15 Funções - Introdução

Ronaldo F. Hashimoto e Carlos H. Morimoto

O objetivo desta aula é introduzir o conceito de função e sua utilidade. Ao final dessa aula você deverá saber:

- Justificar as vantagens do uso de funções em programas.
- Identificar em seus programas oportunidades para criação e uso de funções.
- Definir os parâmetros que a função precisa receber (parâmetros de entrada) e o que a função precisa devolver (o tipo do valor devolvido pela função).

15.1 Exercício de Aquecimento

Faça um programa que leia um real x e um inteiro $n > 0$ e calcule x^n.

Uma solução deste exercício poderia usar a seguinte estratégia:

- Gerar uma sequência de n números reais $x, x, x, \ldots, x$.
- Para cada número gerado, acumular o produto em uma variável pot:

$$pot = pot * x$$

Um trecho de programa para esta solução poderia ser:

```
1       float pot, x;
2       int cont, n;
3       cont = 0; pot = 1;
4       while (cont < n) {
5         pot = pot * x;
6         cont = cont + 1;
7       }
8       /* no final pot contém x^n */
```

15.2 Exercício de Motivação

Faça um programa que leia dois números reais x e y e dois números inteiros $a > 0$ e $b > 0$, e calcule o valor da expressão $x^a + y^b + (x - y)^{a+b}$.

Este programa deve calcular a potenciação de dois números três vezes: uma para x^a, outra para y^b e outra para $(x - y)^{a+b}$.

Para calcular x^a, temos o seguinte trecho de programa:

```
1       float pot, x;
2       int cont, a;
3       cont = 0; pot = 1;
4       while (cont < a) {
5         pot = pot * x;
6         cont = cont + 1;
7       }
8       /* no final pot contem x^a */
```

Para calcular y^b, temos o seguinte trecho de programa:

```
1       float pot, y;
2       int cont, b;
3       cont = 0; pot = 1;
4       while (cont < b) {
5         pot = pot * y;
6         cont = cont + 1;
7       }
8       /* no final pot contém y^b */
```

Para calcular $(x-y)^{a+b}$, temos o seguinte trecho de programa:

```
1       float pot, x, y;
2       int cont, a, b;
3       cont = 0; pot = 1;
4       while (cont < a+b) {
5         pot = pot * (x-y);
6         cont = cont + 1;
7       }
8       /* no final pot contém (x-y)^(a+b) */
```

Tirando as variáveis que mudam, isto é, os expoentes (variáveis a, b e $a+b$) e as bases (variáveis x, y e $x-y$), praticamente temos o mesmo código. Na elaboração de programas, é comum necessitarmos escrever várias vezes uma mesma sequência de comandos, como no exemplo acima.

Assim, seria muito interessante termos uma maneira de escrevermos um código que poderia ser aproveitado, mudando somente os **valores** (note que está escrito **valores** e não nomes) das variáveis que são o expoente e a base da potenciação. Algo como:

```
1       float pot, base;
2       int cont, expoente;
3       cont = 0; pot = 1;
4       while (cont < expoente) {
5         pot = pot * base;
6         cont = cont + 1;
7       }
8       /* no final pot contém base elevado a expoente */
```

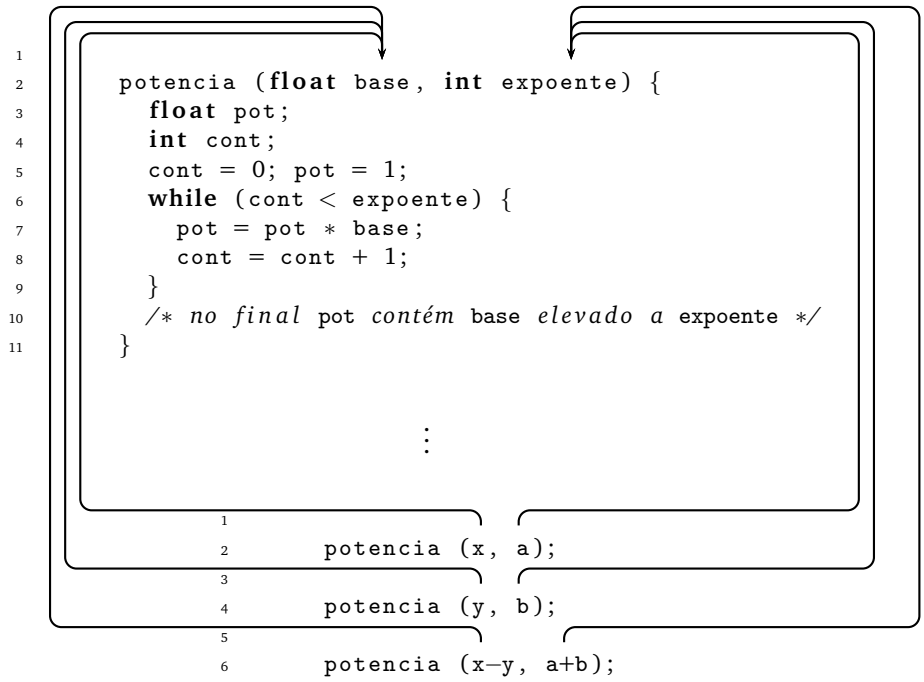
Assim, se quisermos calcular x^a, devemos fazer algo como `expoente ← a` e `base ← x` e rodarmos o código acima. Da mesma forma, para calcular y^b e $(x-y)^{a+b}$ devemos fazer, de alguma forma, `expoente ← b`, `base ← y` e `expoente ← a+b` e `base ← (x-y)`, respectivamente, e rodar o código acima.

A idéia aqui é ainda escrevermos este código somente uma vez! Para isso, vamos dar um nome para o nosso trecho de código. Que tal `potencia`? Assim, toda vez que quisermos utilizar o código acima, simplesmente usuaríamos seu nome que é `potencia` . Poderíamos fazer:

```
1       potencia (float base, int expoente) {
2         float pot;
3         int cont;
4         cont = 0; pot = 1;
5         while (cont < expoente) {
6           pot = pot * base;
7           cont = cont + 1;
8         }
9         /* no final pot contém base elevado a expoente */
10      }
```

O trecho de código acima de nome `potencia` na verdade é uma **função**. A idéia de função é fazer um subprograma de nome `potencia` que seja utilizado, para o exemplo acima, três vezes.

Note agora que as variáveis `base` e `expoente` estão declaradas entre parênteses e depois do nome `potencia`. Por que? A idéia para calcular x^a não é fazer algo como `expoente ← a` e `base ← x`? Então, para calcular x^a, a gente poderia colocar o conteúdo das variáveis `x` e `a` nas variáveis `base` e `expoente`, respectivamente, usando o comando `potencia (x, a)`. Observe no diagrama abaixo:



O comando `potencia (x, a)` coloca o conteúdo das variáveis `x` e `a` (seus valores) nas variáveis `base` e `expoente` respectivamente, como indicado pelas setas. Depois executa os comandos com os respectivos valores das variáveis `base` e `expoente` atualizados. No final da execução da função `potencia`, ou seja, depois que sair do laço, a variável `pot` contém x^a.

Da mesma forma, podemos então calcular y^b e $(x-y)^{a+b}$ usando o comando `potencia (y, b)` e o comando `potencia (x-y, a+b)`, respectivamente. Assim, ao final de cada execução da função `potencia`, a variável `pot` contém y^b e $(x-y)^{a+b}$, respectivamente.

Agora, você deve estar se perguntando o que fazer com cada resultado que está na variável `pot`? Aliás, como `pot` pode guardar três valores se `pot` é uma variável **float**. Na verdade, `pot` guarda somente **um** valor real. Toda vez que o código `potencia` é executado, o valor do último `pot` é perdido!

Ah! Então, para cada vez que o código de `potencia` for executado, a gente poderia imprimir o valor final da variável `pot`. Aí a gente tem o resultado de cada potenciação! Legal, mas, isto não resolveria o problema de calcular $x^a + y^b + (x-y)^{a+b}$. O que teríamos que fazer é guardar o resultado das potenciações em outras três variáveis, por exemplo, `potx`, `poty` e `potxy` e no final imprimir a soma `potx` + `poty` + `potxy`.

Para indicar que o resultado de `potencia (x, a);` será colocado na variável `potx`, vamos escrever o uso da função `potencia` como `potx = potencia (x, a);`. Da mesma forma, para as variáveis `poty` e `potxy`. Assim, teríamos:

```
1       potx = potencia (x, a);
2
3       poty = potencia (y, b);
4
5       potxy = potencia (x-y, a+b);
6
7       soma = potx + poty + potxy;
8       printf ("Resultado = %f\n", soma);
```

Assim, `potx = potencia (x, a);`, `poty = potencia (y, b);` e `potxy = potencia (x-y, a+b);` indicam que o valor da variável `pot` de cada execução da função `potencia` é colocado nas variáveis `potx`, `poty` e `potxy`, respectivamente. Depois é só imprimir a soma das variáveis `potx`, `poty` e `potxy`.

Agora, como fazer com que a função `potencia` coloque corretamente "para fora" o valor da variável `pot`? Nós vamos indicar isso colocando no final da função `potencia` o comando **return** `pot;`. Veja como fica então:

```
1       potencia (float base, int expoente) {
2         float pot;
3         int cont;
4         cont = 0; pot = 1;
5         while (cont < expoente) {
6           pot = pot * base;
7           cont = cont + 1;
8         }
9         /* no final pot contém base elevado a expoente */
10        return pot;
11      }
```

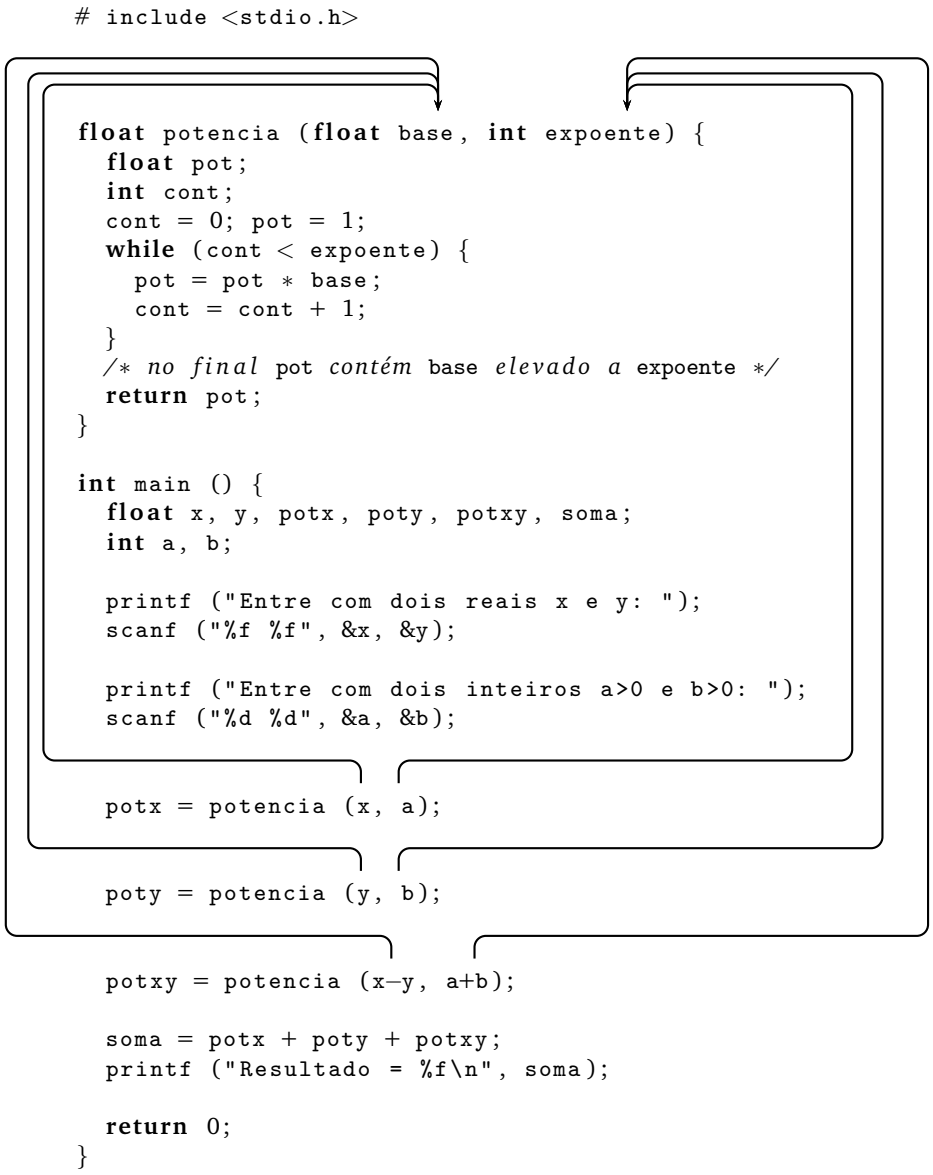
Assim, o uso `potx = potencia (x, a);` faz com que `expoente ← a` e `base ← x` e executa o código de `potencia`; no final da execução, a variável `pot` contém o valor de x^a; o comando **return** `pot;` coloca o valor que está na variável `pot` da função `potencia` em um lugar do computador (CPU) que se chama **acumulador**. **Acumulador?!?** O que é afinal de contas um **acumulador**? O acumulador é um lugar do computador capaz de guardar números. Este acumulador funciona como uma variável: toda vez que um valor é colocado nele, o último valor é perdido. Então, como o resultado final é colocado na variável `potx`? Depois que a função encontrou um **return**, o fluxo do programa volta para o comando `potx = potencia (x, a);`. Neste momento, o resultado da função está no **acumulador**, pois o código da função `potencia` já foi executado e consequentemente o comando **return** também já foi executado (aliás, este é o último comando executado da função). Aí, como temos indicado que a variável `potx` recebe `potencia (x, a)` no comando `potx = potencia (x, a);`, então o valor que está no **acumulador** é colocado em `potx`.

Da mesma forma, o uso `poty = potencia (y, b);` faz com que `expoente ← b` e `base ← y` e executa o código de `potencia`; no final da execução, a variável `pot` contém o valor de y^b; o comando **return** `pot;` coloca o valor que está na variável `pot` da função `potencia` no **acumulador**. Depois que a função encontrou um **return**, o fluxo do programa volta para o comando `poty = potencia (y, b);`. Neste momento, o resultado da função está no **acumulador**, pois o código da função `potencia` já foi executado e consequentemente o comando **return** também já foi executado (aliás, este é o último comando executado da função). Aí, como temos indicado que a variável `poty` recebe `potencia (y, b)` no comando `poty = potencia (y, b);`, então o valor que está no **acumulador** é colocado em `poty`.

Mais uma vez, o uso `potxy = potencia (x-y, a+b);` faz com que `expoente ← a+b` e `base ← x-y` e executa o código de `potencia`; no final da execução, a variável `pot` contém o valor de $(x-y)^{a+b}$; o comando **return** `pot;` coloca o valor que está na variável `pot` da função `potencia` no **acumulador**. Depois que a função encontrou um **return**, o fluxo do programa volta para o comando `potxy = potencia (x-y, a+b);`. Neste momento, o resultado da função está no **acumulador**, pois o código da função `potencia` já foi executado e consequentemente o comando **return** também já foi executado (aliás, este é o último comando executado da função). Aí, como

temos indicado que a variável `potxy` recebe `potencia (x-y, a+b)` no comando `potxy = potencia (x-y, a+b);`, então o valor que está no **acumulador** é colocado em `potxy`.

Você deve estar se perguntando: como coloco tudo isto em um programa em C? Veja o código em C abaixo:



```
1   # include <stdio.h>
2
3
4
5   float potencia (float base, int expoente) {
6     float pot;
7     int cont;
8     cont = 0; pot = 1;
9     while (cont < expoente) {
10      pot = pot * base;
11      cont = cont + 1;
12    }
13    /* no final pot contém base elevado a expoente */
14    return pot;
15  }
16
17  int main () {
18    float x, y, potx, poty, potxy, soma;
19    int a, b;
20
21    printf ("Entre com dois reais x e y: ");
22    scanf ("%f %f", &x, &y);
23
24    printf ("Entre com dois inteiros a>0 e b>0: ");
25    scanf ("%d %d", &a, &b);
26
27
28    potx = potencia (x, a);
29
30
31    poty = potencia (y, b);
32
33
34    potxy = potencia (x-y, a+b);
35
36    soma = potx + poty + potxy;
37    printf ("Resultado = %f\n", soma);
38
39    return 0;
40  }
```

Note que antes do nome da função `potencia` foi colocado um **float**. Este **float** indica que a função "vai jogar para fora" (devolver) um valor do tipo **float** via **return**. De fato, observe que no comando `return pot;`, a variável `pot` guarda um valor real.

A idéia de função no exercício anterior é fazer um subprograma de nome `potencia` que seja utilizado três vezes. Este subprograma deve ter entrada e saída. Por exemplo, para calcular x^a, a entrada do subprograma deve ser o real x e o inteiro a e a saída deve ser o real x^a.

Em resumo, o uso de funções facilita a construção de programas pois possibilita a reutilização de partes de código.

15.3 Exercícios Recomendados

1. Considerando a função fatorial, que parâmetros ela deve receber (incluindo os tipos)? E qual o tipo do valor de saída (real ou inteiro?).

2. Considere uma função `seno` que calcula o valor do seno de um ângulo x com precisão $epsilon$. Que parâmetros ela deve receber (incluindo os tipos)? E qual o tipo do valor de saída dessa função(real ou inteiro?).

3. Escreva um programa que leia dois inteiros `m` e `n`, com `m`$\geq$ `n`, e calcula

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