

LINGUAGEM COBOL

ACUCOBOL



Marcio Adroaldo da Silva
marcio_ad@yahoo.com.br
marcio.infocus@gmail.com
www.controlsyst.com

Janeiro - 2010

INTRODUÇÃO

A organização deste material difere da grande maioria de livros e apostilas que conhecemos, pois o objetivo aqui é fazer com que o interessado conheça aos poucos a linguagem COBOL, de forma que já possa ir utilizando alguns conceitos em atividades práticas mesmo antes de ter visto todo o conteúdo. Isso poderá ser percebido através fato de que os comandos não estão colocados em ordem alfabética e também que o trabalho com arquivos de dados só é abordado após todos os outros comandos, pois as operações com arquivos envolvem detalhes bastante específicos.

Diversos exemplos são apresentados nesse material, e tem apenas o objetivo de demonstrar o uso dos comandos, são exemplos simples, que nem sempre podem ser aplicados a situações reais, mas servem aos propósitos didáticos.

A COBOL é uma linguagem bastante extensa, com recursos poderosos que poucos programadores exploram de verdade. Por isso, não é nosso objetivo esgotar o assunto, o que temos aqui é apenas uma parte básica dos principais comandos. Como em qualquer aprendizado, cabe aos interessados ir além, se esforçar, pesquisar, testar, questionar, procurar, se atualizar e praticar sempre, dessa forma poderá chegar perto do domínio da linguagem.

Sumário

INTRODUÇÃO	2
O QUE É	6
ANSI	6
O futuro	6
PROGRAMA	7
Programa fonte	7
REGRAS DA LINGUAGEM	8
Estrutura e regras de um programa COBOL	8
Regras de codificação	8
Divisões	8
IDENTIFICATION DIVISION	9
ENVIRONMENT DIVISION	9
CONFIGURATION SECTION	9
INPUT-OUTPUT SECTION	9
DATA DIVISION	10
SEÇÕES DA DATA DIVISION	10
CONSTANTES FIGURATIVAS	10
CONDIÇÕES DE CLASSE	10
DEFINIÇÃO DE DADOS	12
NÍVEIS DE DADOS	12
PICTURE	13
PICTURES DE EDIÇÃO	14
NÍVEL 78	15
NÍVEL 88	15
FILE SECTION	17
WORKING-STORAGE SECTION	18
VARIÁVEIS EXTERNAS	18
TABELAS (ARRAY)	19
LINKAGE SECTION	21
SCREEN SECTION	22
PROCEDURE DIVISION	23
ACCEPT	23
DISPLAY	24
GO TO	25
INITIALIZE	26

MOVE	27
MODIFICAÇÃO REFERENCIAL	27
IF, ELSE, END-IF.....	28
EVALUATE, WHEN, OTHER, END-EVALUATE	29
CONTINUE, NEXT SENTENCE	30
CÁLCULOS.....	31
ROUNDED.....	31
ON SIZE ERROR.....	32
COMPUTE.....	33
ADD	34
SUBTRACT.....	35
MULTIPLY	36
DIVIDE	37
FUNÇÕES INTRÍNSECAS.....	38
INTEGER-OF-DATE	40
DATE-OF-INTEGER.....	40
REM	41
CURRENT-DATE	41
WHEN-COMPILED	42
CALL.....	44
ON OVERFLOW.....	44
EXIT.....	45
PERFORM	46
INSPECT	48
STRING.....	50
UNSTRING	52
TRABALHANDO COM ARQUIVOS	53
SELECT	53
O que é uma SELECT ?.....	53
Onde colocamos a SELECT ?.....	53
Tipos de arquivo (ORGANIZAÇÃO).....	53
Arquivos seqüenciais (ORGANIZATION IS SEQUENTIAL).....	53
Arquivos relativos (ORGANIZATION IS RELATIVE).....	53
Arquivos Indexados (ORGANIZATION IS INDEXED).....	54
MÉTODOS DE ACESSO (ACCESS MODE)	54
ÍNDICES DE ACESSO (RELATIVE KEY, RECORD KEY, ALTERNATE RECORD KEY)	54
CONTROLE DE ERROS (FILE STATUS).....	55

MODOS DE BLOQUEIO (LOCK MODE)	56
FD – FILE DESCRIPTION	56
O que é uma FD?	56
Onde colocamos a FD ?	56
COPY de selects e FD	57
Comandos de arquivos.....	58
OPEN	58
CLOSE	58
READ.....	58
WRITE	60
DELETE.....	60
REWRITE.....	60
START	61
INVALID KEY.....	61
Gerando relatórios (arquivos seqüenciais)	64
Definindo a select:	64
Definindo a FD:.....	64
Abrindo a impressora:.....	64
Acucobol.....	67
Compilador.....	67
Executor (runtime)	67
Utilitário de arquivos de dados	67
Utilitário de arquivo objeto.....	68
O arquivo cblconfig	68
Alfred – AcucoboL File Record Editor.....	70
A interface do ALFRED.	71
Principais funções do menu do ALFRED:.....	71
Conclusão	73
Bibliografia	74

O QUE É

COBOL é a abreviação de *Common Business Oriented Language* (Linguagem .

Foi desenvolvida em 1959 por um grupo chamado CODASYL Committee. Conference on Data Systems Languages, era formado por representantes de instituições acadêmicas, grupos de usuários e fabricantes de computadores. O principal objetivo foi desenvolver uma linguagem padrão adequada ao uso comercial para qual todos os grandes fabricantes forneciam compiladores. O Departamento de Defesa dos Estados Unidos da América convocou essa conferência porque, assim como outras agências governamentais, não estava satisfeito com a falta de padrão no campo computacional.

ANSI

Os primeiros compiladores COBOL estavam disponíveis em 1960. Mas os anos se passaram e variações entre os compiladores de fabricantes diferentes começaram a aparecer. Aí o ANSI (American National Standards Institute) entrou em campo para desenvolver padrões para a linguagem COBOL.

Em 1968, estava aprovada a primeira versão ANSI da linguagem (ANSI-68). Em 1974 foi desenvolvida a segunda versão ANSI, tornando a linguagem ainda mais eficiente e padronizada (ANSI-74). A versão de 1985 é a mais largamente utilizada, ela supera as versões anteriores no que diz respeito ao aumento da versatilidade e estrutura da linguagem. O padrão ANSI 2002 definiu ainda mais melhorias ao COBOL, incluindo padronização de OO.

O futuro

O futuro da linguagem tem sido objeto de muita discussão nos últimos anos. Mas é provável que a COBOL permaneça importante ainda por muitos anos. Levemos em consideração os seguintes fatos.

- 1 – De acordo com o Gartner Group, atualmente está em uso uma quantidade estimada de 150 a 250 bilhões de linhas de código COBOL, com programadores adicionando cerca de 5 bilhões de linhas a cada ano*.
- 2 – De acordo com o Datapro Information Services Group, 42,7% de todos os programadores de desenvolvimento de aplicações, em grandes ou médias companhias dos EUA utilizam COBOL*.
- 3 – Espera-se que os ganhos com o desenvolvimento do COBOL sejam superiores a US\$ 200 milhões por volta do ano 2000; esses ganhos aumentaram de US\$ 86,3 milhões, em 1993, a uma taxa média de crescimento de 15,4% ao ano*.

A linguagem é fortemente utilizada principalmente em grandes corporações (bancos, seguradoras, instituições financeiras e de crédito, indústrias, transportes e no governo) além de empresas menores como Software houses que desenvolvem aplicações para pequenas e médias empresas.

Sua utilização ocorre tanto na manutenção de sistemas legados quanto na criação de novos aplicativos.

Um bom indicativo de que o mercado COBOL está forte, é o fato da Microfocus ter adquirido em 2 anos as seguintes empresas: ACUCORP, BORLAND E RM.

* Dados de 2000

PROGRAMA

Um programa é um conjunto de instruções, dispostas de forma lógica e coerente com o objetivo de realizar determinada tarefa.

Programa fonte

É o arquivo texto contendo as instruções a serem executadas, respeitando as normas de uma linguagem. O fonte é o que o programador escreve em editores de texto comuns, e que deve ser compilado para gerar o objeto, que é o arquivo que realmente pode ser executado no computador. O compilador vai avaliar se o fonte está de acordo com as regras da linguagem, e somente se nenhum erro for encontrado o objeto será gerado. No livro “Microsoft COBOL 4.0”, José Eduardo Maluf de Carvalho, compara um programa a uma receita de um prato, pois contém todas as instruções detalhadas para o preparo desse prato. Essa colocação nos permite uma analogia interessante, pois um programa também é um conjunto de instruções detalhadas sobre algo que se deseja realizar.

REGRAS DA LINGUAGEM

Estrutura e regras de um programa COBOL

A COBOL possui a fama de ter uma estrutura extremamente rígida e limitada, o que exige do programador muita atenção a detalhes que não deveriam interferir diretamente no objetivo final do programa. Porém, a linguagem tem evoluído bastante e se tornou mais flexível no que diz respeito às regras de codificação.

Regras de codificação

No seu editor de textos preferido, deve-se respeitar o seguinte:

Colunas	Definição
1 a 6	Uso da linguagem
7	Reservada. Um asterisco nessa coluna ignora a linha toda
8 a 11	Área A. Aqui vão as DIVISIONS, SECTIONS, PARAGRAFOS, e os níveis 77, 78 e 01
12 a 72	Área B. Os comandos propriamente ditos devem ficar nessa área.

Atualmente, em muitos compiladores, isso não é mais necessário. Mas é importante conhecer essas regras, pois dependendo da versão da sua ferramenta COBOL, poderá se deparar com programas utilizando essa estrutura.

Divisões

A COBOL possui 4 DIVISIONS (divisões), que por sua vez possuem SECTIONS (seções). Cada divisão/seção tem um objetivo bem específico, o que torna o programa muito legível e fácil de entender, pois cada definição fica em um local específico.

As divisões são:

IDENTIFICATION DIVISION. *(divisão de identificação)*

ENVIRONMENT DIVISION. *(divisão de ambiente)*

DATA DIVISION. *(divisão de dados)*

PROCEDURE DIVISION. *(divisão de procedimentos)*

A seguir, veremos cada uma dessas divisões com suas respectivas seções.

IDENTIFICATION DIVISION

É a divisão de identificação do programa. Aqui as informações tem o objetivo principal de documentação do fonte. Essa divisão não possui seções.

IDENTIFICATION DIVISION. PROGRAM-ID. TESTE1. AUTHOR. FULANO DE TAL. DATE-WRITTEN. NOVEMBRO/2009. DATE-COMPILED. INSTALLATION. EMPRESA ACME. SECURITY. CADASTRO DE TESTE APENAS. REMARKS. EXEMPLO DO CURSO - NOV/2009.	DIVISÃO DE IDENTIFICAÇÃO. NOME DO PROGRAMA. TESTE1. AUTOR. FULANO DE TAL. DATA EM QUE FOI ESCRITO. NOVEMBRO/2009. DATA EM QUE FOI COMPILADO. INSTALAÇÃO. EMPRESA ACME. COMENTÁRIO. COMENTÁRIO.
--	---

ENVIRONMENT DIVISION.

É a divisão de ambiente, onde se especifica o método de expressão dos aspectos do processamento de dados que dependem das características físicas do computador. E pode ser dividida em duas seções:

CONFIGURATION SECTION

INPUT-OUTPUT SECTION.

CONFIGURATION SECTION

Trata das características do computador fonte e do computador objeto. É subdividida em 3 cláusulas opcionais:

SOURCE-COMPUTER: descreve a configuração na qual é gerado o código intermediário do programa, após compilação.

OBJECT-COMPUTER: descreve a configuração na qual o programa será executado.

SPECIAL-NAMES: relata os nomes das implementações que o seu sistema COBOL usará nos programas fontes. Por exemplo:

DECIMAL POINT IS COMMA : define que o separador decimal será a vírgula (,) e não o ponto (.), que é o padrão. Atenção a isso, pois definirá a forma com que trabalhará com constantes e instruções de valores no seu programa.

INPUT-OUTPUT SECTION

Trata com todas as informações necessárias para controlar a transmissão e manipulação de dados entre meios externos e o programa, e possui dois parágrafos:

FILE-CONTROL. Dá os nomes aos arquivos do programa e os associa aos meios externos.

I-O CONTROL. Define técnicas de controle especiais a serem usadas pelo programa.

DATA DIVISION

Essa é a divisão de dados, todos os dados com que o programa vai trabalhar são definidos nessa divisão.

SEÇÕES DA DATA DIVISION

FILE SECTION.	SEÇÃO DE ARQUIVO
WORKING-STORAGE SECTION.	SEÇÃO DE ARMAZENAMENTO DE TRABALHO
LINKAGE SECTION.	SEÇÃO DE VÍNCULO, CONEXÃO, LIGAÇÃO.
SCREEN SECTION.	SEÇÃO DE TELA

Antes de vermos as seções da DATA DIVISION, vamos detalhar alguns conceitos da linguagem que são utilizados nessa divisão:

CONSTANTES FIGURATIVAS

São literais geradas pelo compilador e usados como palavras reservadas. Algumas dessas palavras são descritas abaixo. Pode-se usar tanto o plural quanto o singular.

ZERO, ZEROS, ZEROES Representa o valor numérico "zero" ou uma ou mais ocorrências do carácter 0.

SPACE, SPACES Representa um ou mais espaços.

HIGH-VALUE, HIGH-VALUES Representa um ou mais caracteres com valores-altos. Usualmente é o hexadecimal "FF".

LOW-VALUE, LOW-VALUES Representa um ou mais caracteres com valores-baixos. Usualmente é o binário 0.

ALL Literal Representa um conjunto de caracteres pré definido.

NULL, NULLS Representa o valor numérico "zero". Também representa um endereço inválido de memória quando usado em conjunto com tipos de dados POINTER. figurative constants for a literal restricted to numeric literals.

CONDIÇÕES DE CLASSE

A condição de classe é usada para testes onde se deseja saber se uma variável é formada ou não de um tipo particular de dados.

NUMERIC Numérico, caracteres de 0 a 9.

ALPHABETIC Alfabético, caracteres de A - Z, de a - z e espaços.

ALPHABETIC-UPPER Alfabético, caracteres de A - Z, e espaços.

ALPHABETIC-LOWER Alfabético, caracteres de a - z, e espaços.

```
IF VARIAVEL IS [NOT] {NUMERIC } {ALPHABETIC } {ALPHABETIC-UPPER}  
    {ALPHABETIC-LOWER} {nome-de-classe }
```

Ex:

```
IF NOME IS NOT ALPHABETIC-UPPER  
    DISPLAY MESSAGE BOX "INFORME APENAS DE A - Z E ESPAÇOS"  
END-IF
```

DEFINIÇÃO DE DADOS

As regras para nomes de variáveis são:

- No máximo 30 caracteres
- Pode começar com um número, mas precisa ter pelo menos uma letra
- Não podem ser utilizadas palavras reservadas da linguagem
- Pode conter apenas os caracteres de A até Z, de 0 até 9 e também “-” e “_”, e nenhum outro tipo de caracter é permitido.

NÍVEIS DE DADOS

As variáveis são definidas por níveis.

Esses níveis lembram um plano de contas, dessa forma é fácil entender estruturas de dados com itens hierárquicos.

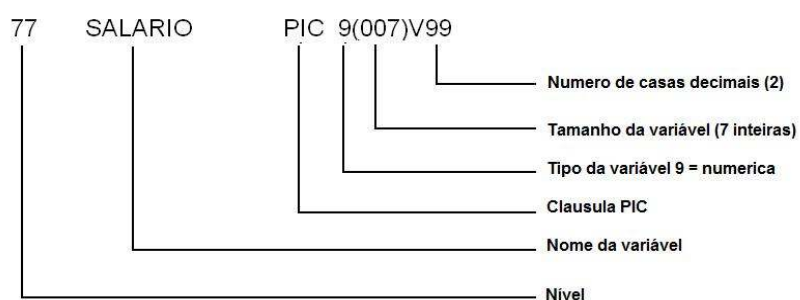
Os níveis são:

01 a 49 – Definição de itens de estruturas.

77 – Itens elementares não fazem parte de nenhuma hierarquia

78 - Constantes

88 – Nomes de condição



Os tipos de dados são definidos através da cláusula PICTURE, ou simplesmente PIC.

EX:

WORKING-STORAGE SECTION.

```
77 SALARIO          PIC 9(007)V99.
77 NOME              PIC X(030).
77 SALDO             PIC S9(010)V99.
77 INDICE            PIC 9(007)V9(008).
01 DATA-DE-HOJE.
   05 ANO             PIC 9(004).
   05 MES             PIC 9(002).
   05 DIA             PIC 9(002).
```

Ao definir variáveis podemos ter a seguinte estrutura:

- Nível
- Nome
- PIC
- Tipo
- Tamanho
- Decimal
- Casas decimais
- Valor inicial
- e outras definições (OCCURS, EXTERNAL, USAGE "comp, comp-3, comp-1"...)

Ex:

Nível	Nome	PIC	Tipo	Tamanho	Decimal	Valor inicial	Outras definições
77	salario	pic	9	(010)	V99		Occurs 10
77	nome	pic	x	(030)			External
01	uf	pic	a	(002)		Value spaces	
01	calculo	pic	9	(010)			Comp

PICTURE

A cláusula Picture identifica o tipo de dado que se está definindo, podendo ser:

Símbolo	Significado
X	Campo alfanumérico
9	Campo numérico
A	Campo alfabético
V	Ponto decimal assumido; usado apenas em campos numéricos
S	Sinal operacional; usado apenas em campos numéricos
Z	<i>Caracter de supressão de zeros</i>
.	<i>Ponto decimal</i>
+	<i>Sinal positivo</i>
-	<i>Sinal negativo</i>
\$	<i>Cifrão</i>
,	<i>Vírgula</i>
CR	<i>Símbolo de crédito</i>
DB	<i>Símbolo de débito</i>
*	<i>Símbolo de proteção de valores</i>
B	<i>Separador de campos - caracter de inserção de espaço</i>
0	<i>(zero) Caracter de inserção de zero</i>
/	<i>Caracter de inserção de barras</i>

PICTURES DE EDIÇÃO

As pictures de edição são muito usadas em relatórios e saídas de tela, pois permitem que os dados sejam mostrados de forma mais amigável ao usuário.

Ex:

```
77 VALOR          pic 9(005)v99 value 456.89.  
quando mostrado na tela apresentará: 045689
```

```
77 VALOR          pic ZZZZ9.99.  
apresentará na tela: 456.89
```

Suponhamos o cgc 76080738001069.

```
77 CGC            pic 9(014).  
apresentará na tela: 76080738001069.
```

```
77 CGC            pic ZZ.ZZZ.ZZ9/9999B99.  
apresentará na tela: 76.080.738/0010 69
```

Observação: As pictures de edição só podem ser usadas em cálculos quando forem armazenar apenas o resultado da operação.

```
77 VALOR          PIC ZZZZZ9,99.  
77 PRECO          PIC 9(005)V99.  
77 QTDE          PIC 9(004).
```

```
COMPUTE VALOR = PRECO * QTDE.
```

VÁLIDO, POIS VALOR VAI APENAS RECEBER O RESULTADO DA OPERAÇÃO

```
77 VALOR          PIC ZZZZZ9,99.  
77 PRECO          PIC ZZZZ9,99.  
77 QTDE          PIC 9(004).
```

```
COMPUTE VALOR = PRECO * QTDE.
```

INVÁLIDO, POIS PREÇO, SENDO DE EDIÇÃO NÃO PODE SER USADO NO CÁLCULO

Exemplos de pictures de edição com o que será apresentado ao usuário

Picture de origem	Valor	Picture de destino	Valor exibido
S9(005)V99	0035488	ZZZZ9.99.	354.88
9(007)V99	000654321	+++++9.99.	+6543.21
9(005)V99	0135862	***.**9,99	**1.358,62
9(008)	12052009	99/99/9999	12/05/2009
9(014)	76080738001069	ZZ.ZZZ.ZZ9/9999B99	76.080.738/0010 69
S9(003)V9	-0056	--9,9	-5,6
9(004)V999	0056987	\$.\$\$9,999	\$56,987
9(005)	98763	9099099	9087063
X(008)	"ABACAXI"	XBXBXBXBXBXBXB	A B A C A X I
X(008)	"ABACAXI"	XXBXXBXXBXX	AB AC AX I

NÍVEL 78

Nível 78 é usado para definir constantes, que são valores imutáveis, por isso não precisam ser definidos em variáveis, normalmente são utilizados para facilitar a codificação, dando nomes significativos a valores.

Ex:

```
78 PI          VALUE 3.14159.
78 AZUL        VALUE 1.
78 BRANCO      VALUE 7.
```

NÍVEL 88

Nível 88 é utilizado para definição de condição de variáveis, sua utilização é opcional, mas costuma tornar o código muito mais legível.

Ex:

```
77 HORA          PIC 9(002).
88 HORA-OK VALUES 0 THRU 23.

IF NOT HORA-OK
    DISPLAY MESSAGE BOX "HORA INVALIDA !!"
END-IF
```

Agora, sem o nível 88

```
77 HORA          PIC 9(002).

IF HORA > 23
    DISPLAY MESSAGE BOX "HORA INVALIDA !!"
END-IF
```



```
77 IDADE          PIC 9(003).  
    88 CRIANCA          VALUE 0 THRU 10.  
    88 ADOLESCENTE      VALUE 11 THRU 19.  
    88 JOVEM            VALUE 20 THRU 30.  
    88 ADULTO           VALUE 31 THRU 60.  
  
...  
EVALUATE TRUE  
    WHEN CRIANCA  DISPLAY MESSAGE BOX "CRIANCA"  
    WHEN ADOLESCENTE  DISPLAY MESSAGE BOX "ADOLESCENTE"  
    WHEN JOVEM  DISPLAY MESSAGE BOX "JOVEM"  
    WHEN ADULTO  DISPLAY MESSAGE BOX "ADULTO"  
    WHEN OTHER  DISPLAY "OUTRA CATEGORIA"  
END-EVALUATE
```

FILE SECTION

Nesta seção são definidas as estruturas dos arquivos de dados, isso é feito utilizando-se as regras de definição de dados, aliadas a cláusulas específicas desta seção. Vamos abortar essa seção mais detalhadamente quando passarmos a trabalhar com arquivos de dados.

WORKING-STORAGE SECTION

Nesta seção serão definidas as variáveis que serão utilizadas durante o processamento, variáveis que somente existirão em memória, pois não serão armazenadas em nenhum outro local. Podemos dizer que é o local da memória que o programa vai utilizar para “rascunho”.

Ex.

```
WORKING-STORAGE SECTION.
01 DESCRICAO                PIC X(030).
77 CONTADOR                 PIC 9(003) VALUE 0.
01 SALDO                    PIC S9(010)V999 VALUE 0.
01 PLACA-VEICULO.
    05 LETRAS                PIC X(003).
    05 NUMEROS               PIC 9(004).
01 DATA-ATUAL              PIC 9(008).
01 FILLER REDEFINES DATA-ATUAL.
    05 ANO-ATUAL             PIC 9(004).
    05 MES-ATUAL             PIC 9(002).
    05 DIA-ATUAL             PIC 9(002).
77 INDEXADOR                PIC 9(003) VALUE 0.
77 INDICE-DE-CALCULO        PIC 9(005)V9(007) VALUE 0.
01 MEUS-CALCULOS.
    05 SALARIO-ATUAL         PIC 9(007)V99.
    05 NOVO-SALARIO         PIC 9(007)V99.
    05 VALOR-DE-AUMENTO     PIC 9(007)V99.
    05 PERCENTUAL-AUMENTO   PIC 9(002)V99.
    05 HORAS-TRABALHADAS    PIC 9(003)V9.
    05 RASCUNHO             PIC 9(005)V999.
```

VARIÁVEIS EXTERNAS

Podemos definir variáveis que serão “enxergadas” por todos os programas de uma seção, isso é bastante útil, pois as variáveis podem ser alimentadas por um programa e referenciadas em outros programas da mesma seção. Um bom exemplo disso é o nome do usuário que se logou no sistema, se for definido como EXTERNAL, todos os sub-programas poderão utilizar essa informação.

IMPORTANTE: variáveis externas devem ser definidas com o mesmo nome, tipo e tamanho, caso contrário não funcionarão de acordo com o desejado.

<pre>PROGRAM-ID. EXT01. WORKING-STORAGE SECTION. 01 USUARIO PIC X(015) EXTERNAL. PROCEDURE DIVISION. DISPLAY "USUARIO: " ACCEPT USUARIO CALL "EXT02" CANCEL "EXT02" GOBACK.</pre>	<pre>PROGRAM-ID. EXT02. WORKING-STORAGE SECTION. 01 USUARIO PIC X(015) EXTERNAL. PROCEDURE DIVISION. DISPLAY MESSAGE BOX "O USUARIO É: " USUARIO GOBACK.</pre>
--	---

Um bom exemplo do uso de variáveis externas é o controle de acesso de um sistema, onde o usuário identifica-se , através de uma tela de login, e os demais programas chamados durante a seção poderão saber qual o usuário logado, sem a necessidade da passagem de parâmetros via linkage.

TABELAS (ARRAY)

Em COBOL, definimos tabelas (é assim que chamamos os arrays), através da cláusula **OCCURS**, que pode estar na WORKING-STORAGE SECTION, na LINKAGE SECTION, ou ainda nas definições de arquivos de dados FILE SECTION. (FD).

O primeiro contato com tabelas, (principalmente com mais de 1 dimensão), pode causar um certo desconforto, mas uma vez dominado é um recurso poderoso, que pode ser usado na solução de diversos problemas.

O uso de tabelas, permite que sejam criadas variáveis sem a necessidade de repetição, isso economiza muito tempo, mas nesse caso, é necessária a utilização de índices, que tem por função determinar exatamente qual ocorrência da variável deve ser utilizada. Sempre que for feita referência a uma variável que contém a cláusula OCCURS, é obrigatório o uso do indexador, caso contrário ocorrerá um erro de compilação.

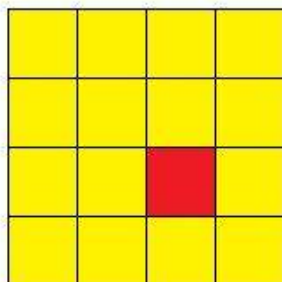


1 Dimensão: apenas um identificador permite localizar o item (linha)

```
77 CODIGO          PIC 9(004) OCCURS 4.
```

...

```
MOVE CODIGO(3) TO CONTROLE.
```



2 Dimensões: para localizar um item precisamos de dois identificadores (linha e coluna)

```
01 OCCURS 4.
```

```
05 OCCURS 4.
```

```
10 CODIGO          PIC 9(004).
```

...

```
MOVE CODIGO(3, 2) TO CONTROLE.
```

Um exemplo de tabela bi-dimensional é o EXCEL, que utiliza linhas numeradas e colunas identificadas por letras, de forma que para saber o conteúdo de uma cédula é necessário utilizar 2 identificadores (coluna e linha), como por exemplo (A 1, B 7).

<pre>77 NOME PIC X(030) OCCURS 10. 77 INDICE PIC 9(002). PROCEDURE DIVISION. INICIO. DISPLAY WINDOW ERASE ADD 1 TO INDICE IF INDICE > 10 DISPLAY MESSAGE BOX "LIMITE DE 10 NOMES ATINGIDO" GO TO ENCERRA END-IF DISPLAY "INFORME O NOME " INDICE ": " ACCEPT NOME(INDICE) GO TO INICIO. ENCERRA. GOBACK.</pre>	<pre>INICIO. MOSTRA TELA EM BRANCO ADICIONE 1 AO INDICE SE O INDICE FOR MAIOR QUE 10 MOSTRE A MENSAGEM "LIMITE DE 10 NOMES ATINGIDO" VA PARA ENCERRA SE-FIM MOSTRE "INFORME O NOME " INDICE ": " RECEBA NOME (IDENTIFICADO PELO INDICE) VA PARA O INICIO. ENCERRA.</pre>
---	--

3 DIMENSÕES (1 CUBO)

01 TABELA.

05 OCCURS 10.

10 OCCURS 10

15 OCCURS 10.

20 TAB-PRECO

PIC 9(005)V99.

LINKAGE SECTION

Nesta seção serão definidas as variáveis que serão utilizadas para a comunicação de parâmetros entre programas, e é definida apenas no programa chamado. Desta forma permite que ele receba dados de diversos outros programas.

Ex.

LINKAGE SECTION.

```
01  DADOS-DE-CALCULO.  
    05  SALARIO-ATUAL      PIC 9(009)V99.  
    05  PERCENTUAL        PIC 9(003)V99.  
    05  NOVO-SALARIO      PIC 9(009)V99.
```

Esta seção não é obrigatória se o programa chamado não receber nenhum parâmetro do chamador, mas do contrário deve ser definida.

Deve-se ter uma atenção especial para que os dados definidos na linkage sejam do mesmo tipo, tamanho e estrutura que foram definidos no programa chamador.

SCREEN SECTION.

A SCREEN SECTION permite a formatação e desenho de layout de telas de forma bastante simples, pois o trabalho é feito através da especificação das características de cada componente da tela. Em cada tela, podemos fazer referência a qualquer item de dado definido anteriormente (FILE SECTION, WORKING-STORAGE SECTION, LINKAGE SECTION). Há muitos anos, trabalhar com SCREEN SECTION, possibilita ao programador COBOL o uso de diversas ferramentas do tipo WYSIWYG, mesmo em ambientes CUI. É extremamente mais eficiente que o tradicional ACCEPT, DISPLAY de itens individuais, onde o programa fica imenso e com dificuldades de manutenção. O uso da SCREEN SECTION, nos permite vislumbrar a programação orientada a eventos e também a interface gráfica.

Ex.

<pre>WORKING-STORAGE SECTION. 01 CLIENTES. 05 CLI-CODIGO PIC 9(004). 05 CLI-NOME PIC X(030). SCREEN SECTION. 01 TELA. 05 LINE 01 COL 01 VALUE "TELA". 05 LINE 03 COL 06 VALUE "Codigo: ". 05 T-CODIGO ENTRY-FIELD USING CLI-CODIGO AUTO REVERSE REQUIRED PIC ZZZ9 AFTER PROCEDURE CONTROLE-CODIGO. 05 LINE 04 COL 06 VALUE "Nome: ". 05 T-NOME ENTRY-FIELD USING CLI-NOME AUTO UPPER REQUIRED. PROCEDURE DIVISION. INICIO. DISPLAY TELA ACCEPT TELA GOBACK. CONTROLE-CODIGO. DISPLAY MESSAGE BOX "ESSE CONTROLE É EXECUTADO" " DEPOIS DE RECEBER O CAMPO CODIGO".</pre>	<pre>TELA. LINHA 1 COLUNA 1 VALE "TELA" LINHA 3 COLUNA 6 VALE "Código: " T-CODIGO CAMPO-DE-ENTRADA USANDO CLI-CODIGO PICTURE ZZZ9 DEPOIS CONTROLE-CODIGO. LINHA 4 COLUNA 6 VALE "Nome: " T-NOME CAMPO-DE-ENTRADA USANDO CLI-NOME. INICIO. MOSTRE TELA RECEBA A TELA ENCERRE. CONTROLE-CODIGO. MOSTRE MENSAGEM "ESSE CONTROLE É EXECUTADO" " DEPOIS DE RECEBER O CAMPO CODIGO".</pre>
---	--

Muitos dos complementos utilizados no comando ACCEPT podem ser colocados nos itens de tela também.

EX: NO-CHO, SECURE, REQUIRED, AUTO, OFF.

PROCEDURE DIVISION

Esta é a divisão de procedimentos, é aqui que são escritos os comandos. Até aqui, tudo o que foi feito foi a definição de estruturas de dados e a preparação do ambiente do programa. A partir de agora é que dizemos o que e como queremos que o programa execute o que desejamos. Para isso utilizaremos os comandos da linguagem, que devem também respeitar regras pré-estabelecidas. Ainda dentro da procedure division, o programador pode definir nomes que permitam o desvio e controle do fluxo de execução do programa, esses nomes podem ser seções (SECTIONS) e/ou parágrafos (PARAGRAPH). Isso funciona como rótulos ou identificadores.

ACCEPT

O ACCEPT é utilizado para receber dados que o usuário informa na tela, e também para obter algumas informações do sistema, como data e hora atual, além de informações mais avançadas. Esse comando tem diversas formas de utilização (11 no Extend), mas aqui veremos apenas as mais simples.

SINTAXE:

ACCEPT [ITEM, VARIÁVEL] AT [LINHA COLUNA] [REVERSE, REQUIRED, UPDATE, FULL, OFF, NO-ECHO, SECURE, PROMPT, UPPER, LOWER, AUTO, COLOR, CONVERT]

CONTROL KEY IN [VARIÁVEL]
BEFORE TIME [CONSTANTE OU VARIÁVEL]

ACCEPT [VARIÁVEL] FROM [CENTURY-DATE, TIME...]

Ex:

<pre>WORKING-STORAGE SECTION. 77 NOME PIC X(030) VALUE SPACES. PROCEDURE DIVISION. ACCEPT NOME AT 1010 REQUIRED REVERSE ACCEPT OMITTED *> AGUARDA UMA TECLA GOBACK.</pre>	<pre>RECEBA NOME NA LINHA 10 COLUNA 10 RECEBA UMA TECLA (ESPERE UMA TECLA)</pre>
<pre>WORKING-STORAGE SECTION. 77 DATAATUAL PIC 9(008). 77 HORAATUAL PIC 9(008). PROCEDURE DIVISION. ACCEPT DATAATUAL FROM CENTURY-DATE ACCEPT HORAATUAL FROM TIME GOBACK.</pre>	<pre>RECEBA DATAATUAL DA DATA DO COMPUTADOR RECEBA HORAATUAL DA HORA DO COMPUTADOR</pre>

DISPLAY

O comando DISPLAY é usado para a exibição de informações na tela, é usado para exibição de mensagens, textos, variáveis, telas e controles de tela. Também possui muitas formas de utilização, e aqui descreveremos as mais comuns.

SINTAXE:

DISPLAY [ITEM, VARIÁVEL] AT [LINHACOLUNA] [REVERSE, UPPER, LOWER, COLOR, SECURE, CONVERT, BOLD]

DISPLAY MESSAGE BOX [MENSAGEM] [TITLE <VARIÁVEL/STRING>]

Ex:

<pre>WORKING-STORAGE SECTION. 77 NOME PIC X(030) VALUE "JOÃO DE SOUZA MELLO". PROCEDURE DIVISION. DISPLAY "Nome do Cliente: " AT 1010 NOME REVERSE GOBACK.</pre>	<p>MOSTRE "NOME DO CLIENTE" NA LINHA 10 COLUNA 10 E O NOME EM VIDEO REVERSO</p>
<pre>WORKING-STORAGE SECTION. 77 DATAATUAL PIC 9(008). 77 HORAATUAL PIC 9(008). PROCEDURE DIVISION. ACCEPT DATAATUAL FROM CENTURY-DATE ACCEPT HORAATUAL FROM TIME DISPLAY DATAATUAL AT 1010 DISPLAY HORAATUAL AT 1210 DISPLAY DATAATUAL(7:2) AT 1310 "/" DATAATUAL(5:2) "/" DATAATUAL(1:4) GOBACK.</pre>	<p>RECEBA DATAATUAL DA DATA DO COMPUTADOR RECEBA HORAATUAL DA HORA DO COMPUTADOR MOSTRE O CONTEUDO DE DATAATUAL MOSTRE O CONTEÚDO DE HORAATUAL MOSTRE OS CARACTERES 7 E 8 DE DATAATUAL "/" OS CARACTERES 5 E 6 E OS CARACTERES DE 1 A 4</p>
<pre>WORKING-STORAGE SECTION. 77 NOME PIC X(030) VALUE "JOÃO DE SOUZA MELLO". PROCEDURE DIVISION. DISPLAY MESSAGE BOX "O nome do cliente é" H"0A" NOME TITLE "Mensagem" GOBACK.</pre>	<p>MOSTRE UMA CAIXA DE MENSAGEM COM "O NOME DO CLIENTE É " PULE UMA LINHA MOSTRE TAMBÉM A VARIÁVEL NOME COLOQUE COMO TITULO "MENSAGEM"</p>
<pre>WORKING-STORAGE SECTION. 77 NOME PIC X(030) VALUE "JOÃO DE SOUZA MELLO". PROCEDURE DIVISION. DISPLAY WINDOW ERASE *> LIMPA A TELA DISPLAY MESSAGE BOX "OK" GOBACK.</pre>	<p>LIMPE A TELA MOSTRE UMA CAIXA DE MENSAGEM COM "OK"</p>

GO TO

Utilizado para desvio no fluxo do programa. Para que possa ser utilizado é necessária a definição de parágrafos, que são rótulos ou endereços que dividem a aplicação em blocos.

SINTAXE:

GO [TO] <PARÁGRAFO> [DEPENDING ON <VARIÁVEL>].

Ex:

<pre>WORKING-STORAGE SECTION. 77 NUMERO PIC 9(004). PROCEDURE DIVISION. INICIO. DISPLAY "INFORME O NUMERO (ZERO ABANDONA): " ACCEPT NUMERO REVERSE CONVERT AUTO IF NUMERO = 0 GO TO FIM END-IF DISPLAY MESSAGE BOX "VOCE DIGITOU O NUMERO: " NUMERO GO TO INICIO. FIM. GOBACK.</pre>	<pre>INICIO. MOSTRE "INFORME O NUMERO (ZERO ABANDONA): " RECEBA NUMERO SE NUMERO = 0 VA PARA FIM SE-FIM MOSTRE MENSAGEM "VOCE DIGITOU O NUMERO: " NUMERO VA PARA INICIO. FIM. ENCERRA.</pre>
<pre>WORKING-STORAGE SECTION. 77 NUMERO PIC 9(001). 88 NUMERO-VALIDO VALUES 1 THRU 3. PROCEDURE DIVISION. INICIO. DISPLAY "INFORME NUMERO DO PROCESSO (0=SAIR): " ACCEPT NUMERO REVERSE CONVERT AUTO IF NUMERO = 0 GO TO FIM END-IF IF NOT NUMERO-VALIDO DISPLAY MESSAGE BOX "INFORME DE 1 A 3 APENAS !" GO TO INICIO END-IF GO TO PROC-1, PROC-2, PROC-3 DEPENDING ON NUMERO. PROC-1. PROC-2. PROC-3. FIM. GOBACK.</pre>	<pre>INICIO. MOSTRE "INFORME NUMERO DO PROCESSO (0=SAIR): " RECEBA NUMERO SE NUMERO = 0 VA PARA FIM SE-FIM SE NÃO FOR NUMERO-VALIDO MOSTRE MENSAGEM "INFORME DE 1 A 3 APENAS !!" VA PARA INICIO SE-FIM VA PARA PROC-1, PROC-2, PROC-3 DEPENDENDO DO NUMERO. PROC-1. PROC-2. PROC-3. FIM. ENCERRA.</pre>

Importante: com a chegada do padrão ANSI 85, a programação estruturada se tornou simples, e a criação de programas COBOL sem GO TO ficou bastante fácil. O uso do GO TO é criticado por muitos, pois alegam que com esse comando é fácil do programador se perder ao direcionar o fluxo de execução do programa. Mas, há muitas aplicações que fazem uso desse comando, que no final das contas exige o mesmo cuidado e atenção que vários outros comandos de qualquer linguagem.

INITIALIZE

Esta declaração proporciona a habilidade de ajustar categorias de dados a valores predeterminados, como por exemplo, ajustar todos os itens alfanuméricos em espaços em branco.

SINTAXE:

INITIALIZE [VARIÁVEL] REPLACING [ALPHABETIC, ALPHANUMERIC, NUMERIC, ALPHANUMERIC-EDITED, NUMERIC-EDITED, DATA] BY [IDENTIFICADOR OU VALOR]

Ex:

<pre>WORKING-STORAGE SECTION. 01 CLIENTE. 05 CODIGO PIC 9(004). 05 NOME PIC X(030). 05 NASCIMENTO PIC 9(008). 05 TELEFONE PIC X(018). PROCEDURE DIVISION. INITIALIZE CLIENTE REPLACING NUMERIC BY ZEROS ALPHANUMERIC BY SPACES.</pre>	<pre>INICIALIZAR CLIENTE TROCANDO O CONTEÚDO NAS NUMÉRICAS POR 0 E DAS ALFANUMÉRICAS POR ESPAÇOS</pre>
---	--

MOVE

Esta declaração é uma das mais usadas em COBOL pois é a que transfere dados, ou seja, associa um determinado valor ou um item a outro ou outros itens de dados.

SINTAXE:

MOVE [VARIÁVEL, CONSTANTE, STRING] TO [VARIÁVEL] [CORR]

Ex:

<pre>WORKING-STORAGE SECTION. 77 NOME PIC X(030) VALUE SPACES. 77 OUTRONOME PIC X(030) VALUE SPACES. PROCEDURE DIVISION. ACCEPT NOME AT 1010 REQUIRED REVERSE MOVE NOME TO OUTRONOME ACCEPT OMITTED *> AGUARDA UMA TECLA GOBACK.</pre>	<p>RECEBA A VARIÁVEL NOME (REQUERIDA) COPIE O CONTEÚDO DE NOME PARA OUTRONOME ESPERE UMA TECLA</p>
---	--

<pre>WORKING-STORAGE SECTION. 01 CLIENTE. 05 CODIGO PIC 9(004). 05 NOME PIC X(030). 05 NASCIMENTO PIC 9(008). 05 TELEFONE PIC X(018). PROCEDURE DIVISION. MOVE SPACES TO NOME, TELEFONE MOVE ZEROS TO CODIGO NASCIMENTO.</pre>	<p>MOVA ESPAÇOS PARA NOME E TELEFONE MOVA ZEROS PARA CODIGO E NASCIMENTO</p>
--	--

<pre>WORKING-STORAGE SECTION. 77 DIADASEMANA PIC 9(001). 77 NOMEODIA PIC X(010). PROCEDURE DIVISION. ACCEPT DIADASEMANA FROM DAY-OF-WEEK EVALUATE DIADASEMANA WHEN 1 MOVE "SEGUNDA" TO NOMEODIA WHEN 2 MOVE "TERÇA" TO NOMEODIA WHEN 3 MOVE "QUARTA" TO NOMEODIA WHEN 4 MOVE "QUINTA" TO NOMEODIA WHEN 5 MOVE "SEXTA" TO NOMEODIA WHEN 6 MOVE "SABADO" TO NOMEODIA WHEN 7 MOVE "DOMINGO" TO NOMEODIA END-EVALUATE DISPLAY NOMEODIA AT 1010 REVERSE ACCEPT OMITTED GOBACK.</pre>	<p>RECEBA DIADASEMANA DO DIA ATUAL AVALIE DIADASEMANA QUANDO 1 MOVA "SEGUNDA" PARA NOMEODIA QUANDO 2 MOVA "TERÇA" PARA NOMEODIA QUANDO 3 MOVA "QUARTA" PARA NOMEODIA QUANDO 4 MOVA "QUINTA" PARA NOMEODIA QUANDO 5 MOVA "SEXTA" PARA NOMEODIA QUANDO 6 MOVA "SABADO" PARA NOMEODIA QUANDO 7 MOVA "DOMINGO" PARA NOMEODIA AVALIE-FIM MOSTRE NOMEODIA ESPERE UMA TECLA</p>
--	--

MODIFICAÇÃO REFERENCIAL

O comando MOVE também permite trabalhar com partes de campos, e isso é determinado informando-se a posição inicial e o número de caracteres a mover separados por “:”.

Ex.:

<pre>MOVE PLACA(1:3) TO LETRAS MOVE CODIGO(4:6) TO NUMERO</pre>	<p>MOVA PLACA(A PARTIR DA POSICAO 1: 3 CARACTERES) PARA LETRAS MOVA CODIGO(A PARTIR DA POSICAO 4: 6 CARACTERES) PARA NUMERO</p>
---	---

IF, ELSE, END-IF

São instruções condicionais que permitem ao programador decidir sobre os rumos da aplicação, são comandos muito utilizados no dia a dia.

<pre>IF NOME = SPACES EXIT PARAGRAPH ELSE DISPLAY NOME END-IF</pre>	<pre>SE NOME = ESPAÇOS SAIA DO PARAGRAFO SENÃO MOSTRE NOME SE-FIM</pre>
<pre>IF OPCA0 = 1 CALL "PRG01" CANCEL "PRG01" ELSE IF OPCA0 = 2 CALL "PRG02" CANCEL "PRG02" ELSE IF OPCA0 = 3 CALL "PRG03" CANCEL "PRG03" ELSE DISPLAY "OPCA0 INVALIDA!!" END-IF END-IF END-IF</pre>	<pre>SE OPCA0 = 1 CHAME "PRG01" CANCELE "PRG01" SENÃO SE OPCA0 = 2 CHAME "PRG02" CANCELE "PRG02" SENÃO SE OPCA0 = 3 CHAME "PRG03" CANCELE "PRG03" SENÃO MOSTRE "OPCA0 INVALIDA !!" SE-FIM SE-FIM SE-FIM</pre>
<pre>IF QTDE NOT NUMERIC DISPLAY MESSAGE BOX "CAMPO NÃO NUMERICO" EXIT PARAGRAPH END-IF</pre>	<pre>SE QTDE NÃO NUMERICO MOSTRE A MENSAGEM "CAMPO NÃO NUMERICO" SAIA DO PARAGRAFO SE-FIM</pre>
<pre>IF SALDO IS NEGATIVE DISPLAY MESSAGE BOX "SALDO NEGATIVO NÃO PERMITIDO !!" EXIT PARAGRAPH END-IF</pre>	<pre>SE SALDO É NEGATIVO MOSTRE A MENSAGEM "SALDO NEGATIVO NÃO PERMITIDO !!" SAIA DO PARAGRAFO SE-FIM</pre>
<pre>IF CODIGO > 10 AND QTDE < 1000 DISPLAY MESSAGE BOX "QTDE INVALIDA PARA ESSE PRODUTO !" EXIT PARAGRAPH END-IF</pre>	<pre>SE CODIGO > 10 E QTDE < 1000 MOSTRE A MENSAGEM "QTDE INVALIDA PARA ESSE PRODUTO !" SAIA DO PARAGRAFO SE-FIM</pre>

Importante: Lembre-se que se a condição definida no IF não for satisfeita, as instruções subordinadas a esse IF não serão realizadas. Utilizamos o ponto (.), o ELSE ou o END-IF para finalizar um IF.

EVALUATE, WHEN, OTHER, END-EVALUATE

<pre> IDENTIFICATION DIVISION. PROGRAM-ID. EVAL01. WORKING-STORAGE SECTION. 77 OPCA0 PIC 9(001). PROCEDURE DIVISION. PERFORM UNTIL 1 <> 1 DISPLAY WINDOW ERASE DISPLAY "OPCA0: " ACCEPT OPCA0 CONVERT EVALUATE OPCA0 WHEN 1 CALL "PRG01" CANCEL "PRG01" WHEN 2 CALL "PRG02" CANCEL "PRG02" WHEN 3 CALL "PRG03" CANCEL "PRG03" WHEN OTHER DISPLAY MESSAGE BOX "OPCA0 INVALIDA !!" END-EVALUATE END-PERFORM GOBACK. </pre>	<pre> EXECUTE ATÉ QUE 1 SEJA DIFERENTE DE 1 LIMPA A TELA MOSTRE "OPCA0: " RECEBA OPCA0 AVALIE OPCA0 QUANDO FOR 1 CHAME "PRG01" CANCELE "PRG01" QUANDO FOR 2 CHAME "PRG02" CANCELE "PRG02" QUANDO FOR 3 CHAME "PRG03" CANCELE "PRG03" QUANDO OUTRA QUALQUER MOSTRE A MENSAGEM "OPCA0 INVALIDA !!" AVALIE-FIM EXECUTE-FIM ENCERRA </pre>
<pre> IDENTIFICATION DIVISION. PROGRAM-ID. EVAL02. WORKING-STORAGE SECTION. 77 IDADE PIC 9(002). 77 SEXO PIC X(001). PROCEDURE DIVISION. PERFORM UNTIL 1 <> 1 DISPLAY WINDOW ERASE DISPLAY "IDADE: " ACCEPT IDADE CONVERT DISPLAY "SEXO: " ACCEPT SEXO UPPER EVALUATE IDADE ALSO SEXO WHEN < 18 ALSO "F" DISPLAY "MULHER MENOR" WHEN => 18 ALSO "F" DISPLAY "MULHER MAIOR" WHEN < 18 ALSO "M" DISPLAY "HOMEM MENOR" WHEN => 18 ALSO "H" DISPLAY "HOMEM MAIOR" WHEN OTHER DISPLAY "ERRO INFORME NOVAMENTE" END-EVALUATE ACCEPT OMITTED END-PERFORM GOBACK. </pre>	<pre> EXECUTE ATÉ QUE 1 SEJA DIFERENTE DE 1 LIMPA A TELA MOSTRE "IDADE: " RECEBA IDADE MOSTRE "SEXO: " RECEBA SEXO COM CAIXA-ALTA AVALIE IDADE TAMBÉM SEXO QUANDO (IDADE) < 18 E TAMBÉM (SEXO) = "F" MOSTRE "MULHER MENOR" QUANDO (IDADE) => 18 E TAMBÉM (SEXO) = "F" MOSTRE "MULHER MAIOR" QUANDO (IDADE) < 18 E TAMBÉM (SEXO) = "M" MOSTRE "HOMEM MENOR" QUANDO (IDADE) => 18 E TAMBÉM (SEXO) = "M" MOSTRE "HOMEM MAIOR" QUANDO OUTRO MOSTRE "ERRO INFORME NOVAMENTE" AVALIE-FIM AGUARDE UMA TECLA EXECUTE-FIM ENCERRA. </pre>

CONTINUE, NEXT SENTENCE

As sentenças CONTINUE e NEXT SENTENCE, são bastante úteis em casos onde pode ser obrigada a colocação de uma instrução. Essas instruções eram muito comuns em programas COBOL antes do padrão ANSI 85 que trouxe os delimitadores de escopo como o END-IF. Alguns programadores lançam mão desses recursos com o objetivo de tornar mais fácil o entendimento de testes com IF.

```
IF SALARIO > 1000
  NEXT SENTENCE
ELSE
  DISPLAY "SALARIO = OU < QUE 1000 !".
```

```
IF SALARIO NOT GREATER 1000
  DISPLAY "SALÁRIO = OU < QUE 1000 !"
END-IF
```

CÁLCULOS

SENTENÇAS:

ROUNDED, ON SIZE ERROR

COMANDOS:

COMPUTE (=, +, -, *, /, **)

ADD, SUBTRACT, MULTIPLY, DIVIDE

ROUNDED

O arredondamento de valores é bastante comum quando um processo envolve cálculos. No COBOL, podemos definir se queremos ou não o arredondamento através do ROUNDED. Caso ele esteja presente na instrução de cálculo ou na fórmula, o arredondamento será automático, e respeitará os limites definidos nas variáveis envolvidas.

Ex:

```
77  Valor          pic 9(003)✓99.
```

O cálculo encontrou o valor 23,458 (3 decimais) e o conteúdo de VALOR será 23,46 se utilizarmos ROUNDED. Se omitirmos o arredondamento, o conteúdo de VALOR será 23,45.

Calculado	PICTURE	COM ROUNDED	SEM ROUNDED
123,7865	9(003)✓99	123,79	123,78
4,654	9(002)✓99	4,64	4,65
7478,5896	9(006)✓9	7478,6	7478,5
4,8	9(001)	5	4
55,55	9(002)✓9	55,6	55,5

ON SIZE ERROR

A cláusula ON SIZE ERROR permite ao programador controlar situações de erro envolvendo cálculos (e outras também), como tentativa de divisão por zero e estouro da variável de resultado, que ocorre quando a variável de destino não suporta o valor encontrado.

EX:

<pre>77 A PIC 9(002) VALUE 6. 77 B PIC 9(002) VALUE 0. 77 C PIC 9(002). PROCEDURE DIVISION. DIVIDE A BY B GIVING C ON SIZE ERROR DISPLAY MESSAGE BOX "ERRO NA DIVISÃO" END-DIVIDE</pre>	<pre>DIVIDA A POR B COLOCANDO O RESULTADO EM C SE DER ERRO (DIVISÃO POR 0) MOSTRE UMA CAIXA DE MENSAGEM "ERRO NA DIVISÃO" DIVIDA-FIM.</pre>
<pre>77 A PIC 9(002) VALUE 6. 77 B PIC 9(002) VALUE 25. 77 C PIC 9(002). PROCEDURE DIVISION. COMPUTE C = A + B ON SIZE ERROR DISPLAY MESSAGE BOX "ESTOURO DE VARIÁVEL !!" NOT ON SIZE ERROR DISPLAY MESSAGE BOX "CALCULO OK " END-COMPUTE</pre>	<pre>CALCULE C = A + B EM CASO DE ERRO (ESTOURO) MOSTRE A CAIXA DE MENSAGEM "ESTOURO DE VARIÁVEL" EM CASO DE <u>NÃO</u> HAVER ERRO MOSTRE A CAIXA DE MENSAGEM "CALCULO OK" CALCULE-FIM</pre>

COMPUTE

O COMPUTE é utilizado para cálculos formados por expressões matemáticas de diferentes complexidades, podendo-se utilizar parênteses para priorizar cálculos e organizar a expressão.

SINTAXE:

```
COMPUTE <VARIABEL> [ROUNDED] = <VARIABEL, CONSTANTE>
    [ +, -, *, /, ** ] <VARIABEL, CONSTANTE>
    ON SIZE ERROR
        [INSTRUCAO CONDICIONAL]
    NOT ON SIZE ERROR
        [INSTRUCAO CONDICIONAL]
END-COMPUTE
```

Ex:

<pre>WORKING-STORAGE SECTION. 77 KM PIC 9(008)V99. 77 DIESEL PIC 9(008)V99. 77 MEDIA PIC ZZZ,ZZ9.999. PROCEDURE DIVISION. COMPUTE MEDIA ROUNDED = KM / DIESEL.</pre>	<pre>CALCULE MEDIA (ARREDONDANDO) = KM / DIESEL</pre>
<pre>WORKING-STORAGE SECTION. 77 ELEMENTS PIC 9(004) VALUE 60. 77 GROUPOFELEMENTS PIC 9(004) VALUE 6. 77 TOTALCOMBINATIONS PIC ZZ,ZZZ,ZZZ,ZZ9. PROCEDURE DIVISION. COMPUTE TOTALCOMBINATIONS = FUNCTION FACTORIAL(ELEMENTS) / FUNCTION FACTORIAL(GROUPOFELEMENTS) * FUNCTION FACTORIAL (ELEMENTS - GROUPOFELEMENTS). DISPLAY MESSAGE BOX "CHANCE DE ACERTAR NA MEGASENA É DE 1 EM: " TOTALCOMBINATIONS. GOBACK.</pre>	<pre>CALCULE TOTALCOMBINATIONS = FATORIAL DE ELEMENTS / FATORIAL DE GROUPOFELEMENTS * FATORIAL DE (ELEMENTS - GROUPOFELEMENTS) MOSTRE A MENSAGEM "CHANCE DE ACERTAR NA MEGASENA É DE 1 EM: " TOTALCOMBINATIONS.</pre>
<pre>WORKING-STORAGE SECTION. 77 A PIC 9(002) VALUE 0. 77 B PIC 9(002) VALUE 0. 77 C PIC Z9. PROCEDURE DIVISION. DISPLAY WINDOW ERASE DISPLAY "INFORME A: " ACCEPT A CONVERT DISPLAY "INFORME B: " ACCEPT B CONVERT COMPUTE C = A * B ON SIZE ERROR DISPLAY MESSAGE BOX "IMPOSSIVEL CALCULAR, VARIÁVEL PEQUENA !!" GOBACK END-COMPUTE DISPLAY " A * B = " C ACCEPT OMITTED GOBACK.</pre>	<pre>LIMPE A TELA MOSTRE "INFORME A: " RECEBA A (NUMERICO) MOSTRE "INFORME B: " RECEBA B (NUMERICO) CALCULE C = A * B EM CASO DE ERRO MOSTRE A MENSAGEM "IMPOSSIVEL CALCULAR, VARIÁVEL PEQUENA !!" ENCERRE CALCULE-FIM MOSTRE " A * B = " C ESPERE UMA TECLA ENCERRE</pre>

ADD

Utilizado para adicionar valores.

SINTAXE:

ADD <VARIÁVEL> TO <VARIÁVEL, CONSTANTE>
GIVING <VARIÁVEL> [ROUNDED]

ADD 1, 2, 3 TO <VARIÁVEL>

Ex:

WORKING-STORAGE SECTION.

77 A PIC 9(002) VALUE 0.

77 B PIC 9(002) VALUE 0.

77 C PIC Z9.

PROCEDURE DIVISION.

DISPLAY WINDOW ERASE

DISPLAY "INFORME A: "

ACCEPT A CONVERT

DISPLAY "INFORME B: "

ACCEPT B CONVERT

ADD A TO B GIVING C

ON SIZE ERROR

DISPLAY MESSAGE BOX

"IMPOSSIVEL CALCULAR, VARIÁVEL PEQUENA !"

GOBACK

END-ADD

DISPLAY " A + B = " C

ACCEPT OMITTED

GOBACK.

LIMPA A TELA

MOSTRE "INFORME A: "

RECEBA A

MOSTRE "INFORME B: "

RECEBA B

ADICIONE A PARA B OBTENDO C

EM CASO DE ERRO

MOSTRE A MENSAGEM

"IMPOSSIVEL CALCULAR, VARIÁVEL PEQUENA !"

ENCERRA

ADICIONE-FIM

MOSTRE "A + B = " C

AGUARDA UMA TECLA

ENCERRA.

SUBTRACT

Utilizado para subtração de valores.

SINTAXE:

```
SUBTRACT <VARIABEL> FROM <VARIABEL, CONSTANTE>  
      GIVING <VARIABEL> [ROUNDED]  
SUBTRACT 1, 2, 3 FROM <VARIABEL>
```

Ex:

<pre>WORKING-STORAGE SECTION. 77 A PIC 9(002) VALUE 0. 77 B PIC 9(002) VALUE 0. 77 C PIC --Z9. PROCEDURE DIVISION. DISPLAY WINDOW ERASE DISPLAY "INFORME A: " ACCEPT A CONVERT DISPLAY "INFORME B: " ACCEPT B CONVERT SUBTRACT A FROM B GIVING C ON SIZE ERROR DISPLAY MESSAGE BOX "IMPOSSIVEL CALCULAR, VARIÁVEL PEQUENA" GOBACK END-SUBTRACT DISPLAY " A - B = " C ACCEPT OMITTED GOBACK.</pre>	
--	--

MULTIPLY

Utilizado para multiplicação de valores.

SINTAXE:

MULTIPLY <VARIÁVEL> [ROUNDED] BY <VARIÁVEL, CONSTANTE> GIVING <VARIÁVEL>

Ex:

<pre>WORKING-STORAGE SECTION. 77 A PIC 9(002) VALUE 0. 77 B PIC 9(002) VALUE 0. 77 C PIC Z,ZZ9. PROCEDURE DIVISION. DISPLAY WINDOW ERASE DISPLAY "INFORME A: " ACCEPT A CONVERT DISPLAY "INFORME B: " ACCEPT B CONVERT MULTIPLY A BY B GIVING C ON SIZE ERROR DISPLAY MESSAGE BOX "IMPOSSIVEL CALCULAR, VARIÁVEL PEQUENA !" GOBACK END-MULTIPLY DISPLAY " A * B = " C ACCEPT OMITTED GOBACK.</pre>	
<pre>WORKING-STORAGE SECTION. 77 A PIC S9(002) VALUE -7. PROCEDURE DIVISION. IF A IS NEGATIVE MULTIPLY -1 BY A END-IF GOBACK.</pre>	

DIVIDE

Utilizado para divisão de valores.

SINTAXE:

DIVIDE <VARIABEL> BY <VARIABEL, CONSTANTE>
GIVING <VARIABEL> [REMAINDER <VARIABEL>] [ROUNDED]

DIVIDE <VARIABEL/CONSTANTE> INTO <VARIABEL/CONSTANTE>.

Ex:

```
WORKING-STORAGE SECTION.  
77 A    PIC 9(002)  VALUE 0.  
77 B    PIC 9(002)  VALUE 0.  
PROCEDURE DIVISION.  
    DISPLAY WINDOW ERASE  
    DISPLAY "INFORME A: "  
    ACCEPT A CONVERT  
    DISPLAY "INFORME B: "  
    ACCEPT B CONVERT  
    DIVIDE A INTO B  
    DISPLAY " B / A  = " B  
        ACCEPT OMITTED  
        GOBACK.
```

```
WORKING-STORAGE SECTION.  
77 A          PIC 9(002)  VALUE 0.  
77 B          PIC 9(002)  VALUE 0.  
77 RESULTADO  PIC Z9.  
77 RESTO      PIC 9(002).  
PROCEDURE DIVISION.  
    DISPLAY WINDOW ERASE  
    DISPLAY "INFORME A: "  
    ACCEPT A CONVERT  
    DISPLAY "INFORME B: "  
    ACCEPT B CONVERT  
    DIVIDE A BY B  GIVING RESULTADO  
        REMAINDER RESTO  
    DISPLAY " A / B  = " RESULTADO  "  
        RESTO: " RESTO  
    ACCEPT OMITTED  
    GOBACK.
```

FUNÇÕES INTRÍNSECAS

Muitas linguagens de programação possuem funções intrínsecas ou internas. Em COBOL, elas foram adicionadas em 1989, como extensão do padrão ANSI 85. São funções financeiras, de calendário, matemáticas, trigonométricas e de manipulação de strings.

Abaixo a listagem das funções suportadas pelo Acucobol (Extend 8.1):

Função	Argumentos	Tipo	Valor retornado
ABSOLUTE-VALUE (ou ABS)	N1	Numérico	Valor absoluto
ACOS	N1	Numérico	Retorna um valor numérico em radianos equivalente ao arco do cosseno do argumento da função.
ANNUITY	N1, N2	Numérico	Retorna um valor numérico equivalente a taxa de uma anuidade paga ao final de um período (N1) para o número de períodos especificado por (N2), para um investimento inicial
ASIN	N1	Numérico	Retorna um valor numérico em radianos equivalente ao arco do seno do argumento da função.
ATAN	N1	Numérico	Retorna um valor numérico em radianos que representa o valor do arco da tangente do argumento fornecido.
CHAR	I1	Alfanumérico	Função alfanumérica que retorna um valor equivalente ao caractere passado para a função através de I1
COS	N1	Numérico	Retorna um valor numérico que corresponde ao cosseno de um ângulo ou arco, expresso em radianos, especificado por N1.
CURRENT-DATE		Alfanumérico	Retorna uma string alfanumérica de 21 caracteres de comprimento que representa a Data do calendário, hora do dia e fatores diferenciais de tempo proporcionados pelo sistema no qual a função é executada.
DATE-OF-INTEGER	I1	Inteiro	Converte uma data inteira para o formato AAAAMMDD. O argumento dessa função deve ser um valor inteiro que indique o número de dias que sucedem 31 de dezembro de 1600 no calendário Gregoriano.
DAY-OF-INTEGER	I1	Inteiro	Converte o inteiro de uma data para o formato Juliano AAAADDD
FACTORIAL	I1	Inteiro	Retorna o valor correspondente ao fatorial de I1
INTEGER	N1	Inteiro	Devolve o maior valor inteiro que é menor ou igual ao argumento utilizado pela função.
INTEGER-OF-DATE	I1	Inteiro	Converte uma data no formato AAAAMMDD para um inteiro (número de dias que sucedem 31 de dezembro de 1600)
INTEGER-OF-DAY	I1	Inteiro	Converte uma data no formato AAAADDD para um inteiro
INTEGER-PART	N1	Inteiro	Retorna a parte inteira de um argumento
LENGTH	A1 ou N1 ou X1	Inteiro	Retorna um valor inteiro igual ao comprimento do argumento em posições de caracteres
LOG	N1	Numérico	Retorna o valor do logaritmo do argumento, que deve ser maior que 0.
LOG10	N1	Numérico	Retorna o logaritmo da base 10 do argumento, que deve ser maior que 0.
LOWER-CASE	A1 ou X1	Alfanumérico	Retorna uma string de caracteres de mesmo comprimento que o argumento da função, com a diferença de que os caracteres foram convertidos para minúsculos.
MAX	A1... ou I1... ou	Depende dos	Função que retorna o argumento que contém o valor máximo. O tipo dessa função depende

	N1... ou X1...	argumentos	do tipo de argumento.
MEAN	N1...	Numérico	Retorna a média aritmética dos seus argumentos
MEDIAN	N1...	Numérico	Retorna o conteúdo do argumento cujo valor é o valor médio da lista de argumentos
MIDRANGE	N1...	Numérico	Retorna a média aritmética entre o maior e o menor argumento.
MIN	A1... ou I1... ou N1... ou X1...	Depende dos argumentos	Retorna o o argumento que contém o menor valor.
MOD	I1, I2	Inteiro	Retorna o módulo de I1 em relação a I2
NUMVAL	X1	Numérico	Retorna o valor numérico de X1
NUMVAL-C	X1, X2	Numérico	Faz o mesmo que NUMVAL, porém qualquer ocorrência de sinal ou vírgula precedendo o ponto decimal especificada pelo X2 será ignorada.
ORD	A1 ou X1	Inteiro	Função que retorna um valor inteiro que representa a posição ordinal do argumento na sequência do programa.
ORD-MAX	A1... ou N1... ou X1...	Inteiro	Retorna um valor representando o número ordinal do argumento que contém o valor máximo.
ORD-MIN	A1... ou N1... ou X1	Inteiro	Retorna um valor representando o número ordinal do argumento que contém o valor mínimo.
PRESENT-VALUE	N1, N2...	Numérico	Retorna o valor presente de uma série de valores futuros especificados por N1, com uma taxa especificada por N2.
RANDOM	I1	Numérico	Retorna um valor pseudo-aleatório.
RANGE	I1... ou N1...	Depende dos argumentos	Retorna a diferença entre o maior e o menor argumento.
REM	N1, N2	Numérico	Retorna o resto da divisão N1 / N2.
REVERSE	A1 ou X1	Alfabético	Inverte a ordem original dos caracteres.
SIN	N1	Numérico	Retorna o valor em radianos do seno de um ângulo ou arco, especificado por N1.
SQRT	N1	Numérico	Retorna a raiz quadrada de N1.
STANDARD-DEVIATION	N1...	N	Retorna o desvio padrão de uma série de argumentos.
SUM	I1... ou N1...	Depende dos argumentos	Retorna a soma de todos os argumentos.
TAN	N1	Numérico	Retorna o valor em radianos da tangente do ângulo ou arco especificado pelo argumento.
UPPER-CASE	A1 ou X1	Alfabético	Retorna uma string de caracteres de mesmo comprimento que o argumento da função, com diferença de que os caracteres em minúsculas foram convertidos para maiúsculos.
VARIANCE	N1...	Numérico	Retorna a variação dos argumentos
WHEN-COMPILED		Alfanumérico	Retorna a data e hora em que o programa foi compilado

Argumentos: I – Inteiro, A – Alfabético, N – Numérico, X - Alfanumérico

Exemplo do uso de funções:

```
COMPUTE RAIZ = FUNCTION SQRT (4)
COMPUTE MAIOR-VALOR = FUNCTION MAX (VALOR-1, VALOR-2, VALOR-3, VALOR-4)
```

Veremos abaixo exemplos de algumas poucas funções que são mais utilizadas no dia a dia:

INTEGER-OF-DATE

Retorna o valor inteiro de uma data (YYYYMMDD), esse valor inteiro é equivalente ao número de dias decorridos desde 31-12-1601. Muito útil no cálculo de diferença de dias entre datas e também quando é necessário somar ou subtrair dias de uma data.

EX: Calcular diferença de dias entre duas datas

<pre>WORKING-STORAGE SECTION. 77 DATA-INICIAL PIC 9(008). 77 DATA-FINAL PIC 9(008). 77 INTEIRO-1 PIC 9(008). 77 INTEIRO-2 PIC 9(008). 77 DIAS PIC 9(005). PROCEDURE DIVISION. DISPLAY "DATA INICIAL YYYYMMDD: " ACCEPT DATA-INICIAL CONVERT DISPLAY "DATA FINAL YYYYMMDD: " ACCEPT DATA-FINAL CONVERT MOVE FUNCTION INTEGER-OF-DATE (DATA-INICIAL) TO INTEIRO-1 MOVE FUNCTION INTEGER-OF-DATE (DATA-FINAL) TO INTEIRO-2 COMPUTE DIAS = INTEIRO-2 - INTEIRO-1 DISPLAY "DIAS: " DIAS ACCEPT OMITTED. GOBACK.</pre>	<pre>MOSTRE "DATA INICIAL YYYYMMDD: " RECEBA DATA-INICIAL MOSTRE "DATA FINAL YYYYMMDD: " RECEBA DATA-FINAL MOVA A FUNÇÃO INTEGER-OF-DATE (DA DATA-INICIAL) PARA INTEIRO-1 MOVA A FUNÇÃO INTEGER-OF-DATE (DA DATA-FINAL) PARA INTEIRO-2 CALCULE DIAS = INTEIRO-2 - INTEIRO-1 MOSTRE "DIAS: " DIAS</pre>
--	--

DATE-OF-INTEGER

Retorna a data no formato (YYYYMMDD) de uma data inteira, fazendo o inverso da função INTEGER-OF-DATE. Muitas vezes, essas duas funções são usadas em conjunto para o trabalho com datas.

EX: Somar um número de dias a uma data.

<pre>WORKING-STORAGE SECTION. 77 DATA-INICIAL PIC 9(008). 77 DATA-FINAL PIC 9(008). 77 DIAS PIC 9(003). PROCEDURE DIVISION. DISPLAY "DATA INICIAL YYYYMMDD: " ACCEPT DATA-INICIAL CONVERT DISPLAY "DIAS A SOMAR NA DATA: " ACCEPT DIAS CONVERT COMPUTE DATA-FINAL = FUNCTION DATE-OF-INTEGER (FUNCTION INTEGER-OF-DATE (DATA-INICIAL + DIAS)) DISPLAY "DATA FINAL YYYYMMDD: " DISPLAY DATA-FINAL CONVERT ACCEPT OMITTED.</pre>	<pre>MOSTRE "DATA INICIAL YYYYMMDD: " RECEBA DATA-INICIAL MOSTRE "DIAS A SOMAR NA DATA: " RECEBA DIAS CALCULE DATA-FINAL = FUNÇÃO DATE-OF-INTEGER (DA FUNÇÃO INTEGER-OF-DATE (DA DATA-INICIAL + DIAS)) MOSTRE "DATA FINAL YYYYMMDD: " MOSTRE DATA-FINAL</pre>
--	---

REM

Esta função devolve o resto da divisão do argumento 1 pelo argumento 2.

Ex: Saber o dia da semana

```
WORKING-STORAGE SECTION.
01 WDATA          PIC 9(008).
01 DIA-SEMANA     PIC 9(001).
01 DIAS-DA-SEMANA.
   05 PIC X(010) VALUE "DOMINGO".
   05 PIC X(010) VALUE "SEGUNDA".
   05 PIC X(010) VALUE "TERCA".
   05 PIC X(010) VALUE "QUARTA".
   05 PIC X(010) VALUE "QUINTA".
   05 PIC X(010) VALUE "SEXTA".
   05 PIC X(010) VALUE "SABADO".
01 NOME-DO-DIA REDEFINES DIAS-DA-SEMANA
               PIC X(010) OCCURS 7.

PROCEDURE DIVISION.
PROCESSO.
  DISPLAY WINDOW ERASE
  DISPLAY "INFORME UMA DATA: YYYYMMDD "
  ACCEPT WDATA CONVERT
  COMPUTE DIA-SEMANA =
    FUNCTION REM
      (FUNCTION INTEGER-OF-DATE(WDATA), 7)
  ADD 1 TO DIA-SEMANA
  DISPLAY "DIA DA SEMANA: "
    NOME-DO-DIA(DIA-SEMANA)
  ACCEPT OMITTED.
  GOBACK.
```

CURRENT-DATE

Esta função devolve a data e hora atuais do sistema.

Ex:

```
WORKING-STORAGE SECTION.
01 DATA-ATUAL.
   05 ANO          PIC 9(004).
   05 MES          PIC 9(002).
   05 DIA          PIC 9(002).
   05 HH          PIC 9(002).
   05 MM          PIC 9(002).
   05 SS          PIC 9(002).
   05 C-SS        PIC 9(002).
   05 FILLER      PIC X(005).

PROCEDURE DIVISION.
PROCESSO.
  DISPLAY WINDOW ERASE
  MOVE FUNCTION CURRENT-DATE(1:16) TO
    DATA-ATUAL
  DISPLAY "DATA: " DIA "/" MES "/" ANO
  DISPLAY "HORA: " HH ":" MM ":" SS ":"
    C-SS
  ACCEPT OMITTED.
  GOBACK.
```

WHEN-COMPILED

Esta função é similar a CURRENT-DATE, mas ao invés de devolver a data e hora do sistema, devolve a data e hora em que o programa foi compilado, isso pode ser bastante útil, pois essa informação pode ser usada para controlar a versão do seu programa.

Algumas funções executam tarefas que podem perfeitamente ser feitas também com comandos simples em COBOL, mas na maioria das vezes ajudam muito, pois diminuem razoavelmente o trabalho de programação em situações que exigiriam maior codificação.

Três formas de fazer uma somatória

```
* USANDO COMPUTE

WORKING-STORAGE SECTION.
01 VALOR-1          PIC 9(007).
01 VALOR-2          PIC 9(007).
01 VALOR-3          PIC 9(007).
01 VALOR-TOTAL      PIC 9(007).
PROCEDURE DIVISION.
...
    COMPUTE VALOR-TOTAL =
        VALOR-1 + VALOR-2 + VALOR-3
```

```
* USANDO ADD

WORKING-STORAGE SECTION.
01 VALOR-1          PIC 9(007).
01 VALOR-2          PIC 9(007).
01 VALOR-3          PIC 9(007).
01 VALOR-TOTAL      PIC 9(007).
PROCEDURE DIVISION.
...
    ADD VALOR-1, VALOR-2, VALOR-3 TO
        VALOR-TOTAL.
```

```
* USANDO COMPUTE COM FUNÇÃO

WORKING-STORAGE SECTION.
01 VALOR-1          PIC 9(007).
01 VALOR-2          PIC 9(007).
01 VALOR-3          PIC 9(007).
01 VALOR-TOTAL      PIC 9(007).
PROCEDURE DIVISION.
...
    COMPUTE VALOR-TOTAL = FUNCTION SUM
        (VALOR-1, VALOR-2, VALOR-3)
```

Duas formas de calcular a média

```
* USANDO COMPUTE DIRETO

WORKING-STORAGE SECTION.
01 VALOR-1          PIC 9(007).
01 VALOR-2          PIC 9(007).
01 VALOR-3          PIC 9(007).
01 MEDIA            PIC 9(007).
PROCEDURE DIVISION.
...
    COMPUTE MEDIA =
        VALOR-1 + VALOR-2 + VALOR-3 / 3.
```

```
* USANDO COMPUTE COM FUNÇÃO

WORKING-STORAGE SECTION.
01 VALOR-1          PIC 9(007).
01 VALOR-2          PIC 9(007).
01 VALOR-3          PIC 9(007).
01 MEDIA            PIC 9(007).
PROCEDURE DIVISION.
...
    COMPUTE MEDIA = FUNCTION MEAN
        (VALOR-1, VALOR-2, VALOR-3)
```

Observação: diversas outras funções foram acrescentadas ao padrão 2002 da linguagem, sem dúvida, as mais importantes dizem respeito a validação de datas e também ao trabalho com horas. Infelizmente essas funções ainda não estão disponíveis em muitos compiladores.

CALL

O CALL é utilizado para chamar outros programas, rotinas internas do COBOL e programas externos como DLLs.

Quando se trata de chamada a outro programa COBOL, se faz necessário respeitar o tamanho e tipo das variáveis envolvidas nessa comunicação (LINKAGE SECTION), caso contrário, os resultados serão confusos, podendo até mesmo ocorrerem erros críticos de execução. No caso de chamadas a funções internas do COBOL deve-se respeitar os parâmetros contidos na documentação (manual).

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. TESTE.  
WORKING-STORAGE SECTION.  
77 NOME PIC X(030).  
PROCEDURE DIVISION.  
    DISPLAY MESSAGE BOX "DIGITE O NOME: "  
    ACCEPT NOME  
    CALL "REVERSO" USING NOME  
    ON OVERFLOW  
        DISPLAY MESSAGE BOX  
            "ERRO AO CARREGAR SUB-ROTINA !!"  
        GOBACK  
    END-CALL  
    DISPLAY "NOME REVERSO: " NOME  
    ACCEPT OMITTED.  
    GOBACK.
```

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. REVERSO.  
LINKAGE SECTION.  
77 NOME-RECEBIDO PIC X(030).  
PROCEDURE DIVISION USING NOME-RECEBIDO.  
    MOVE FUNCTION REVERSE(NOME-RECEBIDO) TO  
        NOME-RECEBIDO  
    GOBACK.
```

Pode-se utilizar ainda algumas sentenças no CALL

BY REFERENCE: Faz com que o programa chamado utilize a mesma área de memória usada nos dados passados via LINKAGE. Tudo o que for mudado no programa chamado refletirá no chamador. Esse é o padrão da linguagem quando nada for especificado.

BY CONTENT: Nesse caso, o programa chamado cria uma cópia dos dados em memória, dessa forma, os dados são preservados sem alteração para o programa chamado.

BY VALUE: Passa apenas os valores das variáveis e não seus endereços. É utilizada para chamadas a rotinas de outras linguagens ou DLLs.

ON OVERFLOW

Ao chamarmos uma sub-rotina ou sub-programa, podem ocorrer erros, que devem ser tratados pelo programador. O erro mais comum é a aplicação não encontrar o programa chamado (ou porque não existe mesmo, ou porque o nome foi digitado errado, ou ainda porque está num formato que não pode ser chamado), também podem ocorrer erros de alocação de memória (não há espaço para carregar a sub-rotina), o que é pouco comum hoje em dia.

Mas o importante é sempre tratar essas possibilidades de erros, para isso contamos com o ON OVERFLOW.

Ex:

```
CALL "LOADDATA" USING AREA-DE-TRANSFERENCIA  
ON OVERFLOW  
    DISPLAY MESSAGE BOX  
    "ERRO AO CARREGAR LOADDATA !!"  
GO TO ENCERRA-ANORMAL
```

```
CHAME "LOADDATA" USANDO AREA-DE-TRANSFERENCIA  
SE HOUVER ERRO  
    MOSTRE A MENSAGEM  
    "ERRO AO CARREGAR LOADDATA !!"  
VÁ PARA ENCERRA-ANORMAL
```

NOT ON OVERFLOW CANCEL "LOADDATA" END-CALL	SE NÃO HOVER ERRO CANCELE "LOADDATA" (REMOVE DA MEMÓRIA) CHAME-FIM
--	--

EXIT

Como se pode imaginar, esse comando é utilizado para sair, mas sair do quê ? De onde? Tudo depende do contexto em que se encontra, pois pode ser utilizado para sair de um parágrafo (EXIT PARAGRAPH), de uma section (EXIT SECTION), de um loop (EXIT PERFORM), de um programa (EXIT PROGRAM) ou ainda de um ciclo de loop (EXIT PERFORM CYCLE).

Ex:

IF CODIGO NOT NUMERIC EXIT PARAGRAPH END-IF	
IF ERROS EXIT SECTION END-IF	

MOVE 0 TO CONTADOR PERFORM UNTIL 1 <> 1 ADD 1 TO CONTADOR IF CONTADOR > 10 EXIT PERFORM END-IF IF CONTADOR = 2 OR 4 OR 7 EXIT PERFORM CYCLE END-IF DISPLAY "CONTADOR: " CONTADOR END-PERFORM	COLOQUE 0 NO CONTADOR EXECUTE ATÉ QUE 1 NÃO SEJA IGUAL A 1 ADICIONE 1 NO CONTADOR SE O CONTADOR FOR MAIOR QUE 10 SAIA DO EXECUTE SE-FIM SE O CONTADOR FOR 2 OU 4 OU 7 SAIA DO CICLO (VA PARA O COMEÇO) SE-FIM MOSTRE "CONTADOR: " CONTADOR EXECUTE-FIM
--	--

Observação: para encerrar um programa, é mais comum encontrarmos a instrução GOBACK (que faz o mesmo papel do EXIT PROGRAM e do STOP RUN), mas ela não faz parte do padrão, embora se encontre disponível na maioria dos compiladores. O EXIT PROGRAM é encontrado como finalização de programas chamados através de call.

PERFORM

Também é utilizado para desvio no fluxo do programa. Pode ser executado para parágrafos, intervalos de parágrafos e sections. Na programação estruturada é utilizado em estruturas de laço ou repetição, onde determinado bloco deve ser processado até que uma situação pré-definida ocorra. Sua utilização permite a programação sem o GO TO.

SINTAXE:

PERFORM [PARAGRAFO/SECTION] [THRU] [PARAGRAFO]

PERFORM [VARIABEL/CONSTANTE/INTEIRO] TIMES

PERFORM UNTIL [CONDIÇÃO]

...

END-PERFORM

PERFORM [WITH TEST AFTER/BEFORE] VARYING [VARIABEL] FROM

[VARIABEL/CONSTANTE/INTEIRO] BY

[VARIABEL/CONSTANTE/INTEIRO] UNTIL

[CONDIÇÃO]

END-PERFORM

EX:

<pre>PROGRAM-ID. PERFORM1. WORKING-STORAGE SECTION. 77 NOME PIC X(015). PROCEDURE DIVISION. PERFORM 5 TIMES DISPLAY "INFORME O NOME: " ACCEPT NOME DISPLAY NOME REVERSE END-PERFORM GOBACK.</pre>	<pre>EXECUTE 5 VEZES MOSTRE "INFORME O NOME:" RECEBA O NOME MOSTRE O NOME EM VIDEO REVERSO EXECUTE-FIM</pre>
--	--

<pre>PROGRAM-ID. PERFORM2. WORKING-STORAGE SECTION. 77 NOME PIC X(015). PROCEDURE DIVISION. PARAGRAFO1. PERFORM PARAGRAFO2 5 TIMES. GOBACK. PARAGRAFO2. DISPLAY "INFORME O NOME: " ACCEPT NOME DISPLAY NOME REVERSE.</pre>	<pre>EXECUTE O PARÁGRAFO2 5 VEZES</pre>
<pre>PROGRAM-ID. PERFORM3. WORKING-STORAGE SECTION. 77 NOME PIC X(015). PROCEDURE DIVISION. PARAGRAFO1. PERFORM UNTIL 1 <> 1 DISPLAY "INFORME O NOME: " ACCEPT NOME IF NOME = SPACES EXIT PERFORM</pre>	<pre>EXECUTE ATÉ QUE 1 SEJA DIFERENTE DE 1 MOSTRE "INFORME O NOME:" RECEBA NOME SE NOME FOR ESPAÇOS SAIA DO EXECUTE</pre>

END-IF DISPLAY NOME REVERSE END-PERFORM. GOBACK.	FIM-SE MOSTRE NOME COM VIDEO REVERSO FIM-EXECUTE.
---	---

PROGRAM-ID. PERFORM4. WORKING-STORAGE SECTION. 77 NOME PIC X(015) VALUE ".". PROCEDURE DIVISION. PARAGRAFO1. PERFORM WITH TEST BEFORE UNTIL NOME = SPACES DISPLAY "INFORME O NOME: " ACCEPT NOME DISPLAY NOME REVERSE END-PERFORM. GOBACK.	EXECUTE "TESTANDO DEPOIS" ATÉ QUE O NOME SEJA ESPAÇOS MOSTRE "INFORME NOME:" RECEBA NOME MOSTRE NOME COM VIDEO REVERSO EXECUTE-FIM
<u>O mesmo processo com GO TO</u> PROGRAM-ID. PERFORM4. WORKING-STORAGE SECTION. 77 NOME PIC X(015) VALUE ".". PROCEDURE DIVISION. PARAGRAFO1. DISPLAY "INFORME O NOME:" ACCEPT NOME DISPLAY NOME REVERSE IF NOME = SPACES GO TO FIM END-IF GO TO PARAGRAFO1. FIM. GOBACK.	PARAGRAFO1. MOSTRE "INFORME O NOME:" RECEBA NOME MOSTRE NOME COM VIDEO REVERSO SE NOME = ESPAÇOS VA PARA FIM SE-FIM VA PARA PARAGRAFO1. FIM.

PROGRAM-ID. PERFORM5. WORKING-STORAGE SECTION. 77 CONTADOR PIC 9(003). 77 VEZES PIC 9(003). PROCEDURE DIVISION. PARAGRAFO1. DISPLAY "CONTAR ATÉ QUANTO: " ACCEPT VEZES CONVERT PERFORM VARYING CONTADOR FROM 1 BY 1 UNTIL CONTADOR > VEZES DISPLAY CONTADOR REVERSE END-PERFORM. ACCEPT OMITTED GOBACK.	EXECUTE VARIANDO CONTADOR DE 1 EM 1 ATÉ O CONTADOR SER MAIOR QUE VEZES MOSTRE CONTADOR EXECUTE-FIM
Com GO TO PROGRAM-ID. PERFORM5. WORKING-STORAGE SECTION. 77 CONTADOR PIC 9(003). 77 VEZES PIC 9(003). PROCEDURE DIVISION. PARAGRAFO1. DISPLAY "CONTAR ATÉ QUANTO: " ACCEPT VEZES CONVERT . MOVE 0 TO VEZES. CONTAR. ADD 1 TO VEZES IF VEZES > CONTADOR GO TO FIM END-IF DISPLAY VEZES REVERSE	

GO TO CONTAR. FIM. GOBACK.	
----------------------------------	--

INSPECT

Essa declaração proporciona a habilidade de contar (TALLY), substituir (REPLACE), ou converter (CONVERT) caracteres simples ou grupos de caracteres em um item de dados. Em outras palavras, ela inspeciona um item qualquer à procura de um ou vários caracteres, para em tão trocá-lo ou convertê-lo em outro predeterminado.

SINTAXE:

```
INSPECT {identificador-1} REPLACING CHARACTERS BY
      {identificador-2 ou literal-1}
      {BEFORE or AFTER} [INITIAL {identificador-3 ou literal-2}]
      {ALL or LEADING ou FIRST} {identificador-4 ou literal-3}
BY {identificador-5 ou literal-4} {BEFORE ou AFTER} INITIAL
   {identificador-6 ou literal-5}
```

```
INSPECT {identificador-1} TALLYING {identificador-2}
      {BEFORE ou AFTER} [INITIAL {identificador-3 ou literal-2}]
      {ALL ou LEADING ou FIRST} {identificador-4 ou literal-3}
BY {identificador-5 ou literal-4} {BEFORE ou AFTER} INITIAL
   {identificador-6 ou literal-5}
```

77 TEXTO PIC X(080). PROCEDURE DIVISION. MOVE "Eu0vou0adicionar0o0UOL0aos0Favoritos" TO TEXTO. INSPECT TEXTO REPLACING ALL ZEROS BY SPACES. DISPLAY TEXTO. GOBACK.	INSPECIONE TEXTO SUBSTITUINDO TODOS OS 0 POR ESPAÇOS
---	---

MOVE "13,89;77,67;124,98;566,7;PRODUTO-1;" TO REGISTRO INSPECT REGISTRO REPLACING ALL "," BY ".".	INSPECIONE REGISTRO TROCANDO TODAS AS "," POR "."
--	--

```

PROGRAM-ID. SRT1.
WORKING-STORAGE SECTION.
77 DESCRICAO          PIC X(060).
77 CONTADOR           PIC 9(003).
PROCEDURE DIVISION.
INICIO.
    DISPLAY WINDOW ERASE
    MOVE "55 KILOS DE CAFÉ DO BRASIL" TO
        DESCRICAO
    DISPLAY DESCRICAO REVERSE
    MOVE 0 TO CONTADOR.
    INSPECT DESCRICAO TALLYING CONTADOR
        FOR ALL "BRASIL".
    DISPLAY "BRASIL = " CONTADOR
    MOVE 0 TO CONTADOR
    INSPECT DESCRICAO TALLYING CONTADOR
        FOR ALL "L".
    DISPLAY "L = " CONTADOR
    MOVE 0 TO CONTADOR
    INSPECT DESCRICAO TALLYING CONTADOR
        FOR ALL "L" BEFORE INITIAL "BRASIL".
    DISPLAY "L ANTES DO BRASIL = "
        CONTADOR
    MOVE 0 TO CONTADOR
    INSPECT DESCRICAO TALLYING CONTADOR
        FOR ALL " " BEFORE INITIAL "D".
    DISPLAY "ESPAÇOS ANTES DO D = "
        CONTADOR
    MOVE 0 TO CONTADOR
    INSPECT DESCRICAO TALLYING CONTADOR
        FOR ALL " "
    DISPLAY "TODOS OS ESPAÇOS EM BRANCO = "
        CONTADOR
    MOVE 0 TO CONTADOR
    INSPECT DESCRICAO TALLYING CONTADOR
        FOR TRAILING " "
    DISPLAY "ESPAÇOS EM BRANCO A DIREITA = "
        CONTADOR
    ACCEPT OMITTED.
    GOBACK.

```

INSPECIONE DESCRICAO CONTANDO CONTADOR POR
TODOS "BRASIL"

INSPECIONE DESCRICAO CONTANDO CONTADOR POR
TODOS OS "L"

INSPECIONE DESCRICAO CONTANDO CONTADOR POR
TODOS OS "L" ANTES DE "BRASIL"

INSPECIONE DESCRICAO CONTANDO CONTADOR POR
TODOS OS " " ANTES DO PRIMEIRO "D"

INSPECIONE DESCRICAO CONTANDO POR TODOS OS
ESPAÇOS EM BRANCO

INSPECIONE DESCRICAO CONTANDO CONTADOR POR
TODOS OS ESPAÇOS A DIREITA

STRING

O comando STRING permite a concatenação de múltiplos valores de item-de-dado enviados em um só receptor.

<pre>IDENTIFICATION DIVISION. PROGRAM-ID. STR001. WORKING-STORAGE SECTION. 01 TODAS-AS-CORES PIC X(100) VALUE SPACES. 77 AZUL PIC X(010) VALUE "AZUL". 77 VERDE PIC X(010) VALUE "VERDE". 77 ROXO PIC X(010) VALUE "ROXO". PROCEDURE DIVISION. DISPLAY WINDOW ERASE STRING AZUL DELIMITED BY " " "-" ROXO DELIMITED BY " " "-" VERDE DELIMITED BY " " INTO TODAS-AS-CORES. DISPLAY TODAS-AS-CORES ACCEPT OMITTED.</pre>	<pre>LIMPA A TELA JUNTE AZUL DELIMITADO POR " " "-" ROXO DELIMITADO POR " " "-" VERDE DELIMITED POR " " PARA TODAS-AS-CORES MOSTRE TODAS-AS-CORES.</pre>
---	--

<pre>IDENTIFICATION DIVISION. PROGRAM-ID. STR001A. DATE-WRITTEN. DEZ-2009. WORKING-STORAGE SECTION. 77 STRING-DA-DATA PIC X(128) VALUE SPACES. 77 DIA-DA-SEMANA PIC 9(001) VALUE 0. 01 DATA-HOJE PIC 9(008). 01 FILLER REDEFINES DATA-HOJE. 05 ANO-HOJE PIC 9(004). 05 MES-HOJE PIC 9(002). 05 DIA-HOJE PIC 9(002). 01 TABELA-MESES. 05 FILLER PIC X(009) VALUE "JANEIRO". 05 FILLER PIC X(009) VALUE "FEVEREIRO". 05 FILLER PIC X(009) VALUE "MARÇO". 05 FILLER PIC X(009) VALUE "ABRIL". 05 FILLER PIC X(009) VALUE "MAIO". 05 FILLER PIC X(009) VALUE "JUNHO". 05 FILLER PIC X(009) VALUE "JULHO". 05 FILLER PIC X(009) VALUE "AGOSTO". 05 FILLER PIC X(009) VALUE "SETEMBRO". 05 FILLER PIC X(009) VALUE "OUTUBRO". 05 FILLER PIC X(009) VALUE "NOVEMBRO". 05 FILLER PIC X(009) VALUE "DEZEMBRO". 01 TB-MES REDEFINES TABELA-MESES PIC X(009) OCCURS 12. 01 TABELA-DIAS. 05 FILLER PIC X(013) VALUE "SEGUNDA-FEIRA". 05 FILLER PIC X(013) VALUE "TERÇA-FEIRA". 05 FILLER PIC X(013) VALUE "QUARTA-FEIRA". 05 FILLER PIC X(013) VALUE "QUINTA-FEIRA". 05 FILLER PIC X(013) VALUE "SEXTA-FEIRA". 05 FILLER PIC X(013) VALUE "SÁBADO". 05 FILLER PIC X(013) VALUE "DOMINGO". 01 TB-DIAS REDEFINES TABELA-DIAS PIC X(013) OCCURS 7. PROCEDURE DIVISION. INICIO.</pre>	
--	--

<pre> DISPLAY WINDOW ERASE ACCEPT DIA-DA-SEMANA FROM DAY-OF-WEEK MOVE FUNCTION CURRENT-DATE(1:8) TO DATA-HOJE STRING "HOJE É " DELIMITED BY SIZE TB-DIAS(DIA-DA-SEMANA) DELIMITED BY " " ", DIA " DELIMITED BY SIZE DIA-HOJE DELIMITED BY SIZE " DE " DELIMITED BY SIZE TB-MES (MES-HOJE) DELIMITED BY " " " DE " DELIMITED BY SIZE ANO-HOJE DELIMITED BY SIZE INTO STRING-DA-DATA. DISPLAY STRING-DA-DATA AT 0301 REVERSE SIZE 80 ACCEPT OMITTED GOBACK. </pre>	<pre> LIMPA A TEMA PEGA O NUMERO DO DIA DA SEMANA DO SISTEMA PEGA A DATA ATUAL DO SISTEMA CONCATENA "HOJE É " DELIMITADO PELO TAMANHO DIA DA SEMANA DELIMITADO POR " " ", DIA " DELIMITADO PELO TAMANHO DIA DO MÊS DELIMITADO PELO TAMANHO " DE " DELIMITADO PELO TAMANHO NOME DO MÊS DELIMITADO POR " " " DE " DELIMITADO PELO TAMANHO ANO DELIMITADO PELO TAMANHO EM STRING-DA-DATA MOSTRA STRING-DA-DATA </pre>
--	---

O string também permite o uso de ponteiros (POINTER), que possibilitam ao programador controlar a posição em que deseja fazer a concatenação, isso é muito útil quando uma string depende de regras condicionais para ser montada. Quando POINTER não é informado, assume 1. Deve-se ter muita atenção quando definir a variável de ponteiro, para que suporte o tamanho da variável se string, senão os resultados estarão comprometidos.

<pre> IDENTIFICATION DIVISION. PROGRAM-ID. STR002A. AUTHOR. MARCIO A. SILVA. DATE-WRITTEN. DEZ-2009. WORKING-STORAGE SECTION. 77 COR PIC X(010). 77 CORES PIC X(050). 77 PONTEIRO PIC 9(003) VALUE 1. PROCEDURE DIVISION. INICIO. DISPLAY WINDOW ERASE DISPLAY "INFORME A COR: " AT 1010 PERFORM UNTIL 1 <> 1 ACCEPT COR AT 1025 REVERSE PROMPT UPPER IF COR = SPACES EXIT PERFORM END-IF STRING COR DELIMITED BY " " " " DELIMITED BY SIZE INTO CORES POINTER PONTEIRO ON OVERFLOW DISPLAY MESSAGE BOX "LIMITE DA STRING ENCONTRADO !" EXIT PERFORM END-STRING DISPLAY CORES AT 1210 REVERSE END-PERFORM GOBACK. </pre>	<pre> LIMPA A TELA MOSTRE "INFORME A COR: " EXECUTE ATÉ QUE 1 SEJA DIFERENTE DE 1 RECEBA COR SE COR = ESPAÇOS SAIA DO EXECUTE SE-FIM CONCATENE(JUNTE) COR DELIMITADA POR " " " " DELIMITADO PELO TAMANHO EM CORES COM PONTEIRO SE DER ESTOURO MOSTRE A CAIXA DE MENSAGEM "LIMITE DA STRING ENCONTRADO !" SAIA DO EXECUTE CONCATENE-FIM MOSTRE CORES EXECUTE-FIM. </pre>
--	--

UNSTRING

O comando UNSTRING, como o nome sugere faz o contrário do comando string, ou seja, possibilita fazer a desmontagem de um item de dado em vários outros. Um exemplo clássico de utilização do unstring, é a importação de dados em arquivos onde cada campo é separado por um caracter específico, onde é bem comum encontrarmos como delimitador o caracter “;”.

<pre>IDENTIFICATION DIVISION. PROGRAM-ID. USTR001. WORKING-STORAGE SECTION. 01 TODAS-AS-CORES PIC X(100) VALUE "AZUL VERDE ROXO". 77 COR-1 PIC X(010). 77 COR-2 PIC X(010). 77 COR-3 PIC X(010). PROCEDURE DIVISION. DISPLAY WINDOW ERASE UNSTRING TODAS-AS-CORES DELIMITED BY SPACES INTO COR-1, COR-3, COR-2 DISPLAY " COR-1: " COR-1 " COR-2: " COR-2 " COR-3: " COR-3 ACCEPT OMITTED.</pre>	<pre>LIMPA A TELA SEPRE TODAS-AS-CORES DELIMITADO POR ESPAÇO PARA COR-1, COR-2, COR-3</pre>
--	---

TRABALHANDO COM ARQUIVOS

Hoje, sem dúvida, a utilização de bancos de dados relacionais representam um grande avanço e são indiscutivelmente vantajosos. Mas aqui iremos tratar apenas de arquivos nativos COBOL, que permitem acesso extremamente rápido e são muito confiáveis, além de não possuir custo adicional.

SELECT

O que é uma SELECT ?

A SELECT possui diversas definições sobre os arquivos. Através dessas definições identificamos o tipo de arquivo, sua organização, método de acesso, índices de acesso, controles de erro e métodos de bloqueio. Dependendo da organização do arquivo, sua SELECT terá ou não algumas dessas definições.

Onde colocamos a SELECT ?

Ela deve ser colocada na ENVIRONMENT DIVISION, na INPUT-OUTPUT SECTION, dentro de FILE-CONTROL.

Tipos de arquivo (ORGANIZAÇÃO)

Em COBOL temos 3 tipos de arquivos:

- SEQUENCIAL
- RELATIVO
- INDEXADO

Arquivos seqüenciais (ORGANIZATION IS SEQUENTIAL)

São arquivos organizados sequencialmente, não possuem índices de acesso, normalmente são arquivos texto. Podemos citar como exemplos desse tipo de arquivo um programa fonte COBOL, um arquivo em fita magnética, um arquivo de transferência de dados. Como se trata de um arquivo não indexado, seu acesso só pode ser seqüencial, ou seja, só é possível localizar um registro ou informação através da leitura seqüencial do mesmo, passando por todos os registros anteriores.

Arquivos relativos (ORGANIZATION IS RELATIVE)

Possuem um identificador de registro que tem como objetivo controlar a posição relativa do registro dentro do arquivo. Esse identificador não é definido como parte do registro, mas deve estar definido na WORKING-STORAGE SECTION. Funciona de forma similar a um arquivo indexado, porém não permite chaves alternadas. Sua utilização não é muito comum, pois o índice controla apenas a posição do registro dentro do arquivo. Esse tipo de arquivo permite acesso seqüencial, randômico ou dinâmico (seqüencial e randômico).

Arquivos Indexados (ORGANIZATION IS INDEXED)

São os arquivos mais utilizados, pois podem possuir índices, que possibilitam a consulta e a ordenação por diversos campos diferentes. Possuem uma organização mais complexa, e, normalmente fisicamente são compostos de 2 ou mais arquivos, sendo um arquivo de dados e outro (ou outros) de índices. Esse tipo de arquivo permite acesso seqüencial, randômico ou dinâmico (seqüencial e randômico).

Ex: Cadastro de clientes, Movimento de peças, Cadastro de veículos...

MÉTODOS DE ACESSO (ACCESS MODE)

São três métodos de acesso, e, estão diretamente relacionadas a organização dos arquivos envolvidos.

		ACCESS MODE		
		SEQUENTIAL	RANDOM	DYNAMIC
ORGANIZATION	SEQUENTIAL	✓	✗	✗
	RELATIVE	✓	✓	✓
	INDEXED	✓	✓	✓

ÍNDICES DE ACESSO (RELATIVE KEY, RECORD KEY, ALTERNATE RECORD KEY)

Índices são exatamente o que o nome sugere (uma forma mais eficaz de localização). Já imaginou localizar um assunto em um livro de 1000 páginas e que você não conhece sem um índice? Os índices nos permitem localizar fácil e rapidamente os registros desejados e nisso se baseia a grande maioria das operações de um sistema.

O uso de índices também está relacionado com a organização do arquivo.

		TIPO DE INDICE	
		RECORD KEY ALTERNATE RECORD KEY	RELATIVE KEY
ORGANIZATION	SEQUENTIAL	✗	✗
	RELATIVE	✗	✓
	INDEXED	✓	✗

CONTROLE DE ERROS (FILE STATUS)

Nos permite identificar se as operações realizadas com arquivos foram bem sucedidas. A Cláusula FILE STATUS nos retorna uma série de códigos que permitem a identificação do que ocorreu com a última operação do arquivo. Essa cláusula deve ser informada na SELECT do arquivo e também definida na WORKING-STORAGE SECTION.

Ex:

<pre> SELECT CLIENTES ASSIGN TO "CLIENTES.DAT" ORGANIZATION IS INDEXED ACCESS MODE IS DYNAMIC RECORD KEY IS CLI-CODIGO ALTERNATE RECORD KEY IS CLI-NOME WITH DUPLICATES FILE STATUS IS <u>CLI-STATUS</u>. WORKING-STORAGE SECTION. 77 <u>CLI-STATUS</u> PIC X(002). 88 <u>CLI-OK</u> VALUES "00" THRU "09". </pre>	<pre> SELECIONE CLIENTES EM "CLIENTES.DAT" ORGANIZAÇÃO É INDEXADA MODO DE ACESSO É DINAMICO CHAVE DO REGISTRO É CLI-CODIGO CHAVE ALTERNATIVA É CLI-NOME E PERMITE DUPLICAÇÕES FILE STATUS É CLI-STATUS. </pre>
---	--

TABELA DE STATUS – ANSI 85

Código		Problema	Em que operações ocorre
1	2		
0	0	Operação bem sucedida	TODAS
0	2	Houve uma duplicação em chave que permite duplicidade (apenas informativo)	WRITE, REWRITE
0	5	O arquivo não existia e foi criado (apenas informativo)	OPEN
1	0	Final do arquivo encontrado	READ NEXT
2	2	Registro duplicado no índice primário ou no índice alternativo que não permite duplicidade	WRITE, REWRITE
2	3	Registro não encontrado	READ, START, DELETE
2	4	Disco cheio	WRITE
3	0	Erro permanente (sistema operacional)	TODAS
3	7	Não há permissão para a operação desejada	OPEN, WRITE, REWRITE, DELETE
3	9	Erro de layout, a definição do programa está diferente da definição física do arquivo de dados	OPEN
4	1	Arquivo já está aberto	OPEN
4	2	Arquivo não aberto	READ, WRITE, REWRITE, DELETE
4	4	Tamanho do registro modificado, ou modo de acesso não permite a operação	WRITE, REWRITE
4	6	Não existe registro corrente Tentativa de ler ou posicionar além do fim/início do arquivo	READ NEXT, START
4	7	Arquivo não aberto para gravação	WRITE, DELETE, REWRITE
4	8	Arquivo aberto de forma que não permite a operação, ou modo de acesso não condiz com a operação	WRITE
9	3	Arquivo bloqueado por outro usuário	OPEN
9	9	Registro bloqueado por outro processo	READ, WRITE, REWRITE, DELETE

IMPORTANTE

Em todas as operações com arquivos de dados, deve-se testar o retorno "FILE-STATUS" para saber se houve falha ou não. É um péssimo hábito ignorar essa regra, pois os resultados podem ser muito comprometedores. Muitos problemas difíceis de localizar em sistemas complexos e grandes ocorrem porque essa regra não é respeitada.

MODOS DE BLOQUEIO (LOCK MODE)

Em ambientes multi-usuário é comum que ocorram acessos simultâneos ao mesmo registro, isso não representa nenhum problema quando se trata apenas de leitura, sem nenhuma atualização desse registro. Porém, há casos onde desejamos que, enquanto fazemos uma atualização num dado registro, outros usuários não consigam alterá-lo também.

FD – FILE DESCRIPTION

O que é uma FD?

A FD (File description) contém a definição da estrutura de dados do arquivo, nela são definidos os campos com seus respectivos atributos. A FD é sempre organizada em forma de registro e as definições de dados seguem as mesmas regras já vistas anteriormente na WORKING-STORAGE SECTION.

Onde colocamos a FD ?

Deve ser colocada na DATA DIVISION, dentro de FILE SECTION.

Exemplos de selects e fds (modelos simples apenas, não se tratam de sugestão)

```
SELECT CLIENTES ASSIGN TO "/SISTEMA/ARQUIVOS/CLIENTES.DAT"
      ORGANIZATION IS INDEXED
      ACCESS MODE IS DYNAMIC
      RECORD KEY IS CLI-CODIGO
      ALTERNATE RECORD KEY IS
      CLI-NOME WITH DUPLICATES
      ALTERNATE RECORD KEY IS CLI-CPF
      FILE STATUS IS STAT-CLIENTES.
```

```
FD  CLIENTES
   LABEL RECORD STANDARD.
01  RG-CLIENTES.
    05  CLI-CODIGO          PIC 9(006).
    05  CLI-NOME            PIC X(030).
    05  CLI-RUA             PIC X(040).
    05  CLI-BAIRRO          PIC X(040).
    05  CLI-CIDADE          PIC X(040).
    05  CLI-UF              PIC A(002).
    05  CLI-CEP             PIC 9(008).
    05  CLI-CPF             PIC 9(014).
```

```
SELECT VEICULOS ASSIGN TO "/SISTEMA/ARQUIVOS/VEICULOS.DAT"
      ORGANIZATION IS INDEXED
      ACCESS MODE IS DYNAMIC
      RECORD KEY IS VEIC-CODIGO
      ALTERNATE RECORD KEY IS VEIC-PLACA
      ALTERNATE RECORD KEY IS VEIC-RENAVAM
      ALTERNATE RECORD KEY IS VEIC-CHASSI
      FILE STATUS IS STAT-VEICULOS.
```

```

FD  VEICULOS
    LABEL RECORD STANDARD.
01  RG-VEICULOS.
     05  VEIC-CODIGO          PIC 9(006).
     05  VEIC-PLACA           PIC X(007).
     05  VEIC-RENAVAM         PIC 9(008).
     05  VEIC-CHASSI          PIC X(025).
     05  VEIC-ANO-FAB         PIC 9(004).
     05  VEIC-ANO-MOD         PIC 9(004).
     05  VEIC-MARCA           PIC X(015).
     05  VEIC-MODELO          PIC X(025).

```

```

SELECT PRODUTOS ASSIGN TO "/SISTEMA/ARQUIVOS/PRODUTOS.DAT"
      ORGANIZATION IS INDEXED
      ACCESS MODE IS DYNAMIC
      RECORD KEY IS PROD-CODIGO
      ALTERNATE RECORD KEY IS
      PROD-DESCRICAO WITH DUPLICATES
      FILE STATUS IS STAT-PRODUTOS.

```

```

FD  PRODUTOS
    LABEL RECORD STANDARD.
01  RG-PRODUTOS.
     05  PROD-CODIGO          PIC 9(006).
     05  PROD-DESCRICAO       PIC X(040).
     05  PROD-MEDIDA          PIC X(005).

```

COPY de selects e FD

Uma definição de arquivos (select e FD), normalmente será utilizada por diversos programas diferentes, e o uso do COPY nos permite minimizar o trabalho de codificação. Para isso, podemos criar arquivos específicos que serão incluídos no programa pelo COPY. No exemplo abaixo, ao invés de escrevermos a select e a fd dos arquivos de clientes e produtos no nosso programa, apenas usaremos o COPY de arquivos separados (.fd e .sl)

```

IDENTIFICATION DIVISION.
PROGRAM-ID.  TESTE.
AUTHOR.  MARCIO A. SILVA.
DATE-WRITTEN.  JAN/2010.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
COPY "CLIENTES.SL".
COPY "PRODUTOS.SL".
DATA DIVISION.
FILE SECTION.
COPY "CLIENTES.FD".
COPY "PRODUTOS.FD".
WORKING-STORAGE SECTION.

```

Comandos de arquivos

A seguir, veremos os comandos utilizados nas operações com arquivos de dados, esses comandos são relativamente simples.

OPEN

Abre um arquivo.

Modos de abertura

INPUT	Apenas para leitura, não permitindo operações de atualização.
OUTPUT	Apaga e recria o arquivo (cuidado com esse modo)
I-O	Leitura e atualização, permite qualquer operação.
EXTEND	Abre o arquivo e posiciona logo no final, de modo a continuar a gravar dali pra frente (para arquivos texto)

OPEN INPUT CLIENTES	ABRA CLIENTES APENAS PARA LEITURA
OPEN OUTPUT TEMPORARIO	RECRIE O ARQUIVO TEMPORARIO
OPEN I-O MOVIMENTO	ABRA O MOVIMENTO PARA ATUALIZAÇÃO/LEITURA
OPEN EXTEND TEXTO	ABRA E POSICIONE NO FINAL DO ARQUIVO TEXTO
OPEN INPUT CLIENTES I-O MOVIMENTO	ABRA CLIENTES PARA LEITURA E MOVIMENTO PARA LEITURA/GRAVAÇÃO

Importante: qualquer operação de atualização (write, rewrite, delete) ou de leitura (read), ou de localização (start) não será executada com o arquivo em questão fechado.

CLOSE

Fecha um arquivo.

CLOSE CLIENTES	FECHA CLIENTES
CLOSE CLIENTES, FILIAIS, EMPRESAS	FECHA CLIENTES, FILIAIS, EMPRESAS

READ

Lê um registro – busca os registros existentes no arquivo, de acordo com os critérios de leitura utilizados. Este comando pode ser usados de várias formas diferentes, de acordo com o tipo de operação desejada.

READ ARQUIVO NEXT - Lê o próximo registro

READ ARQUIVO PREVIOUS - Lê o registro anterior

READ ARQUIVO

- Lê um registro de forma direta, para isso é necessário que os campos que compõem o índice (chave) tenham sido informados para localização.

READ ARQUIVO KEY IS ARQ-NOME

- Lê um registro de forma direta e é informada qual a chave (índice) que deve ser usada nessa leitura, também nesse caso é necessário que o campo chave possua o valor desejado na busca.

<pre> * LEITURA SEQUENCIAL PERFORM UNTIL 1 <> 1 READ CLIENTES <u>NEXT</u> AT END EXIT PERFORM END-READ DISPLAY CLI-NOME END-PERFORM </pre>	<pre> EXECUTE ATÉ QUE 1 SEJA DIFERENTE DE 1 LEIA CLIENTES PROXIMO ATÉ O FINAL SAIA DO EXECUTE LEIA-FIM MOSTRE CLI-NOME EXECUTE-FIM </pre>
<pre> * LEITURA RANDÔMICA (BUSCA DIRETA) MOVE WK-CLIENTE TO CLI-CODIGO READ CLIENTES IF VALID-CLIENTES DISPLAY "CLIENTE ENCONTRADO !!" ELSE DISPLAY "CLIENTE NÃO ENCONTRADO !!" END-IF </pre>	<pre> MOVA WK-CLIENTE PARA CLI-CODIGO LEIA CLIENTES SE CLIENTES-VALIDO MOSTRE "CLIENTE ENCONTRADO !!" SENÃO MOSTRE "CLIENTE NÃO ENCONTRADO !!" FIM-SE </pre>
<pre> * LEITURA SEQUENCIAL (PARA TRÁS) PERFORM UNTIL 1 <> 1 READ CLIENTES <u>PREVIOUS</u> AT END EXIT PERFORM END-READ DISPLAY CLI-NOME END-PERFORM </pre>	<pre> EXECUTE ATÉ QUE 1 SEJA DIFERENTE DE 1 LEIA CLIENTES ANTERIOR ATÉ O FINAL SAIA DO EXECUTE LEIA-FIM MOSTRE CLI-NOME FIM-EXECUTE </pre>
<pre> * LEITURA SEQUENCIAL (SEM BLOQUEIO) PERFORM UNTIL 1 <> 1 READ CLIENTES <u>NEXT</u> WITH NO LOCK AT END EXIT PERFORM END-READ DISPLAY CLI-NOME END-PERFORM </pre>	<pre> EXECUTE ATÉ QUE 1 <> 1 LEIA CLIENTES PROXIMO SEM BLOQUEIO ATÉ O FINAL SAIA DO EXECUTE LEIA-FIM MOSTRE CLI-NOME EXECUTE-FIM </pre>

WRITE

Grava / escreve um registro – adiciona um novo registro no arquivo.

<pre>WRITE RG-CLIENTES IF NOT VALID-CLIENTES DISPLAY MESSAGE BOX "ERRO AO GRAVAR CLIENTE !!" END-IF</pre>	<pre>GRAVE REGISTRO DE CLIENTES SE NÃO FOR VALIDO MOSTRE MENSAGEM "ERRO AO GRAVAR CLIENTE !!" FIM-SE</pre>
---	--

Importante: caso já exista no arquivo um registro com o índice igual ao que se deseja gravar, a gravação não será realizada e será retornado código de status "22". Isso pode ser verificado, através da leitura do arquivo no momento em que a chave é informada, assim é possível avisar antecipadamente ao usuário que o registro já existe.

DELETE

Elimina / exclui um registro.

<pre>DELETE CLIENTES IF NOT VALID-CLIENTES DISPLAY MESSAGE BOX "ERRO AO EXCLUIR CLIENTE !!" END-IF</pre>	<pre>EXCLUA CLIENTE SE NÃO FOR VALIDO MOSTRE MENSAGEM "ERRO AO EXCLUIR CLIENTE !!" FIM-SE</pre>
--	---

Importante: só é possível excluir um registro que já existe, caso se deseje regravar um registro não existente, o status "23" será retornado. Uma boa dica é que tenha ocorrido uma leitura antes da operação de exclusão.

REWRITE

Altera / regrava um registro.

<pre>REWRITE RG-CLIENTES IF NOT VALID-CLIENTES DISPLAY MESSAGE BOX "ERRO AO REGRAVAR CLIENTE !!" END-IF</pre>	<pre>REGRAVE REGISTRO DE CLIENTES SE NÃO FOR VALIDO MOSTRE MENSAGEM "ERRO AO REGRAVAR CLIENTE !!" FIM-SE</pre>
---	--

Importante: assim como ocorre na exclusão, só é possível regravar um registro que já existe, caso se deseje regravar um registro não existente, o status "23" será retornado. Vale a dica de que tenha ocorrido uma leitura antes da operação de regravação.

START

Posiciona o ponteiro do registro de acordo com o valor de um índice (chave) e também define o índice (chave) a ser utilizado.

```
START ARQUIVO KEY IS (NOT) LESS ou <
                        GREATER ou >
                        EQUAL ou = <NOME DA CHAVE>
```

Ex:

<pre>MOVE LOW-VALUES TO CLI-CODIGO START CLIENTES KEY IS > CLI-CODIGO INVALID KEY DISPLAY "ERRO AO LOCALIZAR !!" GO TO ERROS END-START</pre>	<pre>INICIALIZA O CLI CODIGO COM VALORES BAIXOS POSICIONA EM CLIENTES MAIOR QUE CLI-CODIGO SE ERRO DE CHAVE MOSTRA MENSAGEM DE ERRO VA PARA ERROS POSICIONA-FIM</pre>
<pre>MOVE "D" TO CLI-NOME START CLIENTES KEY IS NOT LESS CLI-NOME INVALID KEY DISPLAY "ERRO AO LOCALIZAR !!" GO TO ERROS END-START</pre>	<pre>INICIALIZA CLI-NOME COM A LETRA "D" POSICIONA CLIENTES MAIOR QUE CLI-NOME SE ERRO DE CHAVE MOSTRA MENSAGEM DE ERRO VA PARA ERROS POSICIONA-FIM</pre>

Importante: o *START* apenas posiciona o registro, mas não faz a leitura do mesmo, para isso é necessário o uso do comando *READ* após o *START*.

INVALID KEY

Invalid key pode ser utilizado em operações de leitura e atualização de arquivos (READ, START, DELETE, WRITE, REWRITE) e sua função é estabelecer procedimentos caso ocorram erros de índices nessas operações.

Ex:

<pre>WRITE RG-CLIENTES INVALID KEY DISPLAY "ERRO AO GRAVAR !!" GO TO ERROS END-WRITE</pre>	<pre>GRAVA REGISTRO DE CLIENTES EM CASO DE ERRO DE CHAVE MOSTRA MENSAGEM DE ERRO VA PARA ERROS GRAVA-FIM</pre>
<pre>REWRITE RG-CLIENTES INVALID KEY DISPLAY "ERRO AO REGRAVAR !!" GO TO ERROS END-REWRITE</pre>	<pre>REGRAVA REGISTRO DE CLIENTES EM CASO DE ERRO DE CHAVE MOSTRA MENSAGEM DE ERRO VA PARA ERROS REGRAVA-FIM</pre>
<pre>READ CLIENTES INVALID KEY DISPLAY "REGISTRO NÃO EXISTE !!" END-READ</pre>	<pre>LE CLIENTES EM CASO DE CHAVE INVALIDA MOSTRA MENSAGEM DE ERRO LE-FIM</pre>

Mas então o INVALID KEY substitui o teste de status ?

Já foi observado em alguns compiladores que a cláusula INVALID KEY retorna erros específicos de índices, como registro inexistente ou duplicado e não erros mais complexos. Dessa forma, a opinião pessoal do autor é que se prefira o uso do teste de status para evitar surpresas desagradáveis, pois já passou por situações assim.

Abaixo um programa simples que faz apenas a inclusão de clientes.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CADCLIENTES.
AUTHOR. MARCIO ADROALDO DA SILVA.
DATE-WRITTEN. JAN-2010.
ENVIRONMENT DIVISION.
SPECIAL-NAMES.
        DECIMAL-POINT IS COMMA.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
COPY "CLIENTES.SL".
DATA DIVISION.
FILE SECTION.
COPY "CLIENTES.FD".
WORKING-STORAGE SECTION.
COPY "ACUGUI.DEF".
COPY "ACUCOBOL.DEF".
77 KEYSSTATUS          PIC 9(004)  SPECIAL-NAMES CRT STATUS.
88 ESCAPE-KEY          VALUE 27.
88 WRITE-KEY           VALUE 221.
88 CLEAR-KEY           VALUE 222.
77 STAT-CLIENTES      PIC XX.
88 VALID-CLIENTES     VALUE "00" THRU "09".
SCREEN SECTION.
COPY "TELACLI.SCR".
PROCEDURE DIVISION.
INICIO.
        OPEN I-O CLIENTES
        IF NOT VALID-CLIENTES
                DISPLAY MESSAGE BOX
                "ERRO AO ABRIR CLIENTES !"
                "STATUS: " STAT-CLIENTES
                EXIT PARAGRAPH
        END-IF
        DISPLAY STANDARD GRAPHICAL WINDOW LINES 23 SIZE 80
                TITLE "Inclusão de Clientes"
        DISPLAY TELA
        PERFORM WITH TEST AFTER UNTIL ESCAPE-KEY
                ACCEPT TELA ON EXCEPTION
                PERFORM CONTROLE-TELA
                END-ACCEPT
        END-PERFORM.
FINALIZA.
        CLOSE CLIENTES.
        GOBACK.
CONTROLE-TELA.
        EVALUATE TRUE
                WHEN WRITE-KEY
                        PERFORM GRAVAR
                WHEN CLEAR-KEY
                        PERFORM LIMPA-TELA
        END-EVALUATE.
LIMPA-TELA.
        INITIALIZE RG-CLIENTES
        DISPLAY TELA.
GRAVAR.
        WRITE RG-CLIENTES
        IF NOT VALID-CLIENTES
                DISPLAY MESSAGE BOX
                "ERRO AO GRAVAR CLIENTES !"
                "STATUS: " STAT-CLIENTES
        ELSE
                PERFORM LIMPA-TELA
        END-IF.
```

Arquivo de tela usado no COPY (TELACLI.SCR):

```
01  TELA BLANK SCREEN.
05  LABEL TITLE "Código: " LINE 03 COL 06.
05  T-CODIGO ENTRY-FIELD USING CLI-CODIGO PIC ZZZZ9 COL 14.
05  LABEL TITLE "Nome: " LINE 05 COL 06.
05  T-NOME ENTRY-FIELD USING CLI-NOME COL 14.
05  LABEL TITLE "Rua: " LINE 07 COL 06.
05  T-RUA ENTRY-FIELD USING CLI-RUA COL 14.
05  LABEL TITLE "Nº: " LINE 09 COL 06.
05  T-NUMERO ENTRY-FIELD USING CLI-NUMERO COL 14.
05  LABEL TITLE "Bairro: " LINE 11 COL 06.
05  T-BAIRRO ENTRY-FIELD USING CLI-BAIRRO COL 14.
05  LABEL TITLE "Cidade: " LINE 13 COL 06.
05  T-CIDADE ENTRY-FIELD USING CLI-CIDADE COL 14.
05  LABEL TITLE "UF: " LINE 15 COL 06.
05  T-UF ENTRY-FIELD USING CLI-UF COL 14.
05  LABEL TITLE "CEP: " LINE 17 COL 06.
05  T-CEP ENTRY-FIELD USING CLI-CEP COL 14.
05  LABEL TITLE "CNPJ: " LINE 19 COL 06.
05  T-CNPJ ENTRY-FIELD USING CLI-CNPJ COL 14.
05  B-LIMPAR PUSH-BUTTON TITLE "Limpa tela" LINE 03 COL 65
    EXCEPTION-VALUE 222.
05  B-GRAVAR PUSH-BUTTON TITLE "Gravar" LINE 05 COL 65
    EXCEPTION-VALUE 221.
05  B-SAIR PUSH-BUTTON TITLE "Sair" LINE 07 COL 65
    EXCEPTION-VALUE 27.
```

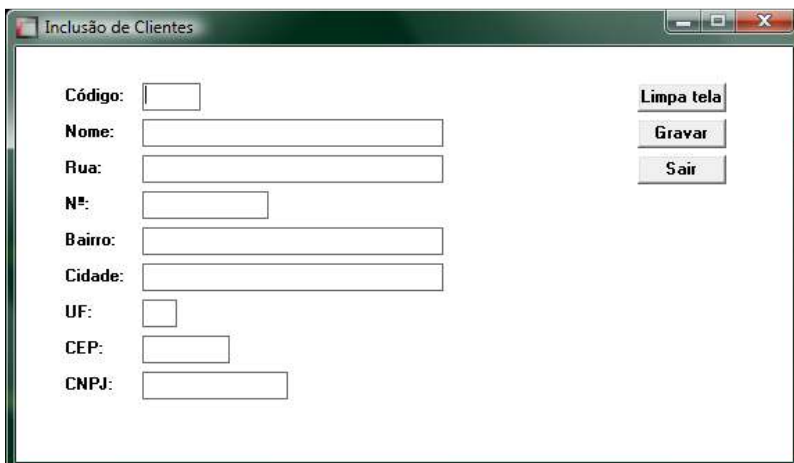
Arquivo de SELECT usado no COPY (CLIENTES.SL):

```
SELECT CLIENTES ASSIGN TO
    "C:\CURSOCOBOL\CLIENTES.DAT"
    ORGANIZATION IS INDEXED
    ACCESS MODE IS DYNAMIC
    RECORD KEY IS CLI-CODIGO
    FILE STATUS IS STAT-CLIENTES.
```

Arquivo de FD usado no COPY (CLIENTES.FD):

```
FD  CLIENTES
    LABEL RECORD STANDARD.
01  RG-CLIENTES.
05  CLI-CODIGO          PIC 9(005).
05  CLI-CNPJ            PIC 9(014).
05  CLI-NOME            PIC X(030).
05  CLI-RUA             PIC X(030).
05  CLI-NUMERO          PIC 9(012).
05  CLI-BAIRRO          PIC X(030).
05  CLI-CIDADE          PIC X(030).
05  CLI-CEP             PIC 9(008).
05  CLI-UF              PIC A(002).
```

Abaixo a tela do programa em execução



Gerando relatórios (arquivos seqüenciais)

Abordaremos aqui apenas a forma mais tradicional de geração de relatórios, uma vez que o Acucobol, infelizmente, não oferece suporte ao uso de Report Section. Aqui, a abordagem será extremamente simples, direcionando apenas relatórios comuns em impressoras matriciais.

Os relatórios em COBOL, não passam de arquivos seqüenciais em linha, que devem ser gravados registro a registro.

Definindo a select:

```
SELECT IMPRESSORA ASSIGN TO PRINTER
      ORGANIZATION IS LINE SEQUENTIAL
      FILE STATUS IS STAT-IMPRESSORA.
```

Temos aqui a select de um arquivo seqüencial, que pode ser utilizado também para geração de arquivos texto, bastando para isso, utilizar ao invés de **PRINTER** um nome de arquivo significativo.

Definindo a FD:

```
FD IMPRESSORA
  LABEL RECORD OMITTED.
01 RG-IMPRESSORA          PIC X(080).
```

Nesse nosso caso, o relatório tem 80 caracteres de largura.

Abrindo a impressora:

```
OPEN OUTPUT IMPRESSORA
```

A impressora sempre deve ser aberta com OUTPUT, uma vez que é um dispositivo de saída.

O processo de impressão é, relativamente simples, pois é feita através da gravação do registro de impressão. A parte mais trabalhosa de se fazer, sem o uso de ferramentas, é a montagem do layout do relatório e os controles de quebra. (Tudo isso, seria bem mais simples utilizando-se Report Section).

Abaixo um exemplo de um relatório simples usando nosso arquivo de clientes e que informa um totalizador de registros no final.

```

IDENTIFICATION DIVISION.
PROGRAM-ID. TESTEREL01.
AUTHOR. MARCIO ADROALDO DA SILVA.
DATE-WRITTEN. JAN-2010.
REMARKS. CURSO COBOL - IMPRESSAO SIMPLIS.
ENVIRONMENT DIVISION.
SPECIAL-NAMES.
    DECIMAL-POINT IS COMMA.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
COPY "CLIENTES.SL".
SELECT IMPRESSORA ASSIGN TO PRINTER
    ORGANIZATION IS LINE SEQUENTIAL
    FILE STATUS IS STAT-IMPRESSORA.
DATA DIVISION.
FILE SECTION.
COPY "CLIENTES.FD".
FD IMPRESSORA
    LABEL RECORD OMITTED.
01 RG-IMPRESSORA          PIC X(080).
WORKING-STORAGE SECTION.
COPY "ACUCOBOL.DEF".
COPY "ACUGUI.DEF".
77 CONTADOR              PIC 9(005) VALUE 0.
77 PAGINA                 PIC 9(004) VALUE 0.
77 LINHAS                 PIC 9(002) VALUE 80.
77 STAT-CLIENTES         PIC XX.
77 88 VALID-CLIENTES     VALUE "00" THRU "09".
77 STAT-IMPRESSORA       PIC XX.
77 88 VALID-IMPRESSORA   VALUE "00" THRU "09".
*****
* AGORA DEFINIMOS O LAY OUT DE IMPRESSÃO
* CABECALHO, LINHA DETALHE E RODAPE
*****
01 TRACOS                 PIC X(080) VALUE ALL "=".
01 CAB01.
    05                   PIC X(068) VALUE
    "RELATORIO DO CADASTRO DE CLIENTES".
    05                   PIC X(008) VALUE "PAGINA: ".
    05 CAB01-PAGINA      PIC ZZZ9.
01 CAB02.
    05                   PIC X(007) VALUE " CODIGO".
    05                   PIC X(003).
    05                   PIC X(025) VALUE "NOME".
    05                   PIC X(003).
    05                   PIC X(020) VALUE "CIDADE".
    05                   PIC X(003).
    05                   PIC X(002) VALUE "UF".
01 LDT.
    05                   PIC X(003).
    05 LDT-CODIGO        PIC ZZZ9.
    05                   PIC X(003).
    05 LDT-NOME          PIC X(025).
    05                   PIC X(003).
    05 LDT-CIDADE        PIC X(020).
    05                   PIC X(003).
    05 LDT-UF            PIC X(002).
01 RODAPE.
    05                   PIC X(010).
    05                   PIC X(030) VALUE
    "TOTAL DE REGISTROS IMPRESSOS: ".
    05 ROD-CONTADOR     PIC ZZ.ZZ9.
PROCEDURE DIVISION.
INICIAL.
    OPEN INPUT CLIENTES
    IF NOT VALID-CLIENTES
        DISPLAY MESSAGE BOX
        "O ARQUIVO DE CLIENTES NÃO FOI ABERTO !!" H"0A"
        "CODIGO DE STATUS: " STAT-CLIENTES
        GOBACK
    END-IF.

    OPEN OUTPUT IMPRESSORA.
    IF NOT VALID-IMPRESSORA

```

```

        DISPLAY MESSAGE BOX
        "IMPOSSIVEL ABRIR IMPRESSORA !" H"0A"
        "CODIGO DE STATUS: " STAT-IMPRESSORA
        GOBACK
    END-IF
    MOVE LOW-VALUES    TO CLI-NOME
    START CLIENTES KEY IS > CLI-NOME
    IF NOT VALID-CLIENTES
        DISPLAY MESSAGE BOX
        "IMPOSSIVEL LOCALIZAR CLIENTES !!" H"0A"
        "CODIGO DE STATUS: " STAT-CLIENTES
        EXIT PARAGRAPH
    END-IF
*****
**  VAMOS FAZER UM LOOP PARA LER NOSSO ARQUIVO DE CLIENTES
**  ATE O FINAL E FAZER A IMPRESSÃO, NO FINAL MANDAMOS
**  IMPRIMIR O RODAPE COM A TOTALIZAÇÃO DOS REGISTROS
*****
        PERFORM UNTIL 1 <> 1
            READ CLIENTES NEXT AT END
            PERFORM RODAPE
            EXIT PERFORM
        END-READ
*****
**  ALIMENTAREMOS OS DADOS DO RELATORIO COM AS INFORMAÇÕES
**  DO NOSSO CADASTRO QUE SERÃO IMPRESSAS
*****
        MOVE CLI-CODIGO  TO LDT-CODIGO
        MOVE CLI-NOME    TO LDT-NOME
        MOVE CLI-CIDADE  TO LDT-CIDADE
        MOVE CLI-UF      TO LDT-UF
*****
**  A VARIÁVEL LINHAS NOS PERMITE FAZER A QUEBRA DE PAGINA
**  PERCEBA QUE INICIOU VALENDO 80, DESSA FORMA FORÇA A IM-
**  PRESSAO DO CABECALHO NA PRIMEIRA PAGINA
*****
        IF LINHAS > 61
            PERFORM CABECALHO
        END-IF
        ADD 1    TO LINHAS, CONTADOR
        WRITE RG-IMPRESSORA FROM LDT AFTER 1
    END-PERFORM.
ENCERRA.
    CLOSE IMPRESSORA, CLIENTES
    DISPLAY MESSAGE BOX "IMPRESSAO ENCERRADA !!"
    GOBACK.

CABECALHO.
    ADD 1    TO PAGINA
    MOVE 5    TO LINHAS
    MOVE PAGINA TO CAB01-PAGINA
*****
**  NÃO QUEREMOS QUE ANTES DE COMEÇAR A IMPRIMIR O RELATÓRIO
**  SEJA FEITO UM SALTO DE PAGINA, POR ISSO, A PRIMEIRA
**  VEZ QUE PASSAR AQUI NÃO EFETUAMOS O AFTER PAGE E SIM
**  O AFTER 1
*****
        IF PAGINA = 1
            WRITE RG-IMPRESSORA FROM TRACOS AFTER 1
        ELSE
            WRITE RG-IMPRESSORA FROM TRACOS AFTER PAGE
        END-IF
        WRITE RG-IMPRESSORA FROM CAB01  AFTER 1.
        WRITE RG-IMPRESSORA FROM TRACOS AFTER 1.
        WRITE RG-IMPRESSORA FROM CAB02  AFTER 1.
        WRITE RG-IMPRESSORA FROM SPACES AFTER 1.

RODAPE.
    MOVE CONTADOR  TO ROD-CONTADOR
    WRITE RG-IMPRESSORA FROM TRACOS AFTER 2.
    WRITE RG-IMPRESSORA FROM RODAPE AFTER 1.

```

Acucobol

O Acucobol possui um conjunto de ferramentas que auxiliam o programador em diversas tarefas, o domínio desses recursos é de extrema importância, pois são importantes no dia a dia, principalmente em situações de emergência.

Compilador

ccbl32 (Windows)

ccbl (LINUX, UNIX, DOS)

Ccbl32 prog01.cbl	Compila prog01, criando um objeto com o nome prog01.acu
Ccbl32 -Zd prog01.cbl	Compila prog01, criando um objeto com o nome prog01.acu com suporte a debugger
Ccbl32 -Z73 prog01.cbl	Compila prog01, criando um objeto com o nome prog01.acu que poderá ser executado com o runtime da versão 7.3
Ccbl32 -o @ prog01.cbl	Compila prog01, criando um objeto com o nome prog01 (sem extensão)
Ccbl32 -Fx prog01.cbl	Compila prog01, criando um objeto com o nome prog01.acu e também gerando XFD dos arquivos usados por esse programa Obs: pode-se definir a versão da XFD colocando-se o número da versão depois do -Fx, Exemplo -Fx3 gera XFD na versão 3 Quando a versão é omitida, será criada a XFD de acordo com a última versão suportada

Executor (runtime)

wrun32 (Windows)

runcbl (LINUX, UNIX, DOS)

Wrun32 prog01.acu	Executa o prog01.acu
Wrun32 -d prog01.acu	Debuga o prog01.acu
Wrun32 -dle c:\erro.log prog01.acu	Debuga o prog01.acu, gravando o log em c:\erro.log
Wrun32 -vv	Mostra a versão do runtime
Wrun32 -c cbl1.cfg prog01.acu	Executa o prog01.acu, mas informa que é para ser utilizado o arquivo de configuração cbl1.cfg

Utilitário de arquivos de dados

vutil32 (Windows)

vutil (LINUX, UNIX, DOS)

Vutil32 -info -x clientes.dat	Mostra informações detalhadas do arquivo clientes.dat
Vutil32 -rebuild clientes.dat	Reindexa o arquivo clientes.dat (recria o arquivo de índice)
	Debuga o prog01.acu, gravando o log em c:\erro.log
Vutil32 -extract clientes.dat > cli.txt	Extrai os dados do arquivo clientes.dat e coloca os dados no arquivo texto cli.txt
Vutil32 -load -t cli.txt clientes.dat	Insere os dados de cli.txt no arquivo clientes.dat

Utilitário de arquivo objeto

cblutl32 (Windows)
cblutil (LINUX, UNIX,DOS)

Cblutl32 –info prog01.acu	Mostra detalhes do objeto prog01.acu (versão do objeto, data e hora que foi compilado, e indica se está com DEBUG)
---------------------------	--

O arquivo cblconfig

O Acucobol utiliza um arquivo de configuração que permite modificar o comportamento da execução da aplicação (variáveis de configuração do runtime), caso esse arquivo não seja encontrado, as configurações padrão serão carregadas, e isso nem sempre é interessante. Por padrão, o arquivo será procurado em /etc/cblconfi ou /etc/cblconfig, dependendo do sistema operacional. Uma forma de indicar um arquivo diferente é através da opção –c junto com o runtime (execução). Existem dezenas de variáveis que podem ser configuradas neste arquivo.

Um exemplo comentado do arquivo cblconfig:

```
# NÃO PERMITE QUE O PROGRAMA FECHÉ AUTOMATICAMENTE
# AO ENCONTRAR UM ERRO DE CHAVE EM UMA LEITURA/ATUALIZAÇÃO
ERRORS_OK 1

# DEFINE QUAL O ARQUIVO DE ICONE QUE DEVE APARECER NA EXECUÇÃO
ICON \ICONES\ICONE1.ICO

# INIBE O RUNTIME DE VALIDAR OS PARAMETROS PASSADOS POR LINKAGE SECTION
# OS RESULTADOS PODEM SER DESAGRADÁVEIS
CHECK_USING 0

# DEFINE QUE ABERTURA DE ARQUIVOS COM OUTPUT PODERÃO CRIAR O ARQUIVO
# CASO NÃO EXISTA
OUTPUT_CREATES 1

# DEFINE QUE ABERTURA DE ARQUIVOS COM OUTPUT PODERÃO CRIAR O ARQUIVO
# CASO NÃO EXISTA
IO_CREATES 1

# DEFINE QUE <ENTER> TEM A MESMA FUNÇÃO DE <TAB> NA SCREEN SECTION
# POR PADRÃO, APENAS <TAB> SALTA DE CAMPOS NESSAS SITUAÇÕES
KEYSTROKE EDIT=Next TERMINATE=13 ^M

# DEFINE OS LOCAIS ONDE O RUNTIME DEVE PROCURAR PELOS OBJETOS(EXECUTÁVEIS)
CODE_PREFIX .;\INFOCUS\OBJECT;\INFOCUS\BRW

# INDICA QUE O CAMINHO DE FILE_PREFIX DEVERÁ SER INCLUÍDO NA CRIAÇÃO DE NOVOS
# ARQUIVOS
APPLY_FILE_PATH 1

# DEFINE OS LOCAIS ONDE O RUNTIME DEVE PROCURAR PELOS ARQUIVOS DE DADOS
FILE_PREFIX \INFOCUS\DATA\
# DEFINE O NUMERO MÁXIMO DE ARQUIVOS ABERTOS E TAMBÉM DE BLOQUEIOS POR ARQUIVO
MAX_FILES 255
MAX_LOCKS 6144

# TRADUZ AS MENSAGENS DO ACUCOBOL
TEXT 1=Pressione Enter !
TEXT 2=Este campo é numérico !!
TEXT 3=Entrada requerida neste campo !
TEXT 4=O campo precisa ser totalmente preenchido !
TEXT 5=Muitas Hot Keys ativadas !
TEXT 6=Programa está faltando ou não está acessível !
TEXT 7=Não é um aplicativo Extend
TEXT 8=Objeto apresenta corrupção
TEXT 9=Memória disponível não é adequada !
TEXT 10=Versão de Objeto não suportada !
```

TEXT 11=Programa já está ativo !
TEXT 12=Muitos segmentos externos !
TEXT 13=Não é suportado LARGE-MODEL !
TEXT 14=Essa aplicação deve ser encerrada primeiro !
TEXT 15=15
TEXT 16=16
TEXT 17=17
TEXT 18=Por favor, encerre esta aplicação primeiro !
TEXT 19=19
TEXT 20=Muitas Linhas !
TEXT 21=Gerenciador de licenças não está em execução !
TEXT 22=Caracter não permitido neste campo !
TEXT 23=&Ok
TEXT 24=&Sim
TEXT 25=&Não
TEXT 26=&Cancelar
TEXT 27=27
TEXT 28=Impossível acessar o arquivo "%s" pois está sendo usado por outros usuários. Deseja esperar mais ?
TEXT 30=Conexao recusada - Acuconnect pode não estar rodando !
TEXT 31=Por favor, entre com um valor entre %ld and %ld
TEXT 32=O programa contém código não suportado para o processador atual !
text 33=Número de licença inválido !
text 34=Conexão recusada - servidor remoto atingiu limite do contador de usuários !
TEXT 35=Erro de licença
text 36=O host remoto não está respondendo.\nPressione Ok para fechar.\nPressione Calcelar para aguardar mais %s segundos
text 1103=Erro 1103

O Alfred é um editor de arquivos indexados bastante versátil e relativamente simples de usar, que permite realizar operações de consulta / atualização em arquivos COBOL através de qualquer índice existente. Aqui entra um personagem novo na nossa história, o arquivo XFD¹.

A XFD do nosso cadastro de clientes:

```
# clientes.xfd - generated by ACUCOBOL-GT v8.1.0
# Generated Tue Feb 02 09:52:24 2010
# [Identification Section]
XFD,06,CLIENTES,CLIENTES,12
0000000161,0000000161,001
000,18,044,046,00
# [Key Section]
01,0,005,0000000000
01
CLI-CODIGO
# [Condition Section]

000

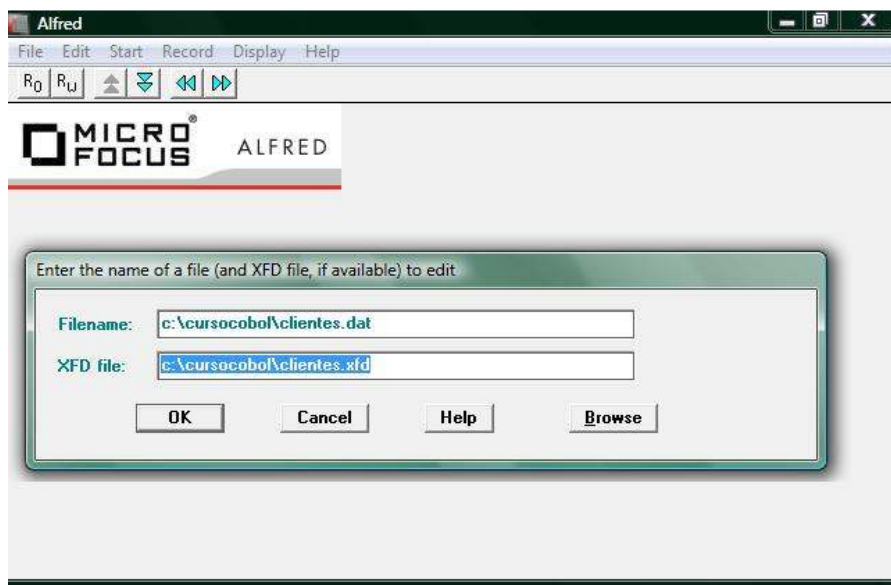
# [Field Section]

0009,00009,00010,00010

0000000000,0000000161,16,0000000161,+00,000,999,01,RG-CLIENTES
0000000000,0000000005,01,0000000005,+00,000,000,05,CLI-CODIGO
0000000005,0000000014,01,0000000014,+00,000,000,05,CLI-CNPJ
0000000019,0000000030,16,0000000030,+00,000,000,05,CLI-NOME
0000000049,0000000030,16,0000000030,+00,000,000,05,CLI-RUA
0000000079,0000000012,01,0000000012,+00,000,000,05,CLI-NUMERO
0000000091,0000000030,16,0000000030,+00,000,000,05,CLI-BAIRRO
0000000121,0000000030,16,0000000030,+00,000,000,05,CLI-CIDADE
0000000151,0000000008,01,0000000008,+00,000,000,05,CLI-CEP
0000000159,0000000002,18,0000000002,+00,000,000,05,CLI-UF
```

¹ Extend File Definition, este arquivo é um dicionário de dados, e, é gerado durante a compilação do seu aplicativo, deste que se utilize o parâmetro -Fx ou -Fa. É gerado um XFD para cada arquivo de dados, e não passa de um “mapa” com o tamanho, tipo e posição dos campos e chaves que compõem o arquivo. Este arquivo também é utilizado para soluções de acesso a RDBMS e XML. Maiores informações sobre XFD no Guia do Usuário Acucobol - Capítulo 5.3

A interface do ALFRED.



No exemplo acima, estamos abrindo o nosso cadastro de clientes, informamos o caminho do arquivo de dados e também do arquivo XFD. E agora podemos navegar pelos registros utilizando as funções do ALFRED.



Principais funções do menu do ALFRED:

File

Open	Abre a caixa de diálogo para que se informe o arquivo a abrir.
Select Key	Permite escolher o índice do arquivo (desde que exista mais de um).
Read/Write	Abre o arquivo de forma a permitir alterações, inclusões e exclusões.
Read Only	Abre o arquivo apenas para leitura
Print	Imprime os dados que estão na tela
Exit	Sai do ALFRED

Start

Not Less than	Solicita o valor da chave e posiciona num valor não menor que ela.
Not Greater than	Solicita o valor da chave e posiciona num valor não maior que ela.
First Record	Vai para o primeiro registro.
Last Record	Vai para o ultimo registro.

Record

Add a New Record	Permite a inserção de um novo registro. . (Desde que aberto com I-O)
Delete This Record	Deleta o registro corrente. (Desde que aberto com I-O)
Save This Record	Salva o registro corrente. . (Desde que aberto com I-O)
Next Record (^N)	Próximo registro.
Previous Record (^P)	Registro anterior.

Conclusão

Nosso objetivo aqui foi de apresentar o COBOL na sua forma mais simples, através de exemplos da utilização de comandos e das regras mais básicas da linguagem, por esse motivo, alguns comandos e conceitos não foram abordados. Esperamos poder melhorar esse material em revisões futuras, de forma a oferecer um conteúdo mais completo e abrangente.

Bibliografia

Stern & Stern – programação estruturada em COBOL para o século XXI

Carvalho – Microsoft COBOL 4.0

Carvalho – Microsoft COBOL 4.5 – Programação avançada

Acucorp – Documentação online

<http://cobol.404i.com>

<http://www.cadcobol.com>

<http://www.controlsyst.com>

Marcio Adroaldo da Silva
marcio_ad@yahoo.com.br
marcio.infocus@gmail.com
www.controlsyst.com

Janeiro - 2010