UNIVERSIDADE FEDERAL DO CEARÁ - CAMPUS CRÁTEUS

CURSOS: CIÊNCIA DA COMPUTAÇÃO E SISTEMA DE INFORMAÇÃO

DISCIPLINA: ESTRTUTURA DE DADOS

PROFS. BRUNO DE CASTRO E WELLINGTON FRANCO

NOTA DE AULA SOBRE RECURSÃO

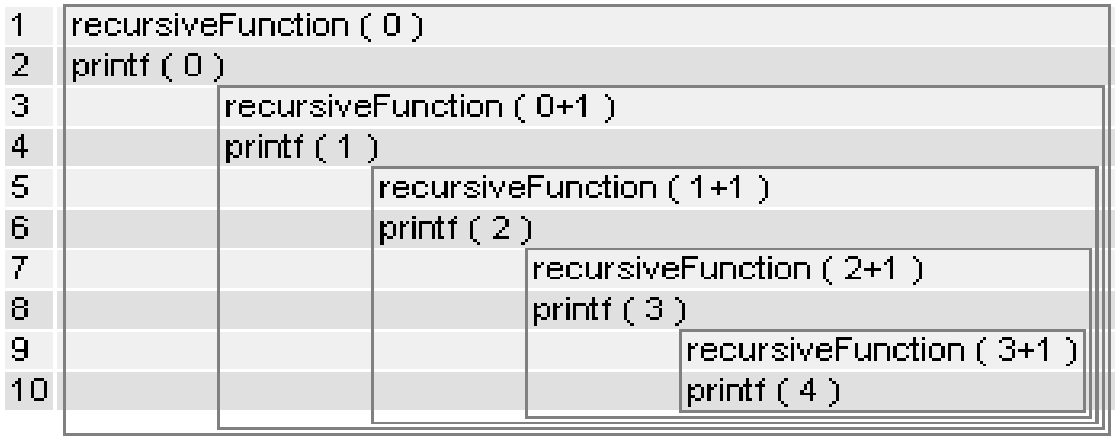
1. RECURSÃO

As funções podem ser chamadas recursivamente, isto é, dentro do corpo de uma função podemos chamar novamente a própria função. Se uma função A chama a própria função A, dizemos que ocorre uma recursão direta. Se uma função A chama uma função B que, por sua vez, chama A, temos uma recursão indireta. Diversas implementações ficam muito mais fáceis usando recursividade. Por outro lado, implementações não recursivas tendem a ser mais eficientes.

Para cada chamada de uma função, recursiva ou não, os parâmetros e as variáveis locais são empilhados na pilha de execução. Assim, mesmo quando uma função é chamada recursivamente, cria-se um ambiente local para cada chamada. As variáveis locais de chamadas recursivas são independentes entre si, como se estivéssemos chamando funções diferentes. A seguir é ilustrado o funcionamento de função recursiva:

```
void recursiveFunction (int num){

   if (num < 4){

      printf("%i\n", num);

      recursiveFunction (num + 1);

   }

}
```

```
void main ()

{

recursiveFunction (0);

}
```



2. RECURSÃO VERSUS ITERAÇÃO

Iteratividade(Estruturas de Repetição) é melhor que recursividade, quando estamos analisando desempenho. A legibilidade de códigos iterativos requer alguma experiência do programador, principalmente em códigos maiores, com muitos laços aninhados. Recursividade confere ao código maior legibilidade, tornando mais simples sua compreensão. Para pequenas aplicações, muitas vezes a recursividade apresenta perda tolerável de desempenho.

Se a iteratividade é sempre melhor, embora menos legível, então por que usar recursividade, uma vez que o importante em um programa é prioritariamente o desempenho? A esta pergunta, podemos responder da seguinte forma: vivemos em um mundo "incompleto", ou seja, que não pode ser completamente representado computacionalmente. Isto significa que, em alguns casos, é tão difícil fazer a modelagem iterativa que optamos por segmentar o problema em funções e usar recursão. Há outras vantagens do uso de recursão, como o processamento paralelo, a organização, e desvantagens como alto consumo de recursos.

Existem problemas que cujas as soluções em computação são inerentemente recursivas, já que elas precisam manter registros de estados anteriores. Um exemplo é são os algoritmos de divisão e conquista, tais como o *Quicksort*. Um algoritmo de divisão e

conquista consiste em dividir um problema maior em problemas menores até que o problema possa ser resolvido diretamente. Então a solução do problema inicial é dada através da combinação dos resultados de todos os problemas menores computados.

3. IMPLEMENTAÇÃO DE UM PROGRAMA RECURSIVO

As implementações recursivas devem ser pensadas considerando-se a definição recursiva do problema que desejamos resolver. Por exemplo, a função para calcular o intervalo de números pares entre 0 e um número n qualquer pode ser implementado da seguinte forma:

```
int numerosParesEntre0eN (int n)

{

   int c = 0;

   while(n>=0){

      if(n % 2 == 0){

         c++;

      }

      n--;

   }

   return c;

}

//Solucao recursiva...

int numerosParesEntre0eNRecursivo (int n)

{

   static int c = 0;

   if(n>=0){

      if(n % 2 == 0){

         c++;

      }

      n--;
```

```
        numerosParesEntre0eNRecursivo(n);
    }
    else
        return c;
}
void main ()
{
    int n;
    int r;
    printf("Digite o valor de n: ");
    scanf("%i",&n);
    r = numerosParesEntre0eN (n);
    printf("A quantidade de numeros pares entre 0 e  %d eh %d \n", n, r);
    r = numerosParesEntre0eNRecursivo (n);
    printf("(Recursivo)A quantidade de numeros pares entre 0 e  %d eh %d \n", n, r);
}
```