamos tambem usar "ft *= i"
6      }
7    return ft;
8    } // chave indica final da funcao 'fat(...)'
9  ...
10 // supondo existir neste contexto variaveis: N e k
11 printf("Combinacao = %f\n", (float)fat(N) / (fat(k) *
   fat(N-k)));
```

Cód. 2. Exemplo de código para computar combinação de n, k-a-k, com uma função para fatorial.

2. Não repetir código: use função/procedimento

Uma das grandes utilidades do emprego de *funções* é simplificar o código, evitando-se que um mesmo código aparece mais de uma vez em seu programa. O código 1 ilustra isso, se não fosse utilizado funções, seria necessário usar mais duas cópias do código para fatorial (uma para computar *fat(N)*, uma para *fat(k)* e uma última para *fat(N-k)*). Isso resultará um código "menor" (menos linhas), além de ser mais fácil mantê-lo funcionando.

Outra vantagem não tão óbvia em código pequenos é reduzir erros e facilitar a manutenção do código.
Uma vez que implementou-se uma função (procedimento) que realiza determinada tarefa, sem erros, pode-se sempre utilizar o mesmo código (apenas copiando-o para o novo programa. Isso reduz a chance de novos erros.
Além disso, por estar bem organizado, em um ponto, a manutenção do código também é simplificada (basta examinar e eventualmente alterar essa função).

Ao declarar funções em *C*, deve-se organizar o código declarando as funções **por ordem de chamada**, ou seja, se seu código usualmente invoca primeiro `f1(.)`, depois `f2(.)` e por último `f3(.)`, então deveria declarar nessa ordem (primeiro `f1(.)` e assim por diante).
Entretanto, existem códigos maiores e mais complexos, nesses casos tente **organizar por assunto**, deixando próximas as funções que tratam do mesmo *assunto*.
Se existir algum caso de *ciclo*, uma função `f1(.)` que invoca `f2(.)` e `f2(.)` que invoca `f1(.)`, será necessário declarar os protótipos das funções. Por exemplo, se ambas as funções devolverem inteiro e a primeira tiver um único parâmetro inteiro e a segunda um inteiro e um flutuante, então deve-se declarar no início os seguintes *protótipos*: `int f1(int);` `int f2(int,float);`.

Outro item que pode ser necessário em códigos mais complexos é declarar uma função dentro do contexto de outra função, como poderia ser o caso de fazer uma função para *combinação*, tendo internamente sua função *fatorial*. Embora nesse caso seria prefarível deixar o *fatorial* disponível para todos os demais códigos (ou seja, como no código 2). Para examinar mais detalhes sobre o *aninhamento* de funções clique aqui.

Para ilustrar as duas vantagens do uso de funções apresentamos um código violando as duas regras, declaração desordenada das funções e repetição de código (que deveria estar em funções). Assim, o código 3 deve ser entendido como **não** se deve programar!

Não codifique com esses 2 erros!

```
1  // Exemplo de codigo ruim:
2  // 1. Codigo repetido que deveria estar em uma
3  funcao/procedimento
4  // 2. Funcoes misturadas com a linha de execucao
5
6  #include <stdio.h>
7
8  void modos (void) {
```

```
 9    // 1. NAO duplique codigo,
10    int N = 3, k = 2, modo, fat, fatN, fatk, i;
11    modo = 2;
12    if (modo==1) { // fat(N)
13      fat = 1; // inicio: codigo que deveria estar como
14  funcao
15      for (i=2; i<N+1; i++)
16        fat *= i;
17      printf("Fatorial de %d = %d\n", N, fat);
18      }
19    else if (modo==2) { // fat(N)/fat(k);
20      fatN = 1; // inicio: codigo duplicado (1)
21      for (i=2; i<N+1; i++)
22        fatN *= i;
23      fatk = 1; // inicio: codigo duplicado (2)
24      for (i=2; i<k+1; i++)
25        fatk *= i;
26      printf("Arranjo %d, %d-a-%d = %d  (%d,%d)\n", N, k, k,
27  (fatN/fatk), fatN, fatk);
28      }
29    }
30
31  void main () {
32    int a = 5, b = 6;
33    printf("%d + %d = %d\n", a,b, soma(a,b));
34    modos();
35    }
36
37  // 2. NAO misture execucao principal com funcoes! Primeiro
38  declare "soma(.)"
    //    Dependendo das configuracoes de seu compilador, nao
    compilaria ou resultaria na advertencia dos 3 comentarios
    abaixo
    int soma (int a, int b) { //  ... warning: implicit
    declaration of function ‘soma’ [-Wimplicit-function-
    declaration]
      return a+b; //              printf("%d + %d = %d", (a,b,
    soma(a,b)));
      } //
    ^~~~
```

Cód. 3. Exemplo de código para computar combinação de n, k-a-k, com uma função para fatorial, mas ordem incoveniente.

Leônidas de Oliveira Brandão
http://line.ime.usp.br

Alterações