RECURSÃO

Estrutura de Dados

Recursão

- A recursão é uma técnica que define um problema em termos de uma ou mais versões menores deste mesmo problema.
- A recursão pode ser utilizada sempre que for possível expressar a solução de um problema em função do próprio problema.
- Uma função é dita recursiva quando dentro do seu código existe uma chamada para si mesma. Por Exemplo: Calcular o Fatorial de um número N inteiro qualquer. Se formos analisar a forma de cálculo temos:

Recursão

$$fat(n) = \begin{cases} 1, \text{ se } n = 0 \text{ (solução trivial)} \\ n \times fat(n-1), \text{ se } n > 0 \text{ (solução recursiva)} \end{cases}$$

- fat(5) = 5 x fat(4)
- fat(4) = 4 x fat(3)
- fat(3) = 3 x fat(2)
- fat(2) = 2 x fat(1)
- fat(1) = 1 x fat(0)
- fat(0) = 1

Recursão

```
#include<stdio.h>
int fatorialrec(int num)
{
  if (num == 0) {
    return 1;
  }
  else {
    return num * fatorialrec(num-1);
  }
}
int main() {
  int num;
  num = 5;
  printf("\nfatR(%d) = %d", num,
fatorialrec(num));
  printf("\n\nfatS(%d) = %d", num,
fatorialsemrec(num));
}
```

```
int fatorialsemrec(int num)
{
  int f, i;
  if (num == 0) {
    return 1;
  }
  else {
   f = 1;
   for(i= num; i > 1; i--){
     f = f * i;
   }
    return f;
  }
}
```

Recursão - Fibonacci

```
int fibb(int n){
   int f1 = 0, f2 = 1,
   f3, i;
   for(i=1; i <= n; i++)
   {
      f3 =  f2 + f1;
      f1 = f2;
      f2 = f3;
      }
      return f1;
      }
```

```
int fib(int n){
   if(n <= 1)
     return m;
   return
     fib(n-1)*fib(n-2);
   }
```

Recursão - Torres de Hanoi

```
void moveTorre(int n, char a, char b,
char c){
  if(n > 0) {
    moveTorre(n-1,a,c,b);
    printf("mover de %c para %c\n",a,b);
    moveTorre(n-1,c,b,a);
    }
    }
```

Recursão

- Em procedimentos recursivos pode ocorrer um problema de terminação do programa, como um "looping interminável ou infinito".
- Portanto, para determinar a terminação das repetições, deve-se:
  - 1) Definir uma função que implica em uma condição de terminação (solução trivial), e
  - 2) Provar que a função decresce a cada passo de repetição, permitindo que, eventualmente, esta solução trivial seja atingida.

Recursão

- Vantagens X Desvantagens
  - Um programa recursivo é mais elegante e menor que a sua versão iterativa, além de exibir com maior clareza o processo utilizado, desde que o problema ou os dados sejam naturalmente definidos através de recorrência.
  - Por outro lado, um programa recursivo exige mais espaço de memória e é, na grande maioria dos casos, mais lento do que a versão iterativa.

Recursão

- Receita básica para escrever um algoritmo recursivo:
  1. se o problema é pequeno, resolva-o diretamente;
  2. se o problema é grande, *reduza-o* a uma versão menor do mesmo problema
     e aplique a receita ao problema menor.

Recursão com Vetores

```
// A função maxr devolve um elemento máximo de v[0..n-1].
// Ela supõe que n >= 1.

int maxr (int v[], int n)
{
   if (n == 1) return v[0];
   else {
      int x;
      x = maxr(v, n-1);
      if (x > v[n-1]) return x;
      else return v[n-1];
   }
}
```

Recursão com Vetores

```
int maxr2 (int v[], int n) {
   return auxiliar(v, 0, n-1);
}

// Recebe v e índices p e r tais que p <= r.
// Devolve um elemento máximo do vetor v[p..r].
//
int auxiliar (int v[], int p, int r) {
   if (p == r) return v[p];
   else {
      int x;
      x = auxiliar(v, p + 1, r);
      if (v[p] < x) return x;
      else return v[p];
   }
```

Recursão com Vetores

```
// A função maxr3 devolve um elemento máximo de
v[0..n-1].
// Ela supõe que n >= 1.
//
int maxr3 (int *v, int n) {
    if (n == 1) return v[0];
    else {
        int x;
        x = maxr3(v + 1, n - 1);
        if (v[0] < x) return x;
        else return v[0];
    }
}
```

Recursão com Vetores

```
float invert(float vetor[], int a, int b){
	if (vetor != NULL){
		if ((b-a)>0) {
			printf("\n ANTES: vetor [%i] = %.2f e vetor [%i] = %.2f",a,
vetor[a], b,vetor[b]); // ==>> para teste
			float aux = vetor[a];
			vetor[a] = vetor[b];
			vetor[b] = aux;
			printf("\n DEPOIS: vetor [%i] = %.2f e vetor [%i] = %.2f",a,
vetor[a], b,vetor[b]); ==>> para teste
			invert(&vetor[tam], a+1, b-1);
		} else {
			return 0;
		}
	}
}
```

Recursão com Vetores

```
int main (){
	float vetor[tam];
	for (int i=0; i<tam; i++){
		vetor[i] = rand () %10;
	}
	for (int i=0; i<tam; i++){
		printf("%.2f ", vetor[i]);
	}
	printf("\n");
	invert(vetor, 0, tam-1);
	for (int i=0; i<tam; i++){
		printf("%.2f ", vetor[i]);
	}
	return 0;
}
```