Recursividade

Definição

- Recursividade é uma técnica de fazer com que uma função ou procedimento chame a si mesmo

- A chamada recursiva é também conhecida como *chamada interna*

- Esta técnica deve ser empregada quando o problema em questão possui natureza recursiva:
  - O problema original pode ser decomposto em subproblemas
  - Cada subproblema possui a mesma lógica do problema original, só que aplicada a subconjunto dos dados
  - Ex: *Cálculo Fatorial*

Cálculo Fatorial

- Especificação:

  n! = 1, se n == 0

  n! = n * (n-1) * (n-2) * ... * 1, se n > 0.

- Ex:

  ```
  5! = 5 * 4 * 3 * 2 * 1 = 120
  ```

- Observamos que:

  ```
  5! = 5 * 4 * 3 * 2 * 1 = 120
            4! = 4 * 3 * 2 * 1 = 24
                      3! = 3 * 2 * 1 = 6
                                2! = 2 * 1 = 2
                                          1! = 1 = 1
  ```

Cálculo Fatorial

- Especificação:

  n! = 1, se n == 0

  n! = n * (n-1) * (n-2) * ... * 1, se n > 0.

- Ex:

  ```
  5! = 5 * 4 * 3 * 2 * 1 = 120
  ```

- Observamos que:

  ```
  5! = 5 * !4                 = 120
          4! = 4 * !3         = 24
                  3! = 3 * !2 = 6
                     2! = 2 * !1 = 2
                          1! = 1 = 1
  ```

- Logo:

  ```
  5! = 5 * 4! = 5 * 24 = 120
  ```

Considerações

- Importante:
  - Toda função recursiva precisa definir, pelo menos, um critério de parada!

- No caso do cálculo fatorial, consideramos que, quando n = 1, cessamos as chama das recursivas e retornamos 1

- Cada caso é reduzido a um caso mais simples até chegarmos ao caso 1!, que é definido imediatamente como 1

- É possível observar que funções recursivas podem ser criadas em quase todos os processos repetitivos para n elementos

Exemplo:

Soma dos n primeiros números inteiros

```
int Soma(int num);
void main(){
        int num, result;
        printf("Digite um n£mero qualquer: ");
        scanf("%d", &num);
        result = Soma(num);
        printf("Resultado = %d", result);
}
int Soma(int num){
        int aux;
        if (num == 0) aux = 0;
        else aux = num + Soma(num-1);
        return aux;
}
```

Exercício:

- Proponha uma solução recursiva para o Cálculo Fatorial

Exercício:

```
int Fatorial(int num){
       int aux;
       if (num < 0) return ERRO;
       if (num == 1 || num == 0) aux = 1;
       else aux = num * Fatorial(num-1);
       return aux;
}
void main(){
       int fat, result;
       printf("Digite um n£mero qualquer: ");
       scanf("%d", &fat);
       result = Fatorial(fat);
       printf("Resultado = %d", result);
       getch();
}
```

Mais Considerações

- Dissemos que a técnica de recursão é aplicável quando a solução de um problema envolve sub-soluções de mesmo procedimento (aplicação repetida da mesma lógica)

- Um procedimento recursivo terá pelo menos, dois passos fundamentais:
  1. Um passo onde o resultado é imediatamente conhecido
  2. Outro passo onde teremos a chamada do mesmo procedimento.

- Dissemos também que é uma técnica apropriada quando o problema a ser resolvido ou os dados a serem tratados são definidos em termos recursivos

- Entretanto, essa máxima não funciona sempre

Sequência de Fibonacci

- A seqüência de fibonacci é a seqüência de inteiros:

  0, 1, 1, 2, 3 , 5, 8, 13 , 21, 34, ...

- Cada elemento nessa seqüência é a soma dos dois elementos anteriores, como por exemplo, 0 + 1 = 1, 1 + 1 = 2, 1 + 2 = 3, 2 + 3 = 5, ...

- Se estabelecermos que `fib(0) = 0, fib(1) = 1`, então poderemos definir a seqüência de Fibonacci por meio da seguinte definição recursiva:

```
f(n) = n se n == 0 ou  n == 1
f(n) = fib(n-2) + fib(n-1) se n >= 2
```

Sequência de Fibonacci

```
int Fibo(int num){
    int aux;
    if (num < 0) return ERROR;
    if (num <=1) aux = num;
    else aux = Fibo(num - 2) + Fibo(num - 1) ;
    return aux;
}
```

Sequência de Fibonacci

```
int Fibo(int num){
    int aux;
    if (num < 0) return ERROR;
    if (num <=1) aux = num;
    else aux = Fibo(num - 2) + Fibo(num - 1) ;
    return aux;
}
```

- Observações:
  - A definição recursiva refere-se a si mesma 2 vezes
  - Ocorre redundância computacional ao aplicar a definição
  - É possível calcular fib(n) através de um método iterativo e esse método é muito mais eficiente

Sequência de Fibonacci

```
int Fibo(int num){
    int aux, fib0 = 0, fib1 = 1;
    if (num <=1) return num;
    for(int i=2; i<=num; i++){
      aux=fib0;
      fib0=fib1;
      fib1=aux+fib0;
    }
    return fib1;
}
```

Sequência de Fibonacci

- Comparação:

N	10	20	30	50	100
Recursivo	8 ms	1s	2 min	21 dias	10^9 anos
Iteração	1/6 ms	1/3 ms	1/2 ms	3/4 ms	1,5 ms

- Conclusão:
  - Devemos evitar o uso de recursividade quando existe uma solução iterativa óbvia

Atividade 1 (em sala):

- Considere a função abaixo, que calcula o elemento máximo de um vetor:

```
int maximo( int n, int v[]) {
  int j, x; x = v[0];
  for (j = 1; j < n; j += 1)
    if (x < v[j]) x = v[j];
  return x;
}
```

- Faz sentido realizar as seguntes alterações? Comente

1. `"x = v[0]"` por `"x = 0"`
2. `"x = v[0]"` por `"x = INT_MIN"`
3. `"x < v[j]"` por `"x <= v[j]"`

Atividade 1 (solução):

- Considere a função abaixo, que calcula o elemento máximo de um vetor:

```
int maximo( int n, int v[]) {
  int j, x; x = v[0];
  for (j = 1; j < n; j += 1)
    if (x < v[j]) x = v[j];
  return x;
}
```

- Faz sentido realizar as seguntes alterações? Comente

1. `"x = v[0]"` por `"x = 0"` (não funciona para valores negativos)
2. `"x = v[0]"` por `"x = INT_MIN"` (funciona mas é deselegante)
3. `"x < v[j]"` por `"x <= v[j]"` (realiza trocas desnecessárias)

Atividade 2 (em sala):

- A função abaixo promete calcular o elemento máximo de um vetor:

```
int maxi( int n, int v[]) {
   int j, m = v[0];
   for (j = 1; j < n; ++j)
      if (v[j-1] < v[j]) m = v[j];
   return m;
}
```

- Ela cumpre a promessa?

Atividade 2 (solução):

- A função abaixo promete calcular o elemento máximo de um vetor:

```
int maxi( int n, int v[]) {
   int j, m = v[0];
   for (j = 1; j < n; ++j)
      if (v[j-1] < v[j]) m = v[j];
   return m;
}
```

- Ela cumpre a promessa?
  - Não. A função compara os elementos do vetor dois-a-dois.
  - Faça um teste de mesa para verificar.

Atividade 3 (em sala):

- Qual o valor de X(4)?

```
int X( int n) {
  if (n == 1 || n == 2)
    return n;
  else
    return X( n-1) + n * X( n-2);
}
```

Atividade 3 (solução):

- Qual o valor de X(4)?

```
int X( int n) {
  if (n == 1 || n == 2)
    return n;
  else
    return X( n-1) + n * X( n-2);
}
```

n		X(n)
1		1
2		2
3	2 + 3*1 =	5
4	5 + 4*2 =	$\boxed{13}$

Exercício (Lab 7)

12. Sem recorrer ao material de aula, implemente as versões recursivas e iterativas para os seguintes problemas
    a. Cálculo da soma dos n primeiros inteiros positivos
    b. Cálculo do fatorial de n (n!)
    c. Cálculo do n-ésimo termo de Fibonacci
    d. Função que calcula o termo máximo de um vetor

    OBS: implemente um programa main que teste (execute) todas as funções acima.

Exercício (Lab 7)

13. Euclides. A seguinte função calcula o maior divisor comum dos inteiros positivos m e n. Escreva e implemente uma função recursiva equivalente.

```
int Euclides( int m, int n) {
  int r;
  do {
    r = m % n;
    m = n;
    n = r;
  } while (r != 0);
  return m;
}
```