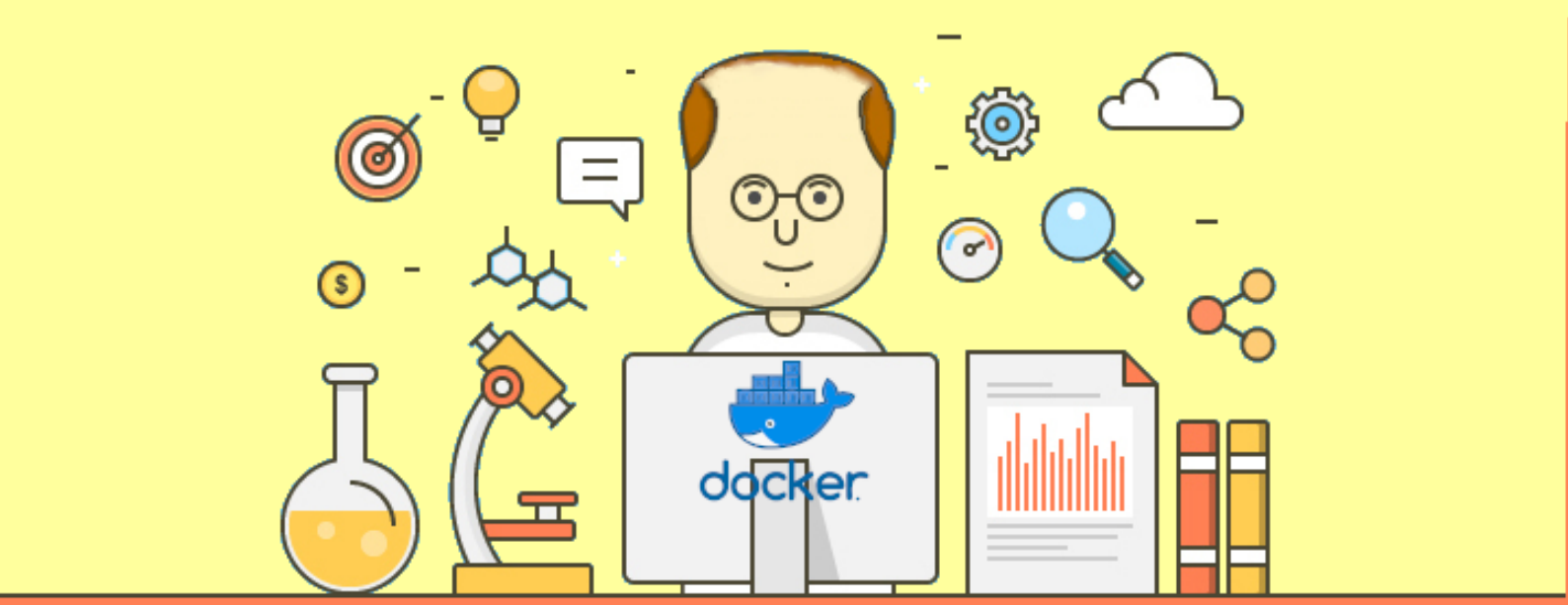
Cobol na Prática

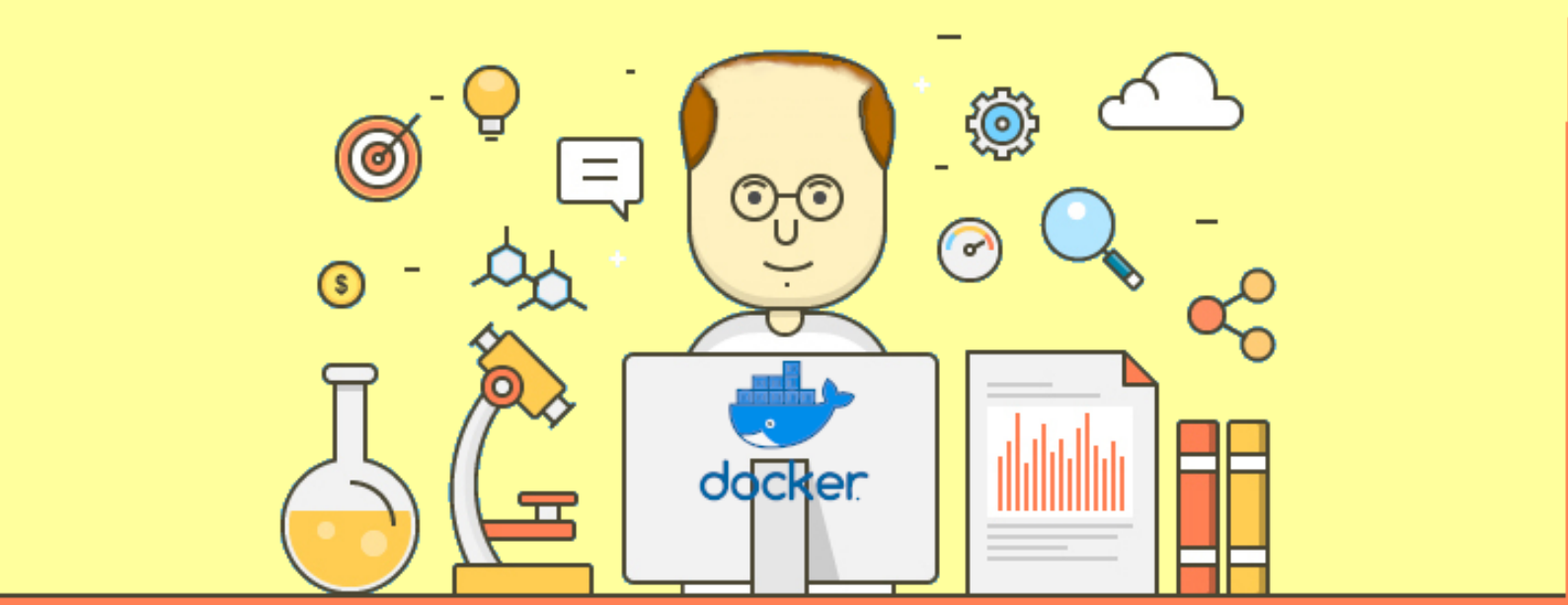
Fernando Anselmo

PUBLICAÇÃO INDEPENDENTE

http:\fernandoanselmo.orgfree.com



Sumário



1. Entendimento Geral

"Não há nada de errado com código legado. Ele te trouxe até onde você está hoje."(*Michael Feathers*, autor de Trabalhando Eficazmente com Código Legado.)

1.1 Do que trata esse livro?

GNU Cobol é uma implementação de código aberto da linguagem **COBOL** (*Common Business-Oriented Language*), uma das linguagens de programação mais antigas, desenvolvida inicialmente na década de 1950. Seu principal objetivo é fornecer uma alternativa gratuita e acessível para os desenvolvedores que trabalham com sistemas legados que dependem do COBOL. É mantido pelo projeto GNU, que visa garantir que softwares essenciais sejam distribuídos de forma aberta e sem custos. Permite que os programadores ainda utilizem o COBOL em sistemas modernos, mantendo a compatibilidade com o código COBOL tradicional e ao mesmo tempo incorporando tecnologias mais recentes.

A linguagem COBOL, historicamente usada em sistemas bancários, governamentais e financeiros, tem uma sintaxe que foca em ser legível e descritiva, facilitando a manutenção de sistemas complexos. O GNU Cobol preserva esses princípios, garante que programas escritos na linguagem permaneçam claros e fáceis de entender. Suporta a maior parte dos recursos do COBOL, isso inclui operações de arquivo, manipulação de dados e estrutura de controle de fluxo, permite que sistemas legados sejam mantidos com pouca ou nenhuma modificação.

Uma das principais vantagens do GNU Cobol é sua integração com outros sistemas e linguagens de programação. Pode ser utilizado para compilar código COBOL e integrá-lo com bibliotecas escritas em outras linguagens, como C e Java, permite que se amplie as opções de interação com sistemas modernos. Isso é especialmente importante em empresas que possuem grandes volumes de código COBOL que precisam ser compatíveis com novas tecnologias ou que desejam melhorar o desempenho sem refazer sistemas inteiros.

O GNU Cobol é amplamente utilizado em ambientes corporativos, onde sistemas legados são críticos para as operações diárias. Embora o COBOL tenha sido considerada uma linguagem em declínio, muitos sistemas ainda dependem de seu uso devido à sua estabilidade e robustez. O GNU Cobol, com sua natureza de código aberto, não só oferece uma forma de preservar esses sistemas, mas também possibilita sua evolução para o futuro, mantendo a compatibilidade com as versões anteriores e ao mesmo tempo permitindo melhorias contínuas.

1.2 Quais são as vantagens de se aprender Cobol atualmente?

Aprender COBOL atualmente pode parecer uma escolha surpreendente, mas oferece várias vantagens, especialmente considerando o contexto de sistemas legados e o mercado de trabalho especializado. Aqui estão algumas razões para estudar COBOL:

- **Demanda por profissionais qualificados**: Embora o COBOL seja uma linguagem antiga, muitos sistemas legados ainda dependem dela, especialmente em setores como bancário, financeiro e governamental. Muitas dessas empresas estão com uma escassez de profissionais qualificados para manter e atualizar esses sistemas. Isso cria uma demanda constante por programadores COBOL, com salários competitivos e uma forte necessidade de manutenção e modernização desses sistemas.
- **Estabilidade e segurança**: O COBOL é amplamente utilizado em sistemas críticos, como os que gerenciam transações bancárias e registros financeiros. A linguagem foi projetada para ser altamente confiável e estável, o que a torna uma escolha popular para ambientes que exigem altos níveis de segurança e precisão. Aprender COBOL pode colocar um profissional em contato com projetos que lidam com grandes volumes de dados de maneira segura e eficiente.
- **Oportunidades de carreira em nichos específicos**: Muitas grandes organizações ainda possuem grandes bases de código COBOL, e estas empresas precisam de especialistas para garantir a continuidade de seus serviços. Como o número de profissionais COBOL diminui ao longo do tempo, aqueles que mantêm o conhecimento da linguagem encontram oportunidades em nichos específicos de mercado, frequentemente com menos concorrência e mais visibilidade.
- **Interação com tecnologias modernas**: Embora o COBOL seja uma linguagem antiga, muitas empresas estão trabalhando para integrar seus sistemas legados com novas tecnologias. Aprender COBOL não significa trabalhar apenas com sistemas desatualizados; muitas vezes, é necessário combinar COBOL com tecnologias mais recentes, como APIs RESTful, sistemas em nuvem, e integrações com linguagens modernas como Java. Esse ambiente híbrido proporciona uma boa oportunidade de desenvolver um conjunto diversificado de habilidades técnicas.

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.

Obviamente, vamos começar com o GNU Cobol, assim na tela de terminal usamos o seguinte comando:
```
$ sudo apt install gnucobol
```

Agora 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 Cobol através dos plugins.
```
$ snap install code
```

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

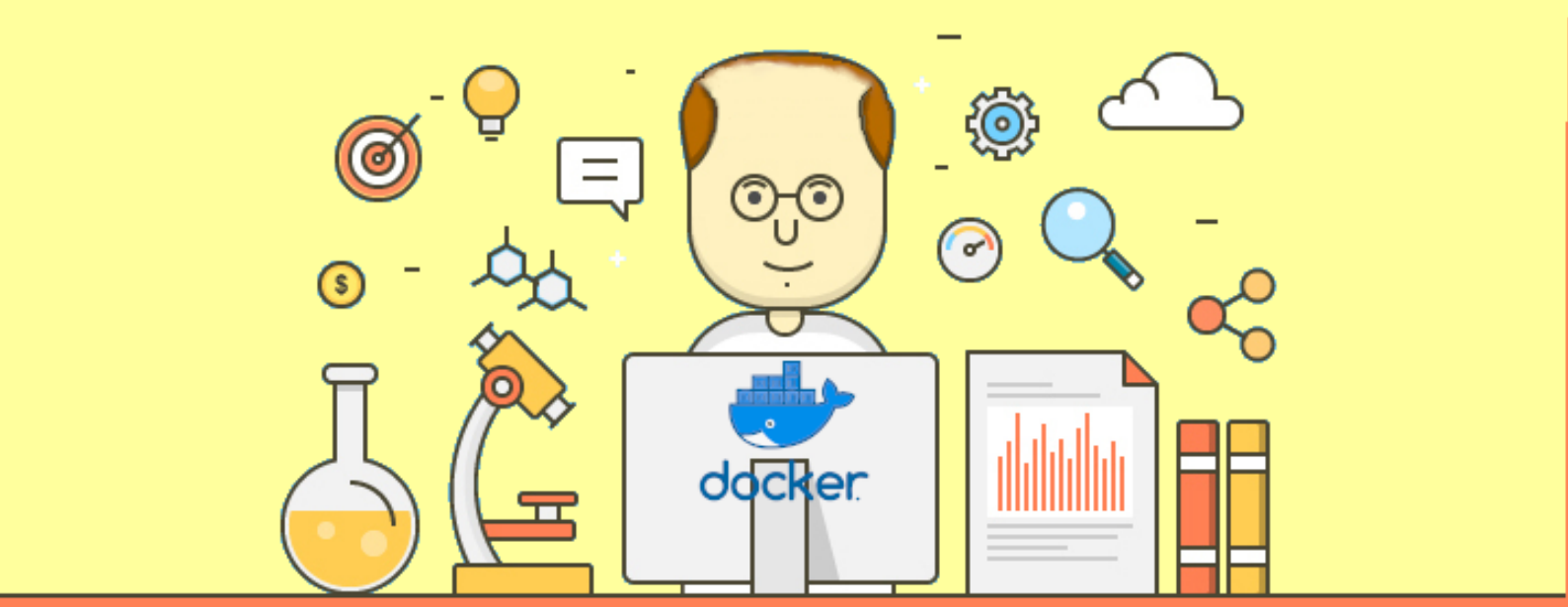
Entramos nessa pasta:
```
$ cd cobolProjects
```

E no editor:
```
$ code .
```

Na seção **Extensões**, instalamos os seguintes plugins:

- COBOL - Fornecedor: Bitlang
- COBOL Language Support - Fornecedor: Broadcom
- COBOL Themes - Fornecedor: BitLang
- COBOL-Lang-Syntax - Fornecedor: Shashi Ranjan

E agora estamos prontos para criarmos e executarmos nosso primeiro programa em máquinas atuais com uma linguagem que nasceu em 1950.



2. Primeiros Programas

"Sistemas antigos não são um fardo; são a base do progresso — refatorá-los é um ato de respeito pelo passado e compromisso com o futuro."(*Sandi Metz*, especialista em design de software.)

2.1 Hello World

Escrever um programa 'Hello World' é uma forma simples e prática de verificar se o ambiente de desenvolvimento está configurado corretamente, e garantir que possamos começar a programar sem obstáculos técnicos. Geralmente e tradicionalmente este é o primeiro passo no aprendizado de qualquer linguagem de programação, assim os iniciantes podem entender a estrutura básica do código e se familiarizar com sua sintaxe.

Primeiramente devemos compreender que o Cobol é uma linguagem que fortemente segue determinados princípios de programação, e qualquer desvio desses resultará em erro.

- As colunas de 1 a 6 do nosso programa é reservada para numeração das linhas.
- A coluna 7 e reservada especialmente para continuação de linha ou início de comentário da linha.
- Nossa codificação sempre deve ser iniciada a partir da coluna 8, e o deslocamento de código é realizado por um TAB.
- TODA instrução deve terminar com um PONTO FINAL.

A estrutura do programa sempre deve conter 4 divisões (mesmo que não sejam utilizadas devem sempre aparecer):

- **Identification Division** - É a primeira divisão do COBOL e contém informações básicas sobre o programa. Usada para identificá-lo, opcionalmente, o autor, a data de criação, e outros detalhes administrativos. O comando mais comum nessa divisão é o **PROGRAM-ID**, que nomeia o programa. É essencial para documentar o propósito do código, especialmente em sistemas onde múltiplos programas podem interagir.
- **Environment Division** - Define o ambiente em que o programa será executado, incluindo as especificações do hardware e software. Subdividida em duas seções principais:

  - **Configuration Section**: Especifica detalhes sobre o sistema, como a máquina onde o programa será executado.
  - **Input-Output Section**: Define os dispositivos de entrada e saída utilizados pelo programa, como arquivos, impressoras ou terminais. Essencial para garantir que o programa funcione corretamente no ambiente de produção.

- **Data Division** - Responsável pela definição de todas as variáveis e estruturas de dados usadas no programa. Além disso, permite que os dados sejam organizados de forma lógica, facilitando o acesso e a manipulação. Subdividida em várias seções:
  - **File Section**: Descreve os arquivos usados pelo programa, incluindo sua estrutura e organização.
  - **Working-Storage Section**: Declara variáveis que mantêm dados em memória durante a execução do programa.
  - **Local-Storage Section**: Contém variáveis que são alocadas apenas durante a execução de um procedimento específico.
  - **Linkage Section**: Define variáveis usadas para comunicar dados entre programas.

- **Procedure Division** - É aqui onde a lógica do programa é implementada. Contém as instruções e os comandos que controlam o fluxo de execução. Os procedimentos são organizados em parágrafos e seções, o que permite modularizar o código para melhorar sua legibilidade e reutilização. É aqui que operações como leitura e escrita de arquivos, cálculos e controle de fluxo são realizadas. Essa divisão é o "coração"do programa, onde os objetivos específicos são alcançados.

E sem mais "delongas", aqui está o nosso programa completo:

```
000001 IDENTIFICATION DIVISION.
000002     PROGRAM-ID. HELLOWORLD.
000003     AUTHOR. Fernando Anselmo.
000004
000005 ENVIRONMENT DIVISION.
000006
000007 DATA DIVISION.
000008*Aqui inicia as ações do programa
000009 PROCEDURE DIVISION.
000010 PRINCIPAL.
000011     DISPLAY "Hello World".
000012
000013     STOP RUN.
000014 END PROGRAM HELLOWORLD.
```

Necessariamente não precisamos inserir os números nas colunas de 1 a 6 para que o programa funcione, porém devemos iniciar sempre na coluna 8, observe que na linha 8 temos um comentário, este é sinalizado com um "*"na coluna 7.

As linhas de 1 a 3 são referentes a identificação do programa, este ao mínimo deve ter **PROGRAM-ID** que identifica o nome do programa e **AUTHOR** com o autor do programa. As linhas 5 e 6 são correspondentes as duas próximas divisões, não é necessário nenhuma informação, porém essas divisões devem estar contidas no programa. E por fim a partir da linha 9 inicia o programa propriamente dito.

Um programa sempre inicia com um "label", que pode ser qualquer um a escolha, colocamos **PRINCIPAL** aqui, do mesmo modo que poderíamos colocar **INICIAL** ou qualquer outro nome. Após esse, temos as

ações que queremos executar, no caso o comando DISPLAY que mostra uma determinada informação na *console*. Para terminar nossas execuções temos o comando **STOP RUN** (linha 13) e para finalizar o programa a instrução **END PROGRAM** com o nome do programa que foi definido pela **PROGRAM-ID** que marca explicitamente o fim do código.

> **Dica 1: Linha em Branco.** Por padrão o GNU Cobol, exige que ao final do programa SEMPRE exista uma linha em final em branco. Sendo assim lembre-se de ao final sempre deixar uma linha em BRANCO.

Salvamos este programa com o nome *HelloWorld.cbl* (outra extensão que poderia ser usada é "cob"), e em seguida devemos compilar com o comando:

```
$ cobc -x HelloWorld.cbl
```

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

```
$ ./HelloWorld
```

E teremos nossa mensagem mostrada.

2.2 Entrada de Dados

Em qualquer linguagem de programação iniciamos com os processos de Entrada/Saída de dados, começamos pela saída, que em Cobol é obtida através do comando **DISPLAY** e agora vamos partir para a entrada, que em Cobol é obtida através do comando **ACCEPT**. Mas não é tão simples assim pois precisamos definir uma variável que receberá a informação do que vamos digitar.

A criação de variáveis em Cobol é realizada na **DATA DIVISION**, mais especificamente na seção **WORKING-STORAGE SECTION** que descreve os dados internos, nesse ponto podemos criar quaisquer variáveis que desejarmos, porém existem regras para isso, a partir da coluna 8 devemos iniciar um "*level number*", esse número sempre inicia com "01", em seguida colocamos o nome da nossa variável, que será utilizada pelo programa, e o tipo de dado que será armazenado, como estamos tratando de variável, usamos a cláusula **PIC** (abreviatura de *PICTURE*) e em seguida o tipo, que podem ser, por padrão:

Símbolo	Significado
x	Campo alfanumérico
9	Campo numérico
A	Campo alfabético
V	Campo decimal assumido; usado apenas em campos numéricos
S	Sinal operacional; usado apenas em campos numéricos

E finalmente colocamos o tamanho do campo, mas vamos com calma e mostrar um exemplo na prática como isso funciona, por exemplo, ao invés de termos um programa estático "Hello World", desejamos que o usuário tenha a possibilidade de colocar seu nome e a data de seu nascimento, como fazemos isso:

```
000001 IDENTIFICATION DIVISION.
000002     PROGRAM-ID. COMOVAI.
```

```
000003     AUTHOR. Fernando Anselmo.
000004
000005 ENVIRONMENT DIVISION.
000006
000007 DATA DIVISION.
000008 WORKING-STORAGE SECTION.
000009
000010 01 NOME        PIC A(020).
000011
000012 01 DATA-ATUAL.
000013    05 ANO-ATUAL PIC 9(004).
000014    05 MES-ATUAL PIC 9(002).
000015    05 DIA-ATUAL PIC 9(002).
000016
000017 PROCEDURE DIVISION.
000018 PRINCIPAL.
000019     DISPLAY "Entre com seu Nome: ".
000020     ACCEPT NOME.
000021     ACCEPT DATA-ATUAL FROM DATE YYYYMMDD.
000022
000023     DISPLAY "Bem vindo " NOME.
000024     DISPLAY "Sabia que hoje é " DIA-ATUAL "/" MES-ATUAL "/"
           ANO-ATUAL.
000025
000026     STOP RUN.
000027 END PROGRAM COMOVAI.
```

O programa é bem similar ao visto anteriormente e como não pretendo ficar me repetindo, assim vamos nos ater aos detalhes modificados, na seção **WORKING-STORAGE SECTION** criamos uma variável (linha 10) chamada NOME que será alfabética e permite de 0 a 20 posições. Em seguida outra variável denominada DATA-ATUAL que contém uma subestrutura (assim o nível deve ser um número menor, por padrão usamos de 5 em 5).

No nosso programa propriamente dito (PROCEDURE DIVISION), na linha 20 existe a espera pelo usuário digitar o NOME, e assim que acontecer na linha 21 é obtida a data atual automaticamente (parâmetro FROM DATE), na linha 23 esse NOME digitado é mostrado e na linha 24 a data atual, como essa linha ultrapassa a coluna 73 devemos continuar a informação na linha abaixo, note que não usamos o caracter de continuação na coluna 7, esse é utilizado apenas quando for um texto que queremos dar continuidade.

Salvamos este programa com o nome *ComoVai.cbl* (outra extensão que poderia ser usada é "cob"), e em seguida devemos compilar com o comando:

```
$ cobc -x ComoVai.cbl
```

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

```
$ ./ComoVai
```

Será solicitado para digitarmos nosso nome e em seguida mostra a mensagem com a data atual.

2.3 Cálculos em Cobol

O comando **COMPUTE** é utilizado para realizar operações aritméticas de forma direta e simplificada, permite cálculos matemáticos como soma, subtração, multiplicação, divisão, e até expressões mais complexas. Combina a funcionalidade dos operadores matemáticos (+, -, *, /, **) para calcular o valor de uma expressão e armazenar o resultado em uma variável destino.

COMPUTE é especialmente útil por sua clareza pois elimina a necessidade de comandos adicionais, como **ADD**, **SUBTRACT**, ou **MULTIPLY**, em situações simples. O resultado da expressão é automaticamente ajustado para o formato definido na variável de destino, incluindo arredondamento quando necessário, de acordo com as especificações da cláusula PICTURE.

Vejamos um caso prático com esse comando através do cálculo do IMC, que mostra o Índice de Massa Corporal, através da seguinte fórmula:

$$\text{IMC} = \frac{\text{PESO}}{\text{ALTURA}^2}$$

Sendo que este PESO é fornecido em quilos enquanto que a ALTURA em metros. Então temos um problema aqui, pois o usuário deve digitar um valor decimal, para conseguirmos isso vamos utilizar o formato: **PIC ZZ9V99**. Que representa um número com 3 dígitos inteiros e 2 casas decimais (exemplo: 51.75) o carácter **Z** elimina os zeros a esquerda. E o **V** é o ponto decimal virtual, que não aparece no armazenamento, mas é considerado no cálculo.

> **Dica 2: Ponto ou Vírgula decimal?.** A maioria das linguagens derivam dos padrões da Língua Inglesa. Cobol não é nenhuma exceção, sendo assim todas as entradas e saídas são mostradas com o uso do ponto (e não vírgula) decimal. Não esqueça disso na hora de digitar a informação.

Acredito que agora estamos prontos para codificar nosso programa:

```
000001 IDENTIFICATION DIVISION.
000002     PROGRAM-ID. IMC.
000003     AUTHOR. Fernando Anselmo.
000004
000005 ENVIRONMENT DIVISION.
000006
000007 DATA DIVISION.
000008 WORKING-STORAGE SECTION.
000009
000010 01 ALTURA    PIC 9V99.
000011 01 PESO      PIC 999.
000012 01 IMC_TOTAL PIC ZZ9.99.
000013
000014 PROCEDURE DIVISION.
000015 PRINCIPAL.
000016     DISPLAY "Entre com sua altura (em Mt): ".
000017     ACCEPT ALTURA.
000018     DISPLAY "Entre com seu peso (em Kg): ".
000019     ACCEPT PESO.
000020
000021     COMPUTE IMC_TOTAL = PESO / (ALTURA ** 2).
000022     DISPLAY "Seu IMC é " IMC_TOTAL.
```

```
000023
000024 STOP RUN.
000025 END PROGRAM IMC.
```

Vamos para os detalhes, na seção WORKING-STORAGE SECTION que declara as variáveis utilizadas durante a execução do programa. Temos:

- **ALTURA**: com formato PIC 9V99 que representa um número com 1 dígito inteiro e 2 casas decimais (exemplo: 1.65).
- **PESO**: com formato PIC 999 que representa um número inteiro de até 3 dígitos (exemplo: 75).
- **IMC_TOTAL**: com formato PIC ZZ9.99 que representa o resultado do IMC com até 3 dígitos antes do ponto decimal e 2 após. Lembrando que carácter **Z** suprime zeros à esquerda, então, ao invés de apresentarmos 030.32 será exibido como 30.32.

Agora vamos na divisão que representa nosso programa, adoto por um simples padrão pessoal o rótulo **PRINCIPAL** que é o ponto de entrada do programa (semelhante a *main* em outras linguagens).

Temos a entrada de duas variáveis a altura em metros e o peso em quilogramas. O comando **ACCEPT** captura os valores fornecidos pelo usuário e armazena nas variáveis correspondentes (ALTURA e PESO).

Em seguida o comando **COMPUTE** realiza o cálculo do IMC, o símbolo **** representa a operação de exponenciação (potenciação). Utilizada para calcular o valor de uma base elevada a uma potência (ou expoente).

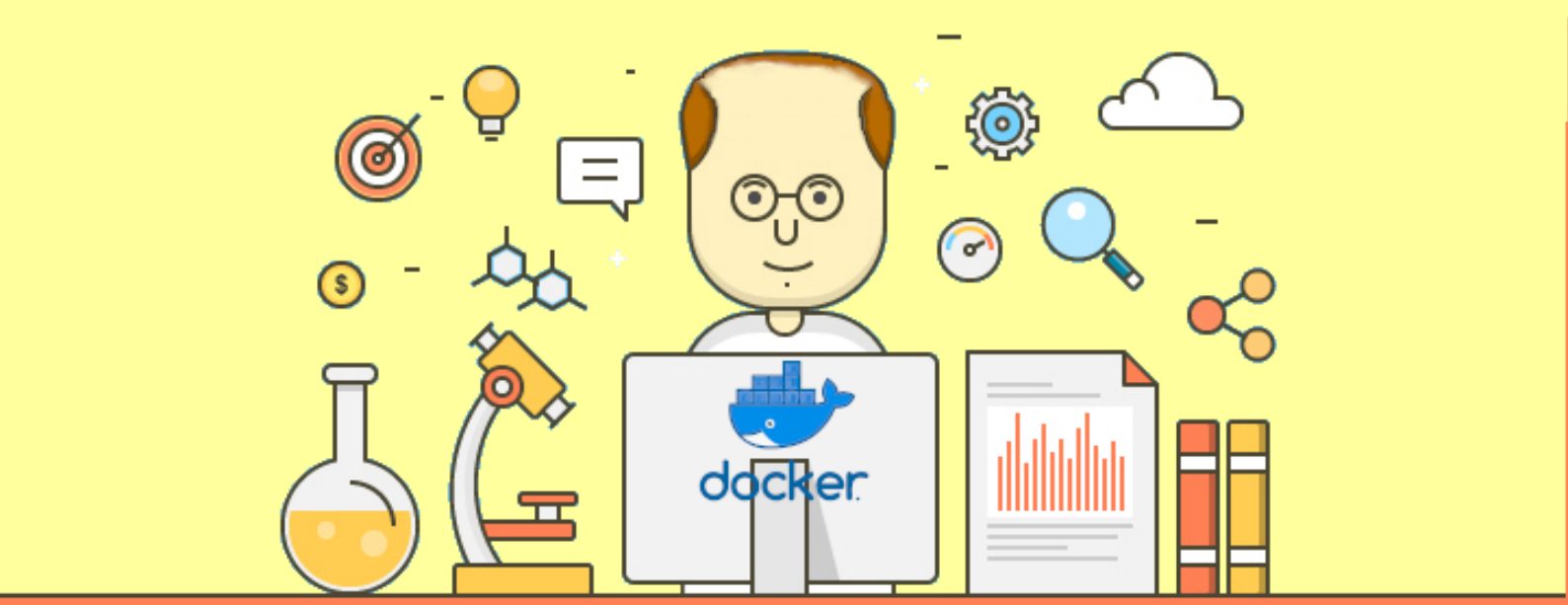
> **Dica 3: Ordem de precedência.** Os parênteses adicionados no comando são usados somente para facilitar a clareza do código pois não são necessários, a exponenciação tem maior precedência do que multiplicação (*) e divisão (/). Mas a utilização de parênteses facilita a clareza e leitura de fórmulas, e não prejudica a performance do programa, assim é recomendável sua utilização.

E por fim, o comando **DISPLAY** mostra a mensagem com o valor do IMC calculado (formatado pela cláusula de edição ZZ9.99).

Salvamos este programa com o nome *IMC.cbl*, e em seguida devemos compilar com o comando:
`$ cobc -x IMC.cbl`

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

Será solicitado para digitarmos a altura (por exemplo, 1.6) e em seguida o peso (por exemplo, 70) e mostra a mensagem com o IMC calculado (que pelo exemplo deve ser 27.34).



3. Cobol como Banco de Dados

"A coisa mais perigosa que você pode dizer é: 'Sempre fizemos assim.'"(*Grace Hopper*, pioneira da computação.)

3.1 Trabalhar com Dados

Obviamente todos sabem que o forte do Cobol é sua capacidade de processar dados, e são a razão da existência do Cobol até os dias atuais. Nenhum outro banco de dados (seja SQL ou NoSQL) possui a capacidade de um sistema em Cobol, pergunte a qualquer instituição financeira se eles trocariam o Cobol por outro sistema (mesmo nos dias atuais com a dificuldade em se encontrar quem forneça manutenção para tais sistemas).

Vamos começar com leitura de dados simples através de um arquivo sequencial, o detalhe mais importante aqui é que o Cobol faz isso através das leituras das POSIÇÕES dos dados, por exemplo o seguinte arquivo:

```
12321Joao               Silva               M
13434Maria              Silva               F
43543Luiza              Albuquerque         F
53453Mario              Oliveira            M
43454Samara             Correia             F
87978Alberico           Soares              M
87886Carlos             Souza               M
76535Joao               Ramalho             M
54543Roberta            Martins             F
```

Observamos que as colunas de 1 a 5 contém a matrícula do funcionários, as 20 próximas colunas (6 a 25) seu nome, as próximas 20 (26 a 35) seu último nome e o carácter 46 contém o gênero. E é dessa forma que devemos informar ao Cobol de modo que a leitura seja informada corretamente.

> **Dica 4: Notação dos códigos.** Apenas para facilitar a leitura a partir desse programa iremos suprimir os números das linhas e o espaçamento inicial, considere que o código do programa SEMPRE deve começar a partir da coluna 7.

Dessa vez vamos por partes no programa que leia os valores contidos neste arquivo e mostre na saída cada

um deles, ao término quantos homens (último campo como "M") e quantas mulheres (último campo como "F") existem na empresa. Vamos ver este programa divisão a divisão.

Na **IDENTIFICATION** não existe muitas modificações:

```
IDENTIFICATION DIVISION.
    PROGRAM-ID. ContagemFuncionarios.
    AUTHOR. Fernando Anselmo.
```

Como sempre colocamos o nome do programa na **PROGRAM-ID** e o autor deste na **AUTHOR**. O nome do programa é o único parâmetro obrigatório os outros totalmente opcionais (tanto que até o momento além do **AUTHOR** não foram usados), para conhecimento, são eles:

- **INSTALLATION**: Indica o nome da instalação ou organização onde o programa será executado.
- **DATE-WRITTEN**: Especifica a data em que o programa foi escrito.
- **DATE-COMPILED**: Indica a data em que o programa foi compilado.
- **SECURITY**: Fornece informações de segurança relacionadas ao programa.
- **REMARKS**: Permite adicionar comentários ou notas livres sobre o programa.

São meramente descritivos e não possuem qualquer impacto na execução do programa, sendo apenas informativos. Por exemplo, aqui está uma **IDENTIFICATION DIVISION** completa:

```
IDENTIFICATION DIVISION.
    PROGRAM-ID. ContagemFuncionarios.
    AUTHOR. Fernando Anselmo.
    INSTALLATION. Empresa Decus in Labore.
    DATE-WRITTEN. 12-JAN-2025.
    DATE-COMPILED. 16-JAN-2025.
    SECURITY. Apenas para uso interno.
    REMARKS. Programa para contagem da equipe de instrutores.
```

A **ENVIRONMENT** começa a mudar um pouco, pois precisamos indicar o arquivo:

```
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT FUNCIONARIOS ASSIGN TO "FUNCIONARIOS.DATA"
        ORGANIZATION IS LINE SEQUENTIAL.
```

Esta divisão descreve o ambiente de execução do programa, incluindo os dispositivos de entrada e saída, como arquivos, impressoras e terminais. Assim, criamos a **INPUT-OUTPUT SECTION** que especifica os arquivos que serão manipulados pelo programa. Nessa temos a **FILE-CONTROL** para informar como associar os arquivos lógicos do COBOL (usados no código) aos arquivos físicos (armazenados no sistema de arquivos).

SELECT no Cobol é diferente do SQL, aqui associa o nome lógico do arquivo no programa (FUNCIONARIOS) ao nome físico no sistema de arquivos. E a cláusula **ASSIGN TO** define o nome físico do arquivo, ou seja, seu nome real no sistema operacional.

ORGANIZATION mostra como os dados no arquivo estão organizados. E finalmente **LINE SEQUENTIAL** significa que o arquivo é baseado em linhas, como um arquivo de texto comum. Cada registro no arquivo termina com um caractere de nova linha (
n) ou sequência equivalente no sistema.

```
DATA DIVISION.
FILE SECTION.
FD FUNCIONARIOS.
01 DETALHEFUNCIONARIO.
   88 FINALREGISTRO VALUE HIGH-VALUES.
   05 MATRICULA-FUNCIONARIO   PIC 9(5).
   05 NOME-FUNCIONARIO.
       10 PRIMEIRO-NOME       PIC X(20).
       10 ULTIMO-NOME         PIC X(20).
   05 GENERO                  PIC X(1).

WORKING-STORAGE SECTION.
01 CONTADORES.
   05 TOTAL-HOMENS            PIC 9(3) VALUE 0.
   05 TOTAL-MULHERES          PIC 9(3) VALUE 0.

01 LEITURA-FINALIZADA          PIC X VALUE "N".
```

Agora temos a **FILE SECTION** que define a estrutura do arquivo que será lido, o nome desse deve ser exatamente o descrito na cláusula **SELECT**. O detalhe que vale atenção é a existencia do registro 88 que é definido para indicar o fim dos registros no arquivo ou quando não há mais dados válidos. Utiliza o valor especial **HIGH-VALUES**, que representa o valor mais alto possível para o conjunto de caracteres em uma tabela de códigos, como **EBCDIC** ou **ASCII**.

Em seguida montamos a estrutura completa como o Cobol armazena cada linha que será lida. A **WORKING-STORAGE SECTION** contém a definição dos contadores que usaremos para definir o total de homens e mulheres. E uma chave que usamos para indicar que terminamos a leitura do programa.

E por fim a **PROCEDURE**:

```
PROCEDURE DIVISION.
INICIO.
    DISPLAY "================================".
    DISPLAY " Contagem de Funcionários".
    DISPLAY "================================".

    OPEN INPUT FUNCIONARIOS.

    PERFORM UNTIL LEITURA-FINALIZADA = "S"
        READ FUNCIONARIOS INTO DETALHEFUNCIONARIO
            AT END
                MOVE "S" TO LEITURA-FINALIZADA
            NOT AT END
                INSPECT PRIMEIRO-NOME REPLACING ALL " "
                    BY LOW-VALUES
                INSPECT ULTIMO-NOME REPLACING ALL " "
                    BY LOW-VALUES
                DISPLAY MATRICULA-FUNCIONARIO " " GENERO " "
                   PRIMEIRO-NOME " " ULTIMO-NOME
```

```
                IF GENERO = "M"
                    ADD 1 TO TOTAL-HOMENS
                ELSE
                    IF GENERO = "F"
                        ADD 1 TO TOTAL-MULHERES
                    END-IF
                END-IF
        END-READ
    END-PERFORM.

    CLOSE FUNCIONARIOS.

    DISPLAY "============================".
    DISPLAY "Resumo:".
    DISPLAY " Total de Homens..: " TOTAL-HOMENS.
    DISPLAY " Total de Mulheres: " TOTAL-MULHERES.
    DISPLAY "============================".

    STOP RUN.
```

Pode parecer bem estranha essa estrutura mas vamos simplificar um pouco, o comando **OPEN INPUT** abre o arquivo para leitura e em seguida temos um laço **PERFORM UNTIL**, que possui a seguinte estrutura simplificada:

```
READ FUNCIONARIOS INTO DETALHEFUNCIONARIO
   AT END
       MOVE "S" TO LEITURA-FINALIZADA
   NOT AT END
       DISPLAY "Registro válido."
END-READ
```

Quando o arquivo alcança o fim ou quando um registro vazio (sem dados válidos) é lido, o COBOL pode preencher aquele campo do registro **HIGH-VALUES**. Temos dois comandos básicos **AT END** quando o fim for atingido e **NOT AT END** enquanto não for.

O detalhe mais importante aqui é a falta do "."(ponto final) nas instruções, toda vez que estamos dentro de um comando NÃO usamos mais o ponto final para indicar o fim do comando, sendo assim esse só será usado no **END-PERFORM**.

A instrução **INSPECT [VARIÁVEL] REPLACING ALL BY LOW-VALUES** retira qualquer espaço em branco, assim temos que ter todo cuidado ao usá-lo, pois se tivéssemos um valor "Meu Nome é Fernando", o resultado seria: "MeuNomeéFernando", ou seja, não pense nisso como um equivalente ao método **TRIM()** de outras linguagens. Esse é um dos motivos de até nos dias atuais em bancos modernos acostumarmos com a separação dos nomes em primeiro e último sem o uso de outros nomes intermediários, mesmo com as linguagens mais modernas.

O comando **IF** do Cobol, possui a cláusula opcional **ELSE** e deve ser terminado com **END-IF**, e no nosso caso é utilizado para totalizar o total de homens e mulheres, é óbvio que poderíamos simplificá-lo para:

```
IF GENERO = "M"
    ADD 1 TO TOTAL-HOMENS
ELSE
```

```
    ADD 1 TO TOTAL-MULHERES
END-IF
```

Mas estaríamos arriscando que se houvesse um erro no registro e não contivesse o valor "M"ou "F"seria totalizado como uma mulher, e apenas para garantirmos a totalização correta adicionamos mais um **IF** para verificarmos se realmente o gênero contém o carácter "F".

Ao término da leitura dos dados fechamos o arquivo **CLOSE FUNCIONARIOS** e mostramos a totalização dos dados.

3.2 Separação em *Labels*

O programa visto anteriormente realiza sua função corretamente, porém está estruturado de forma incorreta o ideal é sempre deixá-lo mais claro e organizado. Em COBOL, pode ser feito com a utilização de parágrafos (chamados de *labels*).

Descrição dos parágrafos:

1. **PROCESSAR-REGISTROS**: que controla o loop principal e chama as funções para leitura do arquivo, exibição dos dados e contagem de gênero.
2. **LER-REGISTRO**: responsável por ler um registro do arquivo e determina se é o fim do arquivo (**AT END**).
3. **EXIBIR-FUNCIONARIO**: para mostrar os dados do funcionário no console.
4. **CONTAR-GENERO**: que incrementa os contadores de homens e mulheres com base no campo GENERO.
5. **EXIBIR-RESUMO**: para mostrar o total de homens e mulheres ao final.

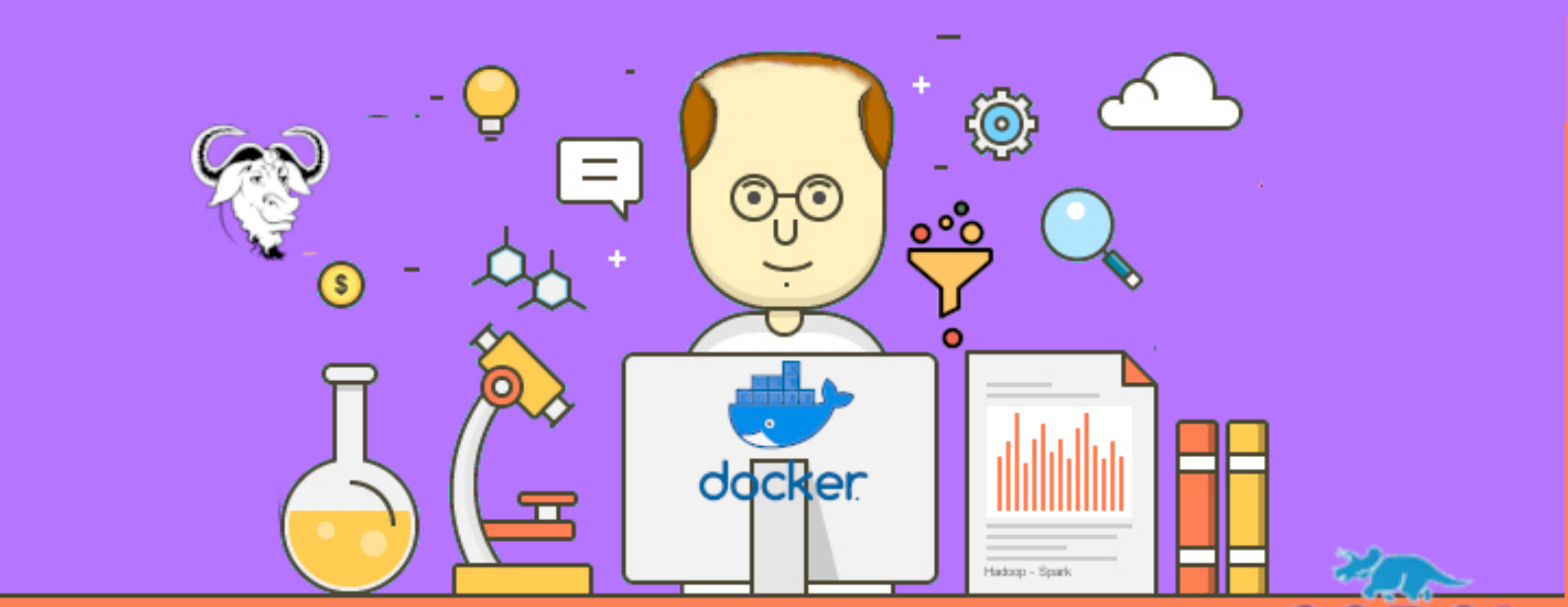
Essa reestruturação sempre deve ser aplicada se pensarmos na **clareza** onde cada funcionalidade é separada em parágrafos específicos, para facilitar a leitura, ou **reutilização** onde os parágrafos podem ser chamados em diferentes partes do programa se necessário. Mas devemos sempre aplicar quando pensamos principalmente na **manutenção** do código, pois alterações em uma parte específica não afetam outras funcionalidades e são muito mais fáceis de serem manejadas.

O comando **PERFORM** é o responsável para a chamada de cada um desses *labels*, o que acontece, o **label** é chamado e ao seu término retorna a execução para o ponto de chamada. Assim a **PROCEDURE DIVISION** vista no programa anterior seria assim redefinida:

```
PROCEDURE DIVISION.
INICIO.
    DISPLAY "==============================".
    DISPLAY " Contagem de Funcionários".
    DISPLAY "==============================".

    OPEN INPUT FUNCIONARIOS.
    PERFORM PROCESSAR-REGISTROS.
    CLOSE FUNCIONARIOS.
```

```
    PERFORM EXIBIR-RESUMO.
    STOP RUN.

PROCESSAR-REGISTROS.
    PERFORM UNTIL LEITURA-FINALIZADA = "S"
       PERFORM LER-REGISTRO
       IF LEITURA-FINALIZADA NOT = "S"
          PERFORM EXIBIR-FUNCIONARIO
          PERFORM CONTAR-GENERO
       END-IF
    END-PERFORM.

LER-REGISTRO.
    READ FUNCIONARIOS INTO DETALHEFUNCIONARIO
        AT END
            MOVE "S" TO LEITURA-FINALIZADA
        NOT AT END
            CONTINUE
    END-READ.

EXIBIR-FUNCIONARIO.
    INSPECT PRIMEIRO-NOME REPLACING ALL " "
        BY LOW-VALUES
    INSPECT ULTIMO-NOME REPLACING ALL " "
        BY LOW-VALUES
    DISPLAY MATRICULA-FUNCIONARIO " " GENERO " "
       PRIMEIRO-NOME " " ULTIMO-NOME.

CONTAR-GENERO.
    IF GENERO = "M"
        ADD 1 TO TOTAL-HOMENS
    ELSE
        IF GENERO = "F"
            ADD 1 TO TOTAL-MULHERES
        END-IF
    END-IF.

EXIBIR-RESUMO.
    DISPLAY "============================".
    DISPLAY "Resumo:".
    DISPLAY " Total de Homens..: " TOTAL-HOMENS.
    DISPLAY " Total de Mulheres: " TOTAL-MULHERES.
    DISPLAY "============================".
```

O resultado da execução é similar ao visto anteriormente, com a grande vantagem que se tivermos que modificar qualquer detalhe do programa saberemos exatamente aonde realizá-lo.



A. Considerações Finais

"Manter sistemas antigos não é apenas preservar o passado, mas garantir que as bases do presente continuem sustentando o futuro."(*Gene Kim*, especialista em DevOps e autor de The Phoenix Project.)

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`

Cobol na Prática

ESTE LIVRO PODE E DEVE SER DISTRIBUÍDO LIVREMENTE

Fernando Anselmo