**Engenharia Informática**

Estruturas de Dados

2019/2020

**Relatório**

**Recursividade**

**António Pereira Nº2019141051**

**David Ferreira Nº2019141281**

**Data: 10/06/2020**

Índice

[1 Introdução 3](#_Toc43403696)

[2 Trabalho realizado 4](#_Toc43403697)

[2.1 Calcular o factorial de um número inteiro, sabendo que *n*! = *n* x (*n* – 1) x (*n* – 2) x ... x 2 x 1. 4](#_Toc43403698)

[2.2 Calcular o *n*–esimo número de Fibonacci sabendo que os *Números de Fibonacci* são termos de uma sucessão (com o mesmo nome) definida por recorrência do seguinte modo: cada termo da sucessão e obtido pela soma dos dois temos anteriores. Os primeiros números de Fibonacci são: 1, 1, 2, 3, 5, 8, 13, 21, … 5](#_Toc43403699)

[2.3 Inverter a ordem dos elementos de um vector de numeros inteiros: 7](#_Toc43403700)

[2.4 Escrever uma sequência de algarismos, dado um n ∈ IN0 9](#_Toc43403701)

[2.5 Limpar uma lista ligada (Ficha de Trabalho n.o 5, alínea n) do exercício 2.), ou seja, eliminar todos os elementos de uma lista ligada. Uma das versões pode ser a usada na resolução da Ficha de Trabalho n.o 5. 11](#_Toc43403702)

[3 Conclusão 18](#_Toc43403703)

[4 Referências 19](#_Toc43403704)

# Introdução

Este trabalho é uma introdução a recursividade e como ela funciona. Neste trabalho vamos desenvolver programas utilizando a recursividade que é basicamente quando dentro de uma função é chamada essa mesma função. Também nesta ficha de trabalho é pedido que criemos uma versão iterativa que nada mais é que uma versão normal de um programa.

# Trabalho realizado

## Calcular o factorial de um número inteiro, sabendo que *n*! = *n* x (*n* -1) x (*n* – 2) x ... x 2 x 1.

Neste primeiro exercício é pedido para calcular o fatorial de um número usando a recursividade e a versão iterativa. Na versão recursiva criamos uma função em que dentro da função é retornado esta mesma função, que tem como parâmetro o n-1, vezes o n que é o número que o utilizador introduz para calcular o fatorial. A função recursiva é a seguinte:

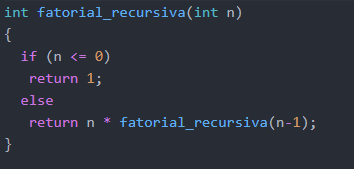


Figura 1- Função recursiva

Já na função iterativa para calcular o fatorial de um número fizemos um for e dentro do for multiplicamos a variável fatorial, que inicialmente é igual a 1, pelo i e a variável fatorial vai ser igual a este valor, deste modo, a variável fatorial vai ficar sempre com o valor da multiplicação, ou seja, vai aumentado o valor e deste modo conseguimos saber o valor do fatorial do número que o utilizador inseriu. Podemos ver a versão iterativa na figura abaixo:

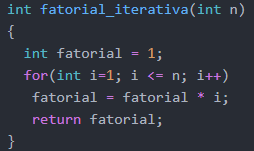


Figura 2- Versão iterativa para calcular o fatorial

O output deste programa é o seguinte:

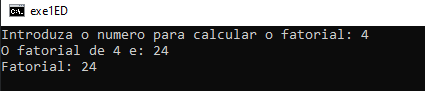


Figura 3- Interface da função recursiva e da função iterativa

## Calcular o *n*–esimo número de Fibonacci sabendo que os *Números de Fibonacci* são termos de uma sucessão (com o mesmo nome) definida por recorrência do seguinte modo: cada termo da sucessão e obtido pela soma dos dois temos anteriores. Os primeiros números de Fibonacci são: 1, 1, 2, 3, 5, 8, 13, 21, …

Neste segundo exercício temos que calcular a serie de Fibonacci de um número que é introduzido pelo utilizador. Na função recursiva, se a variável n, que é o número que o utilizador inseriu, for igual a 1 ou 2 a função retorna 1 senão vai retornar esta mesma função, que tem como parâmetro (n - 1), mais outra vez a chamada da mesma função só que desta vez o parâmetro é (n -2). A função é a seguinte:

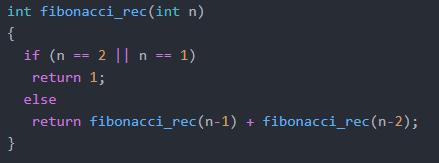
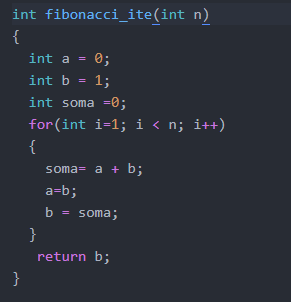


Figura 4- Função recursiva para calcular a serie de Fibonacci

Na função iterativa, para calcular a serie de fibonacci fizemos um for e neste for a variável soma vai ser igual a variável a mais a variável b, ou seja, vai ser a soma dos dois últimos termos, pois, a variável a vai ser sempre igual a variável b e o b vai ser igual a soma até o i ser maior que o n. A função iterativa é a seguinte:

Figura 5- Função iterativa



Output das funções recursiva e iterativa:

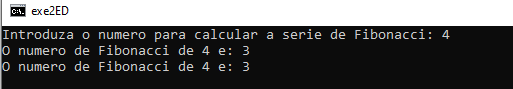


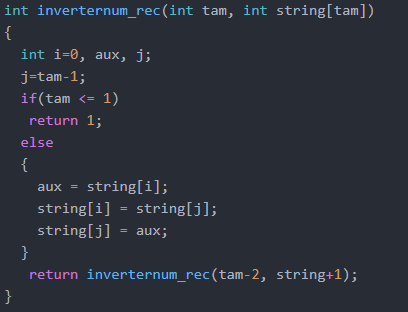
Figura 6- Interface das duas funções para calcular a serie de fibonacci

## Inverter a ordem dos elementos de um vector de numeros inteiros:

Neste exercício, primeiro para inverter a ordem de um vetor tem que se inserir os números do vetor. Para isso, fizemos uma função em que pede ao utilizador para inserir os números do vetor que tem como tamanho o que o utilizador inseriu no início do programa.

Na função recursiva, se o tamanho do vetor for igual ao menor que 1 a função vai retornar 1, pois, não da para inverter um vetor se tiver só um elemento. Caso seja maior que um, então, o vetor vai ser invertido e para isto, primeiro a variável aux vai ser igual ao valor do elemento da string na posição i, depois, a string na posição i vai ser igual ao valor do elemento na posição j, isto é, a variável string[i] vai ter sempre o valor dos últimos elementos do vetor, e por ultimo a variável string[j] vai ser igual a variável aux que vai ter o valor do elemento na posição i, ou seja, a variável string[j] vai ter sempre o valor dos primeiros elementos do vetor. No fim, vai ser retornado a função que tem como argumentos tam-2 e string +1, ou seja, como j é igual a tam -1 e tam vai ser sempre decrementado duas vezes então o valor de j vai sempre diminuir e deste modo o valor da string na posição j vai ser sempre alterado. Já quando é feito string +1 é para a string andar para a frente. A função é a seguinte:

Figura 7- Função para inverter a ordem de um vetor com recursividade



Na função iterativa é realizado um for em que o m vai ser igual a última posição do vetor e enquanto m for maior ou igual a zero vai ser realizado um printf da string na posição m, ou seja, vai ser apresentado os elementos da string na ordem invertida. A função iterativa é a seguinte:

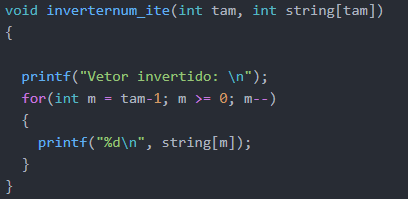


Figura 8- Função iterativa para inverter a ordem de um vetor

O output das duas funções é o seguinte:

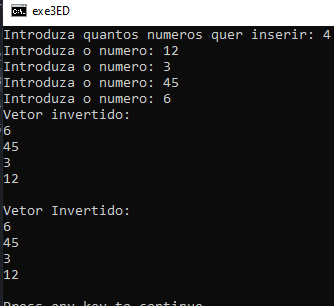


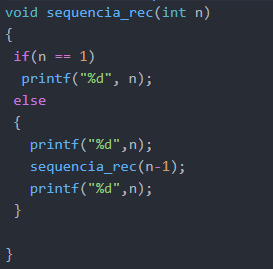
Figura 9- Vetor invertido

## Escrever uma sequência de algarismos, dado um n ∈ IN0

Para este exercício, foi-nos pedido para escrevermos uma sequência de algarismos dado um n introduzido pelo utilizador, ou seja, se o utilizador inserir o número 4 como n a função tem que apresentar 4, 3, 2, 1, 2, 3, 4.

Na função recursiva, se o n for igual vai apresentar o 1 se for maior que 1 então vai ser realizado um printf do valor do n, depois a função vai retornar n -1 e vai ser realizado outra vez um printf ate o n ser 1. Quando o n for 1 então o printf a seguir a chamada da função vai apresentar os valores do n que ficaram esquecidos, isto é, se o n for igual 4 vai escrever o 2, 3,4 pois o 4,3,2,1 já foram escritos no primeiro printf. A função recursiva é a seguinte:

Figura 10- Função recursiva para escrever uma sequência



Para escrever a sequência de algarismos na função iterativa realizamos dois whiles. No primeiro enquanto o n for maior que 1, primeiro vai ser escrito o valor de n e depois o n é decrementado, ou seja, se o n for 4 passa a 3 e assim sucessivamente ate o n ser igual a 1 que é quando o while para. Já no segundo while é onde vai ser escrito segunda parte da sequência, ou seja, enquanto n for menor ou igual a m, variável declarado no início da função e que é igual a n, vai ser apresentado o valor de n e depois incrementado o n, ou seja, como sabemos que o n é 1 depois do primeiro while então o n vai passar a 2 depois a 3 e por ultimo a 4, caso o utilizador no inicio desse o valor de 4 ao n. Na figura abaixo podemos ver a função iterativa:

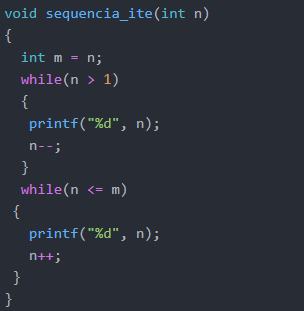
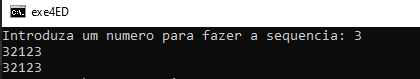


Figura 11- Função iterativa

O output destas duas funções é o seguinte:

Figura 12- Sequencia de algarismos caso o n seja 3

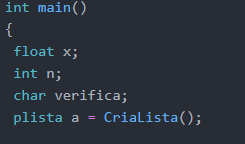


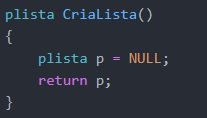
## Limpar uma lista ligada (Ficha de Trabalho n.o 5, alínea n) do exercício 2.), ou seja, eliminar todos os elementos de uma lista ligada. Uma das versões pode ser a usada na resolução da Ficha de Trabalho n.o 5.

Neste último exercício, foi-nos pedido que limpássemos uma lista ligada, isto é, eliminar todos os elementos de uma lista. Para fazer isto, primeiro tivemos que criar uma lista vazia que é basicamente criar uma variável do tipo plista e igualá-la a NULL. A função não tem argumentos, pois no main foi chamada da seguinte forma:

Figura 13- Função para criar lista

Figura 14- Chamada da função CriaLista

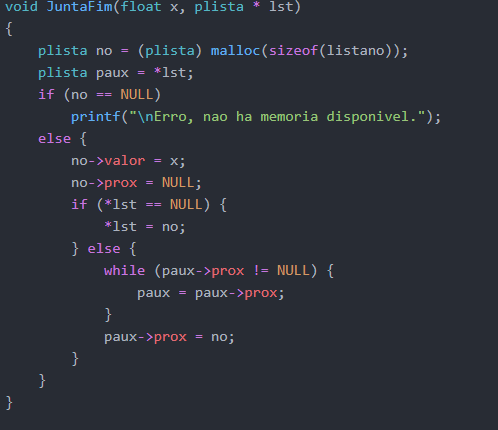




Depois de criada a lista tivemos que fazer uma função para inserir os elementos na lista. Nesta função, pedimos ao utilizador o número de elementos da lista e realizamos um for onde vai ser chamada a função JuntaFim, que é a função para inserir um elemento na lista. Na função JuntaFim, caso a lista esteja vazia, o primeiro elemento será o último. Assim, nesse caso basta igualar o nó ao primeiro elemento da lista e fazer com que o ponteiro passe a apontar para NULL. Caso a lista não esteja vazia, a, temos que percorrer a lista até encontrarmos o último elemento e igualar este último elemento ao que nós queremos colocar lá e passar o valor do último elemento para o lado esquerdo.

No entanto, para o elemento ser, sem dúvida, o último, o ponteiro terá que apontar para NULL. Para não perdermos a lista, criámos um ponteiro auxiliar que aponta para o primeiro elemento da lista. A função para inserir um elemento na lista:

Figura 15- Função JuntaFim que serve para inserir um elemento na lista



Função para inserir mais que um elemento na lista:

Figura 16- Função que serve para inserir mais que um elemento na lista

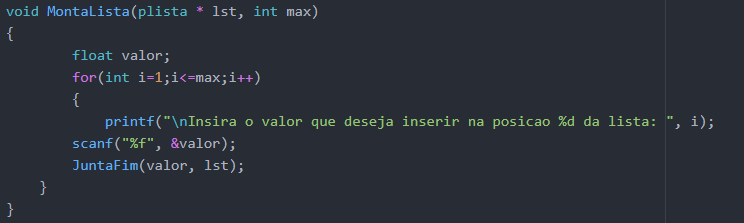
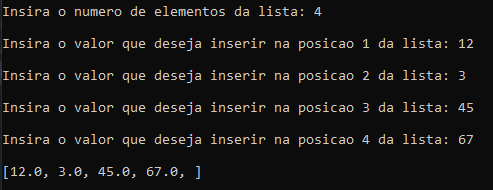
 Output da função MontaLista:

Figura 17- Elementos da lista



Para apresentar os elementos da lista também criamos uma função. Esta função permite o utilizador e nós ver os elementos da lista e ver se tudo correu bem. A função é a seguinte:

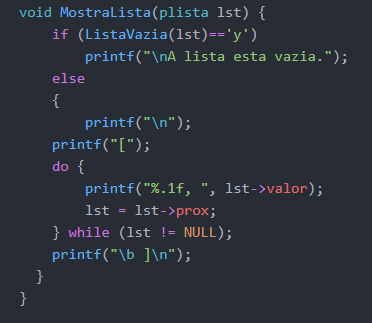


Figura 18- Função para apresentar os elementos de uma lista

Output desta função:

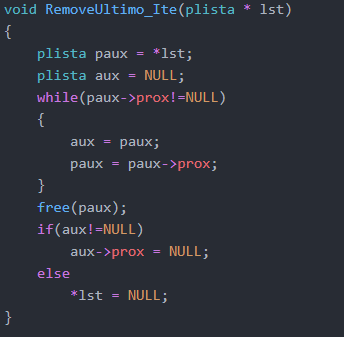
Figura 19- Interface da função MostraLista



Após criarmos a função para inserir os elementos numa lista, fizemos duas funções, uma iterativa outra recursiva, para limpar a lista.

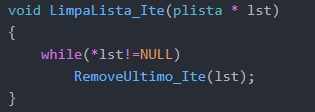
Na função iterativa, fizemos um while e dentro deste é chamada a função RemoveUltimo\_Ite, que é a função, que serve para remover um elemento da lista. Nesta função, utilizamos dois ponteiros em que o ponteiro paux aponta para o início da lista e o ponteiro aux é uma lista vazia. Deste modo, caso a lista não esteja vazia, o ponteiro aux irá andar “atrás” do ponteiro paux, de modo a que, quando é encontrado o último elemento, se possa eliminar o paux, utilizando o free(paux) e, caso a lista tenha mais que um elemento, o aux passa a apontar para NULL, ou seja, passa a ser o último elemento. Caso só tenha um elemento, a lista passa a ser vazia. A função para remover um elemento da lista é a seguinte:

Figura 20- Função que remove um elemento da lista



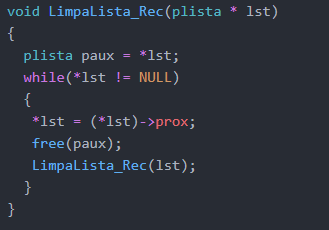
Função iterativa para limpar a lista:

Figura 21- Função que serve para limpar lista



Já na versão recursiva, primeiro criamos um ponteiro auxiliar que aponta para o início da lista, depois, fizemos um while e dentro deste a lista vai andar para a frente através da instrução (\*lst = (\*lst)->prox), vai ser eliminado o valor de paux através do free e ainda dentro do while vai ser chamada esta mesma função que tem como argumento a lista. A função recursiva é a seguinte:

Figura 22- Função recursiva para eliminar os elementos da lista



Output das duas funções para eliminar os elementos de uma lista:

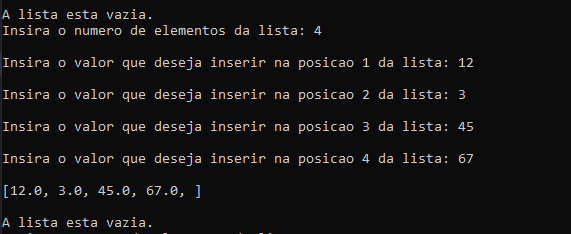


Figura 23- Output da função recursiva

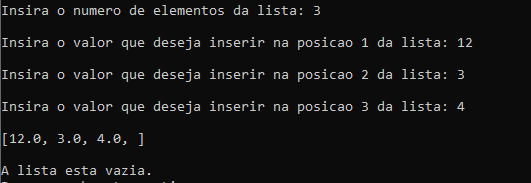


Figura 24- Output da função iterativa

# Conclusão

Concluindo, este trabalho foi interessante e útil para aprendermos a utilizar a recursividade que ao início parecia confuso, mas foi simples de perceber. Assim, achamos que fizemos um bom trabalho e que nos deu novos conhecimentos sobre a linguagem c.

# Referências

<https://www.vivaolinux.com.br/script/Fatorial-Recursivo>

<https://pt.stackoverflow.com/questions/177742/impress%C3%A3o-fibonacci-recursivo>