Linguagem C - Funções

Sidney Batista Filho, M.Sc., SCPJ2P
sidneybf@acm.org

2002-2004

Agenda

- Funções
- Referências

Funções
Definição de função

- Um programa em C é composto por várias funções.

DEFINIÇÃO DE UMA FUNÇÃO:

```
tipo nome(tipo nome1, ..., tipo nomeN) {
   declarações
   instruções
}
```

- A comunicação entre as funções é feita, principalmente, pelos parâmetros e pelos valores retornados

O tipo `void` pode ser usado para declarar funções que não retornam valor algum.

Funções
Exemplo 1

```
//Protótipo (ou declaração) de função
float calcularMenor(float n1, float n2);
//Função principal
int main() {
   float menor;
   float numero1, numero2;

   printf("Digite 2 numeros: ");
   scanf("%f %f", &numero1, &numero2);
   menor = calcularMenor(numero1, numero2);
   printf("menor = %.2f\n", menor);
   return 0;
}
```

Funções
Exemplo 1

```
//Definição de função
float calcularMenor(float n1, float n2) {
   float resultado;

   resultado = (n1 < n2) ? n1 : n2;

   return resultado;
}
```


Funções
Conceitos

- **Protótipo ou declaração de função**

```
float calcularMenor(float n1, float n2);
```

- **Definição de função**

```
float calcularMenor(float n1, float n2)
{ ... }
```

- **Parâmetros reais X formais**

```
//parâmetros reais - chamada da função
menor = calcularMenor(numero1, numero2);
//parâmetros formais - definição da função
float calcularMenor(float n1, float n2)
```

Funções
Exemplo 1 - usando variável global

```
//Variável global
float menor;

//Função principal
int main() {
   float numero1, numero2;

   printf("Digite 2 numeros: ");
   scanf("%f %f", &numero1, &numero2);
   calcularMenor(numero1, numero2);
   printf("menor = %.2f\n", menor);
   return 0;
}
```


Funções
Exemplo 1 - usando variável global

```
//Definição de função
void calcularMenor(float n1, float n2) {
   menor = (n1 < n2) ? n1 : n2;
}
```

- **Variável local: é declarada dentro de um bloco. Ela só existe durante a execução do bloco.**
- **Variável global: é declarada fora de qualquer função. Pode-se fazer acesso à variável global de qualquer função.**

•Os parâmetros formais são tratados como variáveis locais.

Funções
Exercício 1

Fazer um programa para ler uma linha do teclado e converter cada letra minúscula da linha para maiúscula.

Funções
Exercício 1 - Solução

```
#define TAM 80
char paraMaiuscula(char c); //protótipo
main() {
   char linha[TAM], novaLinha[TAM];
   int  i;

   printf("Digite ate' %d letras: ", TAM - 1);
   gets(linha);
   for(i=0;(i<TAM)&&(linha[i]!='\0'); i++) {
      novaLinha[i] = paraMaiuscula(linha[i]);//chamada
   }
   novaLinha[i] = '\0';
   printf("A linha convertida:\n");
   puts(novaLinha);
}
```

Funções
Exercício 1 - Solução (cont.)

```
//Definição de função
char paraMaiuscula(char c) {
  if (('a' <= c) && (c <= 'z'))
    c = c - 'a' + 'A';
  return c;
}
```


Funções
Exercício 2

Fazer um programa para calcular a combinação de um numero n k a k. Os números n e k devem ser obtidos do teclado.

$$c = \frac{(n!)}{(k! \cdot (n-k)!)}$$


```
int fat(int a);
int combinacao(int n, int k);

main() {
        int c, n, k;

        printf("Digite o valor de n e de k: ");
        scanf("%d%d", &n, &k);
        c = combinacao(n, k);
        printf("A combinacao de %d, %d a %d "
               "e' igual a %d\n", n, k, k, c);
}

int combinacao(int n, int k) {
        int c;

        c = fat(n) / (fat(k) * fat(n-k));
        return c;
}

int fat(int a) {
        int i, f = 1;

        for(i = 1; i <= a; i++)
                f *=  i;
        return f;
}
```

Funções
Chamadas por valor e por referência

- Muitas linguagens de programação (Pascal e Object Pascal, por exemplo) possuem 2 maneiras de chamar funções:
  - Chamada por valor e
  - Chamada por referência.
- Na linguagem C, só existe a chamada por valor. No entanto é possível simular a chamada por referência.

Funções
Chamadas por valor e por referência

- Chamada por valor (ou passagem de parâmetro por valor): é feita uma cópia do valor dos argumentos e a mesma é passada para a função chamada. As modificações na cópia não afetam o valor original de uma variável na função que realizou a chamada.

Funções
Chamadas por valor e por referência

- Chamada por referência (ou passagem de parâmetro por referência): a função chamadora permite que a função chamada modifique o valor original da variável na função que realizou a chamada.

Funções
Chamadas por valor e por referência

```
void func(int a);

main() {
   int num = 3;

   func(num);
   printf("num = %d\n",
            num);
}

void func(int a) {
   a = 10;
}
```

Funções
Chamadas por valor e por referência

```
program Teste;
var num : integer;
procedure func (var a:integer);
begin
   a := 10;
end;

begin
   num := 3;
   func(num);
   writeln('num = ', num);
end.
```


Funções
Função troca - versão errada!

```
main() {
   int a = 5, b = 10;

   troca(a, b); //chamada por valor
   printf("a=%d b=%d\n", a, b);
}

void troca(int x, int y) {
   int temp;

   temp = x; x = y; y = temp;
}
```

Funções
Função troca - versão correta!

```
main() {
   int a = 5, b = 10;
   troca(&a, &b); //simula uma
                  //chamada por referência
   printf("a=%d b=%d\n", a, b);
}

void troca(int *x, int *y) {
   int temp;
   temp = *x; *x = *y; *y = temp;
}
```


Funções - Array unidimensional como argumento

- Declaração do array:

```
int vetor[TAM];
```

- Chamada da função:

```
imprimirVetor(vetor, TAM);
```

- definição da função ("por referência"):

```
void imprimirVetor(int v[], int tam)
  {...}
```

Funções - Array unidimensional como argumento

```
#define TAM 5

main() {
   int vetor[TAM] = {1, 2, 3, 4, 5};

   imprimirVetor(vetor, TAM);
   alterarVetor(vetor, TAM, 8);
   imprimirVetor(vetor, TAM);
}
```


Funções - Array unidimensional como argumento

```
void imprimirVetor(int v[], int max) {
   int i;

   printf("Vetor = [ ");
   for (i = 0; i < max; i++) {
      printf("%d ", v[i]);
   }
   printf("]\n");
}
```

Funções - Array unidimensional como argumento

```
void alterarVetor(int v[], int max,
                  int novoValor) {
   int i;

   for (i = 0; i < max; i++) {
      v[i] = novoValor;
   }
}
```


Saída:

```
Vetor = [ 1 2 3 4 5 ]

Vetor = [ 8 8 8 8 8 ]
```

Funções - Array multidimensional como argumento

- **Declaração do array:**

```
int matriz[MAXLIN][MAXCOL];
```

- **Chamada da função:**

```
imprimirMatriz(matriz,MAXLIN,MAXCOL);
```

- **definição da função ("por referência"):**

```
void imprimirMatriz(int m[][MAXCOL],
  int lin, int col) {...}
```

Funções - Array multidimensional como argumento

```
main() {
   int matriz[MAXLIN][MAXCOL] = {{1, 2, 3},
                                 {4, 5, 6},
                                  {7, 8, 9}
                                  };

   imprimirMatriz(matriz, MAXLIN, MAXCOL);
   alterarMatriz(matriz, MAXLIN, MAXCOL, 8);
   imprimirMatriz(matriz, MAXLIN, MAXCOL);
}
```


Funções - Array multidimensional como argumento

```
void imprimirMatriz(int m[][MAXCOL],
                    int lin, int col) {
   int i, j;

   printf("Matriz = \n[ ");
   for (i = 0; i < lin; i++) {
      printf("\t");
      for (j = 0; j < col; j++) {
         printf("%d ", m[i][j]);
      }printf("\n");
   }printf("]\n");
}
```

Funções - Array multidimensional como argumento

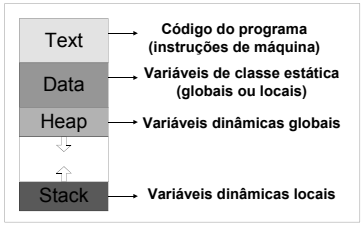
```
void alterarMatriz(int m[][MAXCOL], int lin,
                   int col, int novoValor) {
   int i, j;

   for (i = 0; i < lin; i++) {
      for (j = 0; j < col; j++) {
        m[i][j] = novoValor;
      }
   }
}
```


Saída:

```
Matriz =
[       1 2 3
        4 5 6
        7 8 9
]
Matriz =
[       8 8 8
        8 8 8
        8 8 8
]
```

Regiões de um programa



Funções
Pilha de chamadas de funções

- **Pilha** (stack) é uma estrutura de dados na qual a inserção e a remoção de itens (ou entradas) é feita no mesmo lugar, chamado de topo da pilha.
- Cada chamada de função cria uma nova entrada na **pilha de chamadas de funções** com informações sobre a mesma.
- Política LIFO (last in, first out).

Funções
Pilha de chamadas de funções

```
void funcaoA();
void funcaoB();
void funcaoC();

main() {
   funcaoA();
   printf("main\n");
}

void funcaoA() {
   funcaoB();
   printf("funcaoA\n");
}
```

Funções
Pilha de chamadas de funções

```
void funcaoB() {
   funcaoC();
   printf("funcaoB\n");
}

void funcaoC() {
   printf("funcaoC 1\n");
   printf("funcaoC 2\n");
}
```


Saída:

```
funcaoC 1
funcaoC 2
funcaoB
funcaoA
main
```

Funções - Recursão

- Ocorre quando uma função chama a si mesma, direta ou indiretamente (quando chama várias outras funções que, em determinado ponto, chamam a primeira função novamente).
- A essência da recursão é: Para se obter a resposta de um grande problema, um método geral é usado para reduzir o grande problema em um ou mais problemas de tamanho menor.

Funções - Recursão
Exemplo: Fatorial

- Definição informal

$$n! = n \times (n-1) \times ... \times 1$$

Exemplo:

$$3! = 3 \times 2 \times 1$$

Funções - Recursão
Exemplo: Fatorial

- Definição formal

n! é igual a: 1, se n = 0, ou

$$n \times (n-1)!$$, se n > 0.

Exemplo:

$$3! = 3 \times 2!$$
$$3! = 3 \times (2 \times 1!)$$
$$3! = 3 \times (2 \times (1 \times 0!))$$
$$3! = 3 \times (2 \times (1 \times 1))$$

Funções - Recursão

- Partes da recursão:
  - Caso básico, que é processado sem recursão
  - Um método geral para reduzir um caso particular em um ou mais casos básicos, usando recursão.

Funções - Recursão
Exemplo Fatorial

```
long fatorial(long n) {
  int resultado;
  if (n == 0)
    resultado = 1;
  else
    resultado = n * fatorial(n-1);
  return resultado;
}
```

Funções - Recursão
Exercício

- Calcular o n-ésimo termo da série de Fibonacci usando uma função recursiva.
- Mostrar, através de algum diagrama (um grafo, por exemplo), a execução do cálculo do 4o. termo da série de Fibonacci.

Série de Fibonacci:

0, 1, 1, 2, 3, 5, 8, 13, 21, ...

Funções
Recursão - Exercício

```
long fibo(long n) {
  long resultado;
  if(n == 1) {
    resultado = 0;
  } else if (n == 2){
    resultado = 1;
  } else {
    resultado = fibo(n - 1)
                + fibo(n - 2);
  return resultado;
}
```

Referências

- Kernighan, B.W.; Ritchie, D.M. The C Programming Language. New Jersey: Prentice-Hall, Inc. Second Edition, 1988.
- Deitel, H.M.; Deitel, P.J. Como Programar em C. Rio de Janeiro: Editora LTC. Segunda Edição, 1999.
- http://equipe.nce.ufrj.br/adriano/c/apostila

Referências

- Robbins, Kay A. e Robbins, Steven. Practical Unix Programming. New Jersey: Prentice-Hall, Inc. 1996.
- Kruse, Robert *et.al.* Data Structures & Program Design in C. New Jersey: Prentice-Hall, Inc. 1997.