Computação para Informática - Prof. Adriano Joaquim de Oliveira Cruz

O objetivo desta aula prática é exercitar funções com recursão.

1 Recursão

Recursão é:

- um modo de pensar sobre problemas;
- um método de resolver problemas;
- relacionada à indução matematica.

Uma função é recursiva se ela chama a si mesmo, podendo ser diretamente (listagem 1) ou indiretamente (listagem 2).

Listing 1: Chamada recursiva direta 1.

```
void f() {
   /* ... */
   f()
   /* ... */
}
```

Listing 2: Chamada recursiva indireta 1.

```
void f() {
    /* ... */
    g()
    /* ... */
}
void g() {
   ... f() ...
}
```

Exemplo de recursão:

> Uma criança não conseguia dormir, então sua mãe contou-lhe uma história sobre um sapinho que não conseguia dormir, então a mãe do sapinho contou-lhe uma história sobre ursinho que não conseguia dormir, então a mãe do ursinho contou-lhe uma história sobre um gatinho que dormiu, então o ursinho dormiu, então o sapinho dormiu e finalmente a criança dormiu.

Perguntas:

P: Usar recursão faz o programa rodar mais rapidamente?

R: Não.

P: Usar recursão faz o programa usar menos memória?

R: Não.

P: Então, por que usar recursão?

R: Algumas vezes faz o código ficar mais simples.

Para facilitar pode-se pensar que quando uma chamada recursiva é feita, a função faz uma cópia de si mesmo, das variáveis locais, com seus valores iniciais, e dos parâmetros.

Cada cópia da função inclui uma marcação indicando o ponto atual onde a função está sendo executada. Quando uma chamada recursiva é feita, o marcador na função que chamou fica logo após a chamada. O marcador na função chamada vai para o início da função.

Quando a função retorna, a cópia some, e a função que chamou continua após a chamada, posição indicada pela marcação. As cópias anteriores continuam sabendo de onde continuar porque as marcações estão em suas posições.

Funções podem ser recursivas com cauda ou não. Em uma função recursiva com cauda, nenhuma das chamadas recursivas executam algo após a chamada recursiva completar, exceto para retornar o valor da função.

A listagem 3 mostra o return de uma típica função recursiva com cauda e a listagem 4 mostra uma sem cauda. A chamada sem cauda é caracterizada porque primeiro `f` é feita e depois soma 5, portanto, algo é executado após a chamada.

Listing 3: Chamada recursiva com cauda.

```
int f(int x, int y) {
    return f(x, y);
}
```

Listing 4: Chamada recursiva sem cauda.

```
int f(int x, int y) {
    return f(x, y) + 5;
}
```

1.1 Como pensar recursivamente?

Primeiro escreva o protótipo da função. Defina o que a função recursiva deve fazer em português. Considere o seguinte exemplo.

```
/* Soma os primeiros n elementos de um vetor */
int soma( int vetor[], int n ) ;
```

Quando a chamada `soma( vetor, 10 );` for executada sabemos que os primeiros 10 elementos do vetor `vetor` serão somados.

Em seguida, repita várias vezes para si mesmo o seguinte raciocínio:

> **recursão resolve um grande problema (de tamanho n, por exemplo), resolvendo um ou mais problemas de tamanho menor, e usando as soluções destes pequenos problemas para resolver o grande problema.**

Caso básico

Portanto, recursão é um processo que quebra um problema em problemas menores que serão quebrados em problemas menores até que chegamos no menor problema possível e na sua solução (**caso básico**) e retornamos esta solução.

Uma vez que você escreveu o protótipo, pense em como resolver o próximo problema menor. Então se a chamada é

```
soma(vetor, n),
```

o próximo caso menor é

```
soma(vetor, n-1)
```

Suponha que alguém lhe daria a resposta para esta segunda soma. O que você teria? Você teria a soma dos $n-1$ primeiros elementos do vetor. Agora que você tem esta solução (por hipótese), o que falta para resolver todo o problema? Achar a solução do caso base, que é o menor problema. Neste exemplo seria:

```
sum(vetor, 1 );
```

Deste modo a solução para este problema fica como mostrado na listagem 5.

Listing 5: Soma de vetor recursiva.

```
int soma( int vetor[], int n )
{
  int menor;
  if ( n == 0 )  {/* caso base */
    return 0 ;    /* sem chamada recursiva */
  }
  else {
      menor = soma(vetor, n - 1 ) ; /* resolve problema menor */
      /* usa solucao do menor para resolver o maior */
      return menor + vetor[ n - 1 ] ;
    }
}
```

Os passos para escrever uma função recursiva são:

1. Escreva um protótipo da função recursiva.
2. Escreva um comentário que descreve o que a função deve fazer.
3. Determine o caso base (pode haver mais que um) e sua solução.
4. Determine qual é o problema menor a resolver. Se facilitar use variáveis para armazenar o resultado em variáveis locais.
5. Use a solução do problema menor para resolver o problema maior.

1.2 O que faz recursão trabalhar?

Recursão somente funciona quando um problema tem uma estrutura recursiva. Isto significa que a solução do problema é a mesma para diferentes tamanhos.

Caso reduzir o tamanho do problema faz com que a solução seja diferente então recursão não é a solução. Do mesmo modo, você deve ser capaz de usar os problemas menores para resolver o problema maior.

Todavia, pela mesma razão, deve ser difícil resolver usar laços para resolver o problema. Laços também dependem de fazer a mesma coisa diversas vezes, em diferentes índices.

Se você conseguir resolver um problema por meio de um laço, deve ser possível resolve-lo por meio de recursão.

2 Exercícios

Exercício 1: Escreva um programa que imprima os números de 1 até 10 em ordem inversa e em ordem direta usando recursividade. Não use laços.

Dica para quem não veio na aula teórica: Solução nas transparências.

Exercício 2: Escreva uma função recursiva chamada `int balasDeBoliche(int n)` que determina o número de bolas de boliche em uma pilha organizada na forma de uma pirâmide como mostra a figura 1. Como você pode observar, no topo da pilha existe uma única bola de boliche apoiada em um quadrado formado por quatro bolas, que por sua vez se apoia em um quadrado composto por nove bolas, e assim por diante. Sua função recebe como argumento o número de camadas na pilha (i.e., sua altura) e retorna o número de bolas de boliche na pilha.

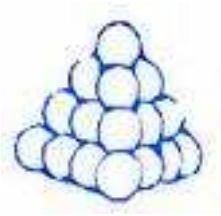

Figura 1: Pilha de bolas de boliche

Exercício 3: O diagrama mostrado na tabela 1 mostra como podemos calcular o máximo divisor comum de dois números inteiros positivos. Escreva uma função iterativa e uma recursiva para calcular o mdc entre dois números inteiros positivos. Use estas funções um programa que lê dois números inteiros e calcula o mdc.

quociente	2	4	3
348	156	36	12
resto	36	12	0

Tabela 1: Algoritmo de Euclides para mdc.

Dica do Sr. Euclides:

> Se x é divisível por y, então y é o maior divisor comum entre os dois números. Caso contrário, o máximo divisor comum entre x e y é sempre igual ao maior divisor comum entre y e o resto da divisão de x por y.

Exercício 4: Escreva um programa que leia uma frase de até 80 caracteres e a imprima em ordem inversa usando recursividade. O protótipo da função é void `reverso(char v[], int n)` onde v é o vetor de caracteres e n é o número de caracteres a serem impressos.

Exercício 5: Complete o programa mostrado na listagem 6. Este programa lê uma série de números inteiros e calcula e imprime os fatoriais. A série termina quando um número negativo é lido. O programa deve ser resolvido com recursividade.

Uma função recursiva é uma função que chama a si mesma. Neste caso o fatorial é calculado da seguinte maneira:

$$\begin{aligned} n! &= n*(n-1)! \\ (n-1)! &= (n-1)*(n-2)! \\ (n-2)! &= (n-2)*(n-3)! \\ \vdots &= \vdots \\ 1! &= 1 \text{ (acabou)} \end{aligned}$$

Listing 6: Programa do problema 5.

```
#include<stdio.h>
long long int fat (long int n);
int main (void) {
     long long int numero;
     while (1)      {
          scanf("%Ld", &numero);
          if (numero < 0) break;
          printf("O fatorial de %ld vale %Ld\n",
                numero, fat(numero));
     }
     return 0;
}
long int fat (long int n) {
}
```

Exercício 6: Na Matemática, os "Números de Leonardo Pisano Fibonacci" formam uma seqüência matemática (sucessão, em Portugal) definida, recursivamente, pela fórmula 1:

$$F(n) = \begin{cases} 0 & \text{se } n = 0 \\ 1 & \text{se } n = 1; \\ F(n-1) + F(n-2) & \text{outros casos.} \end{cases} \quad (1)$$

Na prática você começa com 0 e 1, e então produz o próximo número de Fibonacci somando os dois anteriores para formar o próximo. Os primeiros números são `0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946...`

Escreva uma função recursiva que a partir um número inteiro e positivo n retorne os n primeiros termos da seqüencia. O protótipo da função é : `int Fibonacci(int a);` Escreva um programa que use esta função para imprimir os primeiros n números da sequência.

Exercício 7: Verifique o funcionamente do programa 7.

Listing 7: Programa do problema 7.

```
#include<stdio.h>

#define MAX 10  /* maximo numero de discos */

void TorresDeHanoi (int , char, char, char);

int main (void) {
    int numeroDeDiscos;

     printf("Quantos discos?  =====> ");
     scanf("%d", &numeroDeDiscos);

     TorresDeHanoi(numeroDeDiscos, 'A', 'C', 'B');
    return 0;
}
void TorresDeHanoi(int numeroDeDiscos, char hasteOrigem,
                              char hasteDestino, char hasteAuxiliar) {
     if (numeroDeDiscos == 1) {
          printf("Mover disco 1 da haste %c para a haste %c\n",
                                                  hasteOrigem, hasteDestino);
     }
     else {
          TorresDeHanoi(numeroDeDiscos-1, hasteOrigem, hasteAuxiliar, hasteDestino);
          printf("Mover o disco %d da haste %c para a haste %c\n",
                                     numeroDeDiscos, hasteOrigem, hasteDestino);
          TorresDeHanoi(numeroDeDiscos-1,  hasteAuxiliar, hasteDestino, hasteOrigem);
     }
}
```

3 Desafios

Exercício 8:

É Palíndromo?

Tarefa

Um palíndromo é uma palavra ou uma frase que tenha a propriedade de poder ser lida tanto da direita para a esquerda como da esquerda para a direita. Num palíndromo, normalmente são desconsiderados os sinais ortográficos (diacríticos ou de pontuação), assim como o espaços entre palavras. As seguintes frases são exemplos de palíndromo:

- Ande Edna.
- Após a sopa.
- Socorram-me subi no ônibus em Marrocos.

A sua tarefa é escrever um programa que descubra se cadeias de caracteres lidas do teclado são ou não um palíndromos. Cada cadeia pode ter até 80 caracteres. O seu programa deve usar a seguinte função (escrita por você):

```
int ehPalindromo(char *s);
```

Esta função deve retornar 1 se a cadeia apontada por `s` for um palíndromo e 0 no caso contrário.

Escreva primeiro esta função como um laço e depois recursivamente.

Entrada

A entrada é composta de vários casos de teste. O programa deve ler várias cadeias de caracteres e imprimir se cada cadeia é um palíndromo. Considere que estas cadeias não contém caracteres em branco, sinais de pontuação ou caracteres acentuados (os diacríticos mencionados acima) e todos os caracteres são minúsculas. A entrada para quando uma cadeia de tamanho 0 é lida, para isto use a função `strlen` incluida em #**include**<`string.h`>

Saída

A saída deve imprimir uma mensagem para cada frase lida informando se a frase lida é ou não ou palíndromo.

Exemplo de entrada e saída

Entrada:	**Saída:**
ameaema	ameaema E PALINDROMO
olaalo	olaalo E PALINDROMO
amormeamaemroma	amormeamaemroma E PALINDROMO
evaviuauva	evaviuauva NAO E PALINDROMO
andeedna	andeedna E PALINDROMO
aposasopa	aposasopa E PALINDROMO
ola	ola NAO E PALINDROMO