RECURSIVIDADE

Técnica de programação, na qual um subprograma (função) chama a si mesmo.

Todas as funções recursivas devem encerrar a recursão a partir de um teste de valor ou condição!!!

Recursividade

➔ chamada condicional

(condição de fim da recursão)

```
void rec (x)
 {
 ...
 if ...
    rec (...);
 ...
```

➔ cada chamada ativa uma rotina no *stack*

➔ sempre pode ser substituído por programação com comandos iterativos

Vantagens da Recursividade

- código mais compacto
- especialmente conveniente para estruturas de dados definidas recursivamente tais como árvores
- código pode ser mais fácil de entender

Desvantagens da Recursividade

- maior ocupação de memória
- maior tempo de processamento

O que muda se inverter a ordem dos 2 comandos abaixo?

```
void  pares( int i, n);
{
    if (i < n) {
          pares( i+2,  n );
          printf("%d", i );
     }
    else
        { printf("%d", i ); };
}
```

pares(2,10);

10
8
6
4
2

```
printf("%d", i );
pares( I + 2,  n );
```

2
4
6
8
10

Fatorial de um número

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

Definição:

0! = 1;

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

Fatorial de um número

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

```
int fatorial (int n) // solução linear
{
  int fat;   // valor gerado
  int aux;  // variável auxiliar, para acumular o valor do fatorial
  fat = 1;
  for (aux = n; aux > 1; aux--) // se n=0 ou 1, não entra no for
    {
       fat = fat * aux ;
    }
  return fat; // encerra função, retornando valor
}
```

Fatorial de um número

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

4!

4! = 4 * 3 * 2 * 1 **= 4 * 3!**

3!

3! = 3 * 2 * 1 **= 3 * 2!**

2!

se N = 0 (ou 1)

fatorial de n ← 1

senão

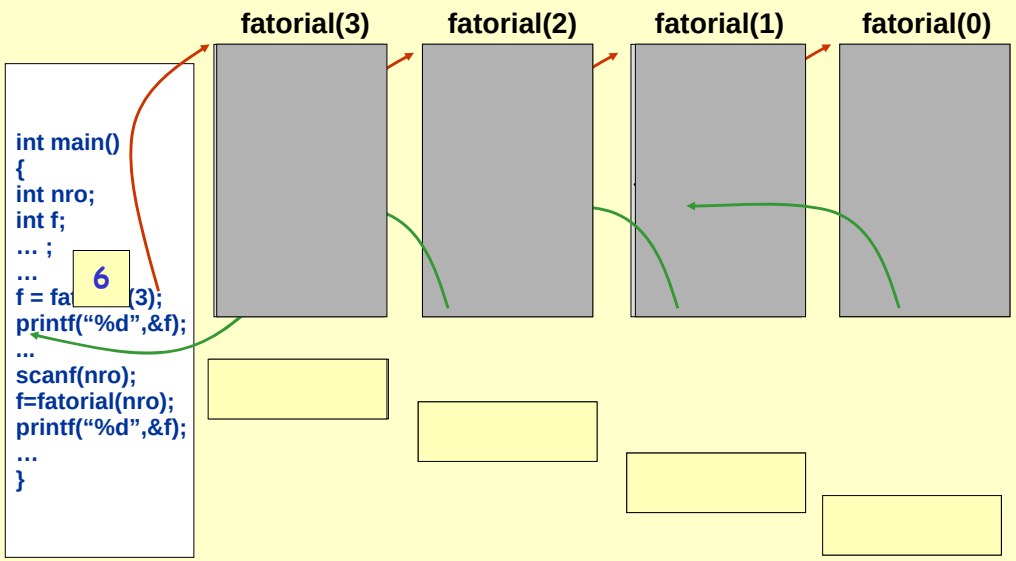
fatorial de n ← n * fatorial de n-1

Se N = 0 então Fatorial de N ← 1
Se N > 0 então
Fatorial de N ← N * fatorial de N-1

```
int fatorial (int n) // solução linear
{
  int fat;   // valor gerado
  int aux;  // variável auxiliar, para acumular o valor do fatorial
  fat = 1;
  for (aux = n; aux > 1; aux--) // se n=0 ou 1, não entra no for
    {
       fat = fat * aux ;
    }
  return fat;  // encerra função, retornando valor
}
```

```
int fatorial (int n) // solução recursiva
{
 int fat;
 if (n == 0) //com n<=1, evita execução adicional e funciona para 0
     fat = 1;  // encerra a recursão e inicia o retorno
 else
     fat = n * fatorial ( n - 1 ); // chamada recursiva
 return fat; // encerra função
} // fatorial
```

Fatorial de um número: execução



Execução de uma chamada de um subprograma recursivo

Início da execução

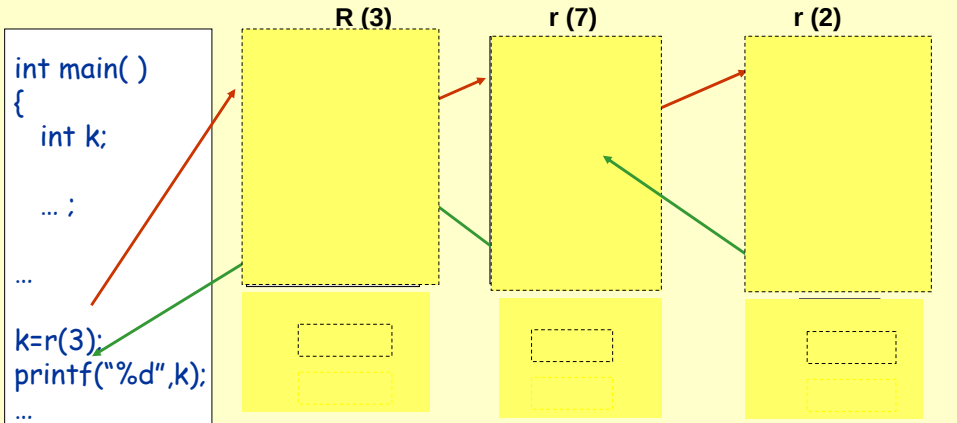
- alocação de variáveis locais
- parâmetros

Processamento

- se a execução for interrompida por nova chamada recursiva, ocorre o “empilhamento” dos indicadores de execução atuais (valores, ponto de retorno)

Fim da execução

- liberação de variáveis locais - “desempilhamento”
- retorno ao ponto de chamada




```
int r (int p)
{
  int x;
  printf(“informe valor:”);
  scanf(“%d”,&x);
  if (x != 5)
     return(r(x));
  else
     return(x);
}
```

Exercício: fazer um programa contendo uma função que calcula a soma dos n elementos inteiros de um vetor, de forma recursiva. No programa principal, ler os números e informar a soma.

```
…
#define N 10
int soma_rec (int v[ ], int ultimo)
{
  if (ultimo == 0)
      return v[0];
  else
      return v[ultimo] + soma_rec (v, ultimo - 1);
}
int main (void)
{
   int numeros[N];
   int i;
   for (i = 0; i < N; i++)
       scanf ("%d", &numeros[i]);  // lê valores: incluir printf explicativo

   printf("soma dos valores lidos eh %d", soma_rec(numeros, N-1) );
   system(“pause”);
   return 0;
}
```

Busca Recursiva em Arranjo

```
/* Retorna o índice no vetor onde está o valor
      ou -1,  caso o valor não esteja no vetor */

int busca(int vet[], int i, int f, int v)    {
  int k;
  if (i > f) //se inicio maior que o fim termina
    return -1;
  else {
    k = (i+f)/2; //busca a partir do meio do vetor
    if (vet[k] == v)
      return k;
    else
      if (vet[k] < v)
      return busca(vet,k+1, f, v); //busca parte
                                   //superior
      else
      return busca(vet,i,k-1 , v); //busca parte
                                   //inferior
  }   }
```

```
int main() {
  int k, v;
  int vet[10] = {3, 5, 7, 9, 12, 15, 16, 22, 26, 32};
  printf ("v: ");
  scanf("%d", &v);
   k = busca(vet, 0, 9, v);
   if (k == -1)
       printf("O valor %d nao está no vetor.\n", v);
  else
       printf("O valor %d esta na posicao %d.\n",
      v, k);
system ("pause");
return 0;
  }
```

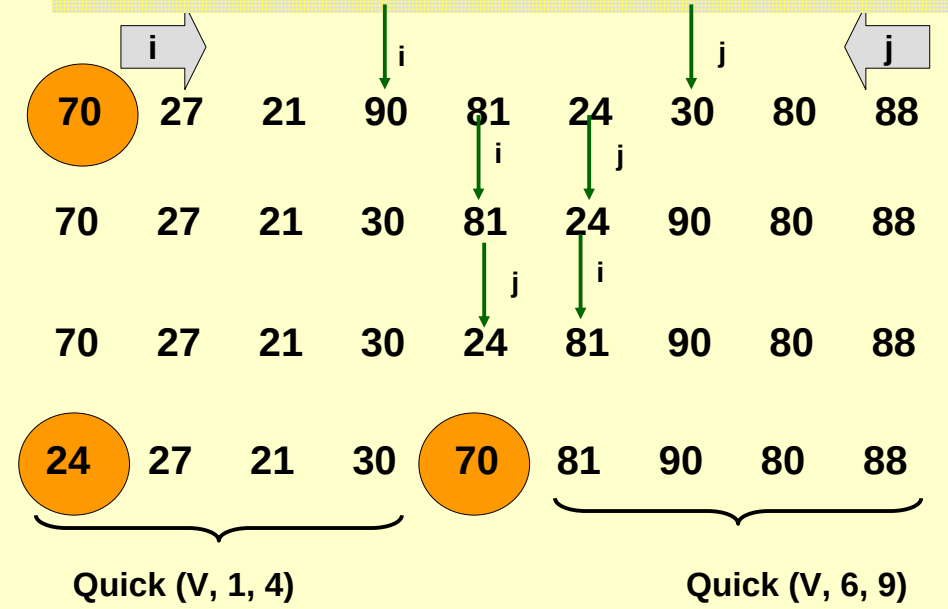
Busca Recursiva em Arranjo

```
V = 15
O valor 15 esta na posicao 5

Pressione qualquer tecla para continuar...


V= 14
O valor 14 não esta no vetor

Pressione qualquer tecla para continuar...
```

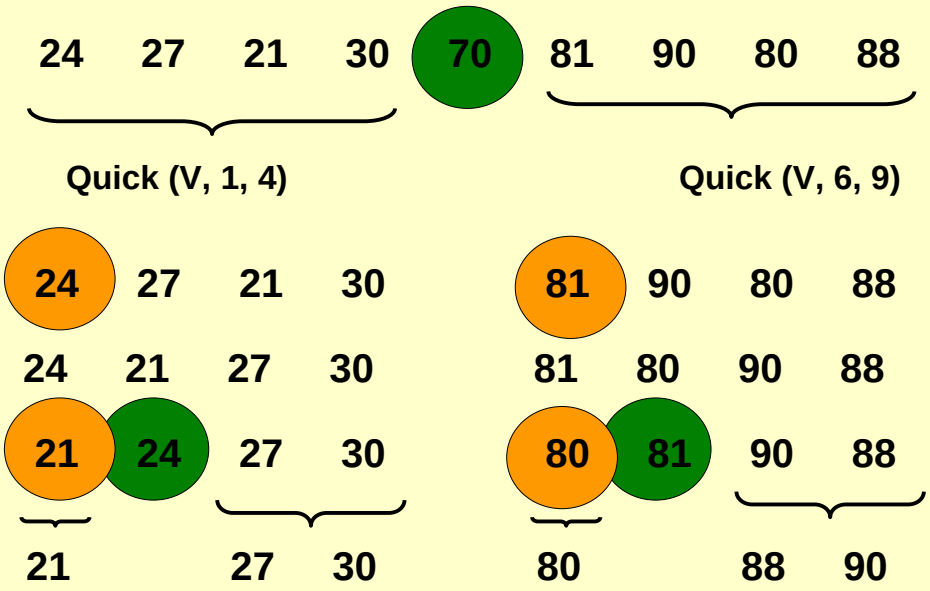
Quicksort

- Divide o problema de **ordenação** em problemas menores, até encontrar um caso que possa ser resolvido trivialmente.
- Um elemento do vetor é escolhido arbitrariamente (pivô) e divide-se a lista em duas listas menores: uma contendo os elementos menores que o pivô e outra com os elementos maiores que o pivô.
- Uma posição intermediária fica disponível para o próprio pivô
- Após a divisão da lista, ordena-se as duas sublistas geradas recursivamente.
- Encerra a recursão quando a sublista tem um único elemento ou está vazia

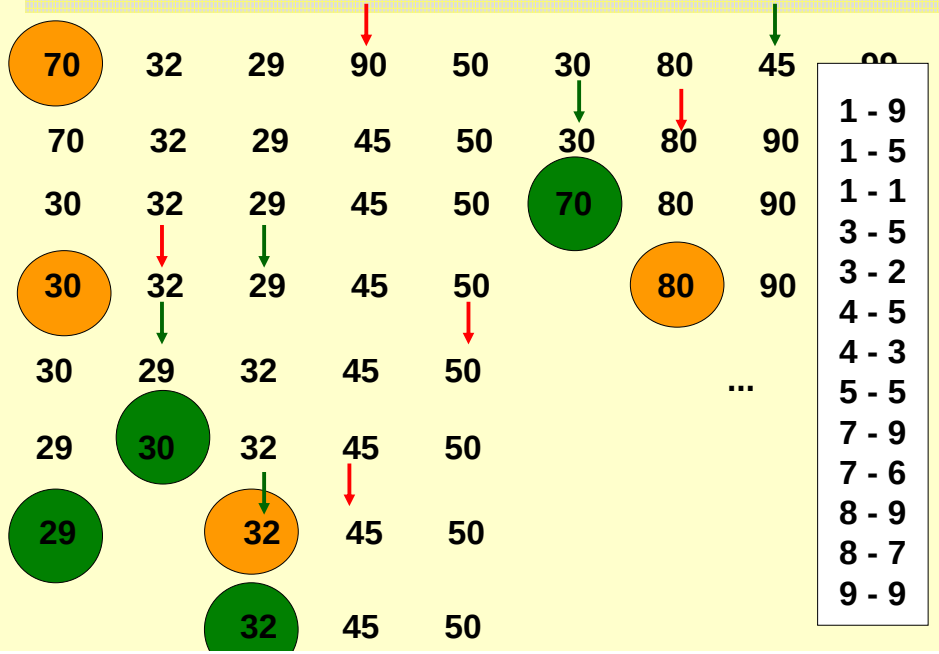
Ex 3 QuickSort



QuickSort



QuickSort



Quicksort

```
int separa (int v[ ], int p, int r) {

  int c = v[p]; /*Pivô*/
  int i = p+1;
  int j = r;
  int t;

  while (i <= j) {
    if (v[i] <= c) ++i;
    else if (c < v[j]) --j;
    else {  // inverte maior e menor
      t = v[i];
      v[i] = v[j];
      v[j] = t;
      ++i; --j;
      }   }
  t = v[p];
  v[p] = v[j];
  v[j] = t;
  for (i = 0; i < TAMANHO; i++)
            printf (", %2d", v[i]);
  }  return j;
}
```

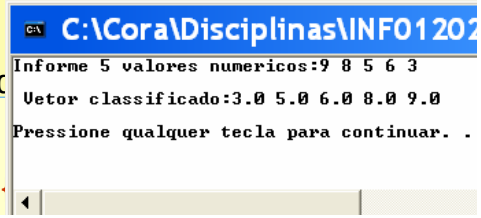
```
void quicksort (int v[  ], int p, int r) {
  int j;
  if (p < r) {
    j = separa (v, p, r);
    quicksort (v, p, j-1);
    quicksort (v, j+1, r);
  }
}

int main () {
  int vetor [TAMANHO] =
     {13,7,2,5,9,11,4,15,0,10,1,12,6,14,3,8};
  mostra_vetor(vetor);
  quicksort (vetor, 0, TAMANHO - 1);
  mostra_vetor(vetor);
  system ("pause");
  return 0;
}
```

Exemplos de problemas recursivos

- ordenar arranjo: colocar o menor elemento na primeira posição e ordenar o restante do arranjo
- inverter uma lista: trocar os dois extremos, e inverter o restante da lista

posição e repetir o processo para os demais elementos.
Faça o algoritmo e o programa que implemente este método, composto por uma função **recursiva** **ordena**, uma função **pos_menor** que retorna a posição do menor elemento de um vetor (vetor, posição inicial de busca e posição final de busca devem ser passados como parâmetro) e uma função **troca** que inverte o conteúdo de 2 variáveis passadas como parâmetros. No programa principal, incluir a leitura de um vetor de 10 elementos [illegible] mpressão do vetor

Algoritmo Ordena_Selecao;
{ordena vetores pelo método de seleção}
Entrada: vetor;
Saída: vetor classificado em ordem crescente;
1. Inicio
2. Para (ind ← 1;ind < num_elem; inc
lê vetor[ind]
3. Ordena(vetor,pos_inic, pos_fin)



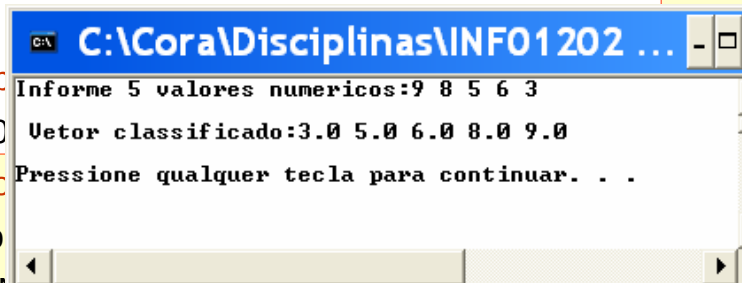
recursiva **ordena**, uma **função pos_menor** que retorna a posição do menor elemento de um vetor (vetor, posição inicial de busca e posição final de busca devem ser passados como parâmetro) e uma **função troca** que inverte o conteúdo de 2 variáveis passadas co

Algoritmo ordena (vetor, pos_inic, pos_fin);
{Identifica o menor elemento entre posição inicial e posição final e substitui este elemento pelo que está na posição inicial}
1. inicio
2. se pos_inic < pos_fin { mesma posição: está ordenado!}
então início
menor ← pos_menor(vetor, pos_inic, pos_fin);
troca(vetor[pos_inic],vetor[menor]);
{ordena o subvetor restante}

```
/* Ordena vetor - método seleção - recursivo */
#include <stdio.h>
#include <stdlib.h>
#define NUMELEM 5
void troca(float *a, float *b) // devolve conteúdos trocados
{
    float aux;
    aux = *a;
    *a = *b;
    *b = aux;
}

int pos_menor(float vet[NUMELEM], int pos_inic, int
   pos_final)
{
```

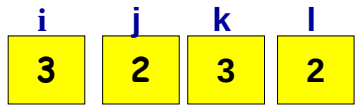
```
// função recursiva que ordena o vetor
void ordena(float vet[ ], int pos_inic, int pos_final)
 {
    int pos_elem_menor;
    if (pos_inic < pos_final) // se mesma posição, está
  classificado
     {
        pos_elem_menor =
  pos_menor(vet,pos_inic,pos_final);
        troca(&vet[pos_inic], &vet[pos_elem_menor]);
        // ordena o restante do vetor
        ordena(vet,pos_inic + 1, pos_final);
     }
 }
```

```
// programa principal:
int main ()
 {
 int ind;
 float vetor[NUMELEM];
 system ("color f0");
 //lê vetor
 printf ("Informe %d valores numericos:",NUMELEM);
 for(ind=0;ind<NUMELEM;ind++)
     scanf("%f",
 // envia vetor p
 ordena(vetor,0
 // imprime vetc
 printf ("\n Veto
```



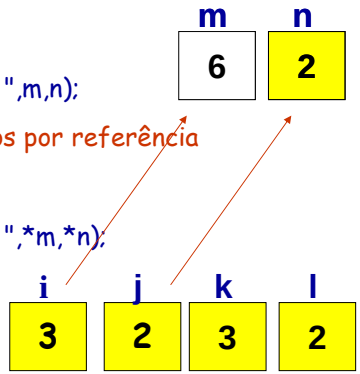
Exercício: Qual o resultado da execução do programa abaixo?

```
/*   Testa recursividade */
#include <stdio.h>
#include <stdlib.h>
void mult_val(int m, int n) // parâmetros por valor
{
  m = m*n;
  printf("\nPor valor: \n");
  printf("\nValores locais: m = %d, n = %d. ",m,n);
}
void mult_ref(int *m, int *n) // parâmetros por referência
{
  *m = *m * *n;
  printf("\nPor referência: \n");
  printf("\nValores locais: m = %d, n = %d. ",*m,*n);
}
int main()
{
  int i=3,j=2,k=3,l=2;
  system ("color f1");
  mult_val(i,j);
  mult_val(i,i); // var i usada para os 2 parâmetros
  mult_ref(&k,&l);
  mult_ref(&k,&k); // var k usada para os 2 parâmetros
  printf("\nValores finais: i = %d, j = %d, k = %d, l = %d. ",i,j,k,l);
  printf("\n");
  system("pause");
  return 0;
}
```



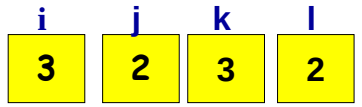
Exercício: Qual o resultado da execução do programa abaixo?

```
/*   Testa recursividade */
#include <stdio.h>
#include <stdlib.h>
void mult_val(int m, int n) // parâmetros por valor
{
  m = m*n;
  printf("\nPor valor: \n");
  printf("\nValores locais: m = %d, n = %d. ",m,n);
}
void mult_ref(int *m, int *n) // parâmetros por referência
{
  *m = *m * *n;
  printf("\nPor referência: \n");
  printf("\nValores locais: m = %d, n = %d. ",*m,*n);
}
int main()
{
  int i=3,j=2,k=3,l=2;
  system ("color f1");
  mult_val(i,j);
  mult_val(i,i); // var i usada para os 2 parâmetros
  mult_ref(&k,&l);
  mult_ref(&k,&k); // var k usada para os 2 parâmetros
  printf("\nValores finais: i = %d, j = %d, k = %d, l = %d. ",i,j,k,l);
  printf("\n");
  system("pause");
  return 0;
}
```



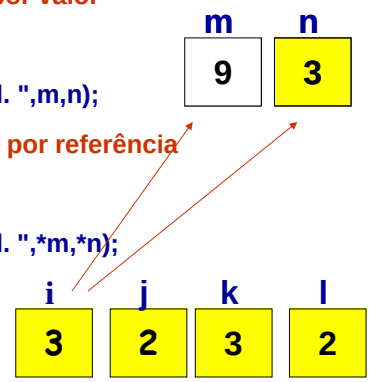
Exercício: Qual o resultado da execução do programa abaixo?

```
/*   Testa recursividade */
#include <stdio.h>
#include <stdlib.h>
void mult_val(int m, int n) // parâmetros por valor
{
  m = m*n;
  printf("\nPor valor: \n");
  printf("\nValores locais: m = %d, n = %d. ",m,n);
}
void mult_ref(int *m, int *n) // parâmetros por referência
{
  *m = *m * *n;
  printf("\nPor referência: \n");
  printf("\nValores locais: m = %d, n = %d. ",*m,*n);
}
int main()
{
  int i=3,j=2,k=3,l=2;
  system ("color f1");
  mult_val(i,j);
  mult_val(i,i); // var i usada para os 2 parâmetros
  mult_ref(&k,&l);
  mult_ref(&k,&k); // var k usada para os 2 parâmetros
  printf("\nValores finais: i = %d, j = %d, k = %d, l = %d. ",i,j,k,l);
  printf("\n");
  system("pause");
  return 0;
}
```



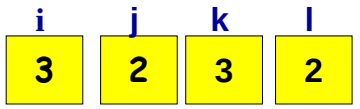
Exercício: Qual o resultado da execução do programa abaixo?

```
/*   Testa recursividade */
#include <stdio.h>
#include <stdlib.h>
void mult_val(int m, int n) // parâmetros por valor
{
  m = m*n;
  printf("\nPor valor: \n");
  printf("\nValores locais: m = %d, n = %d. ",m,n);
}
void mult_ref(int *m, int *n) // parâmetros por referência
{
  *m = *m * *n;
  printf("\nPor referência: \n");
  printf("\nValores locais: m = %d, n = %d. ",*m,*n);
}
int main()
{
  int i=3,j=2,k=3,l=2;
  system ("color f1");
  mult_val(i,j);
  mult_val(i,i); // var i usada para os 2 parâmetros
  mult_ref(&k,&l);
  mult_ref(&k,&k); // var k usada para os 2 parâmetros
  printf("\nValores finais: i = %d, j = %d, k = %d, l = %d. ",i,j,k,l);
  printf("\n");
  system("pause");
  return 0;
}
```



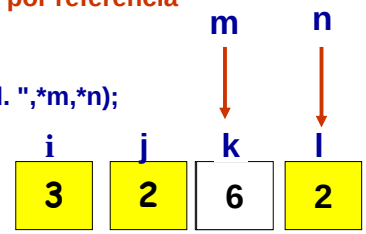
Exercício: Qual o resultado da execução do programa abaixo?


```
/*   Testa recursividade */
#include <stdio.h>
#include <stdlib.h>
void mult_val(int m, int n) // parâmetros por valor
{
  m = m*n;
  printf("\nPor valor: \n");
  printf("\nValores locais: m = %d, n = %d. ",m,n);
}
void mult_ref(int *m, int *n) // parâmetros por referência
{
  *m = *m * *n;
  printf("\nPor referência: \n");
  printf("\nValores locais: m = %d, n = %d. ",*m,*n);
}
int main()
{
  int i=3,j=2,k=3,l=2;
  system ("color f1");
  mult_val(i,j);
  mult_val(i,i); // var i usada para os 2 parâmetros
  mult_ref(&k,&l);
  mult_ref(&k,&k); // var k usada para os 2 parâmetros
  printf("\nValores finais: i = %d, j = %d, k = %d, l = %d. ",i,j,k,l);
  printf("\n");
  system("pause");
  return 0;
}
```

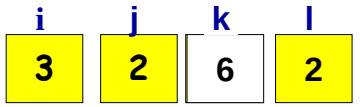
programa abaixo?


```
/*   Testa recursividade */
#include <stdio.h>
#include <stdlib.h>
void mult_val(int m, int n) // parâmetros por valor
{
  m = m*n;
  printf("\nPor valor: \n");
  printf("\nValores locais: m = %d, n = %d. ",m,n);
}
void mult_ref(int *m, int *n) // parâmetros por referência
{
  *m = *m * *n;
  printf("\nPor referência: \n");
  printf("\nValores locais: m = %d, n = %d. ",*m,*n);
}
int main()
{
  int i=3,j=2,k=3,l=2;
  system ("color f1");
  mult_val(i,j);
  mult_val(i,i); // var i usada para os 2 parâmetros
  mult_ref(&k,&l);
  mult_ref(&k,&k); // var k usada para os 2 parâmetros
  printf("\nValores finais: i = %d, j = %d, k = %d, l = %d. ",i,j,k,l);
  printf("\n");
  system("pause");
  return 0;
}
```

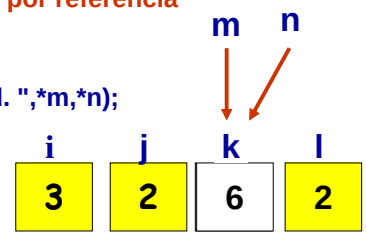
programa abaixo?


```
/*   Testa recursividade */
#include <stdio.h>
#include <stdlib.h>
void mult_val(int m, int n) // parâmetros por valor
{
  m = m*n;
  printf("\nPor valor: \n");
  printf("\nValores locais: m = %d, n = %d. ",m,n);
}
void mult_ref(int *m, int *n) // parâmetros por referência
{
  *m = *m * *n;
  printf("\nPor referência: \n");
  printf("\nValores locais: m = %d, n = %d. ",*m,*n);
}
int main()
{
  int i=3,j=2,k=3,l=2;
  system ("color f1");
  mult_val(i,j);
  mult_val(i,i); // var i usada para os 2 parâmetros
  mult_ref(&k,&l);
  mult_ref(&k,&k); // var k usada para os 2 parâmetros
  printf("\nValores finais: i = %d, j = %d, k = %d, l = %d. ",i,j,k,l);
  printf("\n");
  system("pause");
  return 0;
}
```

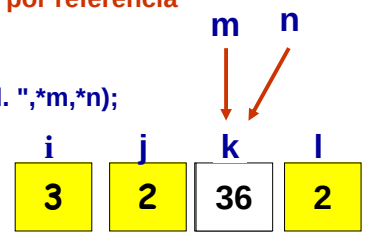
Exercício: Qual o resultado da execução do programa abaixo?

```
/*   Testa recursividade */
#include <stdio.h>
#include <stdlib.h>
void mult_val(int m, int n) // parâmetros por valor
{
  m = m*n;
  printf("\nPor valor: \n");
  printf("\nValores locais: m = %d, n = %d. ",m,n);
}
void mult_ref(int *m, int *n) // parâmetros por referência
{
  *m = *m * *n;
  printf("\nPor referência: \n");
  printf("\nValores locais: m = %d, n = %d. ",*m,*n);
}
int main()
{
  int i=3,j=2,k=3,l=2;
  system ("color f1");
  mult_val(i,j);
  mult_val(i,i); // var i usada para os 2 parâmetros
  mult_ref(&k,&l);
  mult_ref(&k,&k); // var k usada para os 2 parâmetros
  printf("\nValores finais: i = %d, j = %d, k = %d, l = %d. ",i,j,k,l);
  printf("\n");
  system("pause");
  return 0;
}
```



Exercício: Qual o resultado da execução do programa abaixo?

```
/*   Testa recursividade */
#include <stdio.h>
#include <stdlib.h>
void mult_val(int m, int n) // parâmetros por valor
{
  m = m*n;
  printf("\nPor valor: \n");
  printf("\nValores locais: m = %d, n = %d. ",m,n);
}
void mult_ref(int *m, int *n) // parâmetros por referência
{
  *m = *m * *n;
  printf("\nPor referência: \n");
  printf("\nValores locais: m = %d, n = %d. ",*m,*n);
}
int main()
{
  int i=3,j=2,k=3,l=2;
  system ("color f1");
  mult_val(i,j);
  mult_val(i,i); // var i usada para os 2 parâmetros
  mult_ref(&k,&l);
  mult_ref(&k,&k); // var k usada para os 2 parâmetros
  printf("\nValores finais: i = %d, j = %d, k = %d, l = %d. ",i,j,k,l);
  printf("\n");
  system("pause");
  return 0;
}
```



Exercício: Qual o resultado da execução do programa abaixo?

```
/*   Testa recursividade */
#include <stdio.h>
#include <stdlib.h>
void mult_val(int m, int n) // parâme
{
  m = m*n;
  printf("\nPor valor: \n");
  printf("\nValores locais: m = %d, n
}
void mult_ref(int *m, int *n) // parâm
{
  *m = *m * *n;
  printf("\nPor referência: \n");
  printf("\nValores locais: m = %d, n = %d. ",*m,*n);
}
int main()
{
  int i=3,j=2,k=3,l=2;
  system ("color f1");
  mult_val(i,j);
  mult_val(i,i); // var i usada para os 2 parâmetros
  mult_ref(&k,&l);
  mult_ref(&k,&k); // var k usada para os 2 parâmetros
  printf("\nValores finais: i = %d, j = %d, k = %d, l = %d. ",i,j,k,l);
  printf("\n");
  system("pause");
  return 0;
}
```

```
Por valor:

Valores locais: m = 6, n = 2.
Por valor:

Valores locais: m = 9, n = 3.
Por referÛncia:

Valores locais: m = 6, n = 2.
Por referÛncia:

Valores locais: m = 36, n = 36.
Valores finais: i = 3, j = 2, k = 36, l = 2.
Pressione qualquer tecla para continuar. . .
```