Introdução ao uso de funções em C: variáveis locais, globais e aninhamento de funções

Nesta seção apresento um conceito mais "sutil" de programação, para entendê-lo é essencial que você copie os códigos apresentados, "cole" em seu editor preferido e os teste até que esteja tudo claro. Experimente alterar os exemplos e criar outros: grave-os, compile-os e os *rode*. **Para aprender programação é preciso programar!**

Se você precisar de um *algoritmo* com um objetivo muito bem definido (e.g. um *algoritmo* para encontrar o menor valor dentro uma *lista* dada) é melhor **encapsulá-lo** em um bloco de modo a poder invocá-lo sempre que precisar desse *algoritmo*. Geralmente todas as linguagens de programação possibilita esse *encapsulamento* e isso é feito na forma de uma função (ou de um procedimento). A grande vantagem dessa abordagem é deixar o seu código melhor organizado, além de permitir que a função seja utilizada em outros trechos do código sem a necessidade de codificá-la novamente!

Para ampliar as vantagens do uso de *funções* as *linguagens de programação* permitem: o *aninhamento* de funções (funções dentro de funções); a declaração de variáveis locais.

Atenção, lembre que no **C** padrão, **não** é possível declarar variável em qualquer lugar, uma variável **só** pode ser declarada após o *ínicio de um bloco* (após abertura de chaves). Assim, pode-se declarar novas variáveis como primeiras instruções na declaração de uma função, como as variáveis `i` e `fat` na função `fatorial` no *exemplo 1*.

Sobre o aninhamento de funções na linguagem *C*

Em várias linguagens de programação, *C* em particular, é possível declarar função dentro de função (**aninhar**). Por exemplo, dentro de uma função de nome *funcao2*, pode-se declarar outra função de nome *funcao3*. O resultado é que, dentro da primeira função, pode-se invocar a segunda, mas em nenhum outro local do código seria possível invocar esssa função *funcao3*.

Mas vale destacar que esse recurso é efetivamente útil apenas quando a função interna (*funcao3*) só fizer sentido dentro da função que a contém (*funcao2*). Pois em caso contrário, você poderia invocar a função interna em outros contextos (o que não seria possível se declarar *funcao3* dentro de *funcao2*!). Um exemplo dessa situação indesejada é ilustrado no *exemplo 1*, a função *fatorial* deveria ser declarada fora da função *combinacao* (pois ela tem potencial de ser invocada em outros locais).

Um exemplo simples de função contendo outra função, poderia ser o caso do cálculo da combinação de *n*, tomados *k*-a-*k*: $C_{n,k} = n!/(k!(n-k)!)$. Desse modo podemos implementar uma função de nome `combinacao`, com parâmetros `n` e `k`, para computar $C_{n,k}$. Por outro lado, como é necessário computar 3 fatoriais, o código ficaria mais claro (e mais compacto) se definirmos uma função específica para cálculo do fatorial, que denominaremos por `fatorial`.

Além disso, se estamos seguros de não precisar da função `fatorial` em nenhum outro local, podemos implementá-la dentro de `combinacao`. Vide exemplo 1.

Porém, como anteriormente comentado, como a função *fatorial* tem potencial de aplicação mais amplo, ela ficaria melhor se declarada **fora** da função *combinacao* (vide a crítica a esse organização de código logo após o exemplo 1).

Exemplo 1. Declaração aninhada de funções.

```
int combinacao (int n, int k) { // combinacao de n, k-a-k
  int fatorial (int n) { // fatorial de n (supondo n natural)
    int i, fat = 1;
    for (i=2; i<=n; i++)
      fat *= i;
    return fat;
    }
  return fatorial(n) / (fatorial(k)*fatorial(n-k));
  }

void main (void) {
  int  n = 5, k = 1;
  printf("Combinacao %d, %d-a-%d = %d\n", n, k, k, combinacao(n,k));
  n = 5; k = 2;
  printf("Combinacao %d, %d-a-%d = %d\n", n, k, k, combinacao(n,k));
  n = 5; k = 3;
  printf("Combinacao %d, %d-a-%d = %d\n", n, k, k, combinacao(n,k));
  // Problema de declarar 'fatorial()' dentro de 'combinacao()': a linha
abaixo daria erro!
  // Erro: exemplo_funcao_aninhada.c:(.text+0xd3): undefined reference to
fatorial'
  // printf("fat(0)=%d, fat(1)=%d, fat(2)=%d, fat(3)=%d", fatorial(0),
fatorial(1), fatorial(2), fatorial(3));
  }
```

Note que o código acima apresenta a linha comentada `printf("fat(0)=%d, fat(1)=%d, fat(2)=%d, fat(3)=%d", fatorial(0), fatorial(1), fatorial(2), fatorial(3));`. A razão é que ela resultaria erro, pois a função *fatorial* foi declarada dentro da função *combinacao* e portanto só pode ser usando (invocada) dentro dessa segunda!

Uma vez que o uso do cálculo de fatorial é muito comum e pensando que o código acima seria parte de um maior, então provavelmente o desenho de código usado não é bom, pois implicaria em precisarmos definir outra função *fatorial*, essa fora do contexto da função *combinacao*. Nesse sentido, poderia ser melhor usar o código (pensando que ele seja o início de um sistema maior) do exemplo a seguir.

Exemplo 2. Declaração de funções sem aninhamento.

```
int fatorial (int n) { // fatorial de n (supondo n natural)
  int i, fat = 1;
  for (i=2; i<=n; i++)
    fat *= i;
  return fat;
  }
```

```
int combinacao (int n, int k) { // combinacao de n, k-a-k
  return fatorial(n) / (fatorial(k)*fatorial(n-k));
  }

void main (void) {
  int n = 5, k = 1;
  printf("Combinacao %d, %d-a-%d = %d\n", n, k, k, combinacao(n,k));
  n = 5; k = 2;
  printf("Combinacao %d, %d-a-%d = %d\n", n, k, k, combinacao(n,k));
  n = 5; k = 3;
  printf("Combinacao %d, %d-a-%d = %d\n", n, k, k, combinacao(n,k));
  // Note que agora pode-se invocar 'fatorial()' dentro da 'main'
  printf("fat(0)=%d, fat(1)=%d, fat(2)=%d, fat(3)=%d", fatorial(0),
fatorial(1), fatorial(2), fatorial(3));
  }
```

Variáveis globais e locais

Uma vez mais, deve-se destacar que no **C** padrão, **não** é possível declarar variável em qualquer lugar de seu código, uma variável **só** pode ser declarada após uma abertura de chaves.

Ao declarar uma lista de variáveis como primeiras instruções de uma função, isso significa que essas variáveis são "conhecidas" apenas dentro dessa função. Cada uma dessas variáveis será uma **variável local** à função. Por exemplo, no *exemplo 2* acima, a variável *fat* é conhecida apenas dentro da função *fatorial*, portanto seu uso deve-se restringir a esse contexto.

Por outro lado, pode-se usar o mesmo nome para variáveis que estão em contextos distintos. Por exemplo, no código do *exemplo 1*, se fosse necessário poderíamos usar o nome de variável local *i* dentro da função *combinacao* e não haveria qualquer colisão com a variável *i* de *fatorial*.

Entretanto é necessário um critério que elimine *ambiguidades*, isto é, que permita identificar unicamente cada uma das variáries. Para isso usa-se o **princípio da proximidade** (ou da **localidade**), quer dizer, ao usar uma variável considera-se como sua declarção aquela mais próxima. Por isso, que poderiamos usar *i* tanto em *combinacao*, quanto em *fatorial*,

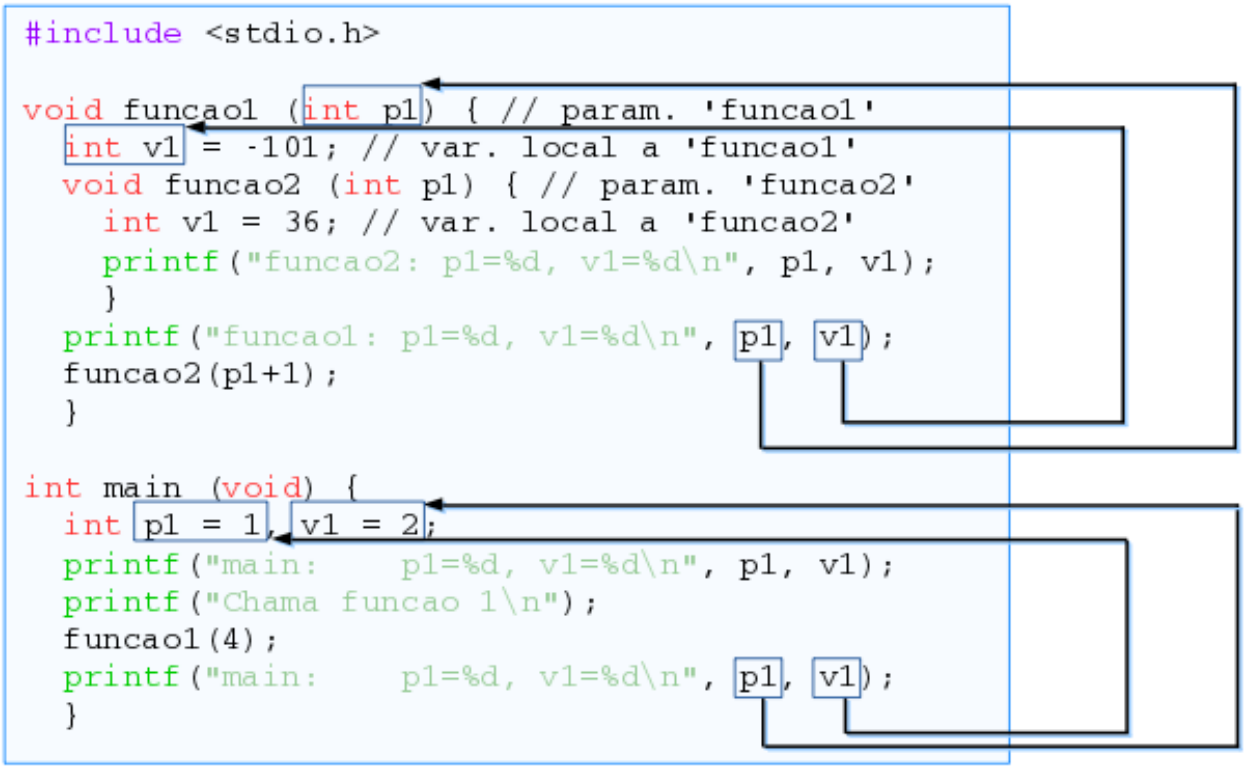

Fig. 1. Código ilustrando escopo de variáveis: `v1` e `v2` correspondem à declaração mais próxima.

main: p1=1, v1=2 Chama funcao 1 funcao1: p1=4, v1=-101 funcao2: p1=5, v1=36 main: p1=1, v1=2*Cód. 1. Saídas para o código da figura 1.*

Já uma variável **global** pode ser acessada em qualquer ponto do código. Mas por essa mesma razão deve evitada, sendo necessária apenas para casos excepcionais, sempre com um nome significativo, que evite qualquer confusão.

Em *C*, toda variável declarar fora do contexto de qualquer função, é uma variável **global**, ou seja, uma variável que pode ser acessada em qualquer ponto do código (em qualquer função). A ideia de variável global é ilustrado no próximo exemplo.

Exemplo 3. Declaração de variável global.

```
char var_global1[] = "essa variavel e' uma variavel global";

int funcao () {
  printf("funcao1: var_global1 = %s\n", var_global1);
  }

void main (void) {
  printf("main   : var_global1 = %s\n", var_global1); // uso da global
  funcao();
  }
```

Aninhando funções e variáveis locais e globais

Deve-se notar que é possível aninhar tantas funções quantas forem necessárias e o mesmo para variáveis locais. O exemplo abaixo ilustra esses aninhamentos e o princípio da proximidade. Examine-o com atenção, copie-o em seu computador, utilizando um compilador *C* e altere seu código, compile-o e rode-o até que esteja certo de ter assimilado os conceitos.

Exemplo 4. Declaração de variáveis locais, usando globais e aninhamento de funções.

```
#include <stdio.h>

char glob1[] = "glob1: variavel global, conhecida em qualquer local";

void funcao1 () {
  char loc1[] = "loc1: conhecida apenas dentro da funcao 'funcao1'";
  char glob1[] = "funcao1.glob1: declarada dentro de 'funcao1' => uma variavel local `a funcao
'funcao1'";
  printf("funcao1: loc1 = %s\n", loc1); // note que: pode haver m
  printf("funcao1: glob1 = %s\n\n", glob1);
  }

void funcao2 (param1) {
  char loc1[] = "loc1: conhecida apenas dentro da funcao 'funcao2 - NAO sou a 'funcao1.loc1'!'";
  char loc2[] = "loc2: conhecida apenas dentro da funcao 'funcao2'";
  void funcao3 () { // essa funcao so' e' conhecida dentro de 'funcao2'!
    char loc1[] = "loc1: conhecida apenas dentro da funcao 'funcao3' - NAO sou a 'funcao1.loc1' e
nem a 'funcao2.loc1'!";
    char glob1[] = "funcao3.glob1: declarada dentro de 'funcao3' => var. local `a funcao
'funcao3' - NAO e' global 'glob1' e nem 'funcao1.glob1'!";
    printf("funcao3: loc1 = %s\n", loc1); // note que: e' local com outra com esse nome;
"principio do mais proximo" em acao!
    printf("funcao3: loc2 = %s\n", loc2); // note que: e' local `a funcao 'funcao2' - variavel
declarada fora de funcao3!
    printf("funcao3: glob1 = %s\n", glob1);
    }
  printf("funcao2: param1 = %s\n", param1);
  printf("funcao2: loc1 = %s\n", loc1); // note que: e' local com outra com esse nome; "principio
do mais proximo" em acao!
  printf("funcao2: loc2 = %s\n", loc2); // note que: e' local
  printf("funcao2: glob1 = %s\n", glob1);
  printf("funcao2: chama funcao3\n\n");
  funcao3();
  printf("funcao2: chama funcao1\n\n");
  funcao1();
  }

void main () {
  char loc1[] = "loc1: conhecida apenas dentro da funcao 'main'";
  printf("main   : loc1 = %s\n", loc1); // note que: e' local com outra com esse nome; "principio
do mais proximo" em acao!

  printf("main   : glob1 = %s\n", glob1);

  printf("main   : chama funcao1\n\n");
  funcao1();
  printf("main   : chama funcao2\n\n");
  funcao2("parametro efetivo passado para a funcao 'funcao2'");

  // Atencao: como a funcao 'funcao3' foi declarada dentro da funcao 'funcao2', entao ela so'
pode se invocada em 'funcao2'
  // Por exemplo, a linha abaixo resultaria no erro:
sobre_variaveis_global_local.c:(.text+0x4d2): undefined reference to funcao3'
  // funcao3();
  }
```

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

Alterações