INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE GOIÁS – CAMPUS GOIÂNIA – GO.
ENGENHARIA DE CONTROLE E AUTOMAÇÃO
ESTRUTURAS DE DADOS

Prof. MSc. Raphael Gomes (raphael@ifg.edu.br)

Módulos em C

Um **módulo** ou **função** agrupa um conjunto de comandos e associa a ele um nome. O uso deste nome é uma chamada da função. Após sua execução, programa volta ao ponto do programa situado imediatamente após a chamada. A volta ao programa que chamou a função é chamada de retorno.

A chamada de uma função pode passar informações (argumentos) para o processamento da função.

- Argumentos = lista de expressões.
- Lista pode ser vazia
- Lista aparece entre parênteses após o nome da função
- Ex.

```
int Soma (int x, int y) {
}
```

No seu retorno, uma função pode retornar resultados ao programa que a chamou

```
return (resultados);
```

- O valor da variável local resultados é passado de volta como o valor da função
- Valores de qualquer tipo podem ser retornados

1.1 – Porque usar funções ?

- Para permitir o reaproveitamento de código já construído(por você ou por outros programadores);
- Para evitar que um trecho de código que seja repetido várias vezes dentro de um mesmo programa;
- Para permitir a alteração de um trecho de código de uma forma mais rápida. Com o uso de uma função é preciso alterar apenas dentro da função que se deseja;
- Para que os blocos do programa não fiquem grandes demais e, por consequência, mais difíceis de entender;
- Para facilitar a leitura do programa-fonte de uma forma mais fácil;
- Para separar o programa em partes (blocos) que possam ser logicamente compreendidos de forma isolada.

1.2 – Primeiro Exemplo

Em primeiro lugar, imaginemos que você necessite várias vezes em seu programa imprimir a mensagem “Pressione a tecla ENTER” e esperar que o usuário tecle ENTER, caso o usuário tecle algo diferente o programa deve imitir um BEEP.

Você pode fazer um laço de WHILE sempre que isto fosse necessário.

Uma alternativa é criar uma função. Com o uso de funções, este processo de repetição fica simplificado. Observe o exemplo a seguir.

```
#include <conio.h>
#include <dos.h>
#include <stdio.h>

void EsperaEnter()  // Definição da função "EsperaEnter"
{
    int tecla;
    printf("Pressione ENTER\n");
    do {
        tecla = getch();
```

```
        if (tecla !=13) // Se nao for ENTER
        {
            sound(700); // Ativa a emissão de um BEEP
            delay(10);  // Mantém a emissão do som por 10 ms
            nosound();  // Para de emitir o som
        }
    } while(tecla != 13); // 13 e' o codigo ASCII do ENTER
}

void main() {
    EsperaEnter();      // Chamada da função definida antes
    // ...........
    EsperaEnter();      // Chamada da função definida antes
    // ...........
    EsperaEnter();      // Chamada da função definida antes
}
```

1.3 – Formato Geral de uma Função em C

```
tipo_da_funcao  NomeDaFuncao (Lista_de_Parametros) {
    // corpo da função
}
```

A `Lista_de_Parametros`, também é chamada de `Lista_de_Argumentos`, é opcional.

1.4 – Parâmetros

A fim de tornar mais amplo o uso de uma função, a linguagem C permite o uso de parâmetros. Este parâmetros possibilitam que se definida sobre quais dados a função deve operar. A função `sound(freq)`, por exemplo, recebe como parâmetro a frequência do som a ser gerado, permitindo que se defina seu comportamento a partir deste valor.

Para definir os parâmetros de uma função o programador deve explicitá-los como se estive declarando uma variável, entre os parênteses do cabeçalho da função. Caso precise declarar mais de um parâmetro, basta separá-los por vírgulas. No exemplo a seguir temos a função SOMA que possui dois parâmetros, sendo o primeiro um float e o segundo um int.

```
void SOMA(float a, int b)  // basta separar por vírgulas
{
    float result;     // a declaração de variáveis é igual ao que
                      // se faz na função main
    result = a+b;
    printf("A soma de %6.3f com %d é %6.3f\n, a,b,Result);
}
```

Os parâmetros são passados para uma função de acordo com a sua posição. Ou seja, o primeiro parâmetro atual(da chamada) define o valor o primeiro parâmetro formal (na definição da função, o segundo parâmetro atual define o valor do segundo parâmetro formal e assim por diante. Os nomes dos parâmetros na chamada não tem relação com os nomes dos parâmetros na definição da função.

No código a seguir, por exemplo, a função SOMA é chamada recebendo como parâmetros as variáveis “b” e “a”, nesta ordem.

```
#include <stdio.h>

void SOMA(float a, int b)  // basta separar os parâmetros por vírgulas
{
    float result;     // a declaração de variáveis é igual ao que
                      // se faz na função main
    result = a+b;
    printf("A soma de %d com %6.3f é %6.3f\n, a,b,Result);
}
```

```
void main() {
    int a;
    float b;
    a = 10;
    b = 12.3;
    SOMA(b,a);  // Chamada da função SOMA(12.3,10)
}
```

O resultado do programa é a impressão da seguinte mensagem: A soma de 12.300 com 10 é 22.300

1.5 – Localização das Funções no Fonte

A princípio podemos tomar com regra a seguinte afirmativa toda função deve ser declarada antes de ser usada.

A declaração de uma função em linguagem C não é exatamente o que fizemos até agora. O que estamos fazendo é a definição da função antes de seu uso. Na definição da função está implícita a declaração.

Alguns programadores preferem que o início do programa seja a primeira parte de seu programa. Para isto a linguagem C permite que se declare uma função, antes de defini-la. Esta declaração é feita através do protótipo da função. O protótipo da função nada mais é do que o trecho de código que especifica o nome e os parâmetros da função.

No exemplo a seguir a função SOMA é prototipada antes de ser usada e assim pôde ser chamada antes de ser definida.

```
#include <stdio.h>

void SOMA(float a, int b);  // Protótipo da função SOMA

void main() {
    SOMA(16.7,15);  // Chamada da função SOMA antes de sua definição,
}                   // mas após sua prototipação

void SOMA(float a, int b)  // Definição da função SOMA
{
    float result;     // a declaração de variáveis é igual ao que
                      // se faz na função main
    result = a+b;
    printf("A soma de %d com %6.3f é %6.3f\n, a,b,Result);
}
```

Atenção: existem compiladores mais simplificados ou antigos que não obrigam a declaração da função antes de seu uso !!! Muito cuidado com eles !! Veja no item a seguir.

1.6 – Verificação dos Tipos dos Parâmetros

A princípio, dados usados parâmetros atuais(aqueles da chamada da função) devem ser dos mesmos tipos dos parâmetros formais. Se isto não ocorrer, mas a declaração da função vier antes de seu uso, os compiladores C modernos se encarregam de converter automaticamente os tipos como se estivesemos usando um CAST.

Entretanto, tenha muito cuidado se as três condições a seguir se verificarem:

- se você estiver usando um compilador que não exige a declaração antes da função antes de seu uso;
- se você usar uma função antes de tê-la prototipado (declarado);
- se os parâmetros formais e reais não forem exatamente do mesmo tipo

Se as três condições se verificarem o resultado da execução da função é imprevisível !

No caso de você declarar corretamente a função antes de usá-la, a conversão de tipos é feita como no caso de variáveis. No exemplo a seguir a função SOMA é chamada com os parâmetros reais dos tipos int e float, nesta ordem. Como na declaração da função o primeiro par6ametro é float e o segundo é int, é feita uma conversão de int para float no caso do primeiro parâmetro e de float para int no caso do segundo

parâmetro.

```
#include <stdio.h>

void SOMA(float a, int b);  // Protótipo da função SOMA

void main() {
    float f;
    f = 20.7
    SOMA(16,f);
}
```

Neste exemplo o primeiro parâmetro é convertido para 16.0 e o segundo para 20. O que ocorre, de fato, é que a chamada da função é feita como se mesma fosse substituída por:

```
SOMA((float)16,(int)f);
```

1.7 – Escopo de Variáveis

Por escopo de uma variável entende-se o bloco de código onde esta variável é válida. Com base nisto, temos as seguintes afirmações:

As variáveis valem no bloco que são definidas;

- as variáveis definidas dentro de uma função recebem o nome de variáveis locais;
- os parâmetros formais de uma função valem também somente dentro da função;
- uma variável definida dentro de uma função não é acessível em outras funções, MESMO ESTAS VARIÁVEIS TENHAM NOME IDÊNTICOS.

No trecho de código a seguir tem-se um exemplo com funções e variáveis com nomes iguais.

```
#include <stdio.h>
#include <conio.h>

void FUNC1() {
    int B;
    B = -100;
    printf("Valor de B dentro da função FUNC1: %d\n", B);
}

void FUNC2() {
    int B;
    B = -200;
    printf("Valor de B dentro da função FUNC2: %d\n", B);
}

void main() {
    int B;
    clrscr();
    B = 10;
    printf("Valor de B: %d\n", B);
    B = 20;
    FUNC1();
    printf("Valor de B: %d\n", B);
    B = 30;
    FUNC2();
    printf("Valor de B: %d\n", B);
    getch();
}
```

SAÍDA

```
 Valor de B: 10
Valor de B dentro da função FUNC1: -100
Valor de B: 20
Valor de B dentro da função FUNC2: -200
Valor de B: 30
```

Referências

- http://www.inf.pucrs.br/~pinho/LaproI/Funcoes/AulaDeFuncoes.htm