MAC 2166 – Introdução à Computação

POLI - PRIMEIRO SEMESTRE DE 2007

Material Didático

Prof. Ronaldo Fumio Hashimoto

FUNÇÕES - MAIS INTRODUÇÃO

Objetivo

Nesta aula é uma continuação da aula anterior e falaremos mais sobre funções. Veremos como declarar funções usando protótipos, como definir o corpo da função e como utilizar funções em seus programas (chamada de função e passagem de parâmetros).

Introdução

Um programa na linguagem C pode ser organizado na forma de funções, onde cada função é responsável pelo cálculo/processamento de uma parte do programa. Por exemplo, no exercício da aula passada, a função `potencia` era responsável pelo cálculo das potenciações envolvidas no programa que era a solução do exercício.

Em particular, o `main` é uma função! Todo programa em C precisa de uma função chamada `main` (função principal), que, dentre todas as funções que estão em seu programa, ela é a primeira a ser executada.

Função Principal

Para efeitos desta disciplina, quando um exercício de MAC2166 estiver pedindo para você **fazer um programa**, isto significa que estamos pedindo para você escrever a **função principal**, ou seja, a função `main`, como você tem feito até agora. Só que agora, sua função `main` vai fazer uso de outras funções definidas por você ou que fazem parte de alguma biblioteca de funções.

Entrada e Saída da Função Principal

Já que estamos falando da função `main`, vamos relembrar um pouco sobre entrada e saída da função principal.

A entrada de dados do programa principal (ou seja, do "**int** `main` ()") é feita pelo teclado e para isso é usado o comando de leitura `scanf` (veja Fig. 1).

A saída de dados do programa principal é feita por impressões na tela e para isso é usado o comando de impressão `printf` (veja Fig. 1).

Ora, por que estamos falando tudo isso? Porque para outras funções que não seja a função principal, a entrada e saída dos dados normalmente não é feita pelo teclado e nem pela tela. Para entrada de dados, usamos o que chamamos de **parâmetros** e para saída, por enquanto, usaremos o comando **return** (veja Fig. 2).

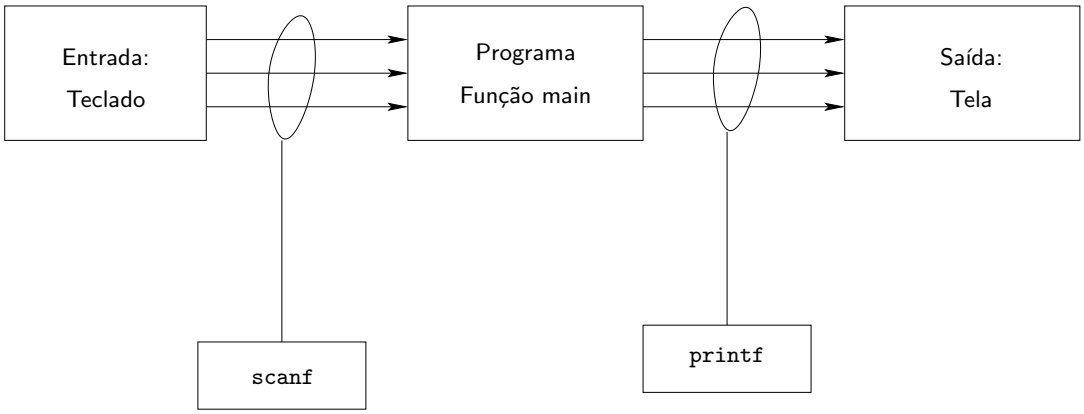

Figura 1: Entrada e Saída do Programa Principal.

As Outras Funções

As "outras funções" são as funções que você usa no seu programa em C com exceção da função principal. Para estas funções, a entrada de dados normalmente é feita por parâmetros. No exemplo da aula anterior, os parâmetros são a base e o expoente da potenciação que se deseja calcular. Para fins deste curso, a saída das "outras funções" é um valor (e somente um único valor) que pode ser um **int**, **float** ou **char**. No exemplo da aula anterior, a saída é um valor do tipo **float** (feita via **return**) e corresponde ao resultado da potenciação que se deseja calcular.

Isto significa que normalmente (note que está escrito **normalmente**) não devemos colocar `scanf` e nem `printf` em funções que não sejam a função principal.

Este conceito de **parâmetros** e **return** para as "outras funções" é muito importante. Se você não entendeu, por favor, releia com atenção a primeira aula de funções. Tem muito aluno que coloca `scanf` e `printf` nas "outras funções" significando que este conceito não foi bem entendido.

Funções Definidas por Você

Para definir uma função, você deve ter em claro quais são as entradas (parâmetros) e qual é a saída (que valor e o tipo do mesmo que a função deve devolver via **return**). Para o exemplo da aula anterior, a função `potencia` tem dois parâmetros: `base` e `expoente` e a saída é um valor **float** resultado da potenciação `base` elevado a `expoente` (veja Fig. 3).

Cabeçalho ou Protótipo de uma Função

O cabeçalho ou protótipo de uma função contém o tipo do valor devolvido (**int**, **char** ou **float**), o nome da função e uma lista de parâmetros, entre parênteses, seguido de um ponto e vírgula dispostos da seguinte forma:

```
<tipo_do_retorno> <nome_da_função> (<lista_de_parâmetros>);
```

onde

- `<tipo_do_retorno>`: define o tipo do valor devolvido pela função (saída) via **return**;
- `<nome_da_função>`: nome da função;

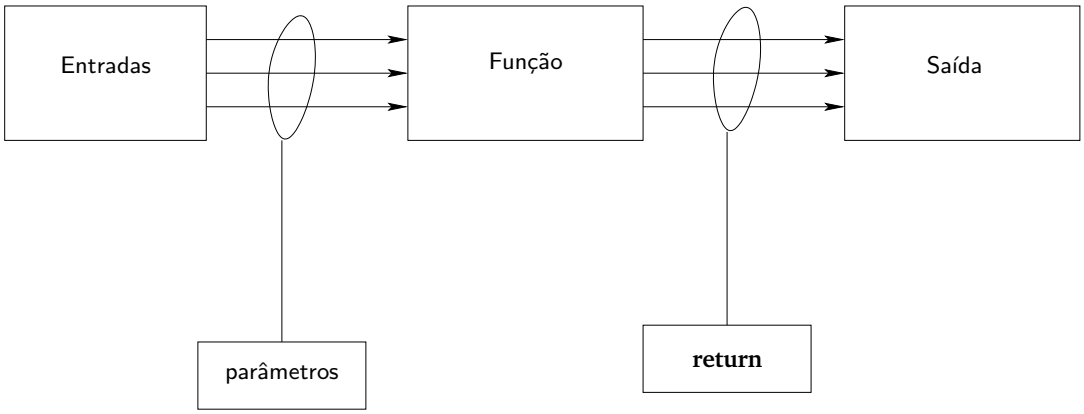

Figura 2: Entrada e Saída de Funções.

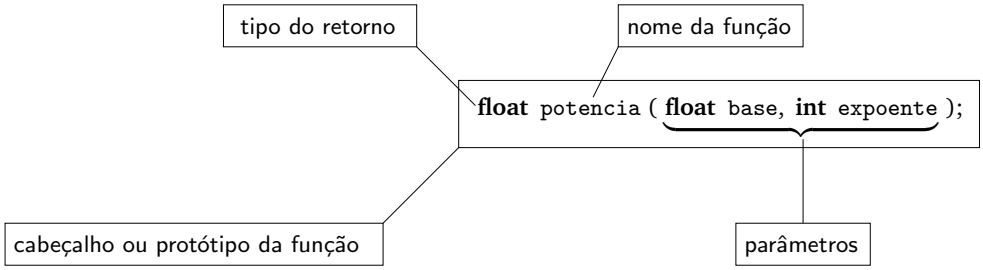

Figura 3: Entrada e Saída da Função `potencia`.

- <`lista_de_parâmetros`>: parâmetros de entrada da função.

Mais uma vez, note que a <`lista_de_parâmetros`> está entre parênteses e o cabeçalho termina com um ponto e vírgula.

A forma de escrever a <`lista_de_parâmetros`> é a mesma forma de declarar variáveis, ou seja, na lista de parâmetros as variáveis são declaradas como <`tipo_da_variável`> <`nome_da_variável`> separadas por vírgulas.

Por exemplo, considere uma função `potencia` da aula passada que calcula `base` elevado a `expoente`, onde é um `base` número real e `expoente` é um número inteiro. Como o resultado de `base` elevado a `expoente` é um número real (uma vez que `base` é um número real), então a função deve ser do tipo **float**. Dessa forma, o seu protótipo seria:

float `potencia` (**float** `base`, **int** `expoente`);

Como Definir uma Função: Corpo de uma Função

Uma função é **definida** com o seu cabeçalho (sem o ponto e vírgula) seguido por um bloco de comandos definidos entre chaves. Chamamos este bloco de comandos dentro da função como *corpo da função*. Por exemplo, no caso da função `potencia`, como vimos na aula passada, sua definição poderia ser:

```
potencia (float base, int expoente) {
  float pot;
  int cont;
  cont = 0; pot = 1;
  while (cont < expoente) {
    pot = pot * base;
    cont = cont + 1;
  }
  /* no final pot contém base elevado a expoente */
  return pot;
}
```

Dentro de cada função, você deve primeiramente declarar as variáveis que serão utilizadas **dentro** da função (ou seja entre as chaves que definem o corpo da função). Dentro da função, os parâmetros de entrada equivalem a variáveis, porém, o valor inicial dessas "variáveis" são definidas na hora de usar (chamar) a função no programa principal, por exemplo.

Os comandos que definem a função são definidos dentro do bloco, ao final, o resultado da função é devolvido ao local da chamada através do comando **return**.

Exercício

Vamos fazer um exercício para fixar estas idéias. Este exercício está dividido em três partes:

(a) Escreva uma função **int** `fat` (**int** `n`) que recebe como parâmetro um inteiro `n` e calcula o fatorial de `n`.

Solução:

```
int fat (int n) {
  int cont = 1, fatorial = 1;

  while (cont <= n) {
    fatorial = fatorial * cont;
    cont = cont + 1;
  }

  return fatorial;
}
```

(b) Utilizando a função do item anterior, faça uma função **int** `comb` (**int** `m`, **int** `n`) que recebe dois inteiros `m` e `n`, com `m`$\geq$ `n`, e calcula

$$\texttt{C(m,n)} = \frac{\texttt{m!}}{\texttt{n!(m-n)!}}$$

Solução:

```
int comb (int m, int n) {
  int resultado;

  resultado = fat (m) / (fat (n)*fat(m-n));

  return resultado;
}
```

(c) Utilizando a função do item anterior, faça um programa que leia (do teclado) um inteiro `n > 0` e imprime (na tela) os coeficientes da expansão de $(a+b)^{\texttt{n}}$.

Exemplo: `n = 2` ⇒ `1, 2, 1`
`n = 4` ⇒ `1, 4, 6, 4, 1`

O nosso programa deve fazer:

para `n = 2`, imprimir: `C(2,0), C(2,1), C(2,2)`
para `n = 4`, imprimir: `C(4,0), C(4,1), C(4,2), C(4,3), C(4,4)`
para `n = x`, imprimir: `C(x,0), C(x,1),.....,C(x,x)`

Solução:

```
1   # include <stdio.h>
2
3   int main () {
4
5     int n, coef, cont;
6
7     printf ("Entre com n > 0: ");
8     scanf ("%d", &n);
9     cont  = 0;
10    while (cont <= n) {
11      coef = comb (n, cont);
12      printf ("%d\n", coef);
13      cont = cont + 1;
14    }
15
16    return 0;
17  }
```

Note que a leitura pelo teclado (`scanf`) e a impressão na tela (`printf`) estão somente na função principal. Nas funções `fat` e `comb` a entrada de dados é feita via parâmetros e a saída via **return**.

Estrutura de um Programa

A estrutura de um programa com funções deve seguir a seguinte forma:

```
/* includes */

/* protótipos das funções */

/* definição das funções */

/* função principal */
```

Nosso programa do exercício anterior com funções seria:

```
1   /* includes */
2
3   # include <stdio.h>
4
5   /* protótipos das funções */
6
7   int fat (int n);
8   int comb (int m, int n);
9
10  /* definição das funções */
11
12  int fat (int n) {
13    int cont = 1, fatorial = 1;
14
15    while (cont <= n) {
16      fatorial = fatorial * cont;
17      cont = cont + 1;
18    }
19
20    return fatorial;
21  }
22
23  int comb (int m, int n) {
24    int resultado;
25
26    resultado = fat (m) / (fat (n)*fat(m-n));
27
28    return resultado;
29  }
30
31  /* função principal */
32
33  int main () {
34
35    int n, coef, cont;
36
37    printf ("Entre com n > 0: ");
38    scanf ("%d", &n);
39    cont  = 0;
40    while (cont <= n) {
41      coef = comb (n, cont);
42      printf ("%d\n", coef);
43      cont = cont + 1;
44    }
45
46    return 0;
47  }
```

Na realidade, a declaração explicita dos protótipos da funções (linhas 7 e 8 do programa) não é necessária, mas é uma boa prática da computação pois permite que você defina as funções em qualquer ordem.

Caso os protótipos não sejam declarados, as funções precisam ser definidas antes de serem utilizadas por outras funções. Por exemplo, no nosso exemplo, como a função `comb` usa (chama) a função `fat`, então a função `fat` deve ser definida antes da função `comb`.

Exercício Especialmente para Você

(a) Faça uma função de nome `primo` que recebe um inteiro `n > 1` e verifica se `n` é primo ou não. Esta função devolve `0` (zero) se `n` não é primo; e devolve `1` (um) se `n` é primo.

(b) Faça um programa que leia um inteiro `n > 2` e verifica se `n` pode ser escrito como `p + q`, com `p > 1`, `q > 1`, `p` e `q` primos.

Por exemplo, n = 5 ⇒ SIM, pois 5 = 2 + 3
n = 11 ⇒ NÃO.

Note que seu programa deve testar se existe um `p` primo tal que `p>0`, `p<n` e `n-p` é primo.

Simulação de Funções

Simule a saída do seguinte programa:

```
1   # include <stdio.h>
2
3   int g (int x, int y) {
4     int z;
5     z = x * x + y * y;
6     return z;
7   }
8
9   int h (int x, int a) {
10    int d;
11    d = g (a, x) + 3;
12    return d;
13  }
14
15  int main () {
16    int y, z, x;
17    z = 2;
18    y = 3;
19    x = h (y, z);
20    printf ("x = %d\n", x);
21
22    return 0;
23  }
```

O programa sempre começa a ser executado na função principal.

	main		
	y	z	x
Início →	?	?	?
Linhas 17 e 18 →	3	2	?

Na Linha 19 é chamada a função `h`. Note a passagem de parâmetros:

- `y` da função `main` → (para) `x` da função `h` e
- `z` da função `main` → (para) `a` da função `h`

	main		
	y	z	x
Início →	?	?	?
Linhas 17 e 18 →	3	2	?

	h		
	x	a	d
Início →	?	?	?
Linha 9 →	3	2	?

Note que a variável `x` da `main` não tem nada a ver com a variável `x` da função `h`. Cada função tem sua variável, ou seja, mesmo sendo de mesmo nome, as variáveis são diferentes, pois estão declaradas em funções diferentes.

Na Linha 11 é chamada a função `g`. Note a passagem de parâmetros:

- `a` da função `h` → (para) `x` da função `g` e
- `x` da função `h` → (para) `y` da função `g`

	h		
	x	a	d
Início →	?	?	?
Linha 9 →	3	2	?

	g		
	x	y	z
Início →	?	?	?
Linha 9 →	2	3	?

Na Linha 5, calcula `z` da função `g`:

	g		
	x	y	z
Início →	?	?	?
Linha 9 →	2	3	13

Na Linha 6, temos um **return**. Na aula passada vimos que o **return**:

- guarda o resultado ou o conteúdo de uma variável no ACUM.
- finaliza a função. O fluxo do programa volta para onde a função foi chamada.

Neste caso, coloca o valor de `z` da função `g` no ACUM e volta para a Linha 11.

	ACUM
Início →	?
Linha 11 →	13

Na Linha 11, calcula `d` da função `h`:

	h		
	x	a	d
Início →	?	?	?
Linha 9 →	3	2	16

Na Linha 12, coloca o valor de `d` da função `h` no ACUM e volta para a Linha 19.

	ACUM
Início →	?
Linha 11 →	13
Linha 12 →	16

Na Linha 19, coloca o conteúdo do ACUM na variável `x` da função `main`:

	main		
	y	z	x
Início →	?	?	?
Linhas 17 e 18 →	3	2	?
Linhas 19 →	3	2	16

No final imprime na tela o contéudo da variável `x` da função `main` na tela:

Tela
x = 16

Funções que fazem parte de alguma Biblioteca

Existe um conjunto de funções pré-definidas na linguagem C que você pode fazer uso. Particularmente, você pode estar interessado em funções matemáticas tais como funções trigonométricas, hiperbólicas, logarítmicas, exponenciais, etc... A seguir, listamos algumas destas funções[1]:

```
/*****************************************************
Seção 1 -- Funções trigonométricas
*****************************************************/

double sin (double x);
double cos (double x);
double tan (double x);

/*****************************************************
Seção 2 -- Exponenciais e logaritmos
*****************************************************/

/* Uso típico: y = exp (x);                           */
/* Devolve e^x, ou seja, o número e elevado à         */
/* potência x.                                        */

double exp (double x);


/* Uso típico: y = log (x);                           */
/* Devolve o logaritmo de x na base e. Não use com    */
/* x negativo.                                        */

double log (double x);


/* Uso típico: y = log10 (x);                         */
/* Devolve o logaritmo de x na base 10. Não use com   */
/* x negativo.                                        */

double log10 (double x);

/*****************************************************
Seção 3 -- Raiz e potência
*****************************************************/

/* Uso típico: y = sqrt (x);                          */
/* Devolve a raiz quadrada de x. Não use com x < 0.   */

double sqrt (double x);


/* Uso típico: p = pow (x, y);                        */
/* Devolve x^y, ou seja, x elevado à potência y.      */
/* Não use com x = 0.0 e y < 0.0. Não use com x < 0.0 */
/* e y não-inteiro.                                   */
/* Caso especial: pow (0.0, 0.0) == 1.0.              */
/* Que acontece se x^y não couber em double? Veja man */
/* pages.                                             */

double pow (double x, double y);
```

[1] retiradas da página <http://www.ime.usp.br/~pf/algoritmos/apend/math.h.html>

```
/*****************************************************
Seção 4 - Valor Absoluto
*****************************************************/

/* Uso típico: i = fabs (x). A função devolve o valor  */
/* absoluto de x.                                      */


double fabs (double x);


/*****************************************************
Seção 5 -- Arredondamentos
*****************************************************/

/* Uso típico: i = floor (x). A função devolve o maior */
/* inteiro que seja menor que ou igual a x, isto é,    */
/* o único inteiro i que satisfaz i <= x < i+1.        */

double floor (double x);


/* Uso típico: j = ceil (x). A função devolve o menor  */
/* inteiro que seja maior que ou igual a x, isto é,    */
/* o único inteiro j que satisfaz j-1 < x <= j.        */

double ceil (double x);
```

Para podermos utilizar essas funções, devemos incluir o arquivo de cabeçalhos # **include** <math.h> no início do seu programa em C. Além disso, se você estiver usando o compilador gcc, você precisa compilar o seu programa com a opção `-lm`:

```
gcc -Wall -ansi -pedantic -O2 <nome_do_programa>.c -o <nome_do_programa> -lm
```

Se você está usando o compilador Dev-C++, você não precisa se preocupar com esta opção.