Lógica de Programação
Funções

Revisão função por referência

```
programa
{

    funcao inicio()
    {
        inteiro a = 1000

        alterarValor(a)
        escreva("a:",a)
    }

    funcao alterarValor(inteiro &v){
        v=1500
        escreva("v:",v,"\n")
    }
}
```

Revisão função por referência

```
programa
{

    funcao inicio()
    {
      inteiro a, vet[10]={5,10,13,10,9,10,10,9,3,2}
      troca(vet,0)
      imprime(vet)
    }

    funcao troca (inteiro &v[], inteiro i){
        v[i]=10
    }

    funcao imprime(inteiro vet[]){
      para(inteiro i=0; i < 10; i++){
        escreva(vet[i],"\n")
      }

    }
}
```

Revisão função por referência

```
programa
{
    funcao inicio()
    {
        cadeia carros[3][3] = {{"KIO-1090","HB20","Sim"},
                               {"ABC-9003","RENEGADE","Não"},
                               {"KAB-230F","KWID","Sim"}}
        alterarDadosCarros(carros)
        //exibirCarros(carros)

    }

    funcao exibirCarros(cadeia carros[][]){
            para(inteiro linha=0; linha < 3; linha++){
                para(inteiro coluna=0; coluna < 3; coluna++){
                    escreva(carros[linha][coluna],"\n")
                }
                escreva("----------------\n")
            }

    }

    funcao alterarDadosCarros(cadeia &matriz[][]){
        matriz[2][2] = "Não"
        exibirCarros(matriz)

    }
}
```

Escreva um programa para entrada de dados em uma matriz[2][3] do tipo cadeia que contenha as seguintes opções:

0 - Fim programa

1 - Entrada de dados (nome, telefone, não)
Deverá ser digitado o nome, telefone e atribuido "não" na coluna porque a pessoa ainda não foi vacinada

2 - Vacinação
Deverá ser procurado na matriz o nome da pessoa caso ele exista a pessoa deverá ser vacinada exibindo a mensagem "vacinação efetuada" alterando a opção na matriz para "sim" caso não exista exibir a mensagem "paciente inexistente"

3- Imprimir a matriz
Obs: As opções 1, 2 e 3 deverão ser utilizadas funções.

Recursividade

Uma função recursiva é uma função que chama a si mesma.

Exemplo:

No algoritmo abaixo, temos uma estrutura de um para que imprime do 20 até 1.

```
programa
{

    funcao inicio()
    {
        para(inteiro i=20; i > 0; i--)
        {
            escreva(i,"\n")
        }
    }
}
```

Podemos fazer a estrutura acima utilizando a recursividade.

```
programa
{

    funcao exibir(inteiro i){
        se (i == 0) {
            escreva(i)
        }senao{
            escreva(i, ", ")
            exibir(i - 1)
        }
    }


    funcao inicio()
    {
        exibir(20)
    }
}
```

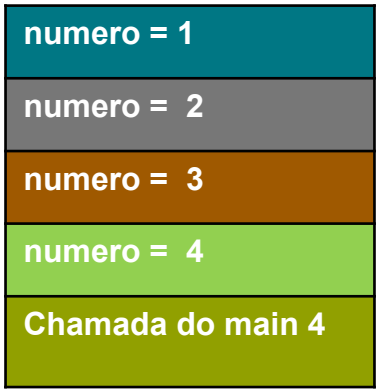
A recursividade sempre tem que ter um ponto de parada para não entrar em loop. No nosso exemplo ao lado é a condição até que **i** for igual a zero.

Entendendo o funcionamento de uma pilha

São estruturas de dados do tipo LIFO (last-in first-out), onde o último elemento a ser inserido, será o primeiro a ser retirado. Assim, uma pilha permite acesso a apenas um item de dados que será o último inserido.

Quando ele faz uma chamada os valores são colocados em uma pilha na memória e no final esses valores vão sendo desempilhados.

```
programa
{

    funcao inicio()
    {
        inteiro numero
        escreva("Digite o número:")
        leia(numero)
        escreva("Resultado:" + somarNumerosAnteriores(numero))
    }

    funcao inteiro somarNumerosAnteriores(inteiro numero) {
        inteiro resultado
        se (numero <= 1) {
            retorne 1
        }senao{
            resultado = somarNumerosAnteriores(numero -1 ) + numero
            retorne resultado
        }
    }
}
```



Entendendo o funcionamento de uma pilha

Desempilhando os valores fazendo a soma dos valores.

```
programa
{

    funcao inicio()
    {
        inteiro numero
        escreva("Digite o número:")
        leia(numero)
        escreva("Resultado:" + somarNumerosAnteriores(numero))
    }

    funcao inteiro somarNumerosAnteriores(inteiro numero) {
        inteiro resultado
        se (numero <= 1) {
            retorne 1
        }senao{
            resultado = somarNumerosAnteriores(numero -1 ) + numero
            retorne resultado
        }
    }
}
```

numero = 1
numero = 2 + 1 = 3
numero = 3 + 3 = 6
numero = 4 + 6 = 10
resultado = 10

Exemplo fatorial não recursivo

```
programa
{

    funcao inicio()
    {
        inteiro numero, fatorial, resultado=1
        escreva("Digite o número:")
        leia(numero)

        para(fatorial=1; fatorial<=numero ; fatorial ++){
            resultado = resultado * fatorial
        }

        escreva("O fatorial de ", numero, " é:", resultado)

    }
}
```

Exemplo fatorial recursivo

```
programa
{
    funcao inteiro fatorial(inteiro i){
        se ( i<=1){
            retorne 1
        }senao{
            i = i * fatorial(i-1)
            retorne i
        }
    }

    funcao inicio()
    {
        inteiro numero
        escreva("Digite um número:")
        leia(numero)
        escreva("O Fatorial de ",numero," é: ", fatorial(numero))
    }

}
```

Exemplo procurar menor valor em um vetor

```
programa
{
    inteiro aux, menor, vetor[] = {3,1,5,9}

    funcao inicio()
    {
        escreva("O menor valor é:", procurarMenorValor(3, vetor))
    }

    funcao inteiro procurarMenorValor(inteiro ultimaPosicao, inteiro vetor[]){

        se(ultimaPosicao == 0){
            retorne vetor[ultimaPosicao]
        }senao{
            aux = vetor[ultimaPosicao]
            menor = procurarMenorValor(ultimaPosicao -1, vetor)
            se (aux < menor) {
                menor = aux
            }
            retorne menor
        }


    }
}
```

ultimaPosicao=0, retorna 3

ultimaPosicao=1, aux =1, menor = ?

ultimaPosicao=2, aux =5, menor = ?

ultimaPosicao=3, aux =9, menor = ? (como é uma chamada recursiva só vamos saber o valor do menor quando chegarmos no final do vetor)

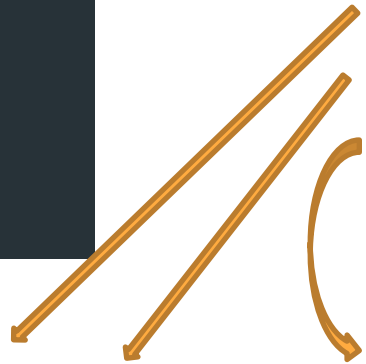
ultimaPosicao=3, vetor

Exemplo procurar menor valor em um vetor desempilhando valores.

```
programa
{
    inteiro aux, menor, vetor[] = {3,1,5,9}

    funcao inicio()
    {
        escreva("O menor valor é:", procurarMenorValor(3, vetor))
    }

    funcao inteiro procurarMenorValor(inteiro ultimaPosicao, inteiro vetor[]){

        se(ultimaPosicao == 0){
            retorne vetor[ultimaPosicao]
        }senao{
            aux = vetor[ultimaPosicao]
            menor = procurarMenorValor(ultimaPosicao -1, vetor)
            se (aux < menor) {
                menor = aux
            }
            retorne menor
        }


    }
}
```

No topo da pilha temos o retorno do valor 3 na posição 0 que será recebido pela variável **menor** que foi quem chamou na segunda linha da tabela abaixo, a partir desse ponto o código continua fazendo a comparação aux < menor

ultimaPosicao=0, retorna 3
ultimaPosicao=1, aux =1, menor = 3
ultimaPosicao=2, aux =5, menor = 1
ultimaPosicao=3, aux =9, menor = 1
main retorna 1



se (1 < 3){
menor = 1
}
retorne **menor**

//A variável menor é retornada para chamada anterior da pilha

se (5 < 1){
menor = 1
}
retorne **menor**

//Continua retornando menor com o valor 1

se (9 < 1){
menor = 1
}
retorne **menor**

//Continua retornando menor com o valor 1