



UNIVERSIDADE PITÁGORAS UNOPAR ANHANGUERA

POLO IBIRITÉ (CENTRO) / MG

SEMESTRE 2024/2

Relatório de Aula Prática:

Programação Estruturada com o Portugol

ALUNO: DENIS DA MATA OLIVEIRA

RA: 3821926601

CURSO: ENGENHARIA DE SOFTWARE - BACHARELADO

MODALIDADE: 100% ONLINE

DISCIPLINA: ALGORITMOS E PROGRAMAÇÃO ESTRUTURADA

PROFESSOR: ANDERSON EMIDIO DE MACEDO GONÇALVES

DATA: 16/11/2024

INTRODUÇÃO

A primeira linguagem de programação foi o Assembly, criada lá na década de 40, ainda na 2ª Guerra Mundial. Ela é usada até hoje pelos compiladores de computador, pois é a linguagem que mais se aproxima da forma como os computadores trabalham. Antes do Assembly, quando os computadores ainda não possuíam memória, a programação era feita fisicamente com os painéis de fiação, diretamente em linguagem de máquina, com zeros e uns, o que rapidamente foi substituído pela linguagem de programação Assembly com os cartões perfurados. O Assembly representava comandos por palavras, ou mnemônicos, como MOV (“mover”), ADD (“adicionar”) e JP (“*jump*” que em português é “pular”), semelhante a linguagem humana, o que facilitou demais a programação. Logo depois do Assembly, na década de 50, veio o FORTRAN, que facilitava mais ainda a programação, pois trazia um nível mais alto e acessível de linguagem, que é usado até hoje para cálculos matemáticos pesados.

Contudo, mesmo com o FORTRAN, as linguagens de programação ainda não eram estruturadas, ou falando de forma mais concreta, não possuíam as estruturas de repetição e condicionais FOR e IF, como conhecemos hoje. A estrutura da linguagem de programação ainda era influenciada pela forma sequencial como a máquina trabalha, com saltos malucos entre linhas, uso do famigerado GO TO e sem indentação, como mostrado na **Figura 1**. Cada linha do código é uma instrução, e para rodar um programa era necessário um cartão perfurado para cada linha, por isso cada linha do programa tinha um número de identificação. Além da coluna do número de identificação de cada linha, havia também uma coluna, sempre à esquerda, que dava a algumas linhas um rótulo na forma de número, como pode ser visto na **Figura 1** na coluna do meio parcialmente vazia. Esses rótulos serviam para saltar livremente de uma linha para outra, semelhante a forma como o computador se desloca na memória para ler os dados, mas isso deixavam o código ruim para ler, o que foi apelidado de “código espaguete”.

Com isso, no final da década de 50, com a linguagem ALGOL e seguindo até o final da década de 70, as linguagens de programação evoluíram para a chamada linguagem estruturada, com a incorporação de estruturas de repetição e condicionais como o FOR, DO e IF ELSE, onde a sequência é definida pela forma como os seres humanos leem o código, e não pela forma com as máquinas trabalham, acabando com o “código espaguete”.

Certos trechos de código de um programa se diferenciam, apenas, por um ou mais parâmetros.

Historicamente, a evolução para linguagem estruturada e procedural ocorreu de forma paralela, ou seja, uma linguagem que era procedural não era necessariamente estruturada. Por exemplo, com o tempo o FORTRAN permitiu o uso de funções, mas ainda manteve o código espaguete. Até hoje o FORTRAN permiti usar o GOTO, mas de forma opcional.

Dado uma tarefa, cujo o resultado é sempre o mesmo, o caminho que você segue para chegar no resultado pode variar dependendo do problema ou contexto. Por exemplo, para contas matemáticas pesadas, como ocorre nos supercomputadores para previsão meteorológica, é melhor evitar uso de matrizes com dimensão alta, pois isso diminui o desempenho do computador. A diferença de desempenho pode ser colossal, apenas dependendo da forma como você escreve o programa. Por outro lado, evitar o uso de matrizes com dimensão alta pode dificultar a leitura do programa por um ser humano. Por isso nasceu os paradigmas de programação, que abordam o caminho que você deve seguir para chegar no resultado, e não no resultado em si. O uso de estruturas de controle e funções foi uma evolução natural e permanente, mas uma linguagem de programação mais moderna, nem sempre é a mais adequada. O paradigma de programação mais popular, nos dia de hoje, é o paradigma de orientação a objeto, que permite que as funções armazenem valores, as chamadas classes, e não só atuem como processadores de informação, evitando variáveis dispersas e expostas por todo o programa, criando uma relação entre processamento e armazenamento. Esse paradigma de programação requer mais recursos da linguagem de programação, para ser usado de forma simples (sem gambiarras usando ponteiros e tuplas), o que eleva o nível de abstração e engenhosidade da linguagem, mas, acontece que nem sempre esse tipo de linguagem de programação é o mais adequado. Em programas onde o desempenho é básico, como na meteorologia e ciências exatas, é comum, até hoje, o uso de programação estruturada com o FORTRAN, evitando o uso excessivo de chamada de funções. Tanto a forma como você escreve o programa, quanto o compilador e linguagem de programação que você adota, impactam drasticamente no desempenho do programa.

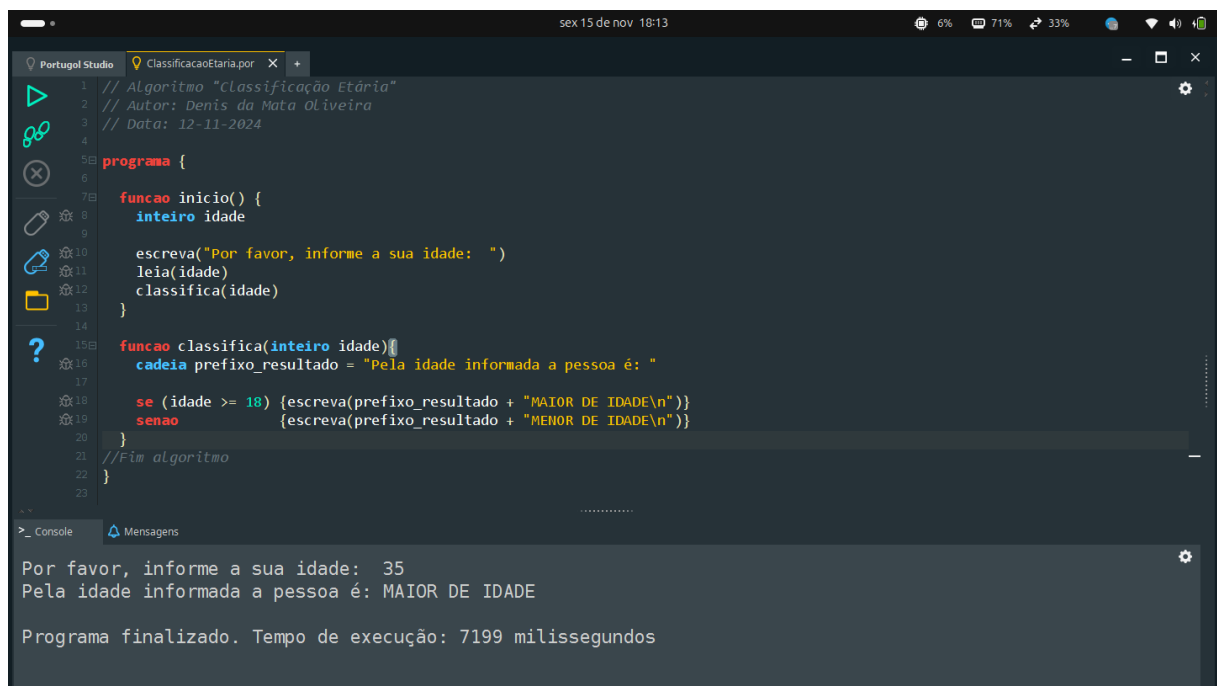
Existe várias ferramentas gratuitas para edição, compilação e execução de programas. O interpretador do Python, por exemplo, já vem junto com o Linux. Felizmente, existe pessoas no Brasil preocupadas com o ensino, que criaram linguagens e compiladores para a aprendizagem de programação, como o Pascal ZIM e Portugol Studio. Ambas as linguagens permitem programação estruturada e procedural, que é o foco da aula. O Portugol é todo em português, extremamente intuitivo e possui muitos recursos de fácil acesso, como execução passo

a passo, monitoramento de variáveis, mapa de estruturas de dados e, principalmente, os códigos de exemplo, muito úteis para quem está começando a programar. Seguindo o roteiro da aula prática, usaremos o Portugol, que possui tanto uma versão offline e mais completa, o Portugol Studio [2], quanto uma versão online e mais acessível, o Portugol Webstudio [3].

OBJETIVO

- Entender a ferramenta Portugol WebStudio.
- Elaborar algoritmos e testar utilizando a ferramenta Portugol WebStudio.

MÉTODO E RESULTADOS



```
1 // Algoritmo "Classificação Etária"
2 // Autor: Denis da Mata Oliveira
3 // Data: 12-11-2024
4
5 programa {
6
7     funcao inicio() {
8         inteiro idade
9
10        escreva("Por favor, informe a sua idade: ")
11        leia(idade)
12        classifica(idade)
13    }
14
15    funcao classifica(inteiro idade){
16        cadeia prefixo_resultado = "Pela idade informada a pessoa é: "
17
18        se (idade >= 18) {escreva(prefixo_resultado + "MAIOR DE IDADE\n")}
19        senao {escreva(prefixo_resultado + "MENOR DE IDADE\n")}
20    }
21 //Fim algoritmo
22 }
23
```

Console Mensagens

Por favor, informe a sua idade: 35
Pela idade informada a pessoa é: MAIOR DE IDADE
Programa finalizado. Tempo de execução: 7199 milissegundos

Figura 2: Programa do cálculo de faixa etária junto com o seu resultado.

O roteiro de aula pediu 2 programas: um bem simples para determinar se a pessoa é maior ou menor de idade, com base na idade informada pelo usuário, e outro para calcular potenciação.

O primeiro programa é muito simples. O código está no **Apêndice A** e o resultado é mostrado na **Figura 2**

Primeiramente, conforme mostrado pela **Figura 2**, podemos ver como a sintaxe do Portugol é simples e acessível. A função `inicio()` é uma função especial, análoga a função `main`

do C/C++. Essa função é executada sempre que o programa é executado, sem ela não tem como rodar o programa. No Portugol, às variáveis devem ser declaradas, uma regra comum de linguagem de programação, mas que não existe em certas linguagens como o Python. A função `inicio` é, geralmente, usada para controle e gerencia, por isso tiramos dessa função partes do programa dedicadas a fazer contas ou tarefas muito específicas, deixando o programa mais legível, modular e escalável. A função que faz a classificação, chamada `classificacao`, apesar de ser muito simples, não é uma função pura, porque gera saída no terminal, e isso, se não bem cuidado, pode acarretar um uso grande de memória com saídas fantasma. Para a função ser considerada pura, ela não deve mexer no sistema operacional, você fornece um argumento, e ela volta um valor. No Portugol, a declaração de variáveis dentro de funções tem escopo local, por isso você pode usar o mesmo nome, usado em outra variável fora da função, que não acarreta em ambiguidade, o que é confortável, mas causa um pouco de confusão para iniciantes. O tipo dos parâmetros da função devem ser especificados, no nosso caso, a função `classifica` tem entradas inteiras. A passagem de argumento no Portugol é por valor (por padrão), mas ele também tem ponteiros e aceita passagem por referência, semelhante ao C/C++, o que evita cópia de estruturas de dados muito grandes, além de permitir modificar variáveis externas a função. A alocação e liberação de memória no Portugol é automática, mas devemos tomar cuidado com a inicialização de variáveis, para evitar erros de inicialização aleatória. As variáveis de texto no Portugol são chamadas de *cadeia*, e são delimitadas por aspas duplas. Quando se trata de apenas um caractere, você pode usar o tipo `caracter`, delimitando apenas com aspa simples. Nas linguagens de programação mais populares, as variáveis de texto são chamadas de *string*. No geral, a sintaxe do Portugol lembra a linguagem C.

A segunda tarefa é calcular potência com a base e expoente fornecidas pelo usuário, em tempo de execução. O resultado do programa é mostrado na **Figura 3**, mas o código é muito longo e é deixado no **Apêndice B**.

Na **Figura 3** é mostrado o resultado para o cálculo da potência de 2 elevado a 16, que dá 65536, contudo, o valor inteiro máximo permitido varia de execução para execução, e o resultado é imprimido errado. Às vezes o programa consegue calcular 2 elevado até 69, o que está além do valor máximo inteiro permitido para computadores 64 bits, que é $2^{63} - 1$, e em outros momentos o programa não consegue nem calcular o valor de 2 elevado a 32. Além disso, ao invés do programa informar que houve uma explosão (*overflow*), falta de memória ou falha

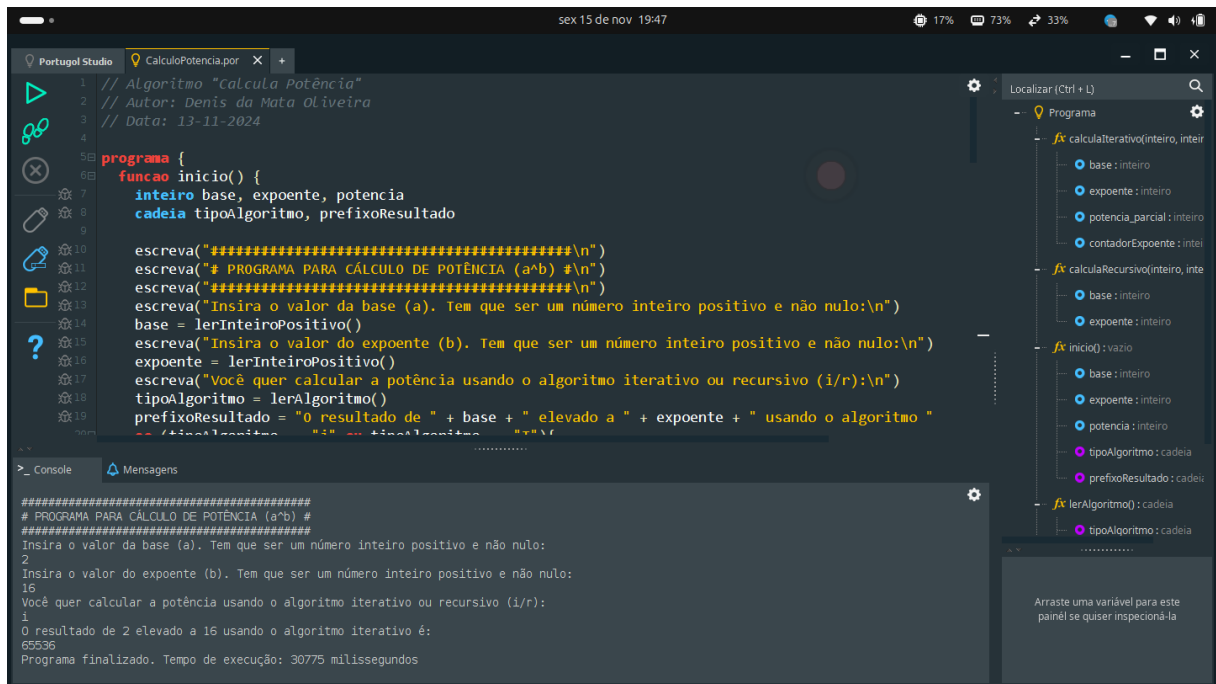


Figura 3: Resultado do programa para o cálculo de potência. O código completo se encontra no Apêndice B.

de segmentação, o programa volta valores como 0, 1 ou o valor da base, o que é errado. Se o valor ultrapassou a capacidade máxima da mantissa, o Portugol deveria informar que houve um *overflow*, ou deveria seguir um ciclo fechado dentro do intervalo $[-2^{63}, 2^{63} - 1]$. O menu de ajuda do Portugol não explica qual é o valor máximo permitido de inteiro [3]. O fato do programa retornar o resultado correto em um determinado momento, mas depois retornar o resultado errado para o mesmo valor, introduzido anteriormente, é um defeito bizarro. O *bug* foi reportado para os desenvolvedores do Portugol no GitHub [4]. Mas enfim, exceto por esse problema, o programa funcionou corretamente para vários valores de base e expoente inteiros e positivos (não-nulos), desde que o expoente não fosse muito grande.

Como o programa foi simples de fazer, aproveitamos para implementar programação defensiva. O programa foi feito de forma modular, procedural e encapsulada. Todas as funções tem contrato bem definido, a entrada é fixa e o resultado da chamada da função é sempre dado pela instrução `retorne`, o que evita acoplamento e nos deixa despreocupados quanto ao funcionamento interno da função. Deixamos a função especial `inicio` mais para controle e gerência do programa, com uma parte da interface de usuário (*view*). Como o roteiro de aula determinou que a base e expoente tem que ser números inteiros e positivos, e o Portugol não tem

o tipo inteiro sem sinal (`unsigned`), criamos uma função para checar se os valores fornecidos obedecem essa regra. Demos ao usuário o direito de escolher entre dois métodos para fazer o cálculo da potência: um método iterativo, mais simples e seguro, e um método recursivo, mais perigoso. No método iterativo, o programa calcula a potência usando uma estrutura simples de repetição para. Inspirado na programação funcional, o método recursivo não realiza mutação de variáveis e chega a ser mais compacto que o método iterativo. Contudo, é bom deixar claro que a recursão é perigosa, pois cada chamada da função aloca um espaço na memória, o que pode fazer a memória explodir. Tem problemas que realmente não tem como evitar a recursão, mas ela sempre deve ser evitada, quando possível. O cálculo de fatorial usando o método iterativo e recursivo é um bom exemplo da diferença absurda que se pode obter de acordo com o método usado. O cálculo do fatorial, pelo método recursivo, leva a um crescimento exponencial, que faz o programa paralisar rapidamente. Além da programação defensiva, para a leitura do valor inteiro, também colocamos uma programação defensiva para a leitura do tipo de método escolhido pelo usuário, emitindo uma mensagem de erro e suportando tanto letras minúsculas quanto maiúsculas. O Portugol é uma ferramenta para finalidades didáticas, mas felizmente ele suportou chamada recursiva, contratos bem definidos e múltiplas condições numa mesma instrução condicional. Apesar do resultado do cálculo de potência crescer exponencialmente com o expoente b , em ambos os métodos, iterativo e recursivo, a complexidade é $O(b)$, ou seja, cresce linearmente com b .

CONCLUSÃO

O objetivo da aula prática é proporcionar um primeiro contato do aluno com a programação de computadores, o que geralmente é feito através da programação estruturada, que é um paradigma de programação mais simples. Além disso, também foi usado o Portugol, que deixa a aula bastante acessível, com sintaxe em português e vários outros recursos didáticos. Apesar do Portugol ter uma versão online, o Portugol WebStudio [3], preferimos a versão offline, o Portugol Studio [2], que é mais sofisticado. Mesmo com a versão offline, o problema do resultado no cálculo de potência persistiu. Devido a vários anos de experiência do autor com programação estruturada, o trabalho foi muito fácil, então aproveitamos para nos aprofundar no conceito e história da programação estruturada, e no cálculo de potência incluímos o método recursivo

para calcular a potência. Tudo foi feito com funções para deixar os programas mais modulares, controláveis, encapsulados, legíveis e desacoplados. Também aplicamos a programação defensiva, que é um bom costume, principalmente quando a aplicação não visa desempenho. O cálculo recursivo, naturalmente, se mostrou mais problemático, e em certos momentos acusava erro de recursão muito longa. Talvez todos os *bugs* bizarros que encontramos pode ser devido a execução do programa com método o recursivo, que pode ter causado algum problema na memória. Os programas criados funcionaram conforme esperado, pelo menos para valores de potência não muito grandes.

REFERÊNCIAS

- [1] Marcello, Loretta & irmãos - 2017, site da internet “Helldragon.eu”, *Forty Years of Computer Languages The Old FORTRAN*.
- [2] Site “Portugol Studio”, <https://univali-lite.github.io/Portugol-Studio/>.
- [3] Site “Portugol Webstudio”, <https://portugol.dev/>.
- [4] Site de versionamento “GitHub”, <https://github.com/UNIVALI-LITE/Portugol-Studio/issues/1172>.
- [5] Inteligência artificial ChatGPT, <https://chatgpt.com/share/6738a84d-4de4-8008-98ea-c6e4acd7929e>.

APÊNDICE A

Programa em Portugol para o classificação de faixa etária.

```
// Algoritmo "Classificação Etária"
// Autor: Denis da Mata Oliveira
// Data: 12-11-2024

programa {
    funcao inicio() {
        inteiro idade

        escreva("Por favor, informe a sua idade: ")
        leia(idade)
        classifica(idade)
    }

    funcao classifica(inteiro idade){
        cadeia prefixo_resultado = "Pela idade informada a pessoa é: "

        se (idade >= 18) {escreva(prefixo_resultado + "MAIOR DE IDADE\n")}
        senao           {escreva(prefixo_resultado + "MENOR DE IDADE\n")}
    }

    //Fim algoritmo
}
```

APÊNDICE B

Programa em Portugol para o cálculo de potência, dado a base e o expoente fornecida pelo usuário. O programa permite escolher entre o método iterativo ou recursivo para o cálculo de potência.

```
// Algoritmo "Calcula Potência"
// Autor: Denis da Mata Oliveira
// Data: 15-11-2024

programa {
    funcao inicio() {
        inteiro base, expoente, potencia
        cadeia tipoAlgoritmo, prefixoResultado

        escreva("#####\n")
        escreva("# PROGRAMA PARA CÁLCULO DE POTÊNCIA (a^b) #\n")
        escreva("#####\n")
        escreva("Insira o valor da base (a).")
        escreva("Tem que ser um número inteiro positivo e não nulo:\n")
        base = lerInteiroPositivo()
        escreva("Insira o valor do expoente (b).")
        escreva("Tem que ser um número inteiro positivo e não nulo:\n")
        expoente = lerInteiroPositivo()
        escreva("Você quer calcular a potência usando o algoritmo ")
        escreva("iterativo ou recursivo (i/r):\n")
        tipoAlgoritmo = lerAlgoritmo()
        prefixoResultado = "O resultado de " + base + " elevado a "
        prefixoResultado += expoente + " usando o algoritmo "
        se (tipoAlgoritmo == "i" ou tipoAlgoritmo == "I"){
            potencia = calculaIterativo(base, expoente)
            escreva(prefixoResultado, "iterativo é:\n")
        }
```

```

        escreva(potencia)
    }
    senao se (tipoAlgoritmo == "r" ou tipoAlgoritmo == "R"){
        potencia = calculaRecursivo(base, expoente)
        escreva(prefixoResultado, "recursivo é:\n")
        escreva(potencia)
    }
}

funcao inteiro lerInteiroPositivo() {
    inteiro numero
    cadeia erroNegativo
    erroNegativo = "Não é permitido zero ou número negativo!"
    erroNegativo += "Por favor, introduza um número inteiro positivo"
    erroNegativo += "e não nulo:\n"

    leia(numero)
    enquanto (numero <= 0) {

        escreva(erroNegativo)
        leia(numero)
    }
    retorne numero
}

funcao cadeia lerAlgoritmo() {
    cadeia tA // Tipo algoritmo
    cadeia erro

    erro = "Não entendi o que você disse! Digite 'i' para escolher "

```

```

    erro += "o algoritmo iterativo, ou 'r' para recursivo:\n"
    leia(tA)
    enquanto (tA != "i" e tA != "I" e tA != "r" e tA != "R"){
        escreva(erro)
        leia(tA)
    }
    retorne tA
}

funcao inteiro calculaIterativo(inteiro base, inteiro expoente){
    inteiro potenciaParcial, contExpoente // Contador expoente

    potenciaParcial = base
    para (contExpoente = 1; contExpoente < expoente; contExpoente++){
        potenciaParcial = potenciaParcial * base
    }
    retorne potenciaParcial
}

funcao inteiro calculaRecursivo(inteiro base, inteiro expoente){
    se (expoente == 1){
        retorne base
    }
    senao {
        retorne base * calculaRecursivo(base, expoente - 1)
    }
}

//Fim algoritmo
}

```