Argumentos e parâmetros de funções

Aula 18

Fábio Henrique Viduani Martinez

Faculdade de Computação
Universidade Federal de Mato Grosso do Sul

Algoritmos e Programação I, 2012

Conteúdo da aula

Introdução

- a interface de uma função especifica quase tudo o que precisamos saber sobre a mesma:
  - o tipo de valor que a função devolve
  - nome/identificador da função
  - parâmetros da função
- podemos entender o que um programa faz sem a necessidade de examinar os detalhes de suas funções
- caso alguns detalhes sejam de nosso interesse, também há a vantagem de que sabemos onde examiná-los

Introdução

- a interface de uma função especifica quase tudo o que precisamos saber sobre a mesma:
  - o tipo de valor que a função devolve
  - nome/identificador da função
  - parâmetros da função
- podemos entender o que um programa faz sem a necessidade de examinar os detalhes de suas funções
- caso alguns detalhes sejam de nosso interesse, também há a vantagem de que sabemos onde examiná-los

Introdução

- a interface de uma função especifica quase tudo o que precisamos saber sobre a mesma:
  - o tipo de valor que a função devolve
  - nome/identificador da função
  - parâmetros da função
- podemos entender o que um programa faz sem a necessidade de examinar os detalhes de suas funções
- caso alguns detalhes sejam de nosso interesse, também há a vantagem de que sabemos onde examiná-los

Argumentos e parâmetros

- um **parâmetro** aparece na definição de uma função e é um nome que representa um valor a ser fornecido quando a função é chamada
- um **argumento** é uma expressão que ocorre em uma chamada de uma função
- conceitualmente, existem três tipos de parâmetros:
  - **de entrada**, que permitem que valores sejam passados *para* a função;
  - **de saída**, que permite que um valor seja devolvido *da* função;
  - **de entrada e saída**, que permitem que valores sejam passados *para* a função e devolvidos *da* função

Argumentos e parâmetros

- um **parâmetro** aparece na definição de uma função e é um nome que representa um valor a ser fornecido quando a função é chamada
- um **argumento** é uma expressão que ocorre em uma chamada de uma função
- conceitualmente, existem três tipos de parâmetros:
  - **de entrada**, que permitem que valores sejam passados *para* a função;
  - **de saída**, que permite que um valor seja devolvido *da* função;
  - **de entrada e saída**, que permitem que valores sejam passados *para* a função e devolvidos *da* função

Argumentos e parâmetros

- um **parâmetro** aparece na definição de uma função e é um nome que representa um valor a ser fornecido quando a função é chamada
- um **argumento** é uma expressão que ocorre em uma chamada de uma função
- conceitualmente, existem três tipos de parâmetros:
  - **de entrada**, que permitem que valores sejam passados *para* a função;
  - **de saída**, que permite que um valor seja devolvido *da* função;
  - **de entrada e saída**, que permitem que valores sejam passados *para* a função e devolvidos *da* função

Argumentos e parâmetros

- até o momento, entramos em contato com parâmetros de entrada e parâmetros de saída nos programas que já fizemos
- os valores dos parâmetros de entrada são passados para uma função através de um mecanismo denominado **passagem por cópia** ou **passagem por valor**

```
x = quadrado(num);
```

Argumentos e parâmetros

- até o momento, entramos em contato com parâmetros de entrada e parâmetros de saída nos programas que já fizemos
- os valores dos parâmetros de entrada são passados para uma função através de um mecanismo denominado **passagem por cópia** ou **passagem por valor**

```
x = quadrado(num);
```

Argumentos e parâmetros

- até o momento, entramos em contato com parâmetros de entrada e parâmetros de saída nos programas que já fizemos
- os valores dos parâmetros de entrada são passados para uma função através de um mecanismo denominado **passagem por cópia** ou **passagem por valor**

```
x = quadrado(num);
```

Argumentos e parâmetros

- os valores dos parâmetros de entrada e saída são passados/devolvidos por um mecanismo chamado **referência**
- temos uma variável especificada na chamada da função e um parâmetro especificado na interface da função que compartilham a mesma área de armazenamento na memória e isso significa que qualquer alteração realizada no conteúdo do parâmetro dentro da função acarreta alteração no conteúdo da variável que é o argumento da chamada

Argumentos e parâmetros

- os valores dos parâmetros de entrada e saída são passados/devolvidos por um mecanismo chamado **referência**
- temos uma variável especificada na chamada da função e um parâmetro especificado na interface da função que compartilham a mesma área de armazenamento na memória e isso significa que qualquer alteração realizada no conteúdo do parâmetro dentro da função acarreta alteração no conteúdo da variável que é o argumento da chamada

Argumentos e parâmetros

```c
#include <stdio.h>

void decompoe(float x, int *parte_int, float *parte_frac)
{
   *parte_int = (int) x;
   *parte_frac = x - *parte_int;
}

int main(void)
{
   float num, b;
   int a;

   scanf("%f", &num);
   decompoe(num, &a, &b);
   printf("Número: %f\n", num);
   printf("Parte inteira: %d\n", a);
   printf("Parte fracionária: %f\n", b);

   return 0;
}
```

Exercícios

1. (a) Escreva uma função com a seguinte interface:

```
void troca(int *a, int *b)
```

que receba dois números inteiros a e b e troque os seus conteúdos.

(b) Usando a função `troca` definida acima, escreva um programa que leia um número inteiro n e um vetor contendo n números inteiros, com $1 \leqslant n \leqslant 100$, ordene seus elementos em ordem crescente usando o método das trocas sucessivas e imprima o vetor resultante na saída.

(c) Repita a letra (b) implementando o método da seleção.

(d) Repita a letra (b) implementando o método da inserção.

Exercícios

```
#include <stdio.h>

#define MAX 100

/* Recebe dois números inteiros a e b
   e devolve seus valores trocados */
void troca(int *a, int *b)
{
   int aux;

   aux = *a;
   *a = *b;
   *b = aux;
}
```

Exercícios

```
/* Recebe um um número inteiro n, com 1 <= n <= 100,
   e uma sequência de n números inteiros, e mostra
   essa sequência em ordem crescente */
int main(void)
{
   int i, j, n, A[MAX];

   scanf("%d", &n);
   for (i = 0; i < n; i++)
      scanf("%d", &A[i]);

   for (i = n - 1; i > 0; i--)
      for (j = 0; j < i; j++)
         if (A[j] > A[j+1])
            troca(&A[j], &A[j+1]);

   for (i = 0; i < n; i++)
      printf("%d ", A[i]);
   printf("\n");

   return 0;
}
```

Exercícios

2. Em um dado país a moeda corrente possui apenas quatro cédulas de papel: $1, $5, $10 e $20.
   (a) Escreva uma função com a seguinte interface:

```
void cedulas(int val, int *um, int *cin, int *dez, int *vin)
```

   que receba um número inteiro não-negativo que representa um valor na moeda do país e determine a menor quantidade de cédulas de 1, 5, 10 e 20 necessárias para pagar o valor especificado.
   (b) Escreva um programa que receba uma série de números inteiros, cada um representando um valor na moeda corrente, e determine a menor quantidade de cédulas para pagar tal valor. O programa termina quando a entrada fornecida é um número inteiro negativo. Use a função do item (a).

Exercícios

3. Dizemos que um número natural n é **palíndromo** se lemos o número da esquerda para direita e também da direita para esquerda e obtemos o mesmo número.
Exemplos: 567765 e 32423 são palíndromos, mas 567675 não é.

   (a) Escreva uma função com a seguinte interface:

```
void quebra(int n, int *prim, int *ult, int *miolo)
```

   que receba um número inteiro $n > 0$ e devolva três números inteiros: o primeiro dígito de n, o último dígito de n e um inteiro que represente o número n sem seu primeiro e último dígitos.
   Exemplo:

valor inicial de n	primeiro dígito	último dígito	miolo de n
732	7	2	3
14738	1	8	473
78	7	8	0
7	7	7	0

Exercícios

3. (*continuação*)

   (b) Usando a função do item (a), escreva um programa que receba um número inteiro $n > 0$ e verifique se n é palíndromo. Suponha que n não contém o dígito 0.

Exercícios

4. O **mínimo múltiplo comum** entre dois números inteiros positivos pode ser calculado conforme o esquema abaixo ilustrado:

8	36	2
4	18	2
2	9	2
1	9	3
1	3	3
1	1	$2^3 \cdot 3^2 = 72$

   (a) Escreva uma função com a seguinte interface:

```
int divisao(int *m, int *n, int d)
```

   que receba três números inteiros positivos m, n e d e devolva 1 se d divide m, n ou ambos, e 0, caso contrário. Além disso, em caso positivo, a função deve devolver, nos parâmetros correspondentes, um valor que representa o quociente da divisão de m por d e outro valor que representa o quociente da divisão de n por d.

Exercícios

4. (*continuação*)

   (b) Escreva um programa que receba um número inteiro $k > 0$ e uma sequência de k pares de números inteiros positivos m e n e calcule, usando a função do item (a), o mínimo múltiplo comum entre m e n.

Exercícios

5. Os babilônios descreveram há mais de 4 mil anos um método para calcular a raiz quadrada de um número. Esse método ficou posteriormente conhecido como método de Newton. Dado um número x, o método parte de um chute inicial y para o valor da raiz quadrada de x e sucessivamente encontra aproximações desse valor, calculando a média aritmética de y e de x/y. O exemplo a seguir mostra o método em funcionamento para o cálculo da raiz quadrada de 3, com chute inicial 1:

x	y	x/y	$(y + x/y)/2$
3	1	3	2
3	2	1.5	1.75
3	1.75	1.714286	1.732143
3	1.732143	1.731959	1.732051
3	1.732051	1.732051	1.732051

O valor devolvido é o valor de y, isto é, $y = \sqrt{x}$.

Exercícios

5. (*continuação*)
   Podemos descrever esta fórmula de uma outra maneira:

$$\begin{cases} y_0 = \alpha\,, \\ y_i = \dfrac{y_{i-1} + x/y_{i-1}}{2}\,, \quad \text{para } i \geqslant 1\,, \end{cases}$$

   onde α é um chute inicial e $y_{i^*} = \sqrt{x}$, para algum $i^* \geqslant 1$ e $|y_{i^*} - y_{i^*-1}| < \varepsilon$.

   (a) Escreva uma função com a seguinte interface:

```
double raiz(double x, double epsilon, int *passos)
```

   que receba um número real x e um número real ε, com $0 < \varepsilon < 1$, e devolva o valor de $\sqrt{x}$ usando o método de Newton descrito acima, até que o valor absoluto da diferença entre dois valores consecutivos de y seja menor que ε. A função deve devolver também, no parâmetro `passos`, a quantidade de passos realizados para obtenção da raiz de x com precisão ε.

Exercícios

5. (*continuação*)

   (b) Escreva um programa que receba um número inteiro $k > 0$ que representa a quantidade de casos de teste. Para cada caso de teste, receba um número real positivo x e um número real ε, com $0 < \varepsilon < 1$, e calcule e imprima $\sqrt{x}$ com precisão dada por ε. Mostre $\sqrt{x}$ com 6 casas decimais na saída.