COBOL

ÍNDICE

1. LINGUAGEM DE PROGRAMAÇÃO COBOL

1.1. História

Há alguns anos atrás, em 1959, alguns profissionais da área de informática como os fabricantes de computadores, estudantes, representantes do governo e usuários, formaram uma conferência: a CODASYL, ou seja, Conference on Data Systems Language. Nesta conferência o principal item abordado foi a necessidade da criação de uma linguagem comercial, objetivando uma maior facilidade na comunicação entre computador e ser humano.

Em 1961, surgiu o COBOL (Common Business Oriented Language), que é uma linguagem voltada para a linguagem humana.

Portanto, podemos denominar COBOL, uma linguagem de alto nível, ou uma linguagem comercial.

A partir desta data, a linguagem COBOL, foi sendo melhorada e daí surgiu a primeira versão em 1968 chamada "ANS", e foi aprovada pela American National Standard Institute.

Podemos fazer algumas comparações das vantagens do uso da linguagem COBOL, ao invés da linguagem pura de máquina:

LINGUAGEM COBOL

1) Facilidade de escrita;
2) Diminuição das instruções para gerar um programa;
3) Manutenção simplificada;
4) Não há total necessidade de uma integração do programa com o computador, pois existe entre os fabricantes de computadores e o programa uma padronização e uniformização, sendo que, caso uma empresa queira mudar de equipamento, não se torna necessário refazer todos os programas existentes, e sim fazer pequenas alterações.

LINGUAGEM PURA MÁQUINA

1) Dificuldade de escrita, pois é uma linguagem de baixo nível composta por números binários, (0, 1), de difícil compreensão;
2) Muitas instruções;
3) Manutenção de programa difícil;
4) Depuração de erros difícil;

5) Necessidade de total integração entre computador e linguagem. Se uma empresa fosse mudar de computador, todos os seus programas teriam que ser convertidos.

1.2. Aplicação da Linguagem COBOL

O COBOL pode ser utilizado em equipamentos de grande porte (Mainframe) e em equipamentos de médio e pequeno porte, respectivamente, minicomputadores e microcomputadores, podendo ser utilizado em diversos ambientes operacionais.

Hoje existem várias versões de COBOL destinadas a Mainframe a minicomputadores e microcomputadores.

As principais diferenças entre COBOL para grande, médio e pequeno porte, estão no tratamento de arquivos e na comunicação entre usuário e sistema e operador e sistema.

1.3. Desenvolvimento de um Programa COBOL

O COBOL é uma linguagem de alto nível. Isto significa que é mais voltada para a linguagem humana e menos voltada para a linguagem de máquina, portanto, o computador não entende o COBOL.

Para podermos elaborar programas em COBOL e executar no computador, é necessário um software de tradução chamado "compilador" COBOL.

O compilador traduzirá o programa fonte (linguagem de alto nível) em programa objeto (linguagem de baixo nível) e irá acusar erros de sintaxe.

Além de traduzirmos um programa COBOL em linguagem de máquina para que o computador possa processar, precisamos também gerar um módulo executável para podermos utilizar o programa que criamos com determinado objetivo, par este ser atingido e concluído.

Este módulo executável será o gerado pela união dos módulos chamados e a resolução dos endereços entre as instruções e comandos que constituem o programa. O Linkage Editor é o software responsável.

1.4. Estrutura e Conteúdo da Linguagem

Um programa COBOL é formado por parágrafos que constituem as seções e estas constituem as divisões.

O COBOL possui 4 (quatro) divisões:

- `IDENTIFICATION DIVISION`: Identificação do programa.

- `ENVIRONMENT DIVISION`: Define o ambiente operacional, equipamento.
- `DATA DIVISION`: Define a origem e as características dos dados.
- `PROCEDURE DIVISION`: Conjunto de instruções e comandos.

1.5. Elemento de Linguagem

1.5.1. Caracteres

A codificação COBOL, é constituída por um conjunto básico de caracteres:

- 26 letras;
- 10 dígitos;
- Caractere branco;
- Ponto e vírgula;
- Aspa ou apóstrofe;
- Parêntesis esquerdo e direito;
- Ponto;
- Vírgula;
- Mais;
- Menos;
- Asterisco;
- Barra;
- Igual;
- Cifrão;
- Maior que;
- Menor que.

1.5.2. Caracteres

Existem as palavras reservadas da linguagem COBOL que veremos no item 1.5.6., existem as palavras atribuídas pelo programador.

Na criação de nomes, o programador deve obedecer a certas regras como:

- Tamanho máximo de 30 caracteres, e estes podem ser: letras, números e caracteres especiais;
- Iniciar a palavra com uma letra.

Exemplos:

```
NOME
DATA_NASC
A0001
```

O programador pode criar até 4 (quatro) tipos de nomes:

- Nomes de campo;
- Nomes de rotina;
- Nomes de condição;
- Nomes externos.

PS: Os nomes de rotina podem começar por um número.

Abordaremos com mais detalhes os tipos de nomes nas divisões correspondentes, nos próximos capítulos.

1.5.3. Literais

São valores constantes utilizados num programa. Existem dois tipos de literais:

- **Literal Numérica:**
  É composta por dígitos de 0 a 9, + ou -, . ou , e o tamanho máximo de um literal numérica é de 18 dígitos sem contar os caracteres diferentes de números. Exemplos:

```
500,00
-300
+1500,00
20
```

- **Literal não Numérica:**
  É composta por um conjunto de caracteres que estão entre aspas ou apóstrofes. Seu tamanho máximo é de 120 caracteres, excluindo-se as aspas ou apóstrofes. Exemplos:

```
"Kyoei Facom - Centro Educacional"
"%$#@! Caracteres especiais"
"001234567 + 7364892"
```

1.5.4. Constantes Figurativas

Fazem parte das palavras reservadas da linguagem COBOL, e possuem um determinado valor. Exemplos:

`HIGH-VALUES` – Maior valor alfanumérico

`LOW-VALUES` – Menor valor alfanumérico

`SPACE(S)` – Caractere branco

`ZERO(S)` – Numero 0

`ALL` – Transforma uma literal alfanumérica em constante figurativa

1.5.5. Regras de Pontuação

O compilador COBOL é um software detalhista quanto à pontuação utilizada no programa. Ao codificarmos devemos seguir as seguintes regras:

1) Ao final de cada sentença devemos usar logo em seguida o ponto final;

2) Entre uma palavra e outra devemos usar no mínimo um espaço em branco;
3) Caracteres de cálculos ou de comparações deverão ser separados por no mínimo um espaço em branco.

1.5.6. Palavras Reservadas

São palavras de uso exclusivo da linguagem, como por exemplo, comandos, nomes de divisões, nomes de seções, comparações, e outros.

Possuímos uma lista das palavras reservadas do COBOL, para que você possa consultar sempre que tiver dúvidas quanto a criação de palavras a serem utilizadas no programa.

1) Palavras Reservadas:

ACCESS	COMP	C07	FILE-CONTROL
ACTUAL	COMP-1	C08	FILE-LIMIT
ADVANCING	COMP-2	C09	FILE-LIMITS
AFTER	COMP-3	C10	FILLER
ALL	COMPUTACIONAL	C11	FINAL
ALPHABETIC	COMPUTACIONAL-1	C12	FIND
ALPHANUMERIC	COMPUTACIONAL-2	DATA	FIRST
ALTER	COMPUTACIONAL-3	DATE	GIVING
ALTERNATE	CONFIGURATION	DATE-COMPILED	GREATER
AND	CONSOLE	DATE-WRITTEN	HIGH-VALUE
APPLY	CONTAINS	DAY	HIGH-VALUES
ARE	CONTROL	DEBUG	I-O
AREA	CONTROLS	DEBUGGING	I-O-CONTROL
AREAS	COPY	DECIMAL-POINT	ID
ASCENDING	CORE-INDEX	DECLARATIVES	IDENTIFICATION
ASSIGN	CURRENT-DATE	DEPENDING	INDEX
AT	C01	DESCENDING	INDEXED
AUTHOR	C02	DIVISION	INPUT
BEFORE	C03	END-IF	INPUT-OUTPUT
CHARACTERS	C04	ERROR	INSTALLATION
COMMA	C05	FD	INVALID
COMMON	C06	FILE	JUST

JUSTIFIED	QUOTE	SKIP3	TRACE
KEY	QUOTES	SOURCE	TRACK
KEYS	RANDOM	SOURCE-	TRACK-AREA
LABEL	READY	COMPUTER	TRACK-LIMIT
LEFT	RECORD	SPACE	TRACKS
LESS	RECORDING	SPACES	TRANSFORM
LIMIT	RECORDS	SPECIAL-NAMES	UPSI-0
LIMITS	REDEFINES	STANDARD	UPSI-1
LINKAGE	REEL	START	UPSI-2
LOCK	REFERENCE	STATUS	UPSI-3
LOW-VALUE	REMAINDER	STOP	UPSI-4
LOW-VALUES	REMARKS	SYNC	UPSI-5
MEMORY	RENAMES	SINCRONIZED	UPSI-6
MODE	REPLACING	SYNIN	UPSI-7
MULTIPLE	RESERVE	SYSIPT	USAGE
NEXT	RESET	SYSLST	USE
NOTE	RETURN	SYSOUT	USING
NUMERIC	REWIND	SYSPCH	VALUE
OBJECT-	ROUNDED	SYSPUNCH	VALUES
COMPUTER	SAME	S01	VARYING
OCCURS	SD	S02	WHEN
OMITTED	SECTION	TALLY	WITH
OUTPUT	SECURITY	TALLYNG	WORDS
PAGE	SEEK	TAPE	WORKING-
PIC	SEGMENT-LIMIT	THAN	STORAGE
PICTURE	SENTENCE	THEN	ZERO
PROCEDURE	SIZE	THRU	ZEROES
PROCESSING	SKIP1	TIME-OF-DAY	ZEROS
PROGRAM-ID	SKIP2	TIMES	

2) Verbos

ACCEPT	COMPUTE	EXHIBIT	OPEN
ADD	CLOSE	GO TO	PERFORM
CALL	DISPLAY	IF	READ
CANCEL	DIVIDE	MOVE	RELEASE
CHAIN	EXAMINE	MULTIPLY	RETURN

REWRITE	SEARCH	STOP	WRITE
RUN	SORT	SUBTRACT	

3) Conectivos

AND	CORRESPONDING	IS	OR
ARE	DOWN	NOT	TO
AT	ELSE	OF	UNTIL
BY	FROM	OFF	UP
CORR	INTO	ON	

1.5.7. Codificação

Existe um padrão de codificação de um programa COBOL relativo às colunas.

As colunas de 1 a 6, são utilizadas para numerar as linhas do programa. Da coluna 1 a coluna 3, a numeração da folha é definida, e da coluna 4 a coluna 6, a numeração da linha é definida. A numeração da página/linha deverá ser feita em ordem crescente.

A coluna 7 é utilizada para indicar que a linha é de comentário, ou para indicar a continuação da linha anterior.

As colunas de 8 a 11 estão situadas na "margem A" da folha. Nessa margem são iniciados os nomes de divisões, seções, parágrafos e alguns números de nível.

As colunas de 12 a 72 estão situadas na "margem B" da folha. Nesta margem todos os parâmetros e instruções restantes são definidos.

A identificação do programa é feita dentre as colunas 73 e 80.

2. AS DIVISÕES DO COBOL

2.1. Identification Division

Esta é a primeira divisão de um programa COBOL. Como o próprio nome já diz, esta divisão é de identificação do programa.

Formato:

```
A   B
89012345
IDENTIFICATION DIVISION.
PROGRAM-ID.     nome-do-programa.
[AUTHOR.        nome-do-programador.]
[INSTALLATION.  nome-da-empresa.]
[DATE-WRITTEN.  data-de-codificação.]
[DATE-COMPILED. data-de-compilação  (Esta  data  o  computador
                                     fornece]
[SECURITY.      comentários do programa.]
[REMARKS.       comentários do programa(Este parágrafo é
                          utilizado somente em grande porte)]
```

2.2. Environment Division

Esta é a segunda divisão de um programa COBOL.

A `ENVIRONMENT DIVISION` está dividida em duas seções: a `CONFIGURATION SECTION` e a `INPUT-OUTPUT SECTION`.

Na `CONFIGURATION SECTION` descrevemos informações do computador e na `INPUT-OUTPUT SECTION` descrevemos informações dos arquivos.

Formato:

```
A   B
89012345
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
[SOURCE-COMPUTER.     nome-do-computador.]
[OBJECT-COMPUTER.     nome-do-computador.]
SPECIAL-NAMES         nome-de-função IS nome-simbólico]
                      DECIMAL-POINT IS COMMA.
```

```
INPUT-OUTPUT SECTION.
FILE-CONTROL.
SELECT nome-do-arquivo ASSIGN TO identificação
           [ORGANIZATION IS SEQUENTIAL]
           [ACCESS MODE IS SEQUENTIAL]
```

O parágrafo SOURCE-COPUTER especifica em qual equipamento o programa será compilado.

O parágrafo OBJECT-COMPUTER especifica em qual equipamento o programa será executado.

No parágrafo SPECIAL-NAMES podemos associar nomes de funções a alguns dispositivos COBOL, como por exemplo, o nome simbólico referente à impressora, "Printer".

Citaremos alguns dos possíveis nome-de-funções utilizados neste parágrafo:

```
SYSLST          Relacionada à impressora
CONSOLE         Relacionada a console do operador
C01             Canais associados à impressora
C02
C03
C04
C05
C06
C07
C08
C09
C10
C11
C12
```

Exemplo: `SPECIAL-NAMES.  C01 IS SALTO.`

Sempre que o nome SALTO for utilizado na PROCEDURE DIVISION, o computador executará um salto de página na impressora. O nome SALTO é simplesmente um nome-simbólico.

A cláusula DECIMAL-POINT IS COMMA tem como objetivo substituir ponto por vírgula e vice-e-versa devido a nossa tradicional notação anglo-saxônica.

Na cláusula SELECT o nome nome-do-arquivo será o nome utilizado pelo programador dentro do programa e a identificação costuma variar de acordo com o ambiente operacional.

No curso da Kyoei Facom. utilizamos o computador VIII da Edisa com o Sistema Operacional UNIX. Um exemplo de SELECT para este ambiente é:

```
SELECT EMP ASSSIGN TO "cademp"
```

Onde, EMP é o nome que o programador atribuiu ao arquivo de empresas dentro do programa, e "cademp" é o nome externo do arquivo criado num processador de textos.

A cláusula ORGANIZATION especifica em que ordem os registros estão ou estarão organizados. Existem vários tipos de organização de arquivos, porém estes serão abordados mais adiante.

Caso a cláusula não seja definida ficará subtendido como organização seqüencial, ou seja, os registros estão ou estarão em ordem seqüencial em que foram ou serão gravados.

A cláusula ACCESS MODE especifica em que ordem os registros serão acessados ou gravados.

Caso a cláusula não seja definida, o acesso será seqüencial aos registros.

Para cada arquivo utilizado no programa, devemos definir uma SELECT.

2.3. Data Division

Esta é a terceira divisão de um programa COBOL.

Esta divisão é dividida em 4 (quatro) seções: A FILE SECTION, a WORKING-STORAGE SECTION, a LINKAGE SECTION e a SCREEN SECTION.

Na FILE SECTION descrevemos os arquivos e seus respectivos registros.

Na WORKING-STORAGE SECTION definimos todas as áreas auxiliares para o processamento.

A LINKAGE SECTION e a SCREEN SECTION são seções opcionais dessa divisão.

A LINKAGE SECTION é uma seção responsável pela comunicação entre os programas.

A SCREEN SECTION é uma seção responsável pela definição de telas.

Estas seções opcionais abordaremos com mais detalhes nos próximos capítulos.

Formato:

```
A   B
89012345
DATA DIVISION.
FILE SECTION.
FD nome-do-arquivo
     ou
SD nome-do-arquivo
     [BLOCK CONTAINS n CHARACTERS
                          ou
                          RECORDS]
     [RECORD CONTAINS n CHARACTERS]
     LABEL RECORD IS OMITTED
                       ou
                    STANDARD
     [VALUE OF FILE-ID     identificação]
     [DATA RECORD IS       nome-do-registro
                 ou
                 ARE       nome-do-registro1
                           [nome-do-registro2]...]
WORKING-STORAGE SECTION.
LINKAGE SECTION.
SCREEN SECTION.
```

A cláusula FD significa File-description, ou seja, descrição do arquivo. Para cada arquivo utilizado no programa, devemos definir uma FD.

A cláusula SD significa Sort-description, ou seja, descrição do arquivo de sort. Para cada arquivo de sort utilizado no programa, devemos definir uma SD.

A cláusula BLOCK CONTAINS indica a quantidade de registros lógicos ou de caracteres que existe em cada bloco ou registro físico do arquivo.

A cláusula RECORD CONTAINS indica o tamanho geral dos registros determinando a quantidade de caracteres.

A cláusula `LABEL RECORD` indica se existe (`STANDARD`) ou não (`OMITTED`), um rótulo de identificação do arquivo correspondente.

A cláusula `VALUE OF FILE-ID` associa o nome-de-arquivo utilizado no programa com o nome de identificação do arquivo.

A cláusula `DATA RECORD` define o nome do registro do arquivo correspondente.

Na `WORKING-STORAGE SECTION`, ao descrevermos as áreas, utilizaremos vários itens que serão abordados a seguir.

2.3.1. Números de Nível

Os números de nível definem a hierarquia dos campos dentro dos registros ou a hierarquia nas áreas auxiliares criadas pelo programador.

O registro também deve ser numerado, pois ele é um item de grupo. A numeração para itens de grupo é "01".

Dentro dos itens de grupo estão os itens elementares, e estes podem receber uma numeração dentre "02" e "49".

Exemplo de codificação de um lay-out de registro:

CODIGO	NOME	DATANASC			SALARIO	ENDER
		DD	MM	AA		
PIC X(05)	X(30)	99	99	99	9(04)V99	X(30)

Formato:

```
A   B
89012345

01   REGISTRO
     05   CODIGO     PIC X(05).
     05   NOME       PIC X(30).
     05   DATANASC
          10   DD    PIC 9(02).
          10   MM    PIC 9(02).
          10   AA    PIC 9(02).
     05 SALARIO      PIC 9(04)V99.
     05 ENDER        PIC X(30).
```

PS: A cláusula PIC será abordada a seguir.

2.3.2. Números de Nível Especiais

77 e 88.

O nível 77 define auxiliares independentes, onde estes não são subdivididos e não possuem subdivisões.

Estes auxiliares são campos que criaremos com o objetivo de, por exemplo:

Precisamos emitir relatórios que deverão ter no máximo por página 60 linhas e devemos mostrar a cada página o número da página, portanto necessitamos de campos auxiliares, um para contar linhas e outro para contar páginas e estes serão criados no nível 77 dentro da `WORKING-STORAGE SECTION`.

```
77    CONTLIN          PIC 9(02)        VALUE 60.
77    CONTPAG          PIC 9(02)        VALUE 0.
```

O nível 88 define nomes de condição que devem ser associados valores definidos ao conteúdo de um campo determinado do registro.

2.3.3. Picture

`PIC` ou `PICTURE` é uma cláusula que define o tipo e o tamanho do campo. O tipo pode ser: numérico, alfanumérico, alfabético e de edição.

Numérico

É representado pelo número 9, e seu universo consiste nos dígitos de 0 a 9. Na representação de um campo numérico, podem aparecer caracteres como a letra "`V`" e/ou a letra "`S`".

"`V`" indica a posição da vírgula decimal e "`S`" indica se o campo armazena um sinal positivo ou negativo.

O tamanho máximo deste tipo de campo é 18 dígitos.

Podemos efetuar cálculos neste tipo de campo.

Exemplos:

```
PIC 9(10)
PIC 9(10)V99
```

```
PIC S9(2)
```

Alfanumérico

É representado pela letra "X", e seu universo consiste em letras do alfabeto, números de 0 a 9 e caracteres especiais.

Apesar de os números serem aceitos neste tipo de campo, não podemos efetuar cálculos no mesmo.

Exemplos:

```
PIC X(30)
```

Alfabético

É representado pela letra "A", e seu universo consiste em apenas letras do alfabeto e o caractere branco.

Exemplos:

```
PIC A(40)
```

Edição

Esta picture é utilizada na apresentação de dados na impressora ou no terminal num formato "mascarado".

Existem dois tipos de picture de edição: Picture de Edição Alfanumérica e Picture de Edição Numérica.

Picture de Edição Alfanumérica

Neste tipo de picture de edição podem ser utilizadas as letras "X", que é o caractere de representação de uma picture alfanumérica, a letra "B", que permite a inclusão de espaços em branco no campo e o número "0", que inclui zeros na posição definida na PIC.

Exemplos:

Picture	Conteúdo Original	Conteúdo Mascarado
0X(2)	AN	0AN
0X(3)BX(2)	PPPPP	0PPP PP
BBBX(4)	ABCD	ABCD

Picture de Edição Numérica

Neste tipo de picture podemos utilizar várias máscaras de formatação, porém as mais comuns são:

9 – Mostra o dígito na posição definida independente de ser “0”;

Z – Troca o dígito “0” pelo caractere BRANCO;

B – Insere o caractere BRANCO na posição definida na picture;

0 – Coloca o dígito “0” na posição definida na picture;

, – Insere uma “,” na posição definida na picture;

. – Insere um “.” na posição definida na picture;

+ – Insere o sinal “+” ou “-” à direita ou à esquerda do conteúdo editado;

- – Insere o sinal “-” à direita ou à esquerda do conteúdo editado;

* – Troca o “0” não significativo por “*” (asterisco);

$ – Este caractere pode aparecer na extremidade esquerda da picture.

Exemplos:

Picture	Conteúdo Original	Conteúdo Mascarado
ZZZ	123	123
Z.ZZ9	1900	1.900
Z.ZZ9,99	0543V55	543,55
*99,99	077V77	*77,77
9B9	66	6 6
99,00	888	88,00
+ZZ9	876	+876
99+	56 (negativo)	56-
99+	56 (positivo)	56+
+99	56 (positivo)	+56
-99	56 (negativo)	-56
99-	56 (negativo)	56-
-99	56 (positivo)	56
$9(4)	8765	$8765

2.3.4. Picture

É uma palavra reservada que preenche determinados espaços definidos na picture.

Exemplo:

```
05    FILLER     PIC X(10)       VALUE "FACOM".
```

De acordo com o exemplo acima, a palavra FACOM será preenchida nas primeiras 5 posições alfanuméricas e espaços em branco nas 5 posições restantes da picture preenchendo portanto as 10 posições. Esta palavra "FACOM" poderá ser apresentada na tela do computador ou na impressora.

2.3.5. Cláusula VALUE

É utilizada para atribuir valores em determinadas áreas de trabalho.

Podemos atribuir valores do tipo: literal-numérica ou literal-não-numérica.

Exemplos:

- Literal-numérica:

```
05    CONT       PIC 99          VALUE 60.
```

- Literal-não-numérica:

```
05    FILLER     PIC X(2)        VALUE "AB".
```

Ainda podemos utilizar os termos: SPACE(S), ZERO, ZEROS, ZEROES e ALL.

Exemplos:

```
05    FILLER     PIC X(4)        VALUE SPACES.
05    FILLER     PIC X(20)       VALUE ALL "*".
          ou definindo sem o ALL;
05    FILLER     PIC X(20)       VALUE.
05    FILLER     PIC 9(3)        VALUE ZEROS.
```

2.3.6. Exemplo de codificação da Working-storage Section

```
A    B
789012345
 WORKING-STORAGE SECTION.
 77    CONTLIN  PIC99           VALUE 60.
 77    CONTPAG  PIC99           VALUE 0.
 77    CH-FIM   PIC X(3)        VALUE "NAO".
*
 01 LINCAB.
    05  FILLER  PIC X(10)       VALUE SPACES.
    05  FILLER  PIC X(34)       VALUE "KYOEI FACO
```

```
   M - CENTRO EDUCACIONAL".
   05  FILLER   PIC X(4)        VALUE "PAG".
   05  PAG          PIC Z9.
01 LINDET.
   05  FILLER  PIC X(5)         VALUE SPACES.
   05  COD-DET PIC X(10).
   05  NOME-DET     PIC X(35).
   05  DIA-DET PIC 99.
   05  FILLER  PIC X            VALUE "/".
   05  MES-DET PIC 99.
   05  FILLER  PIC X            VALUE "/".
   05  ANO-DET PIC 99.
   05  FILLER  PIC X(10).
   05  SAL-DET PIC Z.ZZ9,99.
```

2.4. Procedure Division

Esta divisão é a quarta e última de um programa COBOL. Nela estarão definidos os comandos e instruções que irão resolver um determinado problema de processamento de dados.

Ela é composta por seções e/ou parágrafos e sentenças.

O fluxograma que foi desenvolvido deverá ser codificado nesta divisão do COBOL.

3. COMANDOS BÁSICOS

3.1. OPEN

O comando `OPEN` abre os arquivos que serão processados no programa. Ao abrirmos o arquivo, devemos especificar se este é de `INPUT` (entrada), se é de `OUTPUT` (saída) ou se é de `I-O` (entrada-saída).

Sintaxe: `OPEN INPUT nome-do-arquivo`
`OUTPUT`
`I-O`

OS: O nome-do-arquivo é o mesmo identificado na FD.

3.2. CLOSE

O comando `CLOSE` fecha os arquivos que foram abertos no programa ao final do processamento do programa.

Sintaxe: `CLOSE nome-do-arquivo`

3.3. EXIT PROGRAM

Ao desenvolvermos um sistema, este é composto por um conjunto de programas, ou seja, possui um programa mestre e a partir deste chamamos os outros programas que estarão subordinados ao mestre.

O comando `EXIT PROGRAM` é colocado ao final do processamento do programa e sua função é retornar ao programa que o chamou.

Ao desenvolvermos um programa único, ou seja, um programa independente que não esteja subordinado ao mestre, o comando `EXIT PROGRAM` não será necessário, pois existe um outro comando que faz com que o controle retorne ao Sistema Operacional chamado `STOP RUN`.

Sintaxe: `EXIT PROGRAM.`

3.4. STOP RUN

Encerra a execução do programa, devolvendo o controle ao Sistema Operacional.

Este comando é parecido com o `EXIT PROGRAM`, porém, só pode ser utilizado ao final do programa mestre ou ao final de um programa independente.

Sintaxe: `STOP RUN`

3.5. READ

O comando `READ` executa a leitura do arquivo correspondente e já aberto pelo comando `OPEN`. Somente após executarmos a leitura é que teremos acesso às informações contidas no arquivo. Este é um comando de `INPUT` (entrada).

Sintaxe 1:
```
READ nome-do-arquivo [INTO nome-de-dado]
                     AT END comando-imperativo
```

Sintaxe 2:
```
READ nome-do-arquivo [INTO nome-de-dado]
                     INVALID KEY comando-imperativo
```

A cláusula `INTO` é utilizada quando definimos a estrutura do arquivo numa área da `WORKING-STORAGE SECTION`.

Se definirmos a estrutura do arquivo na própria FD não necessitamos da cláusula `INTO`.

Na sintaxe 1 a cláusula `AT END` trata da condição de fim de arquivo de organização seqüencial e após esta, devemos dar um comando-imperativo finalizado por um ponto.

Na sintaxe 2 a cláusula `INVALID KEY` trata da condição de validação do campo-chave de um arquivo de organização indexada onde `INVALID KEY` significa chave inválida, portanto o campo-chave não existe no arquivo e o computador executará comandos-imperativos. Caso a chave seja válida, significa que o campo-chave foi localizado no arquivo e o computador não executará comandos-imperativos.

3.6. MOVE

Este comando transfere uma informação de um campo ou auxiliar para outro local da memória.

Sintaxe: `MOVE nome-de-campo1 TO nome-de-campo2...`
`constante figurativa`
`literal`
`auxiliar`

3.7. GO TO

Este comando executa um desvio na seqüência lógica do programa, porém possui maior utilização em programas lineares e modulares. Na programação estruturada o uso deste comando, não é necessário.

Sintaxe: `GO TO nome-de-procedimento`

3.8. WRITE

Este comando é utilizado para gravar ou imprimir informações. É um comando de `OUTPUT` (saída).

Na gravação:

Sintaxe: `WRITE nome-de-registro [FROM nome-de-dado]`

Na impressão:

Sintaxe: `WRITE nome-de-registro [FROM nome-de-dado]`
`AFTER ADVANCING nome-de-dado2 LINES`
`ou literal-numérica LINES`
`ou`
`BEFORE PAGE`
`ou nome-simbólico`

A cláusula `FROM` funciona como um comando `MOVE`, porém ao invés de transferir a informação da esquerda para a direita, transfere a informação da direita para a esquerda.

A cláusula `AFTER ADVANCING` permite o espaçamento do papel antes da impressão.

A cláusula `BEFORE ADVANCING` permite o espaçamento do papel depois da impressão.

A palavra reservada PAGE permite o salto de página e o nome-simbólico, se definido no SPECIAL-NAMES, pode estar relacionado a um canal de salto de página.

3.9. ADD

O comando ADD efetua soma de valores em campos.

Sintaxe 1:

```
ADD nome-de-dado1     [nome-de-dado2] ...
    literal-1         [literal-2]
    TO                nome-de-dado3   [ROUNDED]
                      [nome-de-dado4  [ROUNDED]]...
    [ON SIZE ERROR    comandos-imperativos]
```

Sintaxe 2:

```
ADD nome-de-dado1     nome-de-dado2
    literal-1         literal-2 ...
    GIVING            nome-de-dado3   [ROUNDED]
    [ON SIZE ERROR    comandos-imperativos]
```

Na sintaxe 1 todos os valores de literais e/ou nomes-de-dados serão somados e armazenados nos nomes após a cláusula TO.

Na sintaxe 2 todos os valores que antecedem a cláusula GIVING serão somandos e armazenados no nome-de-dado3 como um novo valor.

A cláusula ROUNDED, se utilizada, arredonda o valor do nome receptor.

A cláusula ON SIZE ERROR verifica se há truncamento de números no campo receptor e caso haja, processará um comando-imperativo após seu uso.

3.10. SUBTRACT

Este comando efetua subtrações entre campos.

Sintaxe 1:

```
SUBTRACT nome-de-dado1     [nome-de-dado2]
         literal-1         [literal-2]
         FROM              nome-de-dado3   [ROUNDED]
                           nome-de-dado4   [ROUNDED]...
```

```
                [ON SIZE ERROR     comandos-imperativos]
```

Sintaxe 2:

```
SUBTRACT nome-de-dado1      [nome-de-dado2]
         literal-1          [literal-2]
         FROM               nome-de-dado3
                            Literal-3
                            nome-de-dado4    [ROUNDED]
         [ON SIZE ERROR     comandos-imperativos]
```

Na sintaxe1 os valores que antecedem a cláusula `FROM` serão somados e logo em seguida serão subtraídos dos valores que estão logo após a esta cláusula.

Na sintaxe2 os valores que antecedem a cláusula `FROM` serão somados e em seguida o resultado da soma será subtraída dos valores que estão após a cláusula `FROM` e o resultado da subtração será armazenado em `nome-de-dado4` que está logo após a cláusula `GIVING`.

PS: As cláusulas `ROUNDED` e `ON SIZE ERROR` usadas no comando `SUBTRACT` possuem as mesmas funções explicadas no comando `ADD`.

3.11. MULTIPLY

Este comando efetua multiplicações entre os valores.

Sintaxe 1:

```
MULTIPLY nome-de-dado1 BY  [nome-de-dado2] [ROUNDED]
         literal-1
         [ON SIZE ERROR    comandos-imperativos]
```

Sintaxe 2:

```
MULTIPLY nome-de-dado1 BY  [nome-de-dado2]
         literal-1         literal-2
         GIVING            nome-de-dado3   [ROUNDED]
         [ON SIZE ERROR    comandos-imperativos]
```

Na sintaxe 1, `nome-de-dado1` ou `literal-1` serão multiplicados por `nome-de-dado2` e o resultado será armazenado em `nome-de-dado2`.

Na sintaxe 2, nome-de-dado1 ou literal-1 serão multiplicados por nome-de-dado2 ou literal-2 e o resultado será armazenado em nome-de-dado3.

PS: As cláusulas ROUNDED e ON SIZE ERROR usadas no comando MULTIPLY possuem as mesmas funções explicadas no comando ADD.

3.12. DIVIDE

Este comando efetua divisão entre valores.

Sintaxe 1:

```
DIVIDE nome-de-dado1 INTO  [nome-de-dado2] [ROUNDED]
         literal-1
         [ON SIZE ERROR    comandos-imperativos]
```

Sintaxe 2:

```
MULTIPLY nome-de-dado1 BY  [nome-de-dado2]
         literal-1    INTO  literal-2
         GIVING             nome-de-dado3   [ROUNDED]
         [REMAINDER nome-de-dado4]
         [ON SIZE ERROR    comandos-imperativos]
```

Na sintaxe 1 o nome-de-dado2 será dividido por nome-de-dado1 ou literal-1 e o resultado será armazenado em nome-de-dado2.

Na sintaxe 2, se a opção BY for definida, o nome-de-dado1 ou literal-1 será dividido por nome-de-dado2 ou literal-2 e o resultado será armazenado em nome-de-dado3. Se a opção INTO for definida, o nome-de-dado2 ou literal-2 será dividido por nome-de-dado1 ou literal-1 e o resultado será armazenado em nome-de-dado3.

A cláusula REMAINDER é especificada quando existe a necessidade de sabermos qual é o resto da divisão e este, de acordo com a sintaxe 2, será armazenado em nome-de-dado4.

PS: As cláusulas ROUNDED e ON SIZE ERROR usadas no comando DIVIDE possuem as mesmas funções explicadas no comando ADD.

3.13. COMPUTE

Este comando efetua qualquer operação aritmética. Sua sintaxe é mais simplificada, porém o tempo da máquina para efetuar cálculos é maior.

Sintaxe:

```
COMPUTE nome-de-dado [ROUNDED] = operação-aritmética
        [ON SIZE ERROR     comandos-imperativos]
```

O resultado do calculo será armazenado em `nome-de-dado`.

Na operação-aritmética necessitaremos de operadores matemáticos como:

`+` – adição

`-` – subtração

`*` – multiplicação

`/` – divisão

`**` – exponenciação

Os operadores e o sinal de igual (=) deverão ser precedidos e seguidos no mínimo de um espaço em branco.

As operações serão executadas na mesma hierarquia da matemática.

Caso haja a necessidade de priorizar alguma operação, podemos utilizar parêntesis.

3.14. IF

O comando `IF` significa "SE". Quando necessitamos incluir condições no programa podemos utilizá-lo.

Sintaxe 1: (`IF` simples)

```
IF condição
     comandos-1.
comandos-2.
```

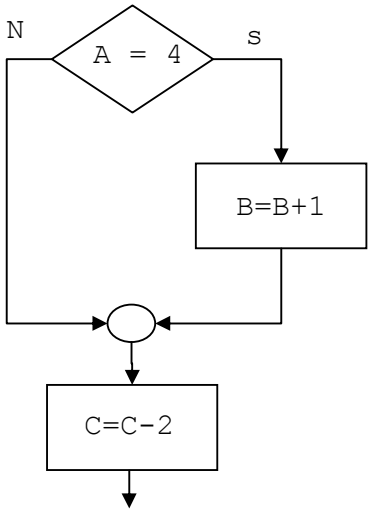
Quando a condição é feita, o computador executará `comandos-1` se a resposta for afirmativa e logo em seguida executará `comandos-2`. Se a resposta for negativa, o computador procurará o primeiro ponto (`.`) dentro da condição e executará o que vem em seguida que é `comandos-2`. Na codificação de uma condição, por motivos de estética, a

resposta afirmativa sempre codificamos em colunas mais a direita do IF, e a resposta negativa na mesma coluna do IF, porém nas linhas seguintes.

Exemplo:

```
IF A=4
      ADD 1 TO B.
SUBTRACT 2 FROM C.
```

ou seja:



Se A=4, o computador executará a adição de 1 em B e em seguida executará a subtração de 2 em C.

Se A for diferente de 4, o computador irá procurar o primeiro ponto (.) dentro da condição e executará a subtração de 2 em C.

Sintaxe 2: (IF THEN ELSE)

```
IF condição
   THEN
      comandos-1
      [NEXT SENTENCE]
   ELSE
      comandos-2
      [NEXT SENTENCE].
instruções
```

Quando a resposta da condição for afirmativa, o computador executará comandos-1 ou NEXT SENTENCE (*) que fazem parte do THEN e logo em seguida irá procurar o primeiro ponto (.) da condição e executar as instruções que estiverem logo após ele. A palavra THEN é

opcional. Se a resposta da condição for negativa, o computador executará `comandos-2` ou `NEXT SENTENCE` (*) que fazem parte do `ELSE` e logo em seguida irá procurar o primeiro ponto (.) da condição e executar as instruções que estiverem logo após dele.

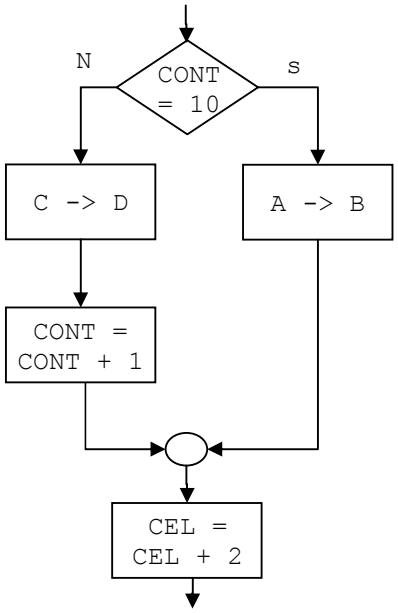
(*) => O `NEXT SENTENCE` é uma palavra que utilizamos quando não temos comandos a executar na resposta afirmativa ou negativa de um `IF THEN ELSE`. Quando o computador encontra um `NEXT SENTENCE`, ele passa para a primeira instrução que estiver logo após o primeiro ponto (.) encontrado dentro da condição.

PS: Podemos encadear vários `IFs` simples ou `IFs THEN ELSE` porém devemos prestar muita atenção com os pontos (.). Uma das regras é que nunca podemos colocar pontos (.) nos comandos que estiverem dentro do `THEN`.

Exemplo:

```
IF CONT = 10
   THEN
      MOVE A TO B
   ELSE
      MOVE C TO D
      ADD 1 TO CONT.
MULTIPLY 2 BY CEL
```

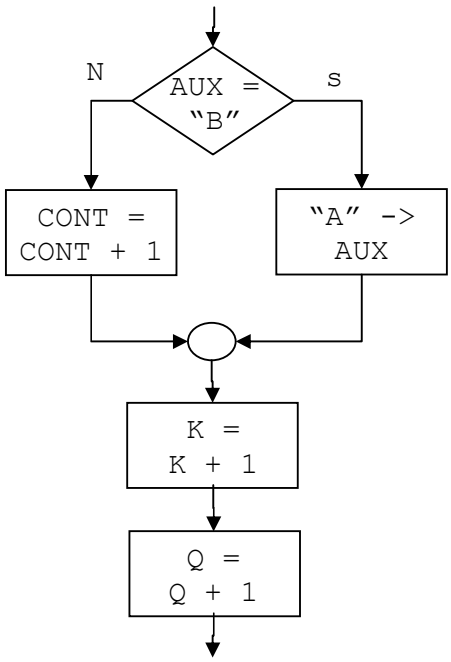
ou seja:



Exemplo 2:

```
IF AUX = ”B”
   THEN
      MOVE A TO AUX
   ELSE
      ADD 1 TO CONT.
ADD 1 TO K
ADD 1 TO Q.
```

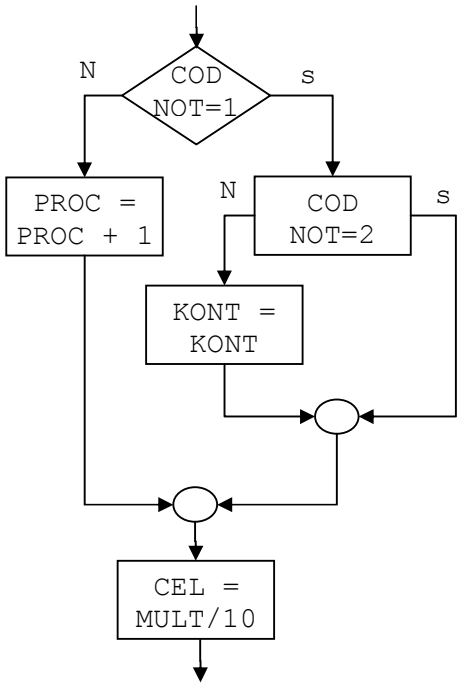
ou seja:



Exemplo 3:

```
IF COD NOT = 1
   THEN
      IF COD NOT = 2
         THEN
            NEXT SETENCE
         ELSE
            ADD 1 TO KONT
   ELSE
      ADD 1 TO PROC.
DIVIDE MULT BY 10 GINVING CEL.
```

ou seja:



3.14.1. Condição de Classe

Existem apenas dois tipos: Numérico e Alfabético.

Sintaxe:

```
IF nome-de-dado is [NOT]   NUMERIC
                             ou
                           ALPHABETIC
```

3.14.2. Condição de Sinal

Existem três tipos: Negative, Positive e Zero.

Sintaxe:

```
IF nome-de-dado is [NOT]   NUMERIC
                           NEGATIVE
                           ZERO
```

3.14.3. Condição de Relação

Verifica a relação entre os operandos.

Sintaxe:

```
IF expressão-1 is [NOT]    GREATER [THAN] ou > expressão-2
                           LESS [THAN]    ou <
                           EQUAL [TO]     ou =
```

3.14.4. Nome de Condição

Este tipo de nome é utilizado para simplificar a codificação. O item independente nível 88, que é definido na DATA DIVISION forma um nome de condição. Exemplo:

```
...
DATA DIVISION.
FILE SECTION.
FD  ARQ
    LABEL RECORD      IS STANDARD
    DATA RECORD       IS REGITRO.
01 REGISTRO.
   05  COD            PIC X(5).
   05  NOME           PIC X(30).
   05  DATANASC.
       10   DIA            PIC X(2).
       10   MES            PIC X(2).
       10   MÊS-OK         VALUE "01" THRU "12".
       10   ANO            PIC X(2).
...
PROCEDURE DIVISION.
...
    IF MÊS-OK
       instruções
    ELSE
       DISPLAY "MES INVALIDO" AT 0101.
          ou
    IF MÊS-OK
       DISPLAY "MES INVALIDO" AT 0101.
    ELSE
       Instruções.
```

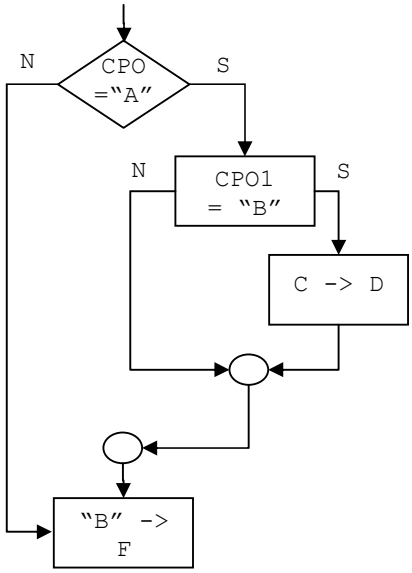
Se o nível 88 não fosse utilizado, a pergunta seria:

```
    IF MES > "00" AND MES < "13" (Mês válido)
                     ou
    IF MES < "01" OR MES > "12" (Mês inválido)
```

3.14.5. Condições Compostas

São várias perguntas num só comando `IF`. Este processo se torna possível com o uso das palavras `AND` e `OR`. Exemplos:

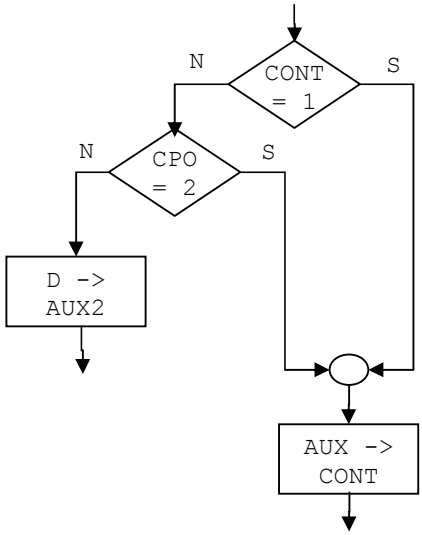
- AND



ou seja:

```
IF CPO = "A" AND CPO1 = "B"
     MOVE C TO D.
MOVE "B" TO F.
```

- OR



ou seja:

```
IF CONT = 1 OR CPO = 2
   THEN
     MOVE AUX TO CONT
   ELSE
     MOVE D TO AUX2
```

AND

V	V	V
V	F	F
F	V	F
F	F	F

OR

V	V	V
V	F	V
F	V	V
F	F	F

3.15. PERMORM

Este comando provoca um desvio no fluxo do programa a fim de executar as instruções contidas numa rotina chamada pelo comando `PERFORM`. Ao final da rotina o processamento retorna ao fluxo normal do programa, para a instrução que vier logo após o comando `PERFORM`.

A diferença que existe entre os comandos `PERFORM` e `GO TO` é que, no `PERFORM` o computador sai do fluxo do programa para executar a rotina que o comando chama, executa as instruções da rotina e ao final retorna para o fluxo do programa para a instrução seguinte ao `PERFORM` executado, e no comando `GO TO` o computador executa um desvio no programa para onde o `GO TO` especificou.

Existem várias sintaxes deste comando.

Sintaxe 1:

```
PERFORM rotina-1
```

O computador procura o nome de parágrafo "`rotina-1`", encontrando, executa as instruções que estão dentro da rotina e ao final desta, retorna para o fluxo do programa para instrução seguinte ao comando `PERFORM` executado.

Sintaxe 2:

```
PERFORM rotina-1 TRHU rotina-2
```

O computador procura a "rotina-1" e executa as instruções e rotinas que estiverem dentro desta até encontrar o nome de parágrafo "rotina-2". Ao encontrar "rotina-2", o computador retorna o fluxo normal do programa para a instrução seguinte ao comando PERFORM executado.

Sintaxe 3:

```
PERFORM rotina-1 UNTIL chave=0
```

O computador procura "rotina-1" e processa as instruções que estão dentro desta até que chave=0. Forma-se portanto um LOOP.

Sintaxe 4:

```
PERFORM rotina-1 VARYING campo-1 FROM 1 BY 1 UNTIL
        campo-2 > 5
```

Este tipo de PERFORM geralmente é utilizado para fazer pesquisas em tabelas. O computador procura "rotina-1" e ao encontrar fica um LOOP, executando as instruções internas variando campo-1 e este começa com 1 e vai sendo acrescentado de 1 até que campo-2>5.

O LOOP é finalizado quando campo-2>5 e o computador retorna ao fluxo normal do programa.

3.16. CALL

Este comando transfere a execução de um programa para outro.

Sintaxe:

```
CALL subprog
```

3.17. CANCEL

Remove da memória a sub-rotina chamada, libertando-a para outras rotinas.

Sintaxe:

```
CANCEL subprog
```

3.18. CHAIN

Este comando transfere a execução de um programa para outro programa definitivamente.

Sintaxe:

```
CHAIN subprog
```

3.19. COPY

Copia conteúdo de arquivos do tipo texto para o programa fonte em uso.

Sintaxe:

```
COPY arquivo-texto
```

3.20. DELETE

Este comando remove o registro em uso do arquivo que está sendo lido.

Sintaxe:

```
DELETE arquivo
```

3.21. ACCEPT

Este comando é utilizado para se obter e preencher valores em variáveis na tela, determinando ou não, linha e coluna.

Sintaxe 1:

```
ACCEPT variável AT LC
```

Sintaxe 2:

```
ACCEPT nome-de-dado FROM nome-simbólico
                         registrador-especial
```

Na sintaxe 1 podemos mostrar e entrar com valores na variável numa determinada linha e coluna da tela.

Na sintaxe 2 podemos transferir informações de um nome-simbólico definido em SPECIAL NAMES ou registrador-especial para um nome-de-dado.

Exemplo:

```
ACCEPT DATA-AUX FROM DATE
```

3.22. DISPLAY

Este comando exibe informações na tela, definindo ou não as linhas e colunas.

Sintaxe 1:

```
DISPLAY nome-de-dado AT LC literal
```

Sintaxe 2:

```
DISPLAY nome-de-dado [UPON nome-simbólico]
        literal
```

Na sintaxe 1 podemos exibir o conteúdo do `nome-de-dado` ou literal numa determinada linha e coluna da tela.

Na sintaxe 2 podemos exibir o conteúdo do `nome-de-dado` ou literal, direcionando para determinados dispositivos através da cláusula `UPON`.

3.23. REWRITE

Este comando é utilizado para regravar registros de arquivos com organização indexada.

Sintaxe:

```
REWRITE registro
```

3.24. SORT

Este comando cria um arquivo de SORT, definido na SD, organizado em ordem ascendente ou decrescente a partir do arquivo em uso.

Sintaxe:

```
SORT arquivo-de-sort [ASCENDING KEY campo-chave
                      DESCENDING KEY]
     INPUT PROCEDURE rot-vai-classificar
     OUTPUT PROCEDURE rot-ja-classificou
```

Na rotina "`rot-vai-classificar`", que faz parte da `INPUT PROCEDURE`, os registros do arquivo em uso serão gravados na ordem definida no comando SORT no "`arquivo-de-sort`".

Na rotina "`rot-ja-classificou`", que faz parte da `OUTPUT PROCEDURE`, os registros do `arquivo-de-sort` que já estão classificados serão gravados ou impressos num outro arquivo, sendo este de saída.

3.25. RELEASE

Este comando transfere os registros lidos no arquivo de entrada para o arquivo de SORT, organizando e gravando no arquivo de SORT.

Sintaxe:

```
RELEASE regsort
```

3.26. RETURN

Este comando lê os registros do arquivo de SORT.

Sintaxe:

```
RETURN arqsort AT END [comandos-imperativos]
```

3.27. Exemplo de Codificação de Programa

1) Proposta

Ler os registros de um arquivo de alunos e gravar num arquivo de alunos aprovados os alunos que possuem média > ou = a 7 (sete).

2) Lay-outs

ARQALU

	COD	NOME	NOTA1	NOTA2	TURMA
PIC	X(3)	X(30)	99V99	99V99	X(5)

ARQAPROV

	COD	NOME	MEDIA	TURMA
PIC	X(3)	X(30)	99V99	X(5)

3) Procedimentos

3.1) Ler o ARQALU.

3.2) Selecionar os alunos que possuem MEDIA > ou = 7,0.

3.3) Gravar em ARQAPROV os alunos selecionados.

3.4) Terminar execução ao final do arquivo de alunos.

```
       A   B       C                D                E     F
12345678901234567890123456789012345678901234567890123456789012345678
001010 IDENTIFICATION DIVISION.
        PROGRAM-ID.                 PROGEXEM.
        AUTHOR.                     ANA CLAUDIA.
        INSTALLATION.               KYOEI FACOM.
        DATE-WRITTEN.               10/03/95.
        DATE-COMPILED.
        SECURITY.
      **************************************************
      * Lê registros de ARQALU, seleciona alunos
      * com média e grava em ARQAPROV.
      **************************************************
        ENVIRONMENT DIVISION.
        CONFIGURATION SECTION.
        SOURCE-COMPUTER.            VIII-EDISA.
        OBJECT-COMPUTER.            VIII-EDISA.
        SPECIAL-NAMES.              DECIMAL-POINT IS COMMA.

        INPUT-OUTPUT SECTION.
        FILE-CONTROL.
002010       SELECT ARQUALU         ASSIGN TO "arqalu".
             SELECT ARQAPROV        ASSIGN TO "arqaprov".

        DATA DIVISION.
        FILE SECTION.
        FD  ARQALU
            LABEL    RECORD         IS STANDARD
            DATA     RECORD         IS REGALU.
        01  REGALU.
            05  COD-ENT         PIC X(3).
            05  NOME-ENT        PIC X(30).
            05  NOTA1-ENT       PIC 99V99.
            05  NOTA2-ENT       PIC 99V99.
            05  TURMA-ENT       PIC X(5).
        FD  ARQAPROV
            LABEL    RECORD         IS STANDARD
```

```
    DATA     RECORD         IS REGAPROV.
01  REGAPROV.
    05  COD-SAI         PIC X(3).
    05  NOME-SAI        PIC X(30).
    05  MEDIA-SAI       PIC 99V99.
    05  TURMA-SAI       PIC X(5).
WORKING-STORAGE SECTION.
77  CH-FIM
77  AUXMED
PROCEDURE DIVISION.
MODULO-MESTRE.
    PERFORM INICIALIZA
    PERFORM PROCESSA UNTIL CH-FIM = "SIM"
    PERFORM FINALIZA
    STOP RUN.
INICIALIZA.
    PERFORM ABERTURA.
    PERFORM LEITURA.
ABERTURA.
    OPEN INPUT ARQALU
         OUTPUT ARQPROV.
LEITURA.
    READ ARQALU AT END
         MOVE "SIM" TO CH-FIM.
PROCESSA.
    PERFORM CALCULA
    PERFORM TESTA.
CALCULA.
    COMPUTE AUXMED = ( NOTA1-ENT + NOTA2-ENT) /2.
TESTA.
    IF AUXMED NOT < 7
       PERFORM GRAVA.
    PERFORM LEITURA.
GRAVA.
    MOVE COD-ENT TO COD-SAI
    MOVE NOME-ENT TO NOME-SAI
    MOVE AUXMED-ENT TO MEDIA-SAI
```

```
    MOVE TURMA-ENT TO TURMA-SAI
    WRITE REGAPROV.
FINALIZA.
    CLOSE ARQALU ARQAPROV.
```

4. TELAS

4.1. Conceito

O uso de tela num programa permite uma comunicação direta entre usuário e sistema.

A tela do monitor do computador possui 24 linhas e 80 colunas.

Para podermos mostrar ou entrar com informações na tela devemos definir linha e coluna.

4.2. Exibir Informações

A exibição das informações na tela é feita pelo comando `DISPLAY`.

Existem três métodos de definição de tela.

Podemos definir a tela numa seção da `DATA DIVISION` chamada, `SCREEN SECTION`, ou numa outra seção da `DATA DIVISION` chamada, `WORKING-STORAGE SECTION` e ainda podemos definir a tela usando somente o próprio comando `DISPLAY` dentro da `PROCEDURE DIVISION`.

Adotaremos o ultimo método para exemplificar um programa de tela.

A sintaxe do comando `DISPLAY` se encontra no capítulo 3.

4.3. Inserir Informações

A inclusão de informações num arquivo através da tela é feita com o comando `ACCEPT`.

Este comando busca a informação inicial da variável e mostra na tela permitindo a alteração de seu conteúdo.

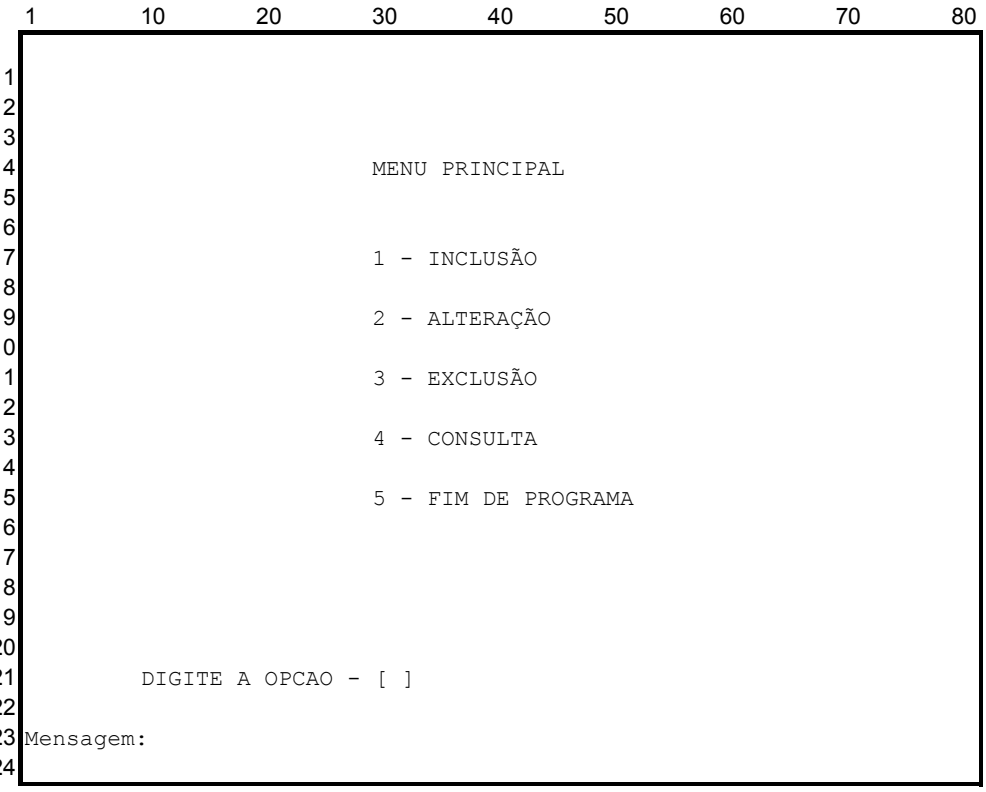
A sintaxe do comando `ACCEPT` se encontra no capítulo 3.

4.4. Exemplo de Codificação de um Programa de Tela

4.4.1. Proposta

Elaborar um programa que permita executar programas subordinados através da escolha de opções na tela do computador.

4.4.2. Lay-out



4.4.3. Procedimentos

1) Limpar a tela.
2) Exibir lay-out.
3) Escolher o programa a executar ou fim de programa, digitando uma opção.
4) Emitir mensagem de erro, caso opção inválida.

```
       A   B       C               D                E     F
12345678901234567890123456789012345678901234567890123456789012345678
001010 IDENTIFICATION DIVISION.
        PROGRAM-ID.                PROGTELA.
        AUTHOR.                    ANA CLAUDIA.
        INSTALLATION.              KYOEI FACOM.
        DATE-WRITTEN.              10/03/95.
        DATE-COMPILED.
        SECURITY.
      ***************************************************
      * Seleciona programas ou final de execução numa tela
```

```
      ***************************************************
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER.            VIII-EDISA.
       OBJECT-COMPUTER.            VIII-EDISA.
       SPECIAL-NAMES.              DECIMAL-POINT IS COMMA.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 OPCAO                     PIC X(1) VALUE SPACES.
       77 CH-FIM                    PIC X(3) VALUE "NAO".
002010 77 W-LIMPA                   PIC X(50) VALUE SPACES.
       PROCEDURE DIVISION.
       MODULO-MESTRE.
           PERFORM INICIALIZA
           PERFORM PROCESSA UNTIL CH-FIM = "SIM"
           PERFORM FINALIZA
           STOP RUN.
       INICIALIZA.
```


```
       PROCESSA.
```


```
       IF OPCAO = "1"
               THEN
```


```
               ELSE
                 IF OPCAO = "2"
                    THEN
```


```
                    ELSE
                      IF OPCAO = "3"
                         THEN
```

```
                CHAMA PROGRAMA DE EXCLUSAO, EXECUTA
                E CANCELA EXECUCAO DESTE
                    ELSE
                      IF OPCAO = "4"
                         THEN
                CHAMA PROGRAMA DE CONSULTA, EXECUTA
                E CANCELA EXECUCAO DESTE
                              ELSE
                                IF OPCAO = "5"
                                   THEN
                                   SAI DO LOOP
                                   ELSE
                                   OPCAO INVALIDA
      MOVE SPACES TO OPCAO.
FINALIZA.
    DISPLAY SPACES AT 0000.
```

5. ORGANIZAÇÃO E MÉTODOS DE ACESSO AOS ARQUIVOS

Existem 3 (três) tipos de organização e arquivos: Seqüência, Indexada e Relativa.

Em cada tipo de organização podemos ter diferentes tipos de acesso aos registros.

5.1. Arquivos com Organização Seqüencial

Neste tipo de organização de arquivo, os registros estão dispostos na ordem da digitação dos dados no processador de textos.

O método de acesso neste tipo de arquivo é exclusivamente seqüencial.

5.1.1. Exemplo de Programa Codificado

5.1.1.1. Proposta

Elaborar um programa que faca a leitura de um arquivo de organização seqüencial e imprima num relatório algumas informações.

5.1.1.2. Lay-outs

ARQEMP:

CODFUNC	NOME	DATADM			ENDER	SALARIO
		DD	MM	AA		
PIC X(7)	X(20)	99	99	99	X(20)	9(04)V99

RELATORIO:

```
  1        10        20        30        40        50        60        70        80
  1   RELACAO DOS FUNCIONÁRIOS DA EMPRESA                          PAG. ZZ9
  2
  3
  4 CODIGO    NOME           ENDERECO                   DATA ADM.   SALARIO
  5
  6 x-----x x--------------x x-------------x           99/99/99   Z.ZZ9,99
  7
  8
  9
 10
 11
 12
 13
 14
 15
 16
 17
 18
 19
 20
 21 ------------------------( NOVA PAGINA )------------------------
 22
 23   TOTAL DE FUNCIONARIOS DA EMPRESA: Z.ZZ9
 24
 25
 26
 27
 28
 29
 30
```

5.1.1.3. Procedimentos

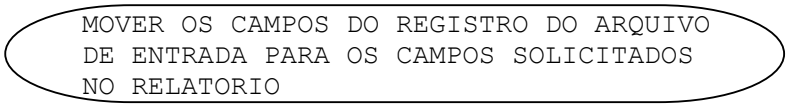
1) Ler ARQEMP.
2) Imprimir registros conforme lay-out no arquivo RELATORIO (60 linhas por folha).
3) Ao final, imprimir a quantidade de funcionários da empresa em nova página e terminar a execução.

```
      A   B       C                D                E     F
12345678901234567890123456789012345678901234567890123456789012345678
001010 IDENTIFICATION DIVISION.
       PROGRAM-ID.                PROGSEQ.
       AUTHOR.                    ANA CLAUDIA.
       INSTALLATION.              KYOEI FACOM.
```

```
       DATE-WRITTEN.              10/03/95.
       DATE-COMPILED.
       SECURITY.
      ***************************************************
      * Imprime registros de um arquivo de organização
      * seqüencial e acesso seqüencial.
      ***************************************************
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER.           VIII-EDISA.
       OBJECT-COMPUTER.           VIII-EDISA.
       SPECIAL-NAMES.             DECIMAL-POINT IS COMMA.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
            SELECT ARQEMP         ASSIGN TO "arqemp".
002010      SELECT RELAT          ASSIGN TO "rel".
       DATA DIVISION.
       FILE SECTION.
       FD ARQEMP
          DEFINIR A DESCRICAO DO ARQUIVO DE EMPRESA
       FD ARQAPROV
          DEFINIR A DESCRICAO DO ARQUIVO DE RELATORIOS
003010 WORKING-STORAGE SECTION.
       77  CONTLIN                PIC 99 VALUE 60.
       77  CH-FIM                 PIC X(3) VALUE "NAO".
       77  CONTPAG                PIC 9(3) VALUE 0.
       77  CONTFUNC               PIC 9(4) VALUE 0.
       01  LINCAB1.
           05  FILLER             PIC X(6) VALUE SPACES.
           05  FILLER             PIC X(55) VALUE "RELACAO
           DE FUNCIONARIOS DA EMPRESA".
           05  FILLER             PIC X(5) VALUE "PAG.".
           05  PAG                PIC ZZ9.
       01  LINCAB2.
           05  FILLER             PIC X(2) VALUE SPACES.
```

```
            05  FILLER             PIC X(8) VALUE "CODIGO".
            05  FILLER             PIC X(21) VALUE "NOME".
            05  FILLER             PIC X(21) VALUE
                                   "ENDERECO".
            05  FILLER             PIC X(10) VALUE
                                   "DATA ADM.".
004010      05  FILLER             PIC X(7) VALUE
                                   "SALARIO".
        01  LINDET.
            05  FILLER             PIC X(2) VALUE SPACES.
            05  CODFUNC-DET        PIC X(8).
            05  NOME-DET           PIC X(21).
            05  ENDER-DET          PIC X(21).
            05  DD-DET             PIC 99.
            05  FILLER             PIC X(1) VALUE "/".
            05  MM-DET             PIC 99.
            05  FILLER             PIC X(1) VALUE "/".
            05  AA-DET             PIC 99.
            05  FILLER             PIC X(2) VALUE SPACES.
            05  SALARIO-DET        PIC Z.ZZ9,99.
        01  LINTOT.
            05  FILLER             PIC X(1) VALUE SPACES.
            05  FILLER             PIC X(34) VALUE "TOTAL
                                   DE FUNCIONARIOS DA
                                   EMPRESA:".
            05  TOT                PIC Z.ZZ9.
005010  PROCEDURE DIVISION.
        MODULO-MESTRE.
            PERFORM INICIALIZA
            PERFORM PROCESSA UNTIL CH-FIM = "SIM"
            PERFORM FINALIZA
            STOP RUN.
        INICIALIZA.
            PERFORM ABERTURA.
            PERFORM LEITURA.
        ABERTURA.
            OPEN INPUT ARQEMP
```

```
IMPRIMIR REGIMP                OUTPUT RELAT.
        LEITURA.
            READ ARQEMP AT END
                 MOVE "SIM" TO CH-FIM.
        PROCESSA.
            PERFORM CARREGA
            PERFORM IMPRIME
            PERFORM LEITURA.
        CARREGA.
```




```
        IMPRIME.
            IF CONTLIN > 59
                 PERFORM CABEC.
            PERFORM DETALHE.
        CABEC.
```


```
007010
        DETALHE.
```


```
        FINALIZA.
            PERFORM TOTALIZA
            CLOSE ARQEMP RELAT.
        TOTALIZA.
            MOVE CONTFUNC TO TOT
            WRITE REGIMP FROM LINTOT AFTER PAGE.
```

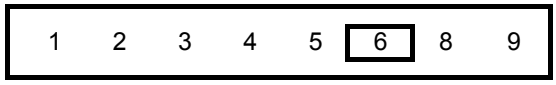
5.2. Arquivos com Organização Indexada

Neste tipo de organização de arquivo, cada registro deverá conter um campo chave que servirá como um endereço de localização do registro.

Os métodos de acesso neste tipo de arquivo são: seqüencial (Sequential), aleatório (Random) ou dinâmico (Dynamic).

No acesso seqüencial (Sequential), os registros são dispostos sequencialmente em ordem crescente do conteúdo da chave.

Ao procurarmos um conteúdo de chave no arquivo, e este conteúdo existindo, o acesso a ele é direto. Exemplo:



Se precisamos acessar o registro correspondente à chave (6) do arquivo, não é necessário ler sequencialmente todos os registros.

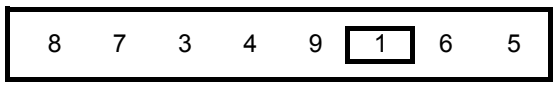
Na `INPUT-OUTPUT SECTION`, especificamente na `FILE-CONTROL`, definimos um arquivo indexado-seqüencial da seguinte maneira:

```
SELECT ARQUIND ASSIGN TO "arqind".
       ORGANIZATION IS      INDEXED
       ACCESS MODE IS       SEQUENTIAL
       RECORD KEY IS        CHAVE.
```

A cláusula `RECORD KEY` é a responsável pela identificação do campo que será a chave do registro. Este campo deverá estar definido dentro da `DATA DIVISION` na definição do registro do arquivo em uso.

No acesso aleatório (Random), os registros são dispostos numa ordem aleatória do conteúdo da chave.

Ao procurarmos um conteúdo de chave no arquivo, e este conteúdo existindo, o acesso a ele é direto. Exemplo:



Se precisamos acessar o registro correspondente à chave (1) do arquivo, não é necessário ler seqüencialmente todos os registros.

Na `INPUT-OUTPUT SECTION`, especificamente na `FILE-CONTROL`, definimos um arquivo indexado-randômico da seguinte maneira:

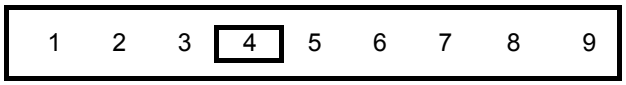
```
SELECT ARQUIND ASSIGN TO "arqind".
```

```
        ORGANIZATION IS      INDEXED
        ACCESS MODE IS       RANDOM
        RECORD KEY IS        CHAVE.
```

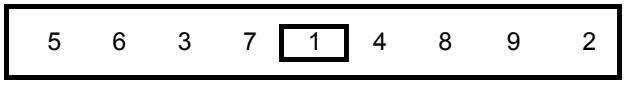
No acesso dinâmico, os registros podem estar dispostos numa ordem seqüencial e aleatória do conteúdo da chave dentro de um arquivo no mesmo programa.

Ao procurarmos um conteúdo de chave no arquivo, e este conteúdo existir, o acesso a ele é direto. Exemplo:

Seqüencial:



Aleatório:



Na INPUT-OUTPUT SECTION, especificamente na `FILE-CONTROL` definimos um arquivo indexado-dinâmico da seguinte maneira:

```
SELECT ARQUIND ASSIGN TO "arqind".
       ORGANIZATION IS      INDEXED
       ACCESS MODE IS       DYNAMIC
       RECORD KEY IS        CHAVE.
```

Os arquivos de organização indexada podem ser de entrada (`INPUT`), de saída (`OUTPUT`) ou ainda de entrada e saída (`I-O`) de informações.

Quando o arquivo é de `I-O` significa que já possuímos registros gravados neste arquivo. No caso da necessidade de excluir um registro selecionado através de sua chave, devemos utilizar o comando `DELETE`. No caso da necessidade de alterar algum campo do registro selecionado através da chave, após a alteração, precisaremos regravar o registro e, portanto devemos utilizar o comando `REWRITE`.

PS: Os comandos `DELETE` e `REWRITE` estão definidos no Capítulo 3.