Linguagem C
Função

Prof. Edwar Saliba Júnior
Fevereiro de 2011

Conceitos

- As técnicas de programação dizem que, sempre que possível, evite códigos extensos, separando o mesmo em funções, visando um fácil entendimento e uma manutenção facilitada;
- De acordo com a técnica, devem-se agrupar códigos correlatos em uma função;
- Uma outra utilização de função é quando um trecho de código será utilizado muitas vezes no programa. Deve-se colocar este trecho em uma função e sempre que for preciso chamar a função;
- A Linguagem C possibilita criar funções, sendo possível passar parâmetros para elas e retornar valores tanto no nome da função como em algum parâmetro passado;
- Enfim, lembre-se que: tudo em C é função.

Função e Protótipo (Assinatura da Função)

- Para ser usada, uma função deve estar previamente definida, isto é, deve-se indicar para o compilador qual o nome da função e quais são os parâmetros esperados;
- Uma maneira simples de se resolver isto, é a colocação da função antes de seu uso;
- Quando se têm sistemas grandes, não é recomendável ter um único arquivo fonte, pois a manutenção seria impraticável;
- Neste caso, é possível ter uma função definida em um programa fonte e seu uso em outro programa fonte. Para resolver este problema a Linguagem C criou uma definição chamada de **protótipo** (assinatura da função);
- No protótipo de uma função é definido somente o necessário para o compilador não acusar erros. A definição do protótipo geralmente é colocada dentro de arquivos *header* (.h) e incluída dentro dos programas fontes;
- No protótipo somente são informados o nome da função, o seu tipo de retorno e o tipo de cada parâmetro esperado.

Tipos de Funções

- Funções pré-definidas pela linguagem:
  - Funções da biblioteca;
  - Definidas nos arquivos da biblioteca da linguagem;
- Funções definidas pelo programador:
  - Escritas pelos programadores nos programas.

Por quê criar funções?

- **Para reduzir a complexidade de um programa:**
  - Elaborar a solução em partes pequenas e bem definidas;
  - Uma tarefa complexa é dividida em funções mais simples;
- **Para evitar a repetição de código ao longo do programa:**
  - Diminuir o tamanho do código;
  - Menos erros, menor custo de manutenção, menor tempo de programação;
- **Reutilização de código:**
  - Código já testado, sem erros => custo menor de programação e maior confiabilidade.

Características das Funções

- **Uma função deve realizar uma única tarefa bem definida;**
- **Toda função tem um nome único. De preferência bem significativo:**
  - **Serve para que a função seja invocada (chamada);**
- **Uma função pode ser invocada a partir de outras funções:**
  - **Por exemplo, a partir de *main*( );**
- **Uma função deve ser programada de forma a fazer exatamente o que se espera dela, sem efeitos colaterais.**

Características das Funções

- **O código da função deve ser independente do programa e deve ser tão genérico quanto possível:**
  - **Para que possa ser utilizado em outros programas;**
- **Uma função pode receber parâmetros de execução, para se adaptar a situações distintas e ser genérica;**
- **Uma função pode retornar um valor como resultado de seu trabalho:**
  - **comando *return*;**
- **Uma função é constituída por instruções em C. De acordo com a sintaxe da linguagem:**
  - **Em C não se pode definir funções dentro de funções, portanto, todas as funções estão no mesmo nível.**

Definição

- Sintaxe:

```
tipo_retorno nome_função(tipo_parâmetro_01 nome_parâmetro_01,
tipo_parâmetro_02 nome_parâmetro_02, ...){

   bloco de comandos

   return(valor);

}
```

- Para se definir uma função deve-se indicar o tipo do retorno da função, seu nome e os parâmetros da mesma;
- Uma função pode ou não retornar um valor. Se uma função não retorna nenhum valor seu `tipo_retorno` deve ser definido como `void`. Os parâmetros devem ser definidos, um por um, indicando o seu tipo e nome separado por vírgula;
- Quando uma função tem o `tipo_retorno` definido como `void`, então o uso do comando `return` no final da função torna-se desnecessário.

Exemplo – Cálculo de Potência

- Faça um programa que imprima na tela, o resultado dos seguintes cálculos 2^9, 3^5 e 7^7.

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int i, total1 = 1, total2 = 1, total3 = 1;

    for(i = 0; i < 9; i++)
        total1 = total1 * 2;

    for(i = 0; i < 5; i++)
        total2 = total2 * 3;

    for(i = 0; i < 7; i++)
        total3 = total3 * 7;

    printf("\nO resultado da potência de 2^9 = %i", total1);
    printf("\nO resultado da potência de 3^5 = %i", total2);
    printf("\nO resultado da potência de 7^7 = %i", total3);

    return 0;
}
```

O mesmo código repetido três vezes. Péssima prática de programação!

Função para Cálculo de Potência

- Vamos resolver novamente, o exemplo anterior. Desta vez usaremos função.

```c
#include <stdio.h>

int potencia(int base, int exp){
    int i, total = 1;

    for(i = 0; i < exp; i++)
        total = total * base;

    return(total);
}

int main()
{
    int total1 = 1, total2 = 1, total3 = 1;

    total1 = potencia(2,9);
    total2 = potencia(3,5);
    total3 = potencia(7,7);

    printf("\nO resultado da potência de 2^9 = %i", total1);
    printf("\nO resultado da potência de 3^5 = %i", total2);
    printf("\nO resultado da potência de 7^7 = %i", total3);

    return 0;
}
```

Exemplo – Função e Protótipo

- Poderíamos ter definido a função “potencia”, depois da função “main”.
- Mas para isto, teríamos que ter definido o protótipo da função “potencia”, para não termos um erro de compilação.

```c
#include <stdio.h>

int potencia(int base, int exp);

int main()
{
    int total1 = 1, total2 = 1, total3 = 1;

    total1 = potencia(2,9);
    total2 = potencia(3,5);
    total3 = potencia(7,7);

    printf("\nO resultado da potência de 2^9 = %i", total1);
    printf("\nO resultado da potência de 3^5 = %i", total2);
    printf("\nO resultado da potência de 7^7 = %i", total3);

    return 0;
}

int potencia(int base, int exp){
    int i, total = 1;

    for(i = 0; i < exp; i++)
        total = total * base;

    return(total);
}
```

Função com Tipo de Retorno *void*

- Uma função pode não ter que retornar valor algum;
- Para este tipo de função, define-se o tipo de retorno como *void*, e neste caso não se usa o comando "*return*" no final da função;
- Em algumas linguagens de programação, este tipo de função é conhecida como "procedimento".

```c
#include <stdio.h>

void imprimeLinha(){
    int i;

    for (i=1; i<=46; i++)
        putchar('*');
    putchar('\n');
}

int main()
{
    imprimeLinha();
    printf("Bem-vindo ao maravilhoso mundo da Linguagem C!\n");
    imprimeLinha();
    return 0;
}
```

```
FuncaoVoid
**********************************************
Bem-vindo ao maravilhoso mundo da Linguagem C!
**********************************************

Process returned 0 (0x0)   execution time : 0.007 s
Press ENTER to continue.
```

Passagem de Parâmetros para Funções

- Por valor:
  - As variáveis criadas como parâmetros da função, se portam como variáveis locais à função, ou seja, elas armazenaram em si, os valores a elas passados;
- Por referência:
  - **Não existe em linguagem C**. É possível fazer uma **simulação** de passagem por referência, em linguagem C, através do uso de ponteiros.

Passagem de Parâmetros para Funções

- Por valor e Por referência (**Simulação**)

```c
#include <stdio.h>

float soma(float a, float b){
    return(a + b);
}

int main()
{
    float val1 = 0, val2 = 0, total = 0;

    printf("Digite o primeiro valor: ");
    scanf("%f",&val1);
    printf("Digite o segundo valor: ");
    scanf("%f",&val2);

    total = soma(val1,val2);

    printf("O total da soma é: %f",total);

    return 0;
}
```

```c
#include <stdio.h>

void soma(float a, float b, float *tot){
    *tot = a + b;
}

int main()
{
    float val1 = 0, val2 = 0, total = 0;

    printf("Digite o primeiro valor: ");
    scanf("%f",&val1);
    printf("Digite o segundo valor: ");
    scanf("%f",&val2);

    soma(val1,val2, &total);

    printf("O total da soma é: %f",total);

    return 0;
}
```

Escopo de Variáveis

- Entende-se como escopo de variáveis, a área onde o valor e o nome dela tem significado no código-fonte.
- Pode-se ter dois tipos de variáveis na Linguagem C:
  - Global - quando a variável é definida fora de qualquer função. Esta variável pode ser usada em qualquer função e o significado dela abrange todo o programa fonte, a partir de sua declaração.
  - Locais - são definidas dentro de funções e o seu significado é válido somente dentro da função que foi declarada. Assim, pode-se ter duas variáveis com o mesmo nome em funções diferentes.

Exemplo de Escopo de Variáveis

- Global e Local

```c
#include <stdio.h>

int valor = 0;

void alteraValor(){
    valor = 456;
}

int main()
{
    printf("\nValor: %i", valor);
    valor = 123;
    printf("\nValor: %i", valor);
    alteraValor();
    printf("\nValor: %i", valor);
    printf("\n");

    return 0;
}
```

```
VariavelGlobal

Valor: 0
Valor: 123
Valor: 456

Process returned 0 (0x0)   execution time : 0.009 s
Press ENTER to continue.
```

```c
#include <stdio.h>

void alteraValor(){
    int valor = 456;
    printf("\nValor na função: %i", valor);
}

int main()
{
    int valor = 123;
    printf("\nValor na função main - 1: %i", valor);
    alteraValor();
    printf("\nValor na função main - 2: %i", valor);
    printf("\n");

    return 0;
}
```

```
VariaveisLocais

Valor na função main - 1: 123
Valor na função: 456
Valor na função main - 2: 123

Process returned 0 (0x0)   execution time : 0.009 s
Press ENTER to continue.
```

Bibliografia

- LAUREANO, Marcos. **Programação em C para ambiente Linux**. Disponível em: <http://br-c.org/doku.php>. Acesso em: 06 fev. 2011.
- MURTA, Cristina Duarte. *Slides* da disciplina de Programação de Computadores I. CEFET-MG, 2010.
- SENNE, Edson Luiz França. **Primeiro Curso de Programação em C**. 2. ed. Florianópolis: Visual Books, 2006.