Funções na linguagem C

Prof. Marcelo Costa, MSc
marcelo.nascimento@uva.edu.br

Agenda

- Funções

Introdução às Funções

- Uma função é um bloco de código de programa que pode ser usado diversas vezes em sua execução. O uso de funções permite que o programa fique mais legível, mais bem estruturado. Um programa em C consiste, no fundo, de várias funções colocadas juntas.

```
tipo_de_retorno nome_da_função (lista_de_argumentos)
{
        código_da_função
}
```

Exemplo

```
#include <stdio.h>
 int mensagem () /* Funcao simples: so imprime Ola! */
{
    printf ("Ola! ");
    return(0);
}
 int main ()
{
    mensagem();
    printf ("Eu estou vivo!\n");
    return(0);
}
```

Main() e outras funções

- A diferença fundamental entre main e as demais funções do problema é que main é uma função especial, cujo diferencial é o fato de ser a primeira função a ser executada em um programa.

Argumentos

- Argumentos são as entradas que a função recebe. É através dos argumentos que passamos parâmetros para a função. Já vimos funções com argumentos. As funções printf() e scanf() são funções que recebem argumentos.

```
#include <stdio.h>

int square (int x) /* Calcula o quadrado de x */

{
    printf ("O quadrado e %d",(x*x));
    return(0);

}

int main ()

{
    int num;
    printf ("Entre com um numero: ");
    scanf ("%d",&num);
    printf ("\n\n");
    square(num);
    return(0);

}
```

Detalhamento do programa

- Na definição de square() dizemos que a função receberá um argumento inteiro x.
- Quando fazemos a chamada à função, o inteiro num é passado como argumento
- Temos que satisfazer aos requisitos da função quanto ao tipo e à quantidade de argumentos quando a chamamos.
- Não é importante o nome da variável que se passa como argumento, ou seja, a variável num, ao ser passada como argumento para square() é copiada para a variável x.

Função de mais de uma variável

- Os argumentos são separados por vírgula e que deve-se explicitar o tipo de cada um dos argumentos, um a um.
- Os argumentos passados para a função não necessitam ser todos variáveis porque mesmo sendo constantes serão copiados para a variável de entrada da função.

Exemplo

```
#include <stdio.h>

 int mult (float a, float b,float c) /* Multiplica 3  numeros */

{
      printf ("%f",a*b*c);
      return(0);

}


 int main ()

{
      float x,y;
      x=23.5;
      y=12.9;
      mult (x,y,3.87);
      return(0);

}
```

Retornando dois floats

```
#include <stdio.h>

float prod (float x,float y)

{

    return (x*y);

}

int main ()

{
    float saida;
    saida=prod (45.2,0.0067);
    printf ("A saida e: %f\n",saida);
    return(0);

}
```

Exercício

- Escreva uma função que some dois inteiros e retorne o valor da soma.

Recursividade

Prof. Marcelo Costa, MSc
marcelo.costa@uva.edu.br

Funções Recursivas

- Na linguagem C, assim como em muitas outras linguagens de programação, uma função pode chamar a si própria. Uma função assim é chamada função recursiva.
- A primeira coisa a se providenciar é um critério de parada. Este vai determinar quando a função deverá parar de chamar a si mesma. Isto impede que a função se chame infinitas vezes.
- As funções recursivas são em sua maioria soluções mais elegantes e simples, se comparadas a funções tradicionais ou iterativas, já que executam tarefas repetitivas sem utilizar nenhuma estrutura de repetição, como for ou while.
- Porém essa elegância e simplicidade têm um preço que requer muita atenção em sua implementação.

Funções recursivas contem duas partes fundamentais:

- Ponto de Parada ou Condição de Parada: que é o ponto onde a função será encerrada, e é geralmente um limite superior ou inferior da regra geral.
- Regra Geral: é o método que reduz a resolução do problema através da invocação recursiva de casos menores, que por sua vez são resolvidos pela resolução de casos ainda menores pela própria função, assim sucessivamente até atingir o “ponto de parada” que finaliza o método.

Algoritmo recursivo

- Para se criar um algoritmo recursivo, deve-se primeiro procurar encontrar uma solução de como o problema pode ser dividido em passos menores. Depois definir um ponto de parada.
- Em seguida, definir uma regra geral que seja válida para todos os demais casos. Deve-se verificar se o algoritmo termina, ou seja, se o ponto de parada é atingido. Para auxiliar nessa verificação, recomenda-se criar uma árvore de execução do programa, como um chinês, mostrando o desenvolvimento do processo.

Quando utilizer recursividade

- Deve-se utilizar a recursividade quando esta forma for a mais simples e intuitiva de implementar uma solução para a resolução de um determinado problema. Se não for (simples e intuitiva), será então melhor empregar outros métodos não recursivos, também chamados de “métodos iterativos”.

O problema do Fatorial

- Ao produto dos números naturais começando em **n** e decrescendo até **1** denominamos de **fatorial de n** e representamos por **n!**.
- Segundo tal definição, o **fatorial de 5** é representado por **5!** e lê-se **5 fatorial**.
- **5!** é igual a **5 . 4 . 3 . 2 . 1** que é igual a **120**, assim como **4!** é igual a **4 . 3 . 2 . 1** que é igual a **24**, como **3!** é igual a **3 . 2 . 1** que é igual a **6** e que **2!** é igual a **2 . 1** que é igual a **2**.
- Por definição tanto **0!**, quanto **1!** são iguais a **1.**

Forma genérica

$$n! = n \cdot (n - 1)! = n \cdot (n - 1) \cdot (n - 2)! = n \cdot (n - 1) \cdot (n - 2) \cdot (n - 3) \cdot \ldots \cdot 1!$$

Fatorial de n sem recursividade

```
int main()
{
    int fat, n;
    printf("Insira um valor para o qual deseja calcular seu fatorial: ");
    scanf("%d", &n);
    for(fat = 1; n > 1; n = n - 1)
        fat = fat * n;
    printf("\nFatorial calculado: %d", fat);
    return 0;
}
```

Fatorial de n recursivo

```
#include <stdio.h>

int fat(int n)

{
    if (n)
    return n*fat(n-1);
    else return 1;

}

int main()

{
    int n;
    printf("\n\nDigite um valor para n: ");
    scanf("%d", &n);
    printf("\nO fatorial de %d e' %d", n, fat(n));
    return 0;

}
```

- Note que, enquanto n não for igual a 0, a função fat chama a si mesma, cada vez com um valor menor. n=0 é critério de parada para esta função.

Observações

- Há certos algoritmos que são mais eficientes quando feitos de maneira recursiva, mas a recursividade é algo a ser evitado sempre que possível, pois, se usada incorretamente, tende a consumir muita memória e ser lenta.
- Lembre-se que memória é consumida cada vez que o computador faz uma chamada a uma função. Com funções recursivas a memória do computador pode se esgotar rapidamente.

Problema de Fibonacci

- Os números de Fibonacci formam uma sequência infinita de números naturais, sendo que a partir do terceiro, os números são obtidos através da soma dos dois números anteriores.
- A sequência definida por Fibonacci inicia-se com 1 e 1, mas por convenção pode-se definir F(0) = 0, isto é, pode-se convencionar que a sequência começa em 0 e 1.

Em termos matemáticos, a sequência é definida recursivamente pela fórmula abaixo, sendo o primeiro termo F_1= 1:

$$F_n = F_{n-1} + F_{n-2},$$

Tabela de Fibonnacci

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
F(n)	0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987	1597	2584	4181	6765	10946

Na primeira linha temos o índice do número de **Fibonacci** na sequência e na segunda linha temos o número propriamente dito, por exemplo, para **n = 7** que corresponde ao oitavo número, temos que **F(7) = 13** , ou seja, o oitavo número da série é o número **13**.

Veja que de fato, a partir do terceiro número, cada um deles é o resultado da soma dos números anteriores, por exemplo, **10946 = 4181 + 6765**.

Algoritmo de Fibonnacci sem recursividade

```
#include "stdio.h"
 void main()
{
   int a, b, auxiliar, i, n;
   a = 0;
  b = 1;
    printf("Digite um número: ");
    scanf("%d", &n);
  printf("Série de Fibonacci:\n");
  printf("%d\n", b);
for(i = 1; i < n; i++)
  {
    auxiliar = a + b;
    a = b;
    b = auxiliar;
    printf("%d\n", auxiliar);
  }
}
```

Série de Fibonacci usando recursividade em linguagem C

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

main()
{
   int n,i;
   printf("Digite a quantidade de termos da sequência de Fibonacci: ");
   scanf("%d", &n);
   printf("\nA sequência de Fibonacci e: \n");
   for(i=0; i<n; i++)
       printf("%d ", fibonacci(i+1));
   getch();
}

int fibonacci(int num)
{
   if(num==1 || num==2)
       return 1;
   else
       return fibonacci(num-1) + fibonacci(num-2);
}
```

- O laço for contido na função principal main, chama a função fibonacci que calcula os valores retornando o valor 1 quando a posição da sequência for igual a 1 ou 2, e posteriormente calcula o restante dos números sempre somando as duas posições anteriores para obter o resultado atual.

Debugar o algoritmo de fibonacci

- Debugar o código anterior para uma sequência com até 5 números

Exercício de recursividade – Somatório

- Realizar o somatório dos números considerando como:

  Somatorio (5) = 5 + Somatorio de (4) e assim por diante

DIRETIVAS DE COMPILAÇÃO

- O pré-processador C é um programa que examina o programa fonte escrito em C e executa certas modificações nele, baseado nas Diretivas de Compilação.
- As diretivas de compilação são comandos que não são compilados, sendo dirigidos ao pré-processador, que é executado pelo compilador antes da execução do processo de compilação propriamente dito.

Modificação do programa-fonte – diretivas ANSI

- Portanto, o pré-processador modifica o programa fonte, entregando para o compilador um programa modificado. Todas as diretivas de compilação são iniciadas pelo caracter #. As diretivas podem ser colocadas em qualquer parte do programa.

#if
#else
#include
#ifdef
#elif
#define
#ifndef
#endif
#undef

Diretiva #include

- Indica ao compilador para incluir, na hora da compilação, um arquivo especificado.

```
#include "nome_do_arquivo"
ou
#include <nome_do_arquivo>
```

- A diferença entre se usar " " e < > é somente a ordem de procura nos diretórios pelo arquivo especificado. Se você quiser informar o nome do arquivo com o caminho completo, ou se o arquivo estiver no diretório de trabalho, use " ".
- Se estiver no caminho pré-especificado use <>

As Diretivas define e undef

- A diretiva #define tem a seguinte forma geral:

  #define nome_da_macro sequência_de_caracteres

- Quando você usa esta diretiva, você está dizendo ao compilador para que, toda vez que ele encontrar o nome_da_macro no programa a ser compilado, ele deve substituí-lo pela sequência_de_caracteres fornecida.

Exemplo

```
#include <stdio.h>
#define PI 3.1416
#define VERSAO "2.02"
int main ()
{
    printf ("Programa versao %s",VERSAO);
    printf ("O numero pi vale: %f",PI);
    return 0;
}
```

Outro uso do #define

- Simplesmente definir uma macro com argumentos para o código

#define nome_da_macro

```
#define max(A,B) ((A>B) ? (A):(B))
#define min(A,B) ((A<B) ? (A):(B))
...
x = max(i,j);
y = min(t,r);
```

- Embora pareça uma chamada de função, o uso de max (ou min) simplesmente substitui, em tempo de compilação, o código especificado. Cada ocorrência de um parâmetro formal (A ou B, na definição) será substituído pelo argumento real correspondente.

#undef

- A diretiva #undef tem a seguinte forma geral:

```
#undef nome_da_macro
```

- Ela faz com que a macro que a segue seja apagada da tabela interna que guarda as macros.
- O compilador passa a partir deste ponto a não conhecer mais esta macro.

Diretivas ifdef e endif

```
#ifdef nome_da_macro
sequência_de_declarações
#endif
```

- A diretiva de compilação #endif é util para definir o fim de uma sequência de declarações para todas as diretivas de compilação condicional.

Diretivas ifdef e endif

```
#define PORT_0 0x378

...

/* Linhas de codigo qualquer... */

...

#ifdef PORT_0

#define PORTA PORT_0

#include "../sys/port.h"

#endif
```

- Demonstram como estas diretivas podem ser utilizadas. Caso PORT_0 tenha sido previamente definido, a macro PORTA é definida e o header file port.h̄ é incluído.

A Diretiva ifndef

- A diretiva #ifndef funciona ao contrário da diretiva #ifdef. Sua forma geral

#ifndef nome_da_macro

sequência_de_declarações

#endifé:

- A sequência de declarações será compilada se o nome da macro não tiver sido definido.

A Diretiva if

- A diretiva #if tem a seguinte forma geral:

```
#if expressão_constante
   sequência_de_declarações
#endif
```

- A sequência de declarações será compilada se a expressão-constante for verdadeira.
- É muito importante ressaltar que a expressão fornecida deve ser constante, ou seja, não deve ter nenhuma variável.

Diretiva #else

- A diretiva #else tem a seguinte forma geral:

```
#if expressão_constante
    sequência_de_declarações
#else
    sequência_de_declarações
#endif
```

- Ela funciona como seu correspondente, o comando else.

Exemplo

```
#define SISTEMA DOS
...
/*linhas de codigo..*/
...
#if SISTEMA == DOS
     #define CABECALHO "dos_io.h"
#else
    #define CABECALHO "unix_io.h"
#endif
     #include CABECALHO
```

Diretiva elif

diretiva #elif serve para implementar a estrutura if-else-if. Sua forma geral é:

```
#if expressão_constante_1
    sequência_de_declarações_1
#elif expressão_constante_2
     sequência_de_declarações_2
#elif expressão_constante_3
      sequência_de_declarações_3


#elif expressão_constante_n
    sequência_de_declarações_n
#endif
```

Protótipo de Funções

- Muitas vezes teremos o nosso programa espalhado por vários arquivos. Ou seja, estaremos chamando funções em um arquivo que serão compiladas em outro arquivo. Como manter a coerência?
- A solução são os protótipos de funções. Protótipos são nada mais, nada menos, que declarações de funções. Isto é, você declara uma função que irá usar.
- O compilador toma então conhecimento do formato daquela função antes de compilá-la. O código correto será então gerado.

Exemplo de protótipo

- Formato: tipo_de_retorno nome_da_função (declaração_de_parâmetros);

onde o tipo-de-retorno, o nome-da-função e a declaração-de-parâmetros são os mesmos que você pretende usar quando realmente escrever a função.

- Os protótipos têm uma nítida semelhança com as declarações de variáveis.

Exemplo de Funções

```
#include <stdio.h>

float Square (float a);

int main ()

{
    float num;
    printf ("Entre com um numero: ");
    scanf ("%f",&num);
    num=Square(num);
    printf ("\n\nO seu quadrado vale: %f\n",num);
    return 0;

}

float Square (float a)

{
    return (a*a);

}
```

Tipo void

- Permite fazer funções que não retornam nada e funções que não têm parâmetros.

  void nome_da_função (declaração_de_parâmetros);

- Numa função, como a acima, não temos valor de retorno na declaração return. Aliás, neste caso, o comando return não é necessário na função.

  tipo_de_retorno nome_da_função (void);

  ou

  void nome_da_função (void);

Exemplo de funções com void

```
#include <stdio.h>

void Mensagem (void);

int main ()

{
	Mensagem();
	printf ("\tDiga de novo:\n");
	Mensagem();
	return 0;

}

void Mensagem (void)

{

    printf ("Ola! Eu estou vivo.\n");

}
```

Retorno de funções void

As duas funções main() abaixo são válidas:

```
main (void)
{
        ....
        return 0;
}
void main (void)
{
        ....
}
```

Declaração

- Suponha que a função 'int EPar(int a)', seja importante em vários programas, e desejemos declará-la num módulo separado. No arquivo de cabeçalho chamado por exemplo de 'funcao.h‘ teremos a seguinte declaração:

função.h

```
int EPar(int a);
```

Função.c

```
int EPar (int a)
{
        if (a%2) /* Verifica se a e divisivel por dois */
        return 0;
else
        return 1;
}
```

Exemplo de utilização

```
#include <stdio.h>

#include "funcao.h"

void main ()

{
    int num;
    printf ("Entre com numero: ");
    scanf ("%d",&num);
    if (EPar(num))
            printf ("\n\nO numero e par.\n");
    else
            printf ("\n\nO numero e impar.\n");

}
```

Exercícios

- Escreva um programa que faça uso da função EDivisivel(int a, int b).
- Organize o seu programa em três arquivos: o arquivo prog.c, conterá o programa principal; o arquivo func.c conterá a função; o arquivo func.h conterá o protótipo da função. Compile os arquivos e gere o executável a partir deles.

Resolução

```
#include <stdio.h>

#include <stdlib.h>

#include "func.h"

int main(int argc, char *argv[]) {

                char div;

                int a,b;

                printf ("Entre com o valor 1 : ");

                scanf ("%d",&a);

                printf ("\nEntre com o valor 2 :");

                scanf ("%d",&b);

                div = EDivisivel(a,b);

                putchar(div);

                return 0;

}
```

Func.h

```
char EDivisivel(int a,int b);
```

Func.c

```
char EDivisivel(int a,int b)
{
        if ( (a % b) == 0 )
            return 'S';
        else
            return 'N';
}
```

Vetores e Matrizes

Vetores

- Vetores nada mais são que matrizes unidimensionais.
- É importante notar que vetores, matrizes bidimensionais e matrizes de qualquer dimensão são caracterizadas por terem todos os elementos pertencentes ao mesmo tipo de dado.

```
tipo_da_variável nome_da_variável [tamanho];
```

- Quando o C vê uma declaração como esta ele reserva um espaço na memória suficientemente grande para armazenar o número de células especificadas em tamanho :

```
float exemplo [20];
```

Acesso a Vetores

- Na linguagem C a numeração começa sempre em zero. Isto significa que, no exemplo acima, os dados serão indexados de 0 a 19. Para acessá-los vamos escrever:

exemplo[0]

exemplo[1]

.

exemplo[19]

- Mas ninguém o impede de escrever:

exemplo[30]

exemplo[103]

- C não verifica se o índice que você usou está dentro dos limites válidos. Este é um cuidado que você deve tomar. Se o programador não tiver atenção com os limites de validade para os índices ele corre o risco de ter variáveis sobreescritas ou de ver o computador travar.

```
#include <stdio.h>
int main ()
{
    int num[100]; /* Declara um vetor de inteiros de 100 posicoes */
    int count=0;
    int totalnums;
do
{
     printf ("\nEntre com um numero (-999 p/ terminar): ");
     scanf ("%d",&num[count]);
      count++;
} while (num[count-1]!=-999);
     totalnums=count-1;
      printf ("\n\n\n\t Os números que você digitou foram:\n\n");
      for (count=0;count<totalnums;count++)
      printf (" %d",num[count]);
      return(0);
}
```

Explicação sobre o programa

- No exemplo acima, o inteiro count é inicializado em 0.
- O programa pede pela entrada de números até que o usuário entre com o Flag -999. Os números são armazenados no vetor num.
- A cada número armazenado, o contador do vetor é incrementado para na próxima iteração escrever na próxima posição do vetor. Quando o usuário digita o flag, o programa abandona o primeiro loop e armazena o total de números gravados.
- Por fim, todos os números são impressos. É bom lembrar aqui que nenhuma restrição é feita quanto a quantidade de números digitados. Se o usuário digitar mais de 100 números, o programa tentará ler normalmente, mas o programa os escreverá em uma parte não alocada de memória, pois o espaço alocado foi para somente 100 inteiros.

Exercício

- Reescreva o programa anterior, realizando a cada leitura um teste para ver se a dimensão do vetor não foi ultrapassada. Caso o usuário entre com 100 números, o programa deverá abortar o loop de leitura automaticamente. O uso do Flag (-999) não deve ser retirado.

Strings

- Strings são vetores de chars
- As strings são o uso mais comum para os vetores. Devemos apenas ficar atentos para o fato deque as strings têm o seu último elemento como um '\0'.

```
char nome_da_string [tamanho];
```

- O tamanho da string deve incluir o '\0' final (Importante)

Copiando Strings

```
#include <stdio.h>

int main ()

{
    int count;
    char str1[100],str2[100];
    .... /* Aqui o programa le str1 que sera
    copiada para str2 */
    for (count=0;str1[count];count++)
      str2[count]=str1[count];
    str2[count]='\0';
    .... /* Aqui o programa continua */

}
```

- A condição no loop for acima é baseada no fato de que a string que está sendo copiada termina em '\0'.

Função gets

- A função gets() lê uma string do teclado. Sua forma geral é:

gets (nome_da_string); #include <stdio.h>

Exemplo:

```
int main ()
{
    char string[100];
    printf ("Digite o seu nome: ");
    gets (string);
    printf ("\n\n Ola %s",string);
    return(0);
}
```

Função strcpy

- Sua forma geral é:

strcpy (string_destino,string_origem);

- A função strcpy() copia a string-origem para a string- destino. Seu funcionamento é semelhante ao da rotina apresentada

```
#include <stdio.h>
#include <string.h>
int main ()
{
    char str1[100],str2[100],str3[100];
    printf ("Entre com uma string: ");
    gets (str1);
    strcpy (str2,str1); /* Copia str1 em str2 */
    strcpy (str3,"Voce digitou a string "); /* Copia "Voce digitou a string" em
    str3 */
    printf ("\n\n%s%s",str3,str2);
    return(0);
}
```

Função strlen

- Sua forma geral é:

strlen (string);

- A função strlen() retorna a quantidade de caracteres da string

Função strcat

- Sua forma geral é:

Strcat (string_destino,string_origem);

- A função strcat() concatena duas strings

Matrizes

- A forma geral da declaração de uma matriz bidimensional é muito parecida com a declaração de um vetor:

tipo_da_variável nome_da_variável [altura][largura];

- É muito importante ressaltar que, nesta estrutura, o índice da esquerda indexa as linhas e o da direita indexa as colunas. Quando vamos preencher ou ler uma matriz no C o índice mais à direita varia mais rapidamente que o índice à esquerda.
- O C não controla os limites de acesso aos índices da matriz

```
#include <stdio.h>
int main ()
{
    int mtrx [20][10];
    int i,j,count;
    count=1;
    for (i=0;i<20;i++)
        for (j=0;j<10;j++)
        {
           mtrx[i][j]=count;
              count++;
        }
  return(0);
}
```

Matrizes de strings

- Matrizes de strings são matrizes bidimensionais. Se fizermos um vetor de strings estaremos fazendo uma lista de vetores. Esta estrutura é uma matriz bidimensional de chars.

  char nome_da_variável [num_de_strings][compr_das_strings];

- Acesso a uma determinada string:

  nome_da_variável [índice]

```
#include <stdio.h>

int main ()

{
    char strings [5][100];
    int count;
    for (count=0;count<5;count++)
    {
        printf ("\n\nDigite uma string: ");
        gets (strings[count]);
    }
    printf ("\n\n\nAs strings que voce digitou foram:\n\n");
    for (count=0;count<5;count++)
            printf ("%s\n",strings[count]);
    return(0);

}
```

Matrizes multidimensionais

- Sua forma geral é:

tipo_da_variável nome_da_variável [tam1][tam2] ... [tamN];

- Uma matriz N-dimensional funciona basicamente como outros tipos de matrizes. Basta lembrar que o índice que varia mais rapidamente é o índice mais à direita.