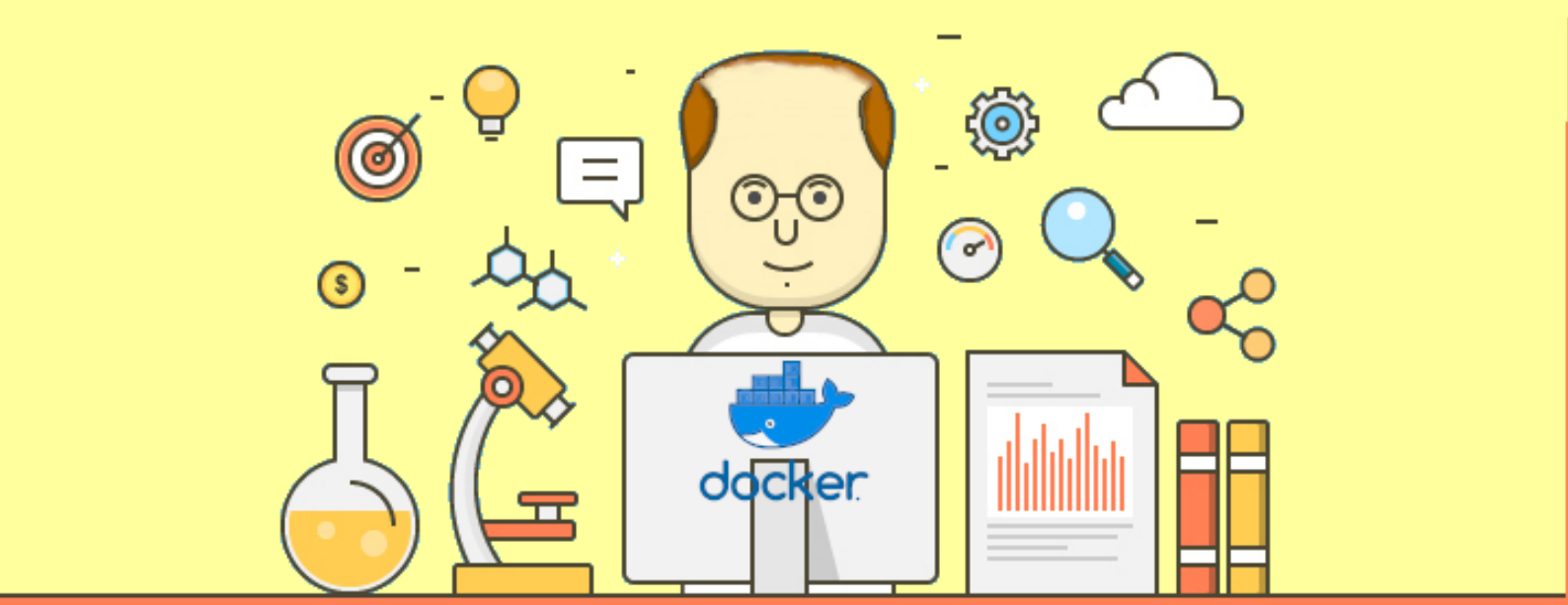
Pascal na Prática

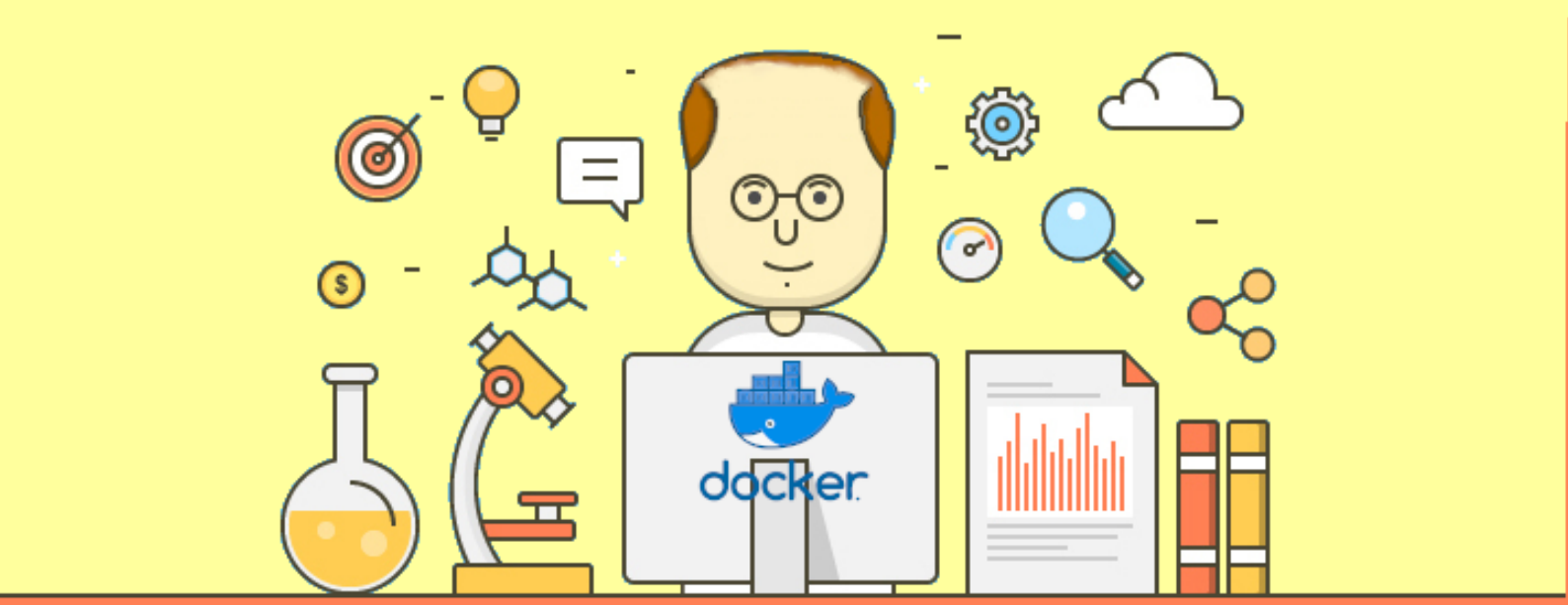
Fernando Anselmo

PUBLICAÇÃO INDEPENDENTE

http:\fernandoanselmo.orgfree.com



Sumário



1. Entendimento Geral

"Um bom programador não se define pelas ferramentas que usa, mas pelo que aprende ao dominá-las — como o Pascal ensina."(*Alan Perlis*, primeiro vencedor do prêmio Turing.)

1.1 Do que trata esse livro?

Pascal foi criada em 1970 pelo cientista da computação suíço **Niklaus Wirth**. Seu objetivo principal foi desenvolver uma linguagem que fosse eficiente, fácil de aprender e que promovesse boas práticas de programação, como por exemplo, o ensino da Lógica de Programação. Inspirada por linguagens anteriores como **ALGOL**, **Pascal** foi projetada para ensinar conceitos fundamentais de programação estruturada e desenvolvimento de software, sendo amplamente adotada em ambientes acadêmicos durante as décadas de 1970 e 1980.

O nome da linguagem é uma homenagem ao matemático e filósofo francês **Blaise Pascal**, conhecido por suas contribuições à matemática e pela invenção de uma das primeiras calculadoras mecânicas, como a *Pascalina*[1]. A linguagem Pascal trouxe inovações importantes, como a ênfase em estruturas de controle claras (condicionais e laços) e na organização modular de programas, o que ajudou os desenvolvedores a criar código legível e fácil de manter.

Embora inicialmente projetada para ensinar, a linguagem logo encontrou aplicações práticas na indústria. Versões como **Turbo Pascal**, lançada pela **Borland** na década de 1980, tornaram-se extremamente populares devido à sua rapidez, facilidade de uso e baixo custo. **Turbo Pascal** oferecia um ambiente de desenvolvimento integrado (IDE) que permitia aos programadores escrever, compilar e executar programas de forma eficiente, algo revolucionário para a época.

Ao longo do tempo, **Pascal** evoluiu e deu origem a variantes como **Object Pascal**, que introduziu conceitos de programação orientada a objetos e serviu de base para o desenvolvimento de ferramentas como o **Delphi**, um ambiente robusto para criar aplicações gráficas para o sistema **Windows** (na época seu único concorrente era o Visual Basic da própria **Microsoft**).

Embora Pascal tenha perdido popularidade com o surgimento de linguagens mais modernas derivadas

[1] A Pascalina foi uma das primeiras calculadoras mecânicas da história, criada pelo matemático, físico e filósofo francês **Blaise Pascal** em 1642, quando ele tinha apenas 19 anos. Foi projetada para ajudar seu pai, **Étienne Pascal**, que trabalhava como coletor de impostos e precisava realizar muitos cálculos repetitivos. A invenção marcou um marco significativo no desenvolvimento das máquinas de cálculo.

do **C**, sua influência persiste. Muitos programadores veteranos aprenderam conceitos fundamentais de computação usando o **Pascal**, e sua ênfase em clareza e estrutura continua sendo uma lição valiosa para a programação até hoje.

1.2 Quais são as vantagens de se aprender Pascal atualmente?

Aprender a linguagem Pascal hoje pode oferecer algumas vantagens, dependendo do contexto e do objetivo de aprendizado. Aqui estão algumas delas:

- **Fundamentos de programação**: é uma linguagem muito boa para aprender os conceitos básicos de programação, como variáveis, estruturas de controle, funções e tipos de dados. Sua sintaxe simples e clara ajuda a focar no raciocínio lógico sem sobrecarregar o aluno com complexidade.
- **Educação**: universidades e escolas a usam como linguagem introdutória em cursos de ciência da computação no passado. Embora outras linguagens tenham substituído em muitos lugares, a base sólida que Pascal oferece ainda é valiosa em cursos que ensinam os fundamentos da programação.
- **Histórico e legado**: o conhecimento pode ser útil caso seja necessário trabalhar em projetos legados ou sistemas que ainda utilizam essa linguagem. Alguns softwares mais antigos, especialmente em áreas como automação industrial e sistemas embarcados, são escritos em Pascal e exigem conhecimento nesta (seja para manutenção ou mesmo migração).
- **Clareza e estrutura**: **Pascal** foi projetada para ser fácil de entender e ensinar. Sua sintaxe é bastante rígida, o que pode ajudar a evitar maus hábitos na programação.
- **Portabilidade**: compiladores modernos para Pascal (como **Free Pascal**) que permitem rodar o código em diferentes plataformas. Isso pode ser interessante para quem deseja experimentar linguagens de baixo nível ou desenvolver software que precise ser altamente eficiente.
- **Desenvolvimento de algoritmos**: sendo uma linguagem estruturada, é muito boa para o desenvolvimento e análise de algoritmos, sem a complexidade de recursos modernos visto em outras linguagens.

Essas vantagens tornam o COBOL uma escolha interessante para quem deseja trabalhar em áreas que envolvem sistemas legados e infraestrutura crítica, permite abrir portas para um conjunto especializado de habilidades que ainda tem grande valor no mercado de trabalho.

1.3 Montagem do Ambiente

Podemos montar nosso ambiente de desenvolvimento sobre diversos sistemas operacionais, oferecendo flexibilidade na escolha da plataforma ideal para o projeto. Neste livro, é utilizado o Ubuntu 24.10, uma das distribuições Linux mais populares e acessíveis, conhecida por sua estabilidade, segurança e suporte a uma vasta gama de ferramentas de desenvolvimento.

O uso de software livre é uma das principais vantagens deste ambiente, pois todos os programas e bibliotecas necessárias para o desenvolvimento estarão disponíveis gratuitamente, sem custos adicionais. Isso inclui editores de código, ferramentas de automação e depuração, que são perfeitamente adequados para projetos profissionais. Além disso, o hardware necessário é simples: um computador (que provavelmente

já possui), sem a necessidade de investimentos adicionais em equipamentos especializados. Com isso, podemos configurar um ambiente de desenvolvimento poderoso e eficiente, aproveitando ao máximo os recursos do Ubuntu e dos softwares livres, sem comprometer o orçamento.

Iremos utilizar o **Free Pascal**, assim na tela de terminal usamos o seguinte comando:
`$ sudo apt install fpc`

Necessitamos de um editor de códigos, recomendo o Visual Studio Code, não apenas pela leveza pois, dentre todos os editores é o que melhor se adapta a linguagem Pascal através dos plugins.
`$ snap install code`

Criamos uma pasta para manter nossos códigos arrumados:
`$ mkdir pascalProjects`

Entramos nessa pasta:
`$ cd pascalProjects`

E no editor:
`$ code .`

Na seção **Extensões**, instalamos o seguinte plugin:

- FreePascal Toolkit - Fornecedor: coolchyni

E agora estamos prontos para criarmos e executarmos nosso primeiro programa em máquinas atuais na linguagem Pascal.

1.4 Mais sobre a Pascalina

As pessoas confundem a **Pascalina** (calculadora criada por **Blaise Pascal**) com a Máquina Diferencial (1822) de **Charles Babbage**, que era também um dispositivo mecânico para calcular e imprimir tabelas matemáticas, como as de logaritmos.

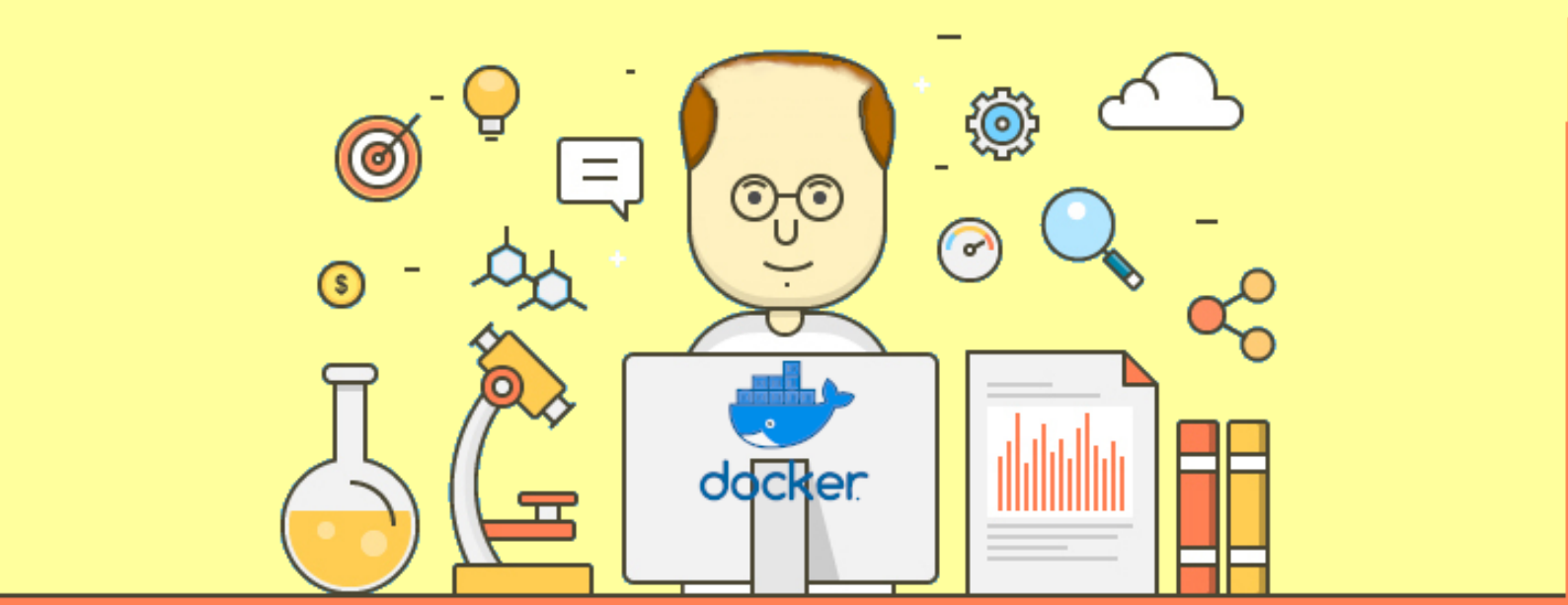
Figura 1.1: Pascalina

A **Pascalina** era uma máquina baseada em engrenagens que podia realizar operações básicas de adição

e subtração. Seu mecanismo utilizava rodas dentadas numeradas de 0 a 9, conectadas de forma que, ao girar uma roda, os números fossem ajustados automaticamente. O sistema implementava a ideia de "transbordo"(ou carry), em que, ao ultrapassar o valor 9, uma roda fazia a próxima avançar, semelhante ao funcionamento dos odômetros.

Embora a Pascalina fosse uma tecnologia revolucionária para a época, ela não teve ampla adoção devido ao seu alto custo de fabricação e à complexidade de operação. Ainda assim, a máquina é considerada um marco na história da computação, pois foi uma das primeiras tentativas de automatizar cálculos, reduzindo o esforço humano e os erros associados ao trabalho manual. Atualmente está preservada em museus e representada em reproduções, e continua sendo um símbolo da engenhosidade humana e um precursor das modernas calculadoras e computadores.

Se deseja conhecer mais sobre esse fantástico projeto, assista o vídeo do Canal "What Will Makes"`https://www.youtube.com/watch?v=E0pJST5mL3A&t=2s`, onde um jovem engenheiro mostra passo a passo da execução de um projeto similar.



2. Passos Iniciais

"Aprender uma linguagem como Pascal é mais do que aprender a programar; é aprender a pensar logicamente."(*Donald Knuth*, cientista da computação.)

2.1 Hello World

É tradição no mundo da programação que devemos começar com o programa "Hello World"quando estamos iniciando o estudo de uma nova linguagem, e quem sou eu para quebrar essa tradição, assim, aqui está o famoso "Hello World"em Pascal:

```
program hello;

begin
  WriteLn('Hello World!');
end.
```

Chega a ser extremamente simples comparando com outras linguagens, o programa sempre inicia com a palavra reservada **program** e seguida o nome deste, finalizamos a instrução com um ";"(ponto e vírgula). O corpo principal do programa está entre na palavra reservada **begin** e seu término **end** que dessa vez deve ser finalizado com um "."(ponto final). E agora temos o comando, ou função se preferir, **WriteLn** que mostra algo na tela, neste caso o literal 'Hello World!', e uma observação muito importante, não utilizamos ASPAS DUPLAS (como os acostumados das linguagens derivadas de **C**) mas ASPAS SIMPLES para indicar as literais.

Salvamos este programa com o nome *Hello.pas* (o nome não precisa ser o mesmo da cláusula **program**), e em seguida devemos compilar com o comando:
`$ fpc Hello.pas`

Nesse momento aconteceu a magia pois o nosso programa foi transformado em um executável do Linux e basta apenas o comando:
`$ ./Hello`

2.2 Obtendo valores

Agora que já agradamos aos deuses da programação, vamos tentar algo mais elaborado, como obter a entrada de duas variáveis numéricas e proceder cálculos aritméticos.

```
program calculadora;
var
  a,b: integer;

begin
  write('Informe o valor de a: ');
  read(a);
  write('Informe o valor de b: ');
  read(b);
  writeln('A soma é: ', a+b);
  writeln('A subtração é: ', a-b);
  writeln('A divisão é: ', a/b);
  writeln('A multiplicação é: ', a*b);
end.
```

Na linguagem Pascal não se importa se escrevemos as funções em letras maiúsculas ou minúsculas (desde que escritas corretamente), assim use a forma que preferir para estas.

Após a definição do programa, temos a palavra chave **var** que define duas variáveis inteiras **a** e **b**. Primeiro solicitamos o valor da primeira, a diferença entre as funções **write()** e **writeln()** e que a primeira escreve o valor e para o cursos no mesmo lugar enquanto que a segunda salta para a próxima linha. A função **read()** aguarda até que o usuário informe um valor e pressione a tecla ENTER e coloca o valor na variável selecionada.

Neste ponto podemos ter um erro no programa, note que nossa variável foi definida com um valor inteiro, caso o usuário informe algo diferente recebemos um **Runtime error**.

Considerando que dois valores foram informados corretamente, mostramos a soma (+), subtração (-), divisão (/) e multiplicação (*) desses, utilizamos a ","(virgula) para separar o literal do resultado.

Salvamos este programa com o nome *Hello.pas* (o nome não precisa ser o mesmo da cláusula **program**), e em seguida devemos compilar com o comando:
`$ fpc Calculadora.pas`

Nesse momento aconteceu a magia pois o nosso programa foi transformado em um executável do Linux e basta apenas o comando:
`$ ./Calculadora`

2.3 Fórmula de Bháskara

Esse programa resolve uma equação do segundo grau com a utilização da **Fórmula de Bhaskara**.

```
program bhaskara;
var
  a, b, c: integer;
var
```

```
  delta: real;

begin
  write('Informe o valor da raiz a: ');
  read(a);
  write('Informe o valor da raiz b: ');
  read(b);
  write('Informe o valor da raiz c: ');
  read(c);

  delta := (b*b) - (4*a*c);

  writeln('Para a função ', a, 'x^2 + ', b, 'x + ', c, ' temos:');
  if delta < 0 then
    writeln('Não existem raízes reais.')
  else if delta = 0 then
    writeln('Existe apenas uma raiz real: ', -b / (2 * a) :5:2)
  else
    begin
      writeln('Delta: ', delta:5:2);
      writeln('x' : ', (-b + sqrt(delta)) / (2*a) :5:2);
      writeln('x'' : ', (-b - sqrt(delta)) / (2*a) :5:2);
    end;
end.
```

Começamos o programa com a definição de dois grupos de variáveis:

- a, b, c: São os coeficientes da equação do segundo grau $ax^2 + bx + c = 0$. Foram declarados como inteiros (integer).
- delta: É valor decimal (real). O delta é a parte da fórmula de Bháskara que determina se existem raízes reais.

A primeira parte, do programa propriamente dito, obtém os valores de **a**, **b** e **c**. Calculamos o valor de Delta, com base na seguinte fórmula:

$$\Delta = b^2 - 4ac$$

A variável **delta** (uma variável auxiliar) armazena esse cálculo. Uma vez que temos esse valor, podemos julgá-lo para verificar se a equação tem soluções reais:

- Se $\delta < 0$, não existem raízes reais (não há solução no conjunto dos números reais).
- Se $\delta = 0$, há uma única raiz real, pois a fórmula se reduz $a - b/2a$.

As raízes são calculadas com a fórmula:

$$x = \frac{-b \pm \sqrt{\Delta}}{2a}$$

Salvamos este programa com o nome *Equacao.pas*, e em seguida devemos compilar com o comando:
`$ fpc Equacao.pas`

E executamos com o comando:
`$ ./Equacao`

O programa funciona perfeitamente porém não é assim que normalmente usamos uma linguagem estruturada, pois dessa forma não estaremos facilitando a manutenção. Vamos utilizar 2 componentes em Pascal.

1. **Função (function)** - Uma função retorna um valor. Ou seja, calcula algo e retorna um determinado resultado.
2. **Procedimento (procedure)** - Um procedimento não retorna valores diretamente. Utilizado para executar ações, como exibir mensagens ou modificar variáveis.

Estrutura de uma função:

```
function NomeDaFuncao(parametros): TipoDeRetorno;
begin
  NomeDaFuncao := valor;  { Retorna um valor }
end;
```

Estrutura de um procedimento:

```
procedure NomeDoProcedimento(parametros);
begin
  { Faz algo, mas não retorna nada }
end;
```

Sabendo disso, agora podemos reescrever o mesmo programa para:

```
program bhaskara;

uses math;  { Para usar a função sqrt() }

{ Função para ler um número inteiro com uma mensagem personalizada }
function lerNumero(msg: string): integer;
begin
  write(msg);
  readln(lerNumero);
end;

{ Função para calcular Delta }
function calcularDelta(a, b, c: integer): real;
begin
  calcularDelta := (b * b) - (4 * a * c);
end;

{ Procedimento para calcular e exibir as raízes, se existirem }
procedure calcularRaizes(a, b: integer; delta: real);
var
  raiz1, raiz2: real;
begin
  if delta < 0 then
    writeln('Não existem raízes reais.')
  else if delta = 0 then
    writeln('Existe apenas uma raiz real: ', -b / (2 * a) :5:2)
  else
    begin
      raiz1 := (-b + sqrt(delta)) / (2 * a);
```

```
      raiz2 := (-b - sqrt(delta)) / (2 * a);
      writeln('Delta: ', delta:5:2);
      writeln('x' : ', raiz1:5:2);
      writeln('x'' : ', raiz2:5:2);
    end;
end;

{ Procedimento para ler os coeficientes da equação }
procedure lerCoeficientes(var a, b, c: integer);
begin
  a := lerNumero('Informe o valor de a: ');
  b := lerNumero('Informe o valor de b: ');
  c := lerNumero('Informe o valor de c: ');
end;

{ Programa principal }
var
  a, b, c: integer;
  delta: real;
begin
  lerCoeficientes(a, b, c);
  delta := calcularDelta(a, b, c);
  writeln('Para a função ', a, 'x^2 + ', b, 'x + ', c, ' temos:');
  calcularRaizes(a, b, delta);
end.
```

Com isso posto, apesar do programa ter crescido muito, observamos uma melhor legibilidade pois o código principal ficou menor e mais fácil de entender. Além disso podemos reusar várias funções ou procedimentos em diversos outros programas.

2.4 Menu de um Sistema

Antigamente os sistemas eram projetados para serem sequenciais, isso é ficávamos presos dentro de uma estrutura escolhida não existia o conceito de Menus *PullDown* ou *PopUps* conhecidos em sistemas gráficos atuais (a vida do programador era bem mais simples).

```
program meu_menu;
var
 opc: integer;
begin
 repeat
  writeln('------ MENU ------');
  writeln('1. Cadastro');
  writeln('2. Busca');
  writeln('3. Ajuda');
  writeln('4. Sair');

  write('Sua opção é: ');
  read(opc);

  if (opc = 1) then
   writeln('Opção cadastro....')
```

```
  else if (opc = 2) then
   writeln('Opção busca....')
  else if (opc = 3) then
   writeln('Mostrar ajuda....')
  else if (opc = 4) then
   writeln('Até a próxima....')
  else
   writeln('Opção inválida!');

 until (opc = 4);
end.
```

O comando **REPEAT-UNTIL** gera um laço de repetição até que alguma coisa aconteça, em linguagem modernas pode ser comparado ao comando **DO-WHILE** (Faça determinada coisa enquanto tal valor lógico for verdadeira), porém a diferença é que no comando REPEAT-UNTIL as ações se repetem **até** tal valor lógico seja verdadeiro. Mas não é a mesma coisa? Vamos escrever o comando **DO-WHILE**, no caso em linguagem **Java**, para entendermos essa diferença:

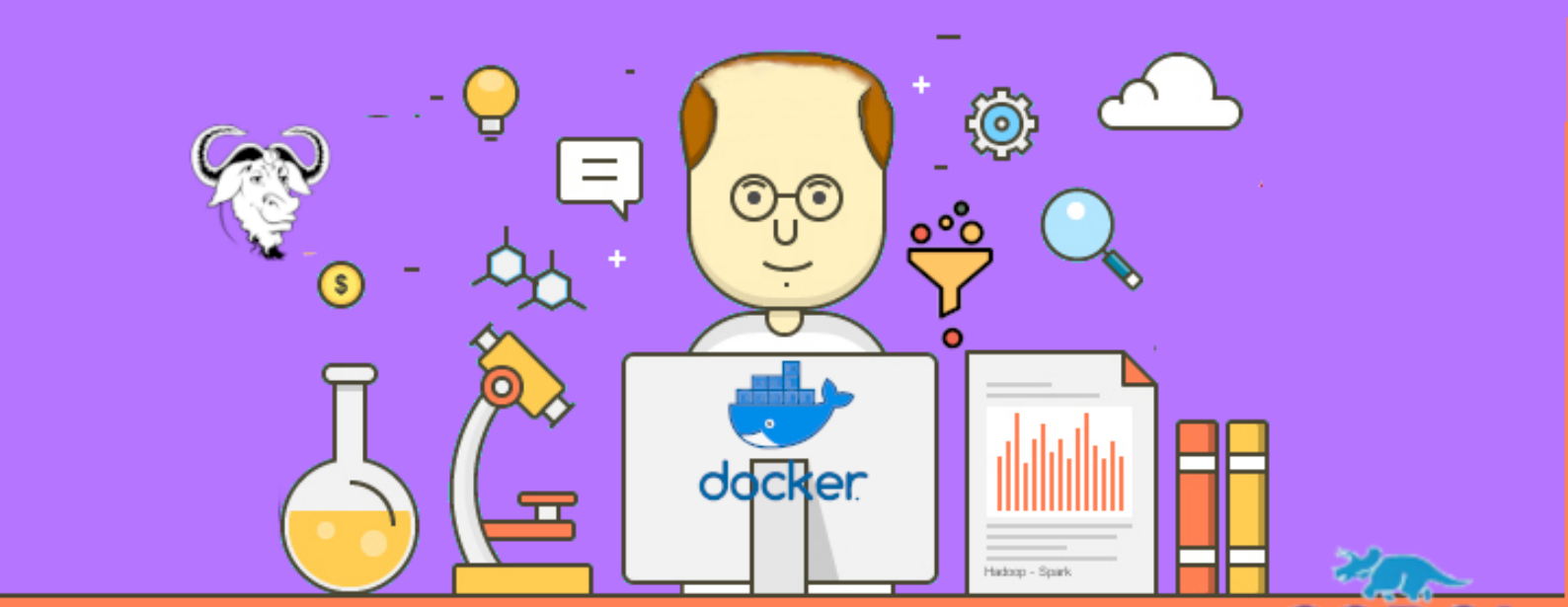
```
do {
   ações a repetir
} while (opc != 4);
```

Observe que no comando **DO-WHILE** a lógica proposta é a negativa do valor, enquanto o valor de OPC **não for** 4; já para o comando REPEAT:

```
   repeat
      ações a repetir
   until (opc = 4);
```

E agora partimos da positividade do valor, **até** o valor de OPC ser 4. Isso permite, em termos lógicos, mais fácil entendimento da situação.

Outro detalhe que devemos observar é a estrutura dos comando **IF-ELSE**. Neste caso a ausência de pontos e vírgulas ao final das linhas. Isso se deve ao fato de o comando não ter terminado e só usamos ponto e vírgula, ao término do comando.



A. Considerações Finais

"A simplicidade é a chave para a eficiência. Foi isso que busquei ao projetar Pascal."(*Niklaus Wirth*, criador de Pascal.)

Não se trata apenas de aprender uma antiga linguagem, se trata de descobrir um novo universo, estou viajando? Provavelmente sim, atualmente vejo os novos programadores e falta neles algo chamado paixão. Programadores da minha época (iniciei nessa área em 1976) não tinha todas as ferramentas que temos atualmente, não tínhamos nem o computador para auxiliar nosso trabalho, como fazíamos? Simples, escrevíamos o código com todo o cuidado e passávamos este para o **digitador**, que por sua vez inseria em um terminal, em seguida solicitávamos ao **operador** para executar o *Job* (trabalho) e recebíamos a listagem de resultado, e todo esse processo era repetido até o programa rodar corretamente. Então todo o trabalho era feito com calma e paciência, e observe que desses três personagens, atualmente, só resta um.

Não sinto saudades daquela época, pois o tempo passa e todo programador deve aprender a se adaptar e não ficar preso a saudosismos, mas o Cobol e existe ainda mercado para esta linguagem, então não conheço por ser um saudosista, e sim por saber que posso ter um chance de entrar nesse concorrido mercado.

Esse não é o fim de uma jornada acredito ser apenas seu começo. Espero que este livro possa lhe servir para criar algo maravilhoso e fantástico que de onde estiver estarei torcendo por você.

A.1 Sobre o Autor

Sou especialista formado em Gestão da Tecnologia da Informação com conhecimentos de programação Java e Python e experiência em Banco de Dados Oracle, PostgreSQL, MS SQL Server além de bancos NoSQL como Hadoop e MongoDB. Realizo desenvolvimento de sistemas com capacidade para analise de dados e detectar tendências, sou autor de 18 livros e diversos artigos em revistas especializadas, palestrante em diversos seminários sobre tecnologia. Focado em aprender e trazer mudanças para a organização com conhecimento profundo do negócio. Atualmente é Professor Universitário em Ciências de Dados e Informática e Desenvolvedor Sênior Java na SEA Tecnologia.

- Perfil no Linkedin: `https://www.linkedin.com/in/fernando-anselmo-bb423623/`
- Endereço do Git: `https://github.com/fernandoans`

Pascal na Prática

ESTE LIVRO PODE E DEVE SER DISTRIBUÍDO LIVREMENTE

Fernando Anselmo