



# C O B O L

## Índice

1	Sistema Operacional .....	5
2	Programa de Processamento .....	5
3	Fluxo de compilação .....	5
4	C O B O L .....	6
4.1	Vantagens do COBOL .....	6
4.2	Divisões .....	6
4.2.1	Divisão de Identificação – IDENTIFICATION DIVISION .....	6
4.2.2	Divisão de Ambiente – ENVIRONMENT DIVISION .....	6
4.2.3	Divisão de Dados – DATA DIVISION .....	6
4.2.4	Divisão de Processamento – PROCEDURE DIVISION .....	6
5	Descrição da Folha de Programação .....	7
6	Conjunto de caracteres válidos .....	8
7	Regras de formatação .....	9
8	Regras de pontuação .....	9
9	Palavras reservadas .....	9
10	Identification Division .....	13
11	Environment Division .....	14
11.1	Configuration Section .....	14
11.2	Input-Output Section .....	15
11.2.1	File-Control .....	15
11.2.1.1	Select .....	16
11.2.2	I-O-Control .....	17
12	Data Division .....	20
12.1	File Section .....	20
12.1.1	Indicadores de níveis: .....	20
12.1.1.1	Nível FD .....	21
12.1.1.1.1	Block Contains .....	21
12.1.1.1.2	Record Contains .....	21
12.1.1.1.3	Recording Mode .....	21
12.1.1.1.4	Label Record .....	22
12.1.1.1.5	Data Record .....	22
12.1.2	Número de níveis .....	23
12.1.2.1	Número de níveis especiais .....	23
12.1.2.1.1	Nível 66 .....	23
12.1.2.1.2	Nível 77 .....	24
12.1.2.1.3	Nível 78 .....	24
12.1.2.1.4	Nível 88 .....	24
12.1.3	Descrição do registro .....	25
12.1.3.1	Picture - PIC .....	25
12.1.3.1.1	Formato Alfabético .....	25
12.1.3.1.2	Formato Alfanumérico .....	26
12.1.3.1.3	Formato Numérico .....	26
12.1.3.1.4	Formato de Edição .....	27
12.1.3.2	Blank when zero .....	29
12.1.3.3	Filler .....	30
12.2	Working-Storage Section .....	30
12.2.1	Value .....	31
12.2.2	Computational .....	31
12.2.2.1	Comp ou Comp-4 (Binário) .....	32
12.2.2.2	Comp-1 ou Comp-2 (Ponto flutuante) .....	32
12.2.2.3	Comp-3 (Compactado) .....	33
12.2.3	Justified .....	34
12.2.4	Redefines .....	35
12.2.5	Constantes figurativas .....	36
12.2.6	Renames .....	36

12.3	Linkage Section.....	38
12.4	Communication Section .....	39
12.5	Report Section .....	39
13	Procedure Division .....	40
13.1	Comandos para manipulação de arquivos.....	41
13.1.1	Close.....	41
13.1.2	Delete.....	42
13.1.3	Open .....	44
13.1.4	Read .....	45
13.1.5	Rewrite.....	47
13.1.6	Start .....	48
13.1.7	Write.....	49
13.2	Comandos aritméticos .....	50
13.2.1	Add.....	51
13.2.2	Compute .....	53
13.2.3	Divide.....	54
13.2.4	Multiply.....	56
13.2.5	Subtract.....	57
13.3	Comandos de decisões.....	59
13.3.1	IF.....	59
13.3.1.1	Testes compostos.....	60
13.3.1.1.1	Teste de classe.....	61
13.3.1.1.2	Teste de nome de condição.....	62
13.3.1.1.3	Teste de relação condicional .....	63
13.3.1.1.4	Teste de sinal.....	64
13.3.1.1.5	Teste de condição composta .....	65
13.3.1.2	Concatenação de IF (ninho de IF) .....	67
13.3.2	Evaluate .....	68
13.4	Comandos Básicos .....	73
13.4.1	Accept .....	73
13.4.2	Alter.....	74
13.4.3	Continue.....	75
13.4.4	Display .....	75
13.4.5	End program .....	76
13.4.6	Examine .....	77
13.4.7	Exhibit .....	80
13.4.8	Exit .....	81
13.4.9	Go to .....	82
13.4.10	Goback.....	83
13.4.11	Initialize .....	84
13.4.12	Inspect .....	87
13.4.13	Move .....	91
13.4.14	On .....	94
13.4.15	Perform .....	95
13.4.16	Ready / Reset .....	98
13.4.17	Stop run .....	99
13.4.18	String.....	100
13.4.19	Synchronized .....	102
13.4.20	Transform.....	103
13.4.21	Unstring.....	104
13.5	Comandos para comunicação entre programas .....	107
13.5.1	Call.....	107
13.5.2	Cancel.....	108
13.5.3	Chain.....	109
13.5.4	Exit Program .....	110
13.5.5	Exemplo de comunicação entre programas.....	111
14	Processamento de arquivos de organização indexada.....	113

15	Tabelas internas.....	115
15.1	Tipo de tabelas.....	115
15.1.1	Identidade .....	115
15.1.2	Não identidade .....	116
15.2	Dimensões de tabelas.....	116
15.2.1	Unidimensional.....	116
15.2.2	Bidimensional.....	118
15.2.3	Tridimensional.....	119
15.3	Comandos de tabelas .....	120
15.3.1	Occurs.....	120
15.3.2	Search.....	122
15.3.3	Set.....	125
16	Tratamento de Impressão .....	126
16.1	Advancing .....	126
16.2	Positioning .....	126
17	File status.....	128
18	Comandos do Sort / Merge .....	132
18.1	Definição na ENVIRONMENT DIVISION.....	132
18.2	Definição na DATA DIVISION.....	133
18.3	Sort .....	134
18.4	Merge.....	135
18.5	Release.....	135
18.6	Return .....	136

## 1 Sistema Operacional

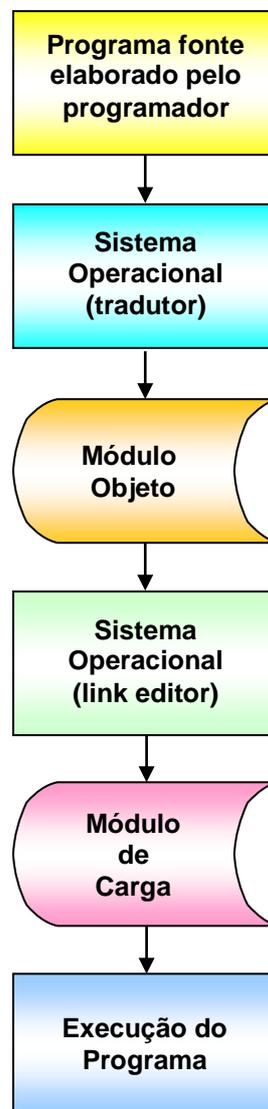
O Sistema operacional são programas de controle que supervisionam os Sistemas, Arquivos e Jobs.

## 2 Programa de Processamento

É composto de:

- Tradutores de linguagens;
- Programas de serviços (utilitários); e;
- Programas de aplicação (COBOL, ASSEMBLER etc)

## 3 Fluxo de compilação



## 4 C O B O L

**Common Business Oriented Language**, é um subconjunto de palavras da língua inglesa, ou seja, um número limitado de palavras inglesas, sujeitas a uma sintaxe própria. É uma linguagem que lida com problemas comerciais, envolvendo arquivos de dados de apreciáveis proporções.

Como segue o padrão nacional americano, é conhecido como ANS (American National Standard)

### 4.1 Vantagens do COBOL

- Independência do tipo do equipamento (IBM, Borroughs, Honeywell etc).
- Facilidade de aprendizado;
- Boa documentação dos programas; e;
- Facilidade de correção e depuração.

### 4.2 Divisões

O COBOL apresenta 4 divisões, que são:

#### 4.2.1 Divisão de Identificação – IDENTIFICATION DIVISION

Identifica o programa fonte (obrigatório) e outras informações como autor, data de compilação, etc.

#### 4.2.2 Divisão de Ambiente – ENVIRONMENT DIVISION

Especifica o equipamento usado para compilação e execução do programa, além de associar os arquivos do programa aos diversos periféricos de Entrada / Saída.

#### 4.2.3 Divisão de Dados – DATA DIVISION

Descreve os arquivos que servirão de entradas ou saídas para o programa.

#### 4.2.4 Divisão de Processamento – PROCEDURE DIVISION

Descreve os procedimentos necessários para a solução do problema.

**5 Descrição da Folha de Programação**

Colunas	Descrição	
01 a 06	Usadas para numeração de linhas (opcional)	
	<b>Posição</b>	<b>Conteúdo</b>
	01 a 03	Número da página
	04 a 06	Número da linha
7	<b>Caractere</b>	<b>Significado</b>
	*	Comentários
	-	Continuação da linha anterior
	/	Salto de página da listagem do programa fonte
08 a 72	Posições reservadas para a digitação do programa fonte	
	<b>08 a 11 (Margem 'A')</b>	<b>12 a 72 (Margem 'B')</b>
	<ul style="list-style-type: none"> <li>- Nomes de divisões;</li> <li>- Nomes de seções;</li> <li>- Procedure-names;</li> <li>- Níveis FD, SD, CD e,</li> <li>- Números de níveis (01, 66, 77 e 78).</li> </ul>	<ul style="list-style-type: none"> <li>- Comandos;</li> <li>- Entradas associadas aos Indicadores de nível e;</li> <li>- Números de níveis (12 ao 49 e 88)</li> </ul>
73 a 80	Identificação do programa (opcional)	

**6 Conjunto de caracteres válidos**

- Numéricos: 0 1 2 3 4 5 6 7 8 9 0
- Alfabéticos: a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I  
J K L M N O P Q R S T U V W X Y Z
- Especiais:
  - ✓ espaço
  - ✓ ponto e vírgula
  - ✓ aspas ou apóstrofos
  - ✓ parêntese esquerdo e direito
  - ✓ ponto
  - ✓ vírgula
  - ✓ mais
  - ✓ menos
  - ✓ asterisco
  - ✓ barra normal
  - ✓ igual
  - ✓ cifrão
  - ✓ maior
  - ✓ menor
  - ✓ dólar
  - ✓ e comercial
  - ✓ arroba

## 7 Regras de formatação

Existem as palavras reservadas da linguagem COBOL, que veremos no item 9, e existem palavras atribuídas ao programador. Na criação de nomes, o programador deve obedecer a certas regras como:

- Não deve exceder a 30 (trinta) caracteres;
- O espaço em branco não é um caractere permitido para a formação de palavras;
- Uma palavra não pode começar e nem terminar com um – (hífen).

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

- Nomes de campos (podem começar por um número);
- Nomes de rotinas;
- Nomes de condição; e;
- Nomes externos.

As palavras existem com propósitos próprios e são classificadas em 3 tipos:

- Verbos - ADD, READ, ENTER etc;
- Necessárias - BY (do comando MULTIPLY); e
- Significado Especial - NEGATIVE, ZERO, SECTION ETC.

**Exemplos:**

- IMPOSTO-DE-RENDA.
- MOVE VENCIMENTO-PREVIO TO DEMITIDO-DA-EMPRESA.

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

## 8 Regras de pontuação

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

- Ao finalizar uma sentença, devemos usar logo em seguida o ponto final;
- Entre uma palavra e outra, devemos usar no mínimo um espaço em branco;
- Caracteres de calculas ou de comparações, deverão ser separados por no mínimo um espaço em branco.

## 9 Palavras reservadas

São palavras de uso exclusivo da linguagem, como por exemplo, comandos, nomes de divisões, nomes de seções, comparações, e outros. Disponibilizaremos abaixo uma lista destas palavras reservadas:

ACCEPT	CHARACTER	DECIMAL-POINT	EXAMINE
ACCESS	CHARACTERS	DECLARATIVES	EXCEPTION
ACTUAL	CLASS	DELETE	EXCESS-3
ADD	CLOCK-UNITS	DELIMITED	EXCLUSIVE
ADDRESS	CLOSE	DELIMITER	EXEC
ADVANCING	COBOL	DEPENDING	EXECUTE
AFTER	CODE	DESCENDING	EXHIBIT
ALL	CODE-SET	DESTINATION	EXIT
ALPHABET	COL	DETAIL	EXTEND
ALPHABETIC	COLLATING	DISABLE	EXTERNAL
ALPHABETIC-LOWER	COLUMN	DISK	FALSE
ALPHABETIC-UPPER	COM-REG	DISP	FD
ALPHANUMÉRIC	COMMA	DISPLAY	FH--FCD
ALPHANUMÉRIC-EDITED	COMMIT	DISPLAY-1	FH--KEYDEF
ALSO	COMMON	DISPLAY-ST	FILE
ALTER	COMMUNICATION	DIVIDE	FILE-CONTROL
ALTERNATE	COMP	DIVISION	FILE-ID
AND	COMP-0	DOWN	FILE-LIMIT
ANY	COMP-1	DUPLICATES	FILE-LIMITS
APPLY	COMP-2	DYNAMIC	FILLER
ARE	COMP-3	ECHO	FINAL
AREA	COMP-4	EGCS	FIRST
AREAS	COMP-5	EGI	FIXED
ASCENDING	COMP-6	EJECT	FOOTING
ASSIGN	COMP-X	ELSE	FOR
AT	COMPUTATIONAL	EMI	FOREGROUND-COLOR
AUTHOR	COMPUTATIONAL-0	EMPTY-CHECK	FOREGROUND-COLOUR
AUTO	COMPUTATIONAL-1	ENABLE	FROM
AUTO-SKIP	COMPUTATIONAL-2	END	FULL
AUTOMATIC	COMPUTATIONAL-3	END-ACEPT	GENERATE
BACKGROUND-COLOR	COMPUTATIONAL-4	END-ADD	GIVING
BACKGROUND-COLOUR	COMPUTATIONAL-5	END-CALL	GLOBAL
BACKWARD	COMPUTATIONAL-6	END-CHAIN	GO
BASIS	COMPUTATIONAL-X	END-COMPUTE	GOBACK
BEEP	COMPUTE	END-DELETE	GREATER
BEFORE	CONFIGURATION	END-DIVIDE	GRID
BEGINNING	CONSOLE	END-EVALUATE	GROUP
BELL	CONTAINS	END-IF	HEADING
BINARY	CONTENT	END-MULTIPLY	HIGH
BLANK	CONTINUE	END-OF-PAGE	HIGH-VALUE
BLINK	CONTROL	END-PERFORM	HIGH-VALUES
BLOCK	CONTROLS	END-READ	HIGHLIGHT
BOTTOM	CONVERT	END-RECEIVE	I-O
BY	CONVERTING	END-RETURN	I-O CONTROL
C01	COPY	END-REWRITE	ID
C02	CORE-INDEX	END-SEARCH	IDENTIFICATION
C03	CORR	END-START	IF
C04	CORRESPONDING	END-STRING	IN
C05	COUNT	END-SUBTRACT	INDEX
C06	CRT	END-UNSTRING	INDEXED
C07	CRT-UNDER	END-WRITE	INDICATE
C08	CSP	ENDING	INITIAL
C09	CURRENCY	ENTER	INITIALIZE
C10	CURRENT-DATE	ENTRY	INITIATE
C11	CURSOR	ENVIRONMENT	INPUT
C12	DATA	EOL	INPUT-OUTPUT
CALL	DATE	EOP	INSERT
CANCEL	DATE-COMPILED	EOS	INSPECT
CBL	DATE-WRITTEN	EQUAL	INSTALLATION
CD	DAY	ERASE	INTO
CF	DAY-OF-WEEK	ERROR	INVALID
CH	DBCS	ESCAPE	IS
CHAIN	DE	ESI	JAPANESE
CHAINING	DEBUG	EVALUATE	JUST
CHANGED	DEBUGGING	EVERY	JUSTIFIED
KANJI	OTHER	REPORTS	SUB-QUEUE-3

KEPT	OTHERWISE	REQUIRED	SUBTRACT
KEY	OUTPUT	REREAD	SUM
KEYBOARD	OVERFLOW	RERUN	SUPPRESS
LABEL	OVERLINE	RESERVE	SYMBOLIC
LAST	PACKED-DECIMAL	RESET	SYNC
LEADING	PADDING	RETURN	SYNCHRONIZED
LEAVE	PAGE	RETURN-CODE	SYSIN
LEFT	PAGE-COUNTER	RETURNING	SYSIPT
LEFT-JUSTIFY	PASSWORD	REVERSE	SYSLST
LEFTLINE	PERFORM	REVERSE-VIDEO	SYSOUT
LENGTH	PF	REVERSED	SYSPUNCH
LENGTH-CHECK	PH	REWIND	TAB
LESS	PIC	REWRITE	TABLE
LIMIT	PICTURE	RF	TALLY
LIMITS	PLUS	RH	TALLYING
LIN	POINTER	RIGHT	TAPE
LINAGE	POS	RIGHT-JUSTIFY	TERMINAL
LINAGE-COUNTER	POSITION	ROLLBACK	TERMINATE
LINE	POSITIONING	ROUNDED	TEST
LINES	POSITIVE	RUN	TEXT
LINE-COUNTER	PREVIOUS	S01	THAN
LINKAGE	PRINT	S02	THEN
LOCAL-STORAGE	PRINTER	SAME	THROUGH
LOCK	PRINTER-1	SCREEN	THRU
LOCKING	PRINTING	SD	TIME
LOW	PROCEDURE	SEARCH	TIME-OF-DAY
LOW-VALUE	PROCEDURE-POINTER	SECTION	TIMES
LOW-VALUES	PROCEDURES	SECURE	TITLE
MANUAL	PROCEED	SECURITY	TO
MEMORY	PROCESSING	SEEK	TOP
MERGE	PROGRAM	SEGMENT	TOTALED
MESSAGE	PROGRAM-ID	SEGMENT-LIMIT	TOTALING
MODE	PROMPT	SELECT	TRACE
MODULES	PROTECTED	SELECTIVE	TRACK-AREA
MORE-LABELS	PURGE	SEND	TRACK-LIMIT
MOVE	QUEUE	SENTENCE	TRACKS
MULTIPLE	QUOTE	SEPARATED	TRAILING
MULTIPLY	QUOTES	SEQUENCE	TRAILING-SIGN
NAME	RANDOM	SEQUENTIAL	TRANSFORM
NAMED	RANGE	SERVICE	TRUE
NATIVE	RD	SET	TYPE
NCHAR	READ	SIGN	UNDERLINE
NEGATIVE	READY	SIZE	UNIT
NEXT	RECEIVE	SKIP1	UNLOCK
NO	RECORD	SKIP2	UNSTRING
NO-ECHO	RECORD-OVERFLOW	SKIP3	UNTIL
NOMINAL	RECORDING	SORT	UP
NOT	RECORDS	SORT-CONTROL	UPDATE
NOTE	REDEFINES	SORT-MERGE	UPON
NULL	REEL	SOURCE	UPSI-0
NUMBER	REFERENCE	SOURCE-COMPUTER	UPSI-1
NUMERIC	REFERENCES	SPACE	UPSI-2
NUMERIC-EDITED	RELATIVE	SPACE-FILL	UPSI-3
OBJECT-COMPUTER	RELEASE	SPACES	UPSI-4
OCCURS	RELOAD	SPECIAL-NAMES	UPSI-5
OF	REMAINDER	STANDARD	UPSI-6
OFF	REMARKS	STANDARD-1	UPSI-7
OMITTED	REMOVAL	STANDARD-2	USAGE
ON	RENAMES	START	USE
OPEN	REORG-CRITERIA	STOP	USER
OPTIONAL	REPLACE	STORE	USING
OR	REPLACING	STRING	VALUE
ORDER	REPORT	SUB-QUEUE-1	VALUES
ORGANIZATION	REPORTING	SUB-QUEUE-2	VARIABLE

VARYING  
WAIT  
WHEN  
WHEN-COMPILED  
WITH

WORDS  
WORKING-STORAGE  
WRITE  
WRITE-ONLY  
ZERO  
ZERO-FILL  
ZEROES  
ZEROS

## 10 Identification Division

É a primeira das quatro divisões, é utilizada para identificar o programa. As informações fornecidas pela Identification Division, são tratadas pelo compilador como comentários, portanto não são traduzidas em linguagem de máquina.

### Formato:

```
IDENTIFICATION DIVISION.  
PROGRAM-ID.      (Nome do programa, com o máximo de 8 caracteres)  
AUTHOR.          (Nome do elaborador / programador )  
INSTALLATION.    (Loção onde o programa será executado)  
DATE-WRITTEN.    (Data da elaboração do programa)  
DATE-COMPILED.   (Área para o compilador inserir data / hora)  
SECURITY.        (Comentários sobre a segurança do programa / arquivo)  
REMARKS.        (Comentários que servem de documentação do programa)
```

### Observações:

- a) Ao informar a opção DATE-COMPILED, não preenchê-la, pois é o próprio sistema que o fará;
- b) Atualmente as linhas INSTALLATION, DATE-WRITTEN, DATE-COMPILED, SECURITY e REMARKS, não são utilizadas.

### Exemplo de preenchimento da Identification Division:

```
IDENTIFICATION DIVISION.  
PROGRAM-ID.      FUTURE01.  
AUTHOR.          FUTURE SCHOOL CURSOS DE COMPUTACAO.
```

## 11 Environment Division

É a divisão que identifica o ambiente, ou seja, o equipamento (computador) que será utilizado para processar o programa, os arquivos e suas respectivas unidades de configuração. Está dividida em duas seções:

- Configuration Section;
- Input-Output Section.

### Formato:

```
ENVIRONMENT      DIVISION.  
CONFIGURATION    SECTION.  
.  
.  
.  
INPUT-OUTPUT     SECTION.  
.  
.  
.
```

### 11.1 Configuration Section

É utilizada para fornecer as características do equipamento onde o programa será compilado e executado:

### Formato:

```
CONFIGURATION    SECTION.  
SOURCE-COMPUTER. (Nome do equipamento onde será compilado o programa)  
OBJECT-COMPUTER. (Nome do equipamento onde será executado o programa)  
SPECIAL-NAMES.  (Relacionar funções existentes no COBOL com nomes  
                  simbólicos dados pelo programador)
```

### Exemplo:

```
CONFIGURATION    SECTION.  
SOURCE-COMPUTER. IBM-3090.  
OBJECT-COMPUTER. IBM-3090.  
SPECIAL-NAMES.  DECIMAL-POINT IS COMMA      1  
                  CURRENCY SIGN IS literal  2  
                  C01           IS CANAL-1.  3
```

### Observações:

- 1 – Definir que o ponto decimal é virgula;
- 2 – Mudar o símbolo monetário (default é \$);
- 3 – Definir que a primeira linha de impressão (C01) será tratada como CANAL-1

## 11.2 Input-Output Section

Controla a transmissão de dados entre o programa e o meio externo, ou seja, define os arquivos utilizados pelo programa efetuando ligações com seus respectivos periféricos. Está dividida em:

- File-Control;
- I-O Control

### Formato:

```
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
    SELECT (nome do arquivo) ASSIGN TO (classe-organização-nome)  
.  
.  
I-O CONTROL.  
.  
.  
.
```

### 11.2.1 File-Control

É a área onde está definido em qual periférico se encontra o arquivo e em que modo será tratado, através da instrução SELECT.

### 11.2.1.1 Select

Tem a função de designar um arquivo para um periférico de Entrada / Saída.



É necessário um 'SELECT' para cada arquivo a ser tratado no programa.

#### Formato:

```
SELECT (nome do arquivo) ASSIGN TO (classe-organização-nome)
```

- **Nome do arquivo**  
Dado pelo programador e é pelo qual será reconhecido nas demais divisões do programa.
- **Classe**  
Especifica o tipo de periférico, podendo ser:
  - ✓ (DA) – Direct Access (Discos Magnéticos);
  - ✓ (UT) – Utility (Fitas e Discos Magnéticos);
  - ✓ (UR) – Unit Record (Leitoras, Impressoras) ;
  - ✓ (UP) – Unit Punch (Perfuradoras)
- **Organização**  
Especifica os métodos de acessos aos arquivos, podendo ser:
  - ✓ (S) – Arquivos Seqüenciais;
  - ✓ (D) – Acesso Direto (classe = DA);
  - ✓ (W) – Acesso Direto com o uso do comando REWRITE
  - ✓ (I) – Arquivos Indexados
- **NOME**  
É o nome que o arquivo será reconhecido no JCL através do cartão DD(não pode ter mais que 8 (oito) caracteres e nem começar com caractere numérico).

**11.2.2 I-O-Control**

Tem as seguintes funções:

- Auxiliar na gravação de arquivos variáveis;
- Efetuar a operação de CHECK-POINT, ou seja, para trabalhos que exijam arquivos com grandes quantidades de registros, garantir a execução do comando caso ocorra algum erro no mesmo, isto é, salvar o que já foi feito até o instante, em uma área pré-estipulada pelo programador;

**Exemplo 01:**

```
I-O-CONTROL.  
  APPLY WRITE-ONLY (Nome do arquivo variável).
```

A cláusula 'APPLY WRITE-ONLY' deve ser utilizada para a gravação de arquivos variáveis. A área na 'FD' deve ser excluída para gravação, não podendo mover dados para lá, para depois gravar o registro, ou seja, caso se faça necessário trabalhar com algum dado que influencie o arquivo a ser gravado este serviço deverá ser feito na 'WORKING-STORAGE SECTION' e depois se grava o registro da WORKING através da opção 'FROM'.

**Exemplo:**

```
  SELECT VARIAVEL ASSIGN TO UT-S-VARIAVEL.  
I-O-CONTROL.  
  APPLY WRITE-ONLY VARIAVEL.  
DATA DIVISION.  
FD VARIAVEL  
  RECORDING V  
  BLOCK 0.  
01 REG1          PIC X(120).  
01 REG2          PIC X(140).  
01 REG3          PIC X(160).  
WORKING-STORAGE SECTION.  
01 REGWORK.  
  05 WS-NOME      PIC X(40).  
  05 WS-ENDERECO  PIC X(40).  
  05 WS-CEP       PIC 9(08).  
  05 WS-CIDADE    PIC X(30).  
  05 WS-ESTADO    PIC X(02).  
  05 WS-RESTO-02  PIC X(20).  
  05 WS-RESTO-03  PIC X(20).  
PROCEDURE DIVISION.  
.....  
MOVE CAD-NOME TO W-NOME.  
MOVE CAD-END TO W-END.  
.....  
IF WS-ESTADO EQUAL 'SP'  
  WRITE REG1 FROM REGWORK  
ELSE  
  IF WS-ESTADO EQUAL 'RJ'  
    WRITE REG2 FROM REGWORK  
  ELSE  
    WRITE REG3 FROM REGWORK.
```

**Exemplo 02:**

```
I-O-CONTROL.
  SAME   RECORD  AREA   FOR (file-01), (file-02), (file-03)...
                                     ou
  SAME   AREA    FOR    (file-01), (file-02), (file-03)...
```

É utilizado para trabalhar com uma única descrição dos dados na 'FD'. Quando não tivermos a opção 'RECORD', os arquivos não podem estar abertos ao mesmo tempo; Primeiro abre-se o FILE-01, faz todo o processamento necessário para o mesmo, fecha-o, depois proceder ao mesmo tratamento para os demais arquivos, seguindo a mesma ordem para cada. A cláusula 'BLOCK CONTAINS nnn RECORDS ' é obrigatória, não podendo ser utilizada a opção 'BLOCK 0'.

**Exemplo:**

```

SELECT ARQ01  ASSIGN TO UT-S-ARQ01.
SELECT ARQ02  ASSIGN TO UT-S-ARQ02.
I-O-CONTROL.
  SAME   RECORD  FOR    AREA  ARQ01
                                     ARQ02.

DATA  DIVISION.
FD  ARQ01
   RECORDING F
   BLOCK 40 RECORDS.
01  REG01.
   05  ARQ1-NOME           PIC X(40).
   05  ARQ1-END            PIC X(40).
   05  ARQ1-CIDADE        PIC X(02).
   05  ARQ1-ESTADO        PIC X(20).
FD  ARQ02
   RECORDING F
   BLOCK 28 RECORDS.
01  REG02                PIC X(92).
PROCEDURE  DIVISION.
  OPEN  INPUT      ARQ01
                                     ARQ02.

  READ  ARQ01
   AT  END
   GO  TO  LER-ARQ02.
  MOVE  ARQ1-NOME  TO  ...
  MOVE  ARQ1-END   TO  ...
.....
LER-ARQ02.
  READ  ARQ02
   AT  END
   GO  TO  FIM-PROCES.
  MOVE  ARQ1-NOME  TO  ...
  MOVE  ARQ1-END   TO  ...
  IF  ARQ1-ESTADO  EQUAL  'SP'
   GO  TO  ROT-SP
  ELSE
   GO  TO  ROT-OUTRO-ESTADO.
.....
```

**Exemplo de preenchimento da Environment Division:**

```
ENVIRONMENT  DIVISION.
CONFIGURATION SECTION.
```

```
SOURCE-COMPUTER. IBM-3090.
OBJECT-COMPUTER. IBM-3090.
SPECIAL-NAMES.
    DECIMAL-POINT IS COMMA.
    C01           IS CANAL-1.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT      CARTAO      ASSIGN    TO    UR-S-CARTAO.
    SELECT      FITA        ASSIGN    TO    UT-S-FITA.
    SELECT      VARIAV      ASSIGN    TO    UT-S-VARIAV.
    SELECT      CADFUNC     ASSIGN    TO    UT-S-CADFUNC.
    SELECT      ARQFUNC     ASSIGN    TO    UT-S-ARQFUNC.
I-O-CONTROL.
    APPLY      WRITE-ONLY VARIAV.
    SAME      RECORD AREA   FOR CADFUNC
                                ARQFUNC.
```

**12 Data Division**

É a divisão que tem a função de descrever os arquivos e seus registros, assim como qualquer área de trabalho necessário ao programa. Possui 5 (cinco) seções que devem aparecer na ordem especificada abaixo, porém se alguma seção não for necessária, deverá ser omitida:

- File Section;
- Working-Storage Section;
- Linkage Section;
- Communication Section; e;
- Report Section.

**12.1 File Section**

Trata da definição dos arquivos descritos na cláusula SELECT, suas características e estrutura, ou seja descreve o conteúdo e a organização dos arquivos.

A primeira informação a ser definida, descrição de arquivos, sort-files, relatórios ou ainda comunicação, é por intermédio de um indicador de nível conforme descrição abaixo:

**12.1.1 Indicadores de níveis:**

Indicador	Definição
FD	File Description – Descrição de arquivos
SD	Sort Description – Descrição de 'sort-files'
RD	Report Description – Descrição de relatórios
CD	Communication Description – Descrição de comunicação

**Observação:** Cada FD descreve o arquivo informado na cláusula SELECT.

A segunda informação a ser definida é descrito por um número de nível, no caso 01.

#### 12.1.1.1 Nível FD

##### Formato:

```
FD nome-do-arquivo
BLOCK CONTAINS nnn RECORDS ou CHARACTERS
RECORD CONTAINS nnn CHARACTERS
RECORDING MODE IS (formato do arquivo)
LABEL RECORD IS/ARE (formato do Label) STANDARD / OMITTED
DATA RECORD IS/ARE (nome do registro)
```

##### 12.1.1.1.1 Block Contains

Especifica o tamanho do registro físico

##### Formato:

```
BLOCK CONTAINS nnn RECORDS ou CHARACTERS
```

##### Explicação da cláusula acima:

- Número de registros que existem por bloco.

**Observação:** Se preenchido com 0 (zero) assume informações do cartão 'DD' do JCL, se não preenchido com 'records' assume 'characters'.

##### 12.1.1.1.2 Record Contains

Especifica o tamanho do registro lógico.

##### Formato:

```
RECORD CONTAINS nnn CHARACTERS
```

##### Explicação da cláusula acima:

- Quantidade de bytes que possui o registro lógico, ou seja, o tamanho lógico do registro.

**Observação:** Se esta cláusula for colocada é feita uma conferência pelo compilador somando-se a quantidade de bytes definidos nos campos do registro, para verificar se o tamanho definido confere com a soma dos bytes dos campos.

##### 12.1.1.1.3 Recording Mode

Designa o formato do arquivo. Pode ser:

- F – Comprimento Fixo;
- V – Comprimento Variável;
- U – Comprimento Indefinido; e;
- S – Estendido (Spanned)

**Formato:**

RECORDING MODE IS F, V, U ou S

**Observações:**

Se não for colocada a cláusula 'Recording Mode', o compilador determinará o formato através do cartão 'DD' do JCL ou catálogo.

Para arquivos seqüenciais usar F, V ou S.

Para arquivos de acesso direto usar F, U ou S.

**12.1.1.1.4 Label Record**

Especifica o formato do Label (rótulo).

**Formato:**

LABEL RECORD IS/ARE (formato do Label) STANDARD / OMITTED

**Observações:**

Quando omitido assume 'Label Record Standard'.

Para geração de arquivos em discos magnéticos sempre utilizar 'Label Record Standard', pois dentro de um determinado disco magnético, pode existir vários arquivos, por isso é necessário dar um nome ao arquivo, no caso, um rótulo.

Para geração de relatórios sempre utilizar 'Label Record Omitted', pois não se pode listar o mesmo relatório em diferentes impressoras simultaneamente durante o processamento de um determinado programa.

**12.1.1.1.5 Data Record**

Serve apenas como documentação para identificação dos registros.

**Formato:**

DATA RECORD IS/ARE (nome do registro)

### 12.1.2 Número de níveis

Servem para estruturar logicamente o registro. Está subdividido em:

- Itens elementares (não possuem divisões);
- Itens de grupo

O números de níveis estão entre 01 e 49, porém quanto menor for o nível, mais importante ele será dentro da definição.

#### Exemplo:

```
FD  ARQUIVO.
01  REGISTRO.
    02  CPO-1          PIC  X(05)
    02  CPO-2 .
        03  CPO-2-A    PIC  9(05) .
        03  CPO-2-B    PIC  X(05) .
    02  CPO-3          PIC  S9(15)V99.
    02  CPO-4.
        03  CPO-4-A    PIC  9(02) .
        03  CPO-4-B.
            04  CPO-4-B-1 PIC  X(02) .
            04  CPO-4-B-2 PIC  9(02) .
            04  CPO-4-B-3 PIC  X.
```

#### Explicação dos níveis informados acima:

- Indicador de nível → FD;
- Número de níveis → 01, 02, 03 e 04;
- Itens de Grupo → REGISTRO, CPO-2, CPO-4, CPO-4-B; e;
- Itens elementares → CPO-1, CPO-2, CPO-3, CPO-4 e CPO-4-B.

### 12.1.2.1 Número de níveis especiais

#### 12.1.2.1.1 Nível 66

Entradas que utilizam a clausula RENAME com o propósito de re-agrupar itens de dados

#### Exemplo:

```
01  REGISTRO.
    02  A              PIC  X(01) .
    02  B              PIC  X(03) .
    02  C              PIC  X(05) .
    66  A1 RENAME X THRU Z.
    66  A2 RENAME X THRU Y.
    66  A3 RENAME Z THRU Y.
```

**12.1.2.1.2 Nível 77**

Entradas que são itens de dados não contíguos, que não são subdivisões de outros, nem podem ser subdivididos. Muito utilizado para acumuladores e/ou áreas de File Status.

**Exemplo:**

```
WORKING-STORAGE      SECTION.
77 ACUM-LINHAS          PIC 9(02) VALUE 60.
77 ACUM-PAG            PIC 9(05) VALUE 0.
77 ACUM-LIDOS          PIC 9(09) VALUE 0.
```

**12.1.2.1.3 Nível 78**

Entradas que especificam constant-names, ou seja, entradas que não sofrerão alterações durante o seu processamento.

**Exemplo:**

```
WORKING-STORAGE      SECTION.
78 CONSTANTE-01       PIC X(06) VALUE 'FUTURE' .
78 CONSTANTE-02       PIC X(06) VALUE 'SCHOOL' .
78 CONSTANTE-03       PIC X(06) VALUE 'CURSOS' .
78 CONSTANTE-04       PIC X(02) VALUE 'DE' .
78 CONSTANTE-05       PIC X(10) VALUE 'COMPUTACAO' .
78 CONSTANTE-06       PIC X(05) VALUE 'COBOL' .
78 CONSTANTE-07       PIC X(03) VALUE 'ANS' .
```

**12.1.2.1.4 Nível 88**

Entradas que especificam condition-names associados a valores particulares de uma variável.

**Exemplo 01:**

```
01 REGISTRO.
  02 SEXO          PIC X(01) .
    88 FEMININO    VALUE 'F' .
    88 MASCULINO   VALUE 'M' .
```

**Exemplo 02:**

```
01 COD-VERBA          PIC 9(03) .
  88 PROVENTOS        VALUE 001 THRU 010, 013, 015 THRU 025.
  88 DESCONTOS        VALUE 011, 012, 30 THRU 50.
```

**Observação:** Os valores definidos no nível 88 devem estar em ordem crescente.

### 12.1.3 Descrição do registro

Uma entrada para cada item (campo), sendo descrito com o número de nível.

#### Formato:

```
05 nome-do-campo          PIC X(nn) .
05 nome-do-campo          PIC S9(07)V99 COMP-3.
```

#### Explicação das linhas acima:

- 05 → Número do nível;
- nome-do-campo → Mnemônico de um determinado campo, por qual vai ser tratado ou chamado dentro do programa;
- PIC ou PICTURE → Formato do campo
- X → Tipo do campo (alfanumérico);
- nn → Tamanho do campo;
- S9 → Campo numérico sinalizado;
- (07)V99 → Tamanho do campo, no caso 7 (sete) inteiros e 2 (duas) decimais;e;
- COMP-3 → Compactado.

#### Exemplos:

```
05 CAMPO1          PICTURE X(25) .
05 CAMPO2          PIC      9(08) COMP-3.
05 CAMPO3          PIC      9(04) VALUE 0.
```

#### 12.1.3.1 Picture - PIC

É usada para descrição de informações sobre os campos ou itens tais como: tamanho, sinal, formato (alfabético, alfanumérico, numérico, ou editadas), posição do ponto decimal ou formato da impressão (editadas). Não podem ser em itens de grupo.

##### 12.1.3.1.1 Formato Alfabético

Representado por letras mais o 'espaço', e é definida pela letra A.

#### Exemplos:

```
05 CAMPO-01        PICTURE IS AAAAAA VALUE 'FUTURE' .
05 CAMPO-02        PIC      IS AAAAAA VALUE 'FUTURE' .
05 CAMPO-03        PIC      A(6) VALUE 'FUTURE' .
```

Se tivermos um conteúdo 'JOSE DA SILVA', teríamos que representar da seguinte maneira:

```
05 CAMPO-04        PIC A(13) VALUE 'JOSE DA SILVA' .
```

### 12.1.3.1.2 Formato Alfanumérico

Representado por letras, números e caracteres especiais do COBOL. É representado pela letra X.

**Exemplos:**

```
05 CAMPO-01          PICTURE IS XXXXXX VALUE 'SCHOOL' .
05 CAMPO-02          PIC      IS XXXXXX VALUE 'SCHOOL' .
05 CAMPO-03          PIC      X(6)   VALUE 'SCHOOL' .
```

Se tivermos um conteúdo '01A02B03C04D', teríamos que representar da seguinte maneira:

```
05 CAMPO-04          PIC X(12) VALUE '01A02B03C04D' .
```

### 12.1.3.1.3 Formato Numérico

Usado para representação exclusiva de itens numéricos. É representado pelo caractere 9.

Sua definição pode apresentar os seguintes conteúdos:

- V → usado para mostrar a posição da vírgula em campos com decimais;
- P → representa um dígito numérico zero e devem ser usado entre o V e o 9; e;
- S → indica a presença de sinal, deve ser colocado antes do caractere 9.

A seguir faremos uma rápida demonstração de acordo com a definição do campo.

Campo		Representação Picture	Valor Exibido
Conteúdo	Definição		
503	9(03)	999	503
125	9V99	9V99	1,25
457	9(03)	999PP	45700
153422	99V9(04)	99VPP9(04)	15,003422
241-	S9(03)	S999	-241
1002	S9V9(03)	S9V999	+1,002
503	9(03)	99	03
153422	9(06)	99VPP	22,00

#### 12.1.3.1.4 Formato de Edição

Utilizada apenas para impressão dos campos. É representado por qualquer combinação abaixo:

Código de Edição	Significado
A	Cada 'A' permite letras do alfabeto e o caractere espaço.
B	Inserção do caractere espaço.
P	Fator de escala para especificar a posição de um ponto decimal assumido.
S	Presença de um sinal operacional.
V	Posição do ponto decimal assumido.
X	Cada 'X' permite qualquer caractere do conjunto de caracteres ASCII.
Z	Substituição de '0' (zeros) não significativos por espaços.
9	Cada '9' permite um caractere numérico.
0	Inserção do '0' (zero).
/	Inserção da '/' (barra).
,	Inserção da ',' (vírgula).
.	Inserção do '.' (ponto).
+ - CR DB	Caracteres de edição de controle de sinal. Cada um representa a posição onde o caractere será colocado. São mutuamente exclusivos.
*	Substituição de '0' (zeros) não significativos por '*' (asteriscos).
\$	Símbolo monetário.

**Exemplos:**

Conteúdo do campo	Definição do campo	Pic	Valor Exibido
001531	9(05)	99.999	01.531
002542	9(05)V99	ZZ.ZZ9,99	25,42
000	9(03)	ZZZ	
000	9(03)	ZZ9	0
422	9(03)	**9	422
002	9(03)	**9	**2
000	9(03)	***	***
0000042	9(05)V99	ZZ.ZZ9,99	0,42
80000(-)	S9(03)V99	999,99CR	800,00CR
2733	9(04)	99.99	27.33
1123	9(04)	990099	110023
12345	9(05)	99B9B99	12 23 5
373	9(03)	\$999	\$373
0373	9(04)	\$ZZZZ	\$ 373
327(-)	S9(03)	-999	-327
327(+)	S9(03)	-999	327
327(-)	S9(03)	+999	327
327(+)	S9(03)	+999	+327
327(-)	S9(03)	999-	327-
238	9(03)	\$\$99	\$238
005	9(03)	\$\$99	\$05
005	9(03)	\$\$ZZ	\$ 5
80000(+)	S9(03)V99	999,99CR	800,00
0012(-)	S9(04)	--.--9	-12

**12.1.3.2 Blank when zero**

Indica que o item descrito deverá ser preenchido com espaços sempre que um valor for zero. Deve ser apenas informado para itens elementares ou numérico de edição.

**Exemplo**

```
01 VALOR          PIC 9(05) BLANK WHEN ZERO.
```

Quando o campo que for movido para o campo valor estiver com conteúdo ZEROS, o campo valor ficará com espaços.

### 12.1.3.3 Filler

Usado para um item elementar cuja informação nunca será referenciada, pode ser usada em qualquer seção da Data Division.

**Exemplo:**

Código	Nome	Endereço					Filler
		Rua	Bairro	CEP	Cidade	Estado	
N	AN	AN	AN	N	AN	AN	AN
3	30	30	15	8	20	2	12

**Legenda:**

AN = Alfanumérico  
 N = Numérico

```

FD  ARQUIVO
   LABEL      RECORD      IS STANDARD
   RECORDING  MODE        IS F
   RECORD     CONTAINS    120 CHARACTERS
   BLOCK      CONTAINS    10 RECORDS
   DATA      RECORD      IS REGISTRO.

01  REGISTRO.
   02  CODIGO          PIC 9(03) .
   02  NOME            PIC X(30) .
   02  ENDERECO.
       03  RUA          PIC X(30) .
       03  BAIXRRO     PIC X(15) .
       03  CEP          PIC 9(08) .
       03  CIDADE      PIC X(20) .
       03  ESTADO      PIC X(02) .
   02  FILLER         PIC X(12) .
  
```

**Exemplo de preenchimento da Data Division – File Section**

```

DATA DIVISION.
FILE SECTION.
FD  ARQUIVO
   LABEL      RECORD      IS STANDARD
   RECORDING  MODE        IS F
   RECORD     CONTAINS    30 CHARACTERS
   BLOCK      CONTAINS    10 RECORDS
   DATA      RECORD      IS REGISTRO.

01  RECIBO.
   02  IDENTIFICACAO  PIC  X(05) .
   02  NUMERO          PIC  9(05) .
   02  VALOR           PIC  S9(15)V99 COMP-3.
   02  FILLER         PIC  X(11) .
  
```

### 12.2 Working-Storage Section

Descreve as informações sobre as áreas de trabalho, descrição de tabelas, etc.

### 12.2.1 Value

É usada para definir um valor inicial para um item da 'Working-Storage Section', não podendo ser usada na 'File Section'.

#### Exemplo:

```
WORKING-STORAGE SECTION.  
01 WRK-DATE-SYS.  
    02 WRK-ANO-SYS          PIC 9(04).  
    02 WRK-MES-SYS         PIC 9(02).  
    02 WRK-DIA-SYS         PIC 9(02).  
  
01 WRK-TIME-SYS.  
    02 WRK-HORA-SYS        PIC 9(02).  
    02 WRK-MINUTO-SYS      PIC 9(02).  
  
01 CAB001.  
    02 FILLER                PIC X(10) VALUE SPACES.  
    02 FILLER                PIC X(50) VALUE  
        'FUTURE SCHOOL CURSOS DE COMPUTACAO'.  
    02 FILLER                PIC X(06) VALUE 'PAG.: '.  
    02 CAB001-PAG           PIC Z.ZZ9.
```

### 12.2.2 Computational

Só podem ser utilizados para itens numéricos. Podem ser definidos como:

- COMP ou COMP-4;
- COMP-1 ou COMP-2; e;
- COMP-3.

### 12.2.2.1 Comp ou Comp-4 (Binário)

Utilizado para definição de campos no formato binário.

#### Exemplo:

```
02 CAMPO-01          PIC 9(08) COMP.
```

Ao definir o formato do campo, no caso binário, devemos lembrar o tamanho físico será da seguinte maneira:

- Para tamanho (PIC) entre 01 e 04, teremos um campo com 2 bytes, também chamado como **HALF-WORD**;
- Para tamanho (PIC) entre 05 e 09, teremos um campo com 4 bytes, também chamado como **FULL-WORD**;
- Para tamanho (PIC) entre 10 e 18, teremos um campo com 8 bytes, também chamado como **DOUBLE-WORD**.

#### Exemplo:

```
1. 01 VALOR1      PIC 9(03) COMP.  
2. 01 VALOR2      PIC 9(01) COMP.  
3. 01 VALOR3      PIC 9(11) COMP.  
4. 01 VALOR5      PIC 9(04) VALUE 1234 COMP.  
5. 01 VALOR6      PIC 9(07) VALUE 499701 COMP.
```

Como podemos ver, os itens **4** e **5**, estão valorizados. Para isso, devemos sempre informar o valor no formato decimal, porém fisicamente a apresentação dos campos será no formato hexadecimal, conforme abaixo:

Item d) **04:D2** (formato hexadecimal convertido para decimal = 1234)

Item e) **00:07:9F:F5** (formato hexadecimal convertido para decimal = 499701)

### 12.2.2.2 Comp-1 ou Comp-2 (Ponto flutuante)

Utilizados para pontos flutuantes interno através dos registradores 0, 2, 4 e 6.

Ao definir o campo como COMP-1, teremos um campo fisicamente com 4 bytes de precisão simples, e ao definir o campo como COMP-2, teremos um campo fisicamente com 8 bytes de precisão simples.

### 12.2.2.3 Comp-3 (Compactado)

Utilizados para compactação de campos numéricos, obedecendo as seguintes regras:

- Campos até 18 dígitos ou 10 bytes (fisicamente);
- Sempre o campo deverá ser sinalizado;
- Para se ter um controle mais exato na contagem dos bytes do registro, é aconselhável que o tamanho do campo a ser compactado seja ímpar;
- 1 byte utiliza-se de 2 dígitos ou 1 dígito mais o sinal.

#### Exemplo:

Conteúdo dos campos:

```
VALOR1 = 4321
VALOR2 = -8765
VALOR3 = -357
VALOR4 = 486
```

```
WORKING-STORAGE SECTION.
01 VALOR1 PIC S9(05) COMP-3.
01 VALOR2 PIC S9(05) COMP-3.
01 VALOR3 PIC S9(05) COMP-3.
01 VALOR4 PIC S9(05) COMP-3.
```

Representação física dos campos:

```
Valor1 = 04:32:1C
Valor2 = 08:76:5D
Valor3 = 00:35:7D
Valor4 = 00:48:6C
```

### 12.2.3 Justified

Posiciona itens alfabéticos ou alfanuméricos à direita no campo receptor, sendo que, quando o item emissor for maior, o campo será truncado à esquerda, e quando for menor será preenchido com brancos.

#### Formato:

```
JUSTIFIED    RIGHT
JUST         RIGHT
```

#### Exemplo:

```
01 NOME-ESCOLA PIC X(40) JUSTIFIED RIGHT.
      OU
01 NOME-ESCOLA PIC X(40) JUST      RIGHT.
.
.
.
      MOVE 'FUTURE SCHOOL CURSOS DE COMPUTACAO' TO NOME-ESCOLA.
```

#### Representação física do campo **NOME-ESCOLA**

```
FUTURE SCHOOL CURSOS DE COMPUTACAO
444444CEEEDC4ECCDDD4CEDEDE4CC4CDDDEECCCD
0000006434950238663034926204503647431316
```

### 12.2.4 Redefines

Utilizado para redefinir uma área já definida, em formatos diferentes, de acordo com o seu uso.

**Exemplo:**

```

01 REGISTRO.
03 CAMPO1          PIC X(06) .
03 CAMPO2.
05 CAMPO2-1       PIC X(03) .
05 CAMPO2-2       PIC 9(04) .
05 CAMPO2-3       PIC X(05) .
03 CAMPO2-R REDEFINES CAMPO2.
05 CAMPO2-R-1     PIC 9(06) .
05 CAMPO2-R-2     PIC 9(06) .
03 CAMPO3         PIC X(03) .
  
```

Representação física do **REGISTRO**

REGISTRO				
CAMPO1	CAMPO2			CAMPO3
	CAMPO2-1	CAMPO2-2	CAMPO2-3	
	CAMPO2-R			
	CAMPO2-R-1	CAMPO2-R-2		

### 12.2.5 Constantes figurativas

São palavras já definidas pelo compilador.

Constantes Figurativas	Picture Aplicáveis
ZERO ZEROS ZEROES	alfanuméricas ou numéricas
SPACE SPACES	alfanuméricas ou numéricas
HIGH-VALUE HIGH-VALUES	alfanuméricas
QUOTE QUOTES	alfanuméricas
ALL 'CHARACTER'	alfanuméricas ou alfabéticas
LOW-VALUE LOW-VALUES	alfanuméricas

#### Exemplo:

```
01 VALOR1    PIC X(50)  VALUE ZEROS .
01 VALOR2    PIC 9(08)  VALUE ZEROS .
01 CAMPO1    PIC X(30)  VALUE SPACES .
01 CAMPO2    PIC X(10)  VALUE HIGH-VALUES .
01 CAMPO3    PIC X(02)  VALUE QUOTES .
01 DADOS1    PIC X(50)  VALUE ALL '*'.
01 DADOS2    PIC X(07)  VALUE LOW-VALUES .
```

### 12.2.6 Renames

Alterna, pela sobreposição, itens elementares.

#### Formato:

```
66 <data-name1> RENAMES <data-name2> THRU | THROUGH <data-name3>.
```

**Exemplo:**

```
01 REGISTRO.  
05 GRUPO1.  
    10 CPO1-GR1      PIC X(02) VALUE 'AB'.  
    10 CPO2-GR1      PIC X(02) VALUE 'CD'.  
    10 CPO3-GR1      PIC 9(02) VALUE '12'.  
05 GRUPO2.  
    10 CPO1-GR2      PIC X(03) VALUE 'FGH'.  
    10 CPO2-GR2      PIC X(03) VALUE 'IJK'.  
    10 CPO3-GR2      PIC 9(03) VALUE '456'.  
66 ARRANJO1  RENAME SGRUPO1.  
66 ARRANJO2  RENAME SGRUPO1 THRU GRUPO2.
```

### 12.3 Linkage Section

Seção que descreve os dados aos quais será feita referência pelo programas, chamador e chamado, e é definida nos programas chamados.

Seu uso deve ser obedecendo as seguintes regras:

- 1) São válidas as mesmas regras apresentadas na Working-Storage Section;
- 2) A cláusula 'Value' pode ser somente especificada para nível 88;
- 3) Assume-se que para cada item passado, tem que ser de nível 01 ou 77;
- 4) Um parâmetro passado, deve ter no máximo 100 bytes;
- 5) Na definição da Linkage Section, quando for pego algum dado pelo parâmetro passado pelo item 4, os dois primeiros bytes devem ser definidos como binário, pois estes bytes conterão o tamanho do parâmetro.

**Exemplo de ligação entre programas.**

**Programa = PROGA (programa chamador)**

```
WORKING-STORAGE SECTION.  
01 DADOS-NOME.  
    03 TAMANHO          PIC 9(02).  
    03 NOME             PIC X(30).  
    .  
    .  
    .  
PROCEDURE             DIVISION.  
    .  
    .  
    CALL 'PROGB' USING DADOS-NOME.  
    .  
    .  
    .  
STOP RUN.
```

**Programa = PROGB (programa chamado)**

```
WORKING-STORAGE SECTION.  
    .  
    .  
    .  
LINKAGE              SECTION.  
01 DADOS.  
    02 TAMANHO-LINK    PIC 9(02).  
    02 NOME-LINK       PIC X(30).  
    .  
    .  
    .  
PROCEDURE            DIVISION USING DADOS.  
    .  
    .  
    .  
GOBACK.
```

**Observação:**

Podemos notar que os nomes das áreas não precisam ser iguais, mas os campos devem possuir as mesmas características (formação e ordem), os dados a serem definidos na Linkage Section devem ter um nível 01, onde será o campo principal que receberá os dados, ou cada campo pode ser referenciado a um nível 77.

#### **12.4 Communication Section**

Seção que descreve os dados que servem de interface entre o Message Control System (MCS) e o programa Cobol.

#### **12.5 Report Section**

Seção que descreve os relatórios que o programa deve emitir. Muito utilizado no Report Writer.

**13 Procedure Division**

É nesta divisão que através dos comando executáveis adequados, os dados são processados de forma a produzir os resultados previstos no programa.

Esquema do processamento da Procedure Division:



### 13.1 Comandos para manipulação de arquivos

#### 13.1.1 Close

Permite o fechamento dos arquivos abertos para o processamento.

**Formato:**

```
CLOSE ARQUIVO.
```

**Exemplos:**

```
CLOSE ARQ-01.
```

**13.1.2 Delete**

Remove logicamente os registros de um determinado arquivo.

**Formato.**

```
DELETE NOME-DO-ARQUIVO  
[invalid key <comando1> | not invalid key <comando2>].
```

**Observações:**

- a) É necessário que o comando Read tenha sido executado;
- b) Pode ser utilizado também com o uso do comando Start.

**Opções utilizadas:**

Opção	Significado
INVALID KEY	Ocorre como resultado de execução mal sucedida.
NOT INVALID KEY	Ocorre como resultado de execução bem sucedida.

**Exemplo a):**

```
MOVE    COD-PECA    TO    CHAVE-PECA.  
  
READ    CADPECA  
        KEY        IS    CHAVE-PECA  
        INVALID KEY  
        DISPLAY `ERRO NA LEITURA DO CODIGO => ` CHAVE-PECA  
        GO          TO    FIM.  
  
DELETE  CADPECA  
        INVALID KEY  
        DISPLAY `DEU NA DELECAO DO CODIGO => ` CHAVE-PECA  
        GO          TO    FIM.  
  
DISPLAY `REGISTRO DELETADO => ` CHAVE-PECA.
```

**Exemplo b):**

```
MOVE    COD-PECA    TO    CHAVE-PECA.  
  
START  CADPECA      KEY IS EQUAL    CHAVE-PECA  
        INVALID KEY  
        DISPLAY `ERRO NO START DO CODIGO => ` CHAVE-PECA  
        GO          TO    FIM.  
  
READ   CADPECA      NEXT  
        INVALID KEY  
        DISPLAY `ERRO NA LEITURA DO CODIGO => ` CHAVE-PECA  
        GO          TO    FIM.
```

```
DELETE CADPECA
      INVALID KEY
      DISPLAY `DEU NA DELECAO DO CODIGO => ` CHAVE-PECA
      GO          TO FIM.

DISPLAY `REGISTRO DELETADO => ` CHAVE-PECA.
```

### 13.1.3 Open

Permite a abertura de um ou mais arquivos para o processamento.

#### Formato:

```
OPEN  INPUT  ARQ-I
      OUTPUT  ARQ-O
      I-O     ARQ-IO
      EXTEND  ARQ-E
```

#### Descrição dos tipos de acessos:

- (INPUT) = ARQ. DE LEITURA (entradas);
- (OUTPUT) = ARQ. DE GRAVAÇÃO E IMPRESSÃO (saídas);
- (I-O) = ARQ. DE LEITURA E GRAVAÇÃO (entradas e saídas);
- (EXTEND) = ARQ. DE GRAVAÇÃO (SEQUENCIAL) (saídas).

#### Exemplo:

```
OPEN  INPUT  CARTAO
      FITA01
      FITA02
      OUTPUT  RELAT
      RELATO
      I-O     ARQUIVO1
      ARQUIVO2
      EXTEND  ARQUIVO3
      ARQUIVO4.
```

#### Observação:

O comando 'OPEN EXTEND' serve para adicionar registros em um arquivo seqüencial.

**13.1.4 Read**

Torna disponíveis os registros de um determinado arquivo.

**Formato:**

```
READ  ARQUIVO  [next | previous]
        [into <área>]
        [at end <comando1> | not at end <comando2>]
        [key is <campo-chave>]
        [invalid key <comando3> | not invalid key <comando4>]
```

**Opções utilizadas:**

Opção	Significado
NEXT	Lê o próximo registro.
PREVIOUS	Lê o registro anterior.
INTO	O registro lido também fica disponível em uma determinada área.
AT END	Condição de fim de arquivo.
NOT AT END	Condição contrária à de AT END.
KEY	Indica o nome do campo-chave, declarado na cláusula SELECT e definido na descrição do registro em nível FD.
INVALID KEY	Ocorre quando o campo-chave informado não consta do índice
NOT INVALID KEY	Ocorre quando o campo-chave informado consta do índice

**Exemplos:**

```
READ  ARQ-01.

READ  ARQ-02
      INTO  WRK-ARQ02.

READ  ARQ-03 NEXT.

READ  ARQ-04 PREVIOUS.

READ  ARQ-05
      AT END MOVE 'FIM' TO WRK-FIM.

READ  ARQ-05
      NOT AT END MOVE 'COMEÇO' TO WRK-COMEÇO.
```

```
READ ARQ-06
KEY IS COD-PECA
INVALID KEY DISPLAY 'REGISTRO NAO CADASTRADO'.
```

```
READ ARQ-07
KEY IS COD-PECA
NOT INVALID KEY DISPLAY 'REGISTRO JA CADASTRADO'.
```

**13.1.5 Rewrite**

Substitui o conteúdo dos registros de um determinado arquivo.

**Observação:**

O comando Rewrite, só pode ser processado, após um comando Read.

**Formato1:**

```
REWRITE <REGISTRO-SAIDA> [from <REGISTRO-ENTRADA>]
      [invalid key <comando1> | not invalid key <comando2>].
```

**Opções utilizadas:**

Opção	Significado
INVALID KEY	Ocorre como resultado de execução mal sucedida.
NOT INVALID KEY	Ocorre como resultado de execução bem sucedida.

**Exemplos:**

```
MOVE    COD-PECA      TO  CHAVE-PECA.
```

```
START   CADPECA      KEY IS EQUAL   CHAVE-PECA
INVALID KEY
DISPLAY `ERRO NO START DO CODIGO => ` CHAVE-PECA
GO      TO  FIM.
```

```
READ    CADPECA      NEXT
INVALID KEY
DISPLAY `ERRO NA LEITURA DO CODIGO => ` CHAVE-PECA
GO      TO  FIM.
```

```
MOVE    00010        TO  QTDE-PECA.
```

```
REWRITE REG-CADPECA
INVALID KEY
DISPLAY `ERRO NO REWRITE DO CODIGO => ` CHAVE-PECA
GO      TO  FIM.
```

**13.1.6 Start**

Utilizado, quando se quer agilizar a leitura de um determinado registro, ou seja efetua o posicionamento em registros para posterior leitura.

**Observação:**

O comando Start, só posiciona o registro, a leitura deverá ser feita pelo comando Read.

**Formato.**

```
START      <nome arquivo>
           [KEY IS {NOT GREATER} TO CHAVE-REGISTRO
             {NOT LESS}
             {LESS}
             {GREATER}
             {NOT EQUAL}
             {EQUAL}
           [invalid key <comando1> | not invalid key <comando2>].
```

**Opções utilizadas:**

Opção	Significado
NOT GREATER	Não maior (menor ou igual)
NOT LESS	Não menor (maior ou igual)
LESS	Menor que
GREATER	Maior que
NOT EQUAL	Diferente de
EQUAL	Equal
INVALID KEY	Ocorre como resultado de execução mal sucedida.
NOT INVALID KEY	Ocorre como resultado de execução bem sucedida.

**Exemplos:**

```
MOVE      COD-PECA      TO CHAVE-PECA.

START     CADPECA       KEY IS EQUAL   CHAVE-PECA
INVALID KEY
DISPLAY  `ERRO NO START DO CODIGO => ` CHAVE-PECA
GO        TO FIM.
```

## 13.1.7 Write

Grava registros em um determina do arquivo.

### Formato1:

```
WRITE <REGISTRO-SAIDA> [from <REGISTRO-ENTRADA>]
      [invalid key <comando1> | not invalid key <comando2>].
```

### Formato2:

```
WRITE <REGISTRO-SAIDA> [from <area>] [after | before]
      {[advancing | positioning ] <literal> <campon1>
      [LINE | LINES] [PAGE]}
```

### Opções utilizadas:

Opção	Significado
FROM	Indica o conteúdo (registro) a ser gravado no arquivo.
INVALID KEY	Ocorre como resultado de execução mal sucedida.
NOT INVALID KEY	Ocorre como resultado de execução bem sucedida.
AFTER ADVANCING	Imprime depois de saltar linhas ou páginas
BEFORE ADVANCING	Imprime antes de saltar linhas ou página
LINE	A quantidade de linhas que serão impressas
PAGE	Salto para uma nova página

### Exemplos:

```
WRITE ARQ-REG-02.
WRITE ARQ-REG-02 FROM ARQ-REG-01.
WRITE ARQ-REG-02 FROM ARQ-REG-01
INVALID KEY DISPLAY 'ERRO NA GRAVACAO' .
WRITE ARQ-REG-02 FROM ARQ-REG-01
NOT INVALID KEY DISPLAY 'GRAVACAO OK' .
WRITE ARQ-RELATO FROM CAB001 AFTER PAGE.
WRITE ARQ-RELATO FROM CAB002
AFTER ADVANCING 1 LINE.
WRITE ARQ-RELATO FROM CAB003
AFTER ADVANCING 2 LINES.
```

**13.2 Comandos aritméticos**

Somente pode ser utilizados com itens elementares numéricos.

Os comandos aritméticos são: ADD, COMPUTE, DIVIDE, MULTIPLY e SUBTRACT.

**Opções utilizadas:**

<b>Opções Comuns</b>	<b>Significado</b>
ROUNDED	Incrementa o valor absoluto de um campo receptor quando o dígito de excesso é maior ou igual a 5.
ON SIZE ERROR	Condição que ocorre quando o resultado a ser armazenado no campo receptor é maior que a capacidade deste.
NOT ON SIZE ERROR	Ocorre na forma contrária à ON SIZE ERROR.

**13.2.1 Add**

Efetua a adição de operandos numéricos.

**Observação:** Pode ser END-ADD nos formatos abaixo:

**Formato 1:**

```
ADD <campon1> | <literaln1> ... TO <campon2> [ROUNDED] ...  
[ON SIZE ERROR <comando1>]  
[NOT ON SIZE ERROR <comando2>]
```

**Exemplo:**

```
ADD      1                TO      CONTADOR.  
ADD      1                TO      CONTADOR1 CONTADOR2..
```

**Explicação do exemplo acima:**

- Adicionar 1 em CONTADOR e o resultado será armazenado em CONTADOR;
- Adicionar 1 em CONTADOR1 e CONTADOR2, simultaneamente, e o resultado será armazenado em CONTADOR1 e CONTADOR2..

**Formato 2:**

```
ADD <campon1> | <literaln1> ... TO <campon2> [ROUNDED]  
GIVING <campo3> [ROUNDED]...  
[ON SIZE ERROR <comando1>]  
[NOT ON SIZE ERROR <comando2>]
```

**Exemplo:**

```
ADD      VALOR1 VALOR2   TO      VALOR3      GIVING RESULTADO.  
ADD      VALOR1 VALOR2   TO      VALOR3      GIVING RESULTADO.
```

**Explicação do exemplo acima:**

- Adicionar o conteúdo dos campos VALOR1, VALOR2 e VALOR3 e o resultado será armazenado em RESULTADO.
- Adicionar o conteúdo dos campos VALOR1 e VALOR2 e o resultado será armazenado em RESULTADO.

**Formato 3:**

```
ADD CORRESPONDING | CORR <item1> TO <item2> [ROUNDED]  
[ON SIZE ERROR <comando1>]  
[NOT ON SIZE ERROR <comando2>]
```

**Exemplo:**

```
01  ITEM1.
    05  VALOR1          PIC  9(02)  VALUE  10.
    05  VALOR2          PIC  9(02)  VALUE  20.
01  ITEM2.
    05  VALOR1          PIC  9(02)  VALUE  30.
    05  VALOR2          PIC  9(02)  VALUE  10.

01  VALOR3              PIC  9(02)  VALUE  18.

01  RESULTADO           PIC  9(02)  VALUE  0.

PROCEDURE  DIVISION.
  ADD      VALOR1 OF ITEM1
           VALOR2 OF ITEM1 TO      VALOR3      GIVING RESULTADO.
  ADD      CORR  ITEM1  TO      ITEM2.
```

**Explicação do exemplo acima:**

- Adicionar o conteúdo dos itens elementares VALOR1 e VALOR2 do item de grupo ITEM1 juntamente com o conteúdo do campo VALOR3 e o resultado será armazenado em RESULTADO.
- Adicionar o conteúdo dos itens elementares VALOR1 e VALOR2 do item de grupo ITEM1 com seus respectivos itens elementares do item de grupo ITEM2 e o resultado será armazenado nos itens elementares VALOR1 e VALOR2 do item de grupo ITEM2.

### 13.2.2 Compute

Calcula e armazena o valor de uma expressão aritmética.

**Caracteres utilizados:**

- Adição +
- Subtração -
- Multiplicação \*
- Divisão /
- Exponenciação \*\*

**Observação:** Pode ser END-COMPUTE no formato abaixo:

**Formato:**

```
COMPUTE <campon1> ROUNDED = <expressão aritmética>  
[ON SIZE ERROR <comando1>]  
[NOT ON SIZE ERROR <comando2>]
```

**Exemplos:**

```
COMPUTE RESULTADO = ((2 + 3) * FATOR) .  
COMPUTE ACUM-PAG = ACUM-PAG + 1 .  
COMPUTE ACUM-LINHA = ACUM-PAG - 18 .  
COMPUTE RESULTADO ROUNDED = ((2 + 3) * FATOR) .  
COMPUTE JUROS-SIMPLES ROUNDED =  
((VALOR-PRESENTE * (TAXA / 100)) * PERIODO) .
```

**Explicação dos exemplos acima:**

- Obter o total da somatório entre 2 e 3, multiplicar pelo conteúdo do campos VALOR e o resultado será armazenado em RESULTADO;
- Somar 1 no campo ACUM-PAG;
- Subtrais 18 do campos ACUM-LINHA;
- Obter o total da somatório entre 2 e 3, multiplicar pelo conteúdo do campos VALOR e o resultado será arredondado e posteriormente armazenado em RESULTADO;
- Dividir o conteúdo do campo TAXA por 100, multiplicar pelo conteúdo do campo VALOR-PRESENTE, multiplicá-lo pelo conteúdo do campo PERIODO e o resultado será arredondado e posteriormente armazenado no campo JUROS-SIMPLES.

### 13.2.3 Divide

Efetua a divisão de operandos numéricos.

**Observação:** Pode ser END-DIVIDE nos formatos abaixo:

#### Formato 1:

```
DIVIDE <campon1> | <literaln1> INTO <campon2> [ROUNDED]
      [ON SIZE ERROR <comando1>]
      [NOT ON SIZE ERROR <comando2>]
```

#### Exemplo:

```
DIVIDE DIVISOR          INTO  DIVIDENDO.
DIVIDE 3                INTO  DIVIDENDO.
```

#### Explicação do exemplo acima:

- Dividir o conteúdo do campo DIVIDENDO pelo campo DIVISOR, e o resultado será armazenado no campo DIVIDENDO;
- Dividir o conteúdo do campo DIVIDENDO por 3, e o resultado será armazenado no campo DIVIDENDO.

#### Formato 2:

```
DIVIDE <campon1> | <literaln1> INTO <campon2> | <literaln2>
      GIVING <campo3> [ROUNDED]...
      [ON SIZE ERROR <comando1>]
      [NOT ON SIZE ERROR <comando2>]
```

#### Exemplo:

```
DIVIDE DIVISOR          INTO  DIVIDENDO GIVING  QUOCIENTE.
DIVIDE 3                INTO  DIVIDENDO GIVING  QUOCIENTE.
```

#### Explicação do exemplo acima:

- Dividir o conteúdo do campo DIVIDENDO pelo campo DIVISOR, e o resultado será armazenado no campo QUOCIENTE;
- Dividir o conteúdo do campo DIVIDENDO por 3, e o resultado será armazenado no campo QUOCIENTE.

#### Formato 3:

```
DIVIDE <campon1> | <literaln1> BY <campon2> | <literaln2>
      GIVING <campo3> [ROUNDED]...
      [ON SIZE ERROR <comando1>]
      [NOT ON SIZE ERROR <comando2>]
```

#### Exemplo:

```
DIVIDE DIVIDENDO       BY  DIVISOR  GIVING  QUOCIENTE.
DIVIDE DIVIDENDO       BY  3        GIVING  QUOCIENTE.
```

**Explicação do exemplo acima:**

- Dividir o conteúdo do campo DIVIDENDO pelo campo DIVISOR, e o resultado será armazenado no campo QUOCIENTE;
- Dividir o conteúdo do campo DIVIDENDO por 3, e o resultado será armazenado no campo QUOCIENTE.

**Formato 4:**

```
DIVIDE <campon1> | <literaln1> BY <campon2> | <literaln2>  
GIVING <campo3> [ROUNDED] REMAINDER <campon4>  
[ON SIZE ERROR <comando1>]  
[NOT ON SIZE ERROR <comando2>]
```

**Exemplo:**

```
DIVIDE DIVISOR INTO DIVIDENDO GIVING QUOCIENTE REMAINDER RESTO.  
DIVIDE 3 INTO DIVIDENDO GIVING QUOCIENTE REMAINDER RESTO.
```

**Explicação do exemplo acima:**

- Dividir o conteúdo do campo DIVIDENDO pelo campo DIVISOR, o resultado será armazenado no campo DIVIDENDO e o resto da operação será armazenado no campo RESTO;
- Dividir o conteúdo do campo DIVIDENDO por 3, o resultado será armazenado no campo DIVIDENDO e o resto da operação será armazenado no campo RESTO;.

**Formato 5:**

```
DIVIDE <campon1> | <literaln1> BY <campon2> | <literaln2>  
GIVING <campo3> [ROUNDED] REMAINDER <campon4>  
[ON SIZE ERROR <comando1>]  
[NOT ON SIZE ERROR <comando2>]
```

**Exemplo:**

```
DIVIDE DIVIDENDO BY DIVISOR GIVING QUOCIENTE REMAINDER RESTO.  
DIVIDE DIVIDENDO BY 3 GIVING QUOCIENTE REMAINDER RESTO.
```

**Explicação do exemplo acima:**

- Dividir o conteúdo do campo DIVIDENDO pelo campo DIVISOR, o resultado será armazenado no campo DIVIDENDO e o resto da operação será armazenado no campo RESTO;
- Dividir o conteúdo do campo DIVIDENDO por 3, o resultado será armazenado no campo DIVIDENDO e o resto da operação será armazenado no campo RESTO;.

### 13.2.4 Multiply

Efetua a multiplicação de operandos numéricos.

#### Formato 1:

```
MULTIPLY <campon1> | <literaln1> BY <campon2> [ROUNDED]  
[ON SIZE ERROR <comando1>]  
[NOT ON SIZE ERROR <comando2>]
```

#### Exemplo:

```
MULTIPLY FATOR1 BY FATOR2.  
MULTIPLY 2 BY FATOR1.
```

#### Explicação do exemplo acima:

- Multiplicar o conteúdo do campo FATOR1 pelo conteúdo do campo FATOR2, o resultado será armazenado no campo FATOR2;
- Multiplicar 2 pelo conteúdo do campo FATOR2, o resultado será armazenado no campo FATOR2..

#### Formato 2:

```
MULTIPLY <campon1> | <literaln1> BY <campon2> | <literaln2>  
GIVING <campo3> [ROUNDED]...  
[ON SIZE ERROR <comando1>]  
[NOT ON SIZE ERROR <comando2>]
```

#### Exemplo:

```
MULTIPLY FATOR1 BY FATOR2 GIVING PRODUTO.  
MULTIPLY 2 BY FATOR1 GIVING PRODUTO.
```

#### Explicação do exemplo acima:

- Multiplicar o conteúdo do campo FATOR1 pelo conteúdo do campo FATOR2, o resultado será armazenado no campo PRODUTO;
- Multiplicar 2 pelo conteúdo do campo FATOR1, o resultado será armazenado no campo PRODUTO.

**13.2.5 Subtract**

Efetua a subtração de operandos numéricos.

**Formato 1:**

```
SUBTRACT <campon1> | <literaln1> FROM <campon2> [ROUNDED]
        [ON SIZE ERROR <comando1>]
        [NOT ON SIZE ERROR <comando2>]
```

**Exemplo:**

```
SUBTRACT      1                      FROM      CONTADOR.
SUBTRACT      VALOR1          VALOR2    FROM      VALOR3.
```

**Explicação do exemplo acima:**

- Subtrair 1 do conteúdo do campo CONTADOR o resultado será armazenado no campo CONTADOR.
- Subtrair a soma dos campos VALOR1 e VALOR2 do campo VALOR3, o resultado será armazenado no campo VALOR3.

**Formato 2:**

```
SUBTRACT <campon1> | <literaln1> FROM <campon2> [ROUNDED]
        GIVING <campo3> [ROUNDED]...
        [ON SIZE ERROR <comando1>]
        [NOT ON SIZE ERROR <comando2>]
```

**Exemplo:**

```
SUBTRACT      1                      FROM      CONTADOR GIVING RESTO.
SUBTRACT      VALOR1          VALOR2    FROM      VALOR3 GIVING RESTO.
```

**Explicação do exemplo acima:**

- Subtrair 1 do conteúdo do campo CONTADOR o resultado será armazenado no campo RESTO;
- Subtrair a soma dos campos VALOR1 e VALOR2 do campo VALOR3, o resultado será armazenado no campo RESTO.

**Formato 3:**

```
SUBTRACT CORRESPONDING | CORR <item1> FROM <item2>      [ROUNDED]
        [ON SIZE ERROR <comando1>]
        [NOT ON SIZE ERROR <comando2>]
```

**Exemplo:**

```
01  ITEM1.  
   05  VALOR1                PIC  9(02)  VALUE  10.  
01  ITEM2.  
   05  VALOR1                PIC  9(02)  VALUE  30.  
  
PROCEDURE  DIVISION.  
  SUBTRACT  CORR  ITEM1  FROM  ITEM2.
```

**Explicação do exemplo acima:**

- Subtrair o item elementar VALOR1 do item de grupo ITEM1 de seus respectivo item elementar do item de grupo ITEM2, o resultado será armazenado no item elementar VALOR1 do item de grupo ITEM2.

### 13.3 Comandos de decisões

#### 13.3.1 IF

Permite que o programador especifique uma série de comandos para o caso de uma condição ser verdadeira. Opcionalmente, uma série de comandos pode ser especificada no caso de a condição ser falsa.

Formato:

```
IF condição
  {NEXT SENTENCE/comando...}
{ELSE
  [NEXT SENTENCE/comando...]}
```

A primeira série de comandos será executada se a condição for verdadeira e a segunda se a primeira condição for falsa.

Seja a condição verdadeira ou falsa, a próxima sentença somente será processada após a execução da série de comandos apropriada. Exceções a esta regra são:

- Ocorrência de um comando GO TO;
- O fluxo normal de o programa ser interrompido por causa de um comando PERFORM ativo.

A cláusula NEXT SENTENCE especifica que determinada condição será executada somente no caso contrário à afirmação estabelecida pelo IF e que o mesmo nada faça se a condição for obedecida.

**Exemplos:**

```
IF CONTA-MAT          EQUAL          123701
  ADD                  PR-UNITARIO TO ACUM-VALOR-CONTA.

IF STATUS-ARQ-FUN    EQUAL          '00'
  NEXT SENTENCE

ELSE
  PERFORM             999-99-TRATAR-ERRO-STATUS.
```

O comando IF, também permite o tratamento de testes compostos, que veremos logo a seguir.

### 13.3.1.1 Testes compostos

Um programa COBOL pode executar qualquer tipo de teste durante o seu processamento, por exemplo:

```
IF  COD-MAT          GREATER          10000
   PERFORM          004-00-TRATAR-CODMAT-MAIOR
ELSE
   PERFORM          004-00-TRATAR-CODMAT-MAIOR.
```

O programa indica a existência de uma decisão em seu processamento através do comando 'IF', seguido de comandos que contenham um teste e também o que fazer conforme o resultado do teste, obtendo com isso uma sentença condicional.

#### Exemplo 1:

```
IF  COD-MAT          LESS          05000
   ADD              1              TO          CONTA-BAIXA
ELSE
   ADD              1              TO          CONTA-ALTA.
```

#### Exemplo 2:

```
IF  COD-MAT          IS          NUMERIC
   COMPUTE          VALOR = VALOR * 1,05
ELSE
   PERFORM          999-99-TRATAR-ERRO.
```

Toda sentença condicional possui pelo menos 4 (quatro) elementos, que são: comando IF, teste, ação verdadeira e ação falsa.

Analisando os exemplos acima, temos:

Exemplo	IF	Teste	Ação Verdadeira	Ação Falsa
01	IF	COD-MAT LESS 05000	ADD 1 TO CONTA-BAIXA	ADD 1 TO CONTA-ALTA
02	IF	COD-MAT IS NUMERIC	COMPUTE VALOR = VALOR * 1,05	PERFORM 999-99-TRATAR-ERRO

Existem 5(cinco)tipo de testes condicionais:

- Teste de classe;
- Teste de nome de condição;
- Teste de relação condicional;
- Teste de sinal; e;
- Teste condição composta.

### 13.3.1.1.1 Teste de classe

Testa o conteúdo de um campo, para verificar se o mesmo é alfabético ou numérico.

#### Formato.

```
IF IDENTIFICADOR IS NUMERIC
IS NOT ALPHABETIC
```

#### Exemplo 1:

```
IF COD-MAT-R IS NUMERIC
PERFORM 005-05-REGISTRO-OK
ELSE
MOVE 'CAMPO NAO NUMERICO' TO WS-MENS-ERRO
PERFORM 999-99-TRATAR-ERRO.
```

#### Exemplo 2:

```
IF NOME-MAT-R IS ALFABETIC
PERFORM 005-05-REGISTRO-OK
ELSE
MOVE 'CAMPO NAO ALFABETICO' TO WS-MENS-ERRO
PERFORM 999-99-TRATAR-ERRO.
```

#### Observação:

Ao efetuar o teste em um campo compactado, para verificar se o conteúdo é numérico, verificar se o campo está sinalizado, caso esteja não esquecer de defini-lo como PIC S9, por exemplo:

```
01 CAMPO-COMPACTADO PIC S(09)V99 COMP-3.
```

### 13.3.1.1.2 Teste de nome de condição

Teste definido pelo uso do nível '88'.

#### Definição:

```
01 REG-CADFUNC.
03 MATRICULA          PIC  9(05).
03 NOME               PIC  X(30).
03 DATA-NASC.
05 DIA-NASC          PIC  9(02).
05 MES-NASC          PIC  9(02).
05 ANO-NASC          PIC  9(04).
03 ESCOLARIDADE      PIC  9(01).

PROCEDURE DIVISION.
IF ESCOLARIDADE          EQUAL 1
PERFORM          NNN-NN-PRIMEIRO-GRAU
ELSE
IF ESCOLARIDADE          EQUAL 2
PERFORM          NNN-NN-SEGUNDO-GRAU
ELSE
IF NACIONALIDADE        EQUAL 3
PERFORM NNN-NN-SUPERIOR
ELSE
PERFORM 999-99-TRATAR-ERRO.
```

Usando palavras significativas, definidas pelo nível '88'.

#### Definição:

```
01 REG-CADFUNC.
03 MATRICULA          PIC  9(05).
03 NOME               PIC  X(30).
03 DATA-NASC.
05 DIA-NASC          PIC  9(02).
05 MES-NASC          PIC  9(02).
05 ANO-NASC          PIC  9(04).
03 ESCOLARIDADE      PIC  9(01).
88 PRIMEIRO-GRAU      VALUE 1.
88 SEGUNDO-GRAU      VALUE 2.
88 SUPERIOR          VALUE 3.
88 ERRO              VALUE 4 THRU 9.

PROCEDURE DIVISION.
IF PRIMEIRO-GRAU
PERFORM          NNN-NN-PRIMEIRO-GRAU
ELSE
IF SEGUNDO-GRAU
PERFORM          NNN-NN-SEGUNDO-GRAU
ELSE
IF SUPERIOR
PERFORM NNN-NN-SUPERIOR
ELSE
PERFORM 999-99-TRATAR-ERRO.
```

### 13.3.1.1.3 Teste de relação condicional

Efetua comparação entre dois operandos.

**Formato:**

```
IF (IDENTIFICADOR-1)      OPERADOR (IDENTIFICADOR-2)
(LITERAL-1)              DE (LITERAL-2)
(EXPRESSÃO ARITMETICA)   RELAÇÃO (EXPRESSÃO ARITMETICA-2)
```

Segue abaixo os operadores de relação mais utilizados e seus respectivos significados:

Operador de relação	Significado
[IS] GREATER [THAN]	MAIOR QUE
[IS] NOT GREATER [THAN]	NÃO MAIOR QUE
[IS] > [THAN]	MAIOR QUE
[IS] NOT > [THAN]	NÃO MAIOR QUE
[IS] LESS [THAN]	MENOR QUE
[IS] NOT LESS [THAN]	NÃO MENOR QUE
[IS] < [THAN]	MENOR QUE
[IS] NOT < [THAN]	NÃO MENOR QUE
[IS] EQUAL [TO]	IGUAL
[IS] NOT EQUAL [TO]	DIFERENTE
[IS] = [TO]	IGUAL
[IS] NOT = [TO]	DIFERENTE

**Exemplo:**

```
IF ACUM-LINHA          GREATER  60
   PERFORM             NNN-NN-CABECALHOS.

IF COD-PECA           EQUAL     015000
   MOVE PR-UNITARIO   TO        WS-PRECO...

IF PR-UNITARIO        NOT LESS  15,00
   PERFORM             NNN-NN-VALOR-ACIMA.
```

#### 13.3.1.1.4 Teste de sinal

Determinar o valor algébrico de um operando aritmético.

**Formato:**

```
IF (IDENTIFICADOR) IS (POSITIVE)
   (EXPRESSÃO ARITMETICA) IS NOT (NEGATIVE)
                                (ZERO)
```

**Exemplo:**

```
IF PR-UNITARIO IS POSITIVE
MOVE PR-UNITARIO TO PR-PARCIAL
ADD PR-UNITARIO TO ACUM-VAL-POSITIVO.
```

### 13.3.1.1.5 Teste de condição composta

Determinado pelas palavras reservadas AND e OR , permite testar várias condições simples, ao mesmo tempo, da seguinte maneira:

- a) AND → A condição composta só será verdadeira, se todas as condições simples forem atendidas, caso alguma não seja atendida, então , podemos dizer que a condição composta é falsa;
- b) OR → A condição composta só será verdadeira, se alguma das condições simples for atendida, caso, todas as condições não atendam, então, podemos dizer que a condição composta é falsa.

#### Exemplos:

```
IF (COD-MAT          GREATER  19999)  AND
   (QTD-MAT          GREATER  ZEROS)
PERFORM      005-00-GRAVAR-REGISTRO.
```

```
IF (COD-MAT          LESS      10001)  OR
   (COD-MAT          GREATER  19999)
PERFORM      005-00-GRAVAR-REGISTRO.
```

#### Explicação dos exemplos acima:

- No primeiro exemplo, notamos que para que a condição composta seja verdadeira, é necessário que o campo COD-MAT seja maior que 19999 e o campo QTD-MAT seja maior que zeros;
- No segundo exemplo, notamos que para que a condição composta seja verdadeira, é necessário que o campo COD-MAT seja menor que 10001 ou maior que 19999.

#### Observação:

O operador NOT também pode aparecer nas condições compostas.

#### Formato:

```
[NOT] condição-1 {[AND/OR] [NOT] condição-2}
```

onde condição-1 e condição-2, podem ser:

1. uma condição simples, por exemplo: A > B;
2. negação de uma condição simples, por exemplo: NOT (A > B);
3. condição composta, por exemplo: (A > B) AND (C IS POSITIVE);
4. a negação de uma condição composta, por exemplo: NOT ((A > B) AND (C IS POSITIVE));
5. Qualquer combinação acima.

As seguintes regras definem o cálculo das condições compostas:

- a) O par de parênteses mais interno é tratado em primeiro lugar;
- b) As expressões aritméticas são reduzidas a um único valor numérico;
- c) As relações condicionais, testes de classe, testes de sinal e testes de nomes de condição são calculados;
- d) Todos os NOT são efetuados da esquerda para a direita;
- e) Todos os AND são efetuados da esquerda para a direita;
- f) Todos os OR são efetuados da esquerda para a direita;
- g) O próximo par de parênteses é tratado de acordo com os itens b) e f).

**Exemplos:**

```

IF (COD-MAT          EQUAL    10001)      OR
   (QTD-MINIMA       LESS     15)        OR
   (QTD-PECA         NOT GREATER 1000)
PERFORM 005-06-COMPRAR-MATERIAL.

IF (COD-MAT          EQUAL    15000)      AND
   (QTD-PECA         EQUAL    0)
PERFORM 005-04-COD15000-COM-ERRO
ELSE
PERFORM 005-05-COD15000-OK.

IF NOT (QTD-MINIMA AND QTD-MAXIMA EQUAL ZEROS)      OR
      (QTD-PECA          NEGATIVE)                  OR
      (PR-UNITARIO      EQUAL ZEROS)
PERFORM 999-99-TRATAR-ERRO.

IF (QTD-MAT          EQUAL    QTD-EST)      AND
   (STATUS-MAT       EQUAL    'I')          OR
   (PR-UNITARIO      EQUAL    ZERO)
PERFORM PROCESSA-REGISTRO.

```

Possibilidades de execução da rotina de acordo com o especificado acima:

Campos e seus respectivos conteúdos				PROCESSA REGISTRO?
QTD-MAT	QTD-EST	STATUS-MAT	PR-UNITARIO	
10	10	'I'	1,00	Sim
10	11	'I'	1,00	Não
10	11	'I'	0,00	Sim
10	10	'M'	1,00	Não
6	3	'M'	0,00	Sim
6	6	'M'	1,00	Não

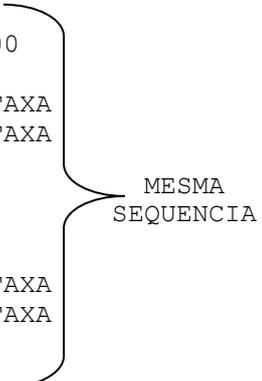
### 13.3.1.2 Concatenação de IF (ninho de IF)

A concatenação ocorre quando há uma intercalação de IFs dentro de uma mesma seqüência.

**Exemplo:**

```
IF  COD-MAT                NOT GREATER      10000
IF  PR-UNITARIO            NOT EQUAL        0
    MOVE 1,10              TO WS-TAXA
    COMPUTE PR-UNITARIO = PR-UNITARIO * WS-TAXA
ELSE
    PERFORM 009-99-ERRO-ABAIXO
ELSE
IF  PR-UNITARIO            NOT EQUAL        0
    MOVE 1,15              TO WS-TAXA
    COMPUTE PR-UNITARIO = PR-UNITARIO * WS-TAXA
ELSE
    PERFORM 009-99-ERRO-ACIMA.

PERFORM 002-01-LER-REGISTRO.
```



**Observação:**

Só podemos usar uma cláusula ELSE para cada IF.

## 13.3.2 Evaluate

O comando EVALUATE implementa a estrutura 'CASE' e apresenta muitas opções de uso.

### Formato:

```
EVALUATE sujeito-1 [ALSO sujeito-2]...
    {{WHEN objeto-1 [ALSO objeto-2]...}comando-imperativo-1}...
    [WHEN OTHER comando-imperativo-2]
END-EVALUATE.
```

### Objetivo:

{	<b>ANY</b>	condição	}										
	<b>TRUE</b>												
	<b>FALSE</b>												
	<b>[NOT]</b>	<table style="border: none;"> <tr> <td style="font-size: 2em; vertical-align: middle; padding-right: 5px;">{</td> <td style="padding: 5px;">                 identificador-1                  literal-1                  exp-aritmetica-1             </td> <td style="font-size: 2em; vertical-align: middle; padding: 0 5px;">}</td> <td style="padding: 5px;"> <table style="border: none;"> <tr> <td style="font-size: 2em; vertical-align: middle; padding-right: 5px;">{</td> <td style="padding: 5px;"> <b>THROUGHT</b>                  ou  <b>THRU</b> </td> <td style="font-size: 2em; vertical-align: middle; padding: 0 5px;">}</td> </tr> </table> </td> <td style="font-size: 2em; vertical-align: middle; padding: 0 5px;">}</td> <td style="padding: 5px;"> <table style="border: none;"> <tr> <td style="font-size: 2em; vertical-align: middle; padding-right: 5px;">{</td> <td style="padding: 5px;">                 identificador-2                  literal-1                  exp-aritmetica-2             </td> <td style="font-size: 2em; vertical-align: middle; padding: 0 5px;">}</td> </tr> </table> </td> </tr> </table>		{	identificador-1 literal-1 exp-aritmetica-1	}	<table style="border: none;"> <tr> <td style="font-size: 2em; vertical-align: middle; padding-right: 5px;">{</td> <td style="padding: 5px;"> <b>THROUGHT</b>                  ou  <b>THRU</b> </td> <td style="font-size: 2em; vertical-align: middle; padding: 0 5px;">}</td> </tr> </table>	{	<b>THROUGHT</b> ou <b>THRU</b>	}	}	<table style="border: none;"> <tr> <td style="font-size: 2em; vertical-align: middle; padding-right: 5px;">{</td> <td style="padding: 5px;">                 identificador-2                  literal-1                  exp-aritmetica-2             </td> <td style="font-size: 2em; vertical-align: middle; padding: 0 5px;">}</td> </tr> </table>	{
{	identificador-1 literal-1 exp-aritmetica-1	}	<table style="border: none;"> <tr> <td style="font-size: 2em; vertical-align: middle; padding-right: 5px;">{</td> <td style="padding: 5px;"> <b>THROUGHT</b>                  ou  <b>THRU</b> </td> <td style="font-size: 2em; vertical-align: middle; padding: 0 5px;">}</td> </tr> </table>	{	<b>THROUGHT</b> ou <b>THRU</b>	}	}	<table style="border: none;"> <tr> <td style="font-size: 2em; vertical-align: middle; padding-right: 5px;">{</td> <td style="padding: 5px;">                 identificador-2                  literal-1                  exp-aritmetica-2             </td> <td style="font-size: 2em; vertical-align: middle; padding: 0 5px;">}</td> </tr> </table>	{	identificador-2 literal-1 exp-aritmetica-2	}		
{	<b>THROUGHT</b> ou <b>THRU</b>	}											
{	identificador-2 literal-1 exp-aritmetica-2	}											

O propósito do comando é explorar as seguintes regras:

- a) Associar com o comando EVALUATE a lista de sujeitos e a de objetos;
- b) A lista de sujeitos é especificada entre a palavra EVALUATE e o primeiro aparecimento do WHEN. O sujeito pode ser um identificador ou um literal. Pode ser também uma expressão / condição aritmética ou as palavras reservadas TRUE / FALSE. Assim, na avaliação, o sujeito conterá um valor numérico, um valor não numérico ou um valor condicional (mostrado pelo TRUE, FALSE ou expressão condicional). Mostraremos alguns exemplos especificando o sujeito:
  1. TRUE – A lista de sujeitos consiste apenas de um elemento o qual avaliará a condição ao valor TRUE;
  2. CODIGO-TRANS – Assume que CODIGO-TRANS é um identificador. Neste caso, também, a lista de sujeitos consiste de apenas um elemento que avaliará o valor de CODIGO-TRANS;
  3. CODIGO-TRANS ALSO DATA-TRANS ALSO CODIGO-CLIENTE – Aqui a lista de sujeitos consiste de três elementos. Os valores dos sujeitos não necessitam ser da mesma classe. Por exemplo, CODIGO-TRANS pode ser numérico enquanto CODIGO-CLIENTE é alfanumérico. Quando a lista contiver mais de um sujeito, a posição do mesmo dentro da lista é importante. Esta posição é designada de ORDINAL.
- c) Cada frase WHEN especifica uma lista de objetos. O valor do objeto pode ser um dos seguintes: um valor numérico / não numérico, uma faixa de valores numéricos / não numéricos, um valor condicional ou qualquer valor. Identificador-1 / literal-1 / expressão aritmética-1 indicam um valor simples numérico / não numérico quanto à frase THROUGHT / THRU é omitida. Quando a frase THROUGHT / THRU for especificada, uma faixa de valores numérico / não numérico é indicada. Condição-1 / TRUE / FALSE indica um valor condicional para o objeto. Alguns exemplos de especificação de lista dos objetos estão a seguir:

1. 3 – O objeto consiste de apenas um elemento que especifica o valor 3;
2. 3 THRU 10 – Aqui também o objeto consiste de um elemento, sendo que este elemento avalia um conjunto de valores;
3. VAL-1 THRU VAL-2 ALSO ANY ALSO QTDE > 500 – A lista de objetos consta de três elementos. O primeiro avalia um conjunto de valores, o segundo avalia qualquer valor e o terceiro avalia uma condição de valor falso ou verdadeiro dependendo do valor de QTDE. Como no caso de uma lista de sujeitos, a posição ordinal de um objeto dentro da lista de objetos é importante;
4. Durante a execução o comando EVALUATE, os valores da lista de sujeitos são comparados com os valores da lista dos objetos na frase WHEN para estabelecer um *match* entre os dois. A comparação a ser processada é a seguinte:
  - ✓ O valor do sujeito é comparado com o valor / conjunto de valores do objeto na correspondência da posição ordinal;
  - ✓ No caso de um único objeto (numérico / não numérico, a comparação sujeito-objeto é feita do modo usual;
  - ✓ Quando um conjunto de valores for especificado para o objeto, a comparação sujeito-objeto resulta em verdadeiro, se o valor do sujeito pertencer ao conjunto;
  - ✓ No caso de valores condicionais, a comparação sujeito-objeto resulta em verdadeiro, se ambos avaliam o mesmo valor, isto é, ambos são verdadeiros ou ambos são falsos;
  - ✓ Se ANY for especificado para o objeto, a comparação sujeito-objeto sempre resulta em verdadeiro; e;
  - ✓ A lista de sujeitos é indicada para um *match* com a lista do objeto, se toda a correspondência de comparação sujeito-objeto resultar em verdadeiro.
5. Cada frase WHEN especifica uma lista de objetos. As frases WHEN são vistas como um *match* na ordem que eles aparecem dentro do comando EVALUATE. Entretanto, no resultado de um *match* o primeiro procedimento que segue a frase WHEN é selecionado para execução e o comando EVALUATE é encerrado. A frase WHEN OTHER, se especificada, é selecionada apenas se previamente o nome da frase WHEN não for escolhida.

O comando EVALUATE sem as frases ALSE torna-se muito simplificado porque é o caso em que existem um sujeito e um objeto (para cada WHEN). Esta forma de comando EVALUATE é útil na implementação da estrutura 'CASE' sem o uso do comando GO TO.

Os exemplos abaixo ilustram o uso do comando EVALUATE.

**Exemplo 1:**

MES e NR-DIAS são campos de dois dígitos numéricos inteiros. Os valores 1, 2, 3 etc para MES indicam respectivamente janeiro, fevereiro, março etc. Dependendo do valor do MES, queremos mover 30, 31 ou 28 para NR-DIAS:

```
EVALUATE      TRUE
      WHEN    MES      EQUAL    04  OU  06  OR  09  OR  11
            MOVE    30      TO NR-DIAS
      WHEN    MES      EQUAL    02
            MOVE    28      TO NR-DIAS
      WHEN    OTHER
            MOVE    31      TO NR-DIAS
END-EVALUATE .
```

**Exemplo 2:**

Vamos supor que NOTAS contém as notas obtidas pelos estudantes, variando de 0 a 100, e GRAU é um campo alfanumérico de uma posição. Queremos calcular o GRAU de acordo com critério mostrado na tabela abaixo:

Notas	Grau
080 – 100	A
060 – 079	B
045 – 059	C
030 – 044	D
000 – 029	E

```
EVALUATE      NOTAS
      WHEN    080 THRU 100      MOVE  'A'      TO  GRAU
      WHEN    060 THRU 079      MOVE  'B'      TO  GRAU
      WHEN    045 THRU 059      MOVE  'C'      TO  GRAU
      WHEN    030 THRU 044      MOVE  'D'      TO  GRAU
      WHEN    000 THRU 029      MOVE  'E'      TO  GRAU
      WHEN    OTHER              MOVE  'W'      TO  GRAU
END-EVALUATE .
```

**Observação:**

No exemplo acima, o literal 'W' foi movido para o campo GRAU, pois se trata de uma NOTA diferente das mencionadas, se tivermos certeza que as notas informadas serão entre 000 a 100, então não precisamos informar essa condição.

### Exemplo 3:

TIPO PRODUTO = 1	S	S	N	N
CATEGORIA = 1	S	N	S	N
COMISSÃO = 10%	S			
COMISSÃO = 8%		S		S
COMISSÃO = 12%			S	

A partir da tabela de decisões acima, mostraremos a diferença de codificação entre o comando IF e o comando EVALUATE.

#### Codificação com o uso do comando IF:

```

IF (TIPO-PRODUTO EQUAL 1) AND
(CATEGORIA EQUAL 1)
MOVE 10 TO COMISSAO
ELSE
IF (TIPO-PRODUTO EQUAL 1) AND
(CATEGORIA EQUAL 2)
MOVE 8 TO COMISSAO
ELSE
IF (TIPO-PRODUTO EQUAL 2) AND
(CATEGORIA EQUAL 1)
MOVE 12 TO COMISSAO
ELSE
IF (TIPO-PRODUTO EQUAL 2) AND
(CATEGORIA EQUAL 2)
MOVE 8 TO COMISSAO
ELSE
IF (TIPO-PRODUTO EQUAL 3) AND
(CATEGORIA EQUAL 1)
MOVE 10 TO COMISSAO
ELSE
IF (TIPO-PRODUTO EQUAL 3) AND
(CATEGORIA EQUAL 2)
MOVE 10 TO COMISSAO
ELSE
MOVE 0 TO COMISSAO.

```

#### Codificação com o uso do comando EVALUATE:

```

EVALUATE TIPO-PRODUTO ALSO CATEGORIA
WHEN 1 ALSO 1 MOVE 10 TO COMISSAO
WHEN 1 ALSO 2 MOVE 8 TO COMISSAO
WHEN 2 ALSO 1 MOVE 12 TO COMISSAO
WHEN 2 ALSO 2 MOVE 8 TO COMISSAO
WHEN 3 ALSO 1 MOVE 10 TO COMISSAO
WHEN OTHER MOVE 0 TO COMISSAO
END-EVALUATE.

```

**Exemplo 4:**

O comando EVALUATE do exemplo 3 também pode ser escrito da seguinte maneira:

```
EVALUATE      TIPO-PRODUTO      ALSO      CATEGORIA
  WHEN        1      ALSO 1
  WHEN        3      ALSO 1
                MOVE      10      TO  COMISSAO
  WHEN        1      ALSO 2
  WHEN        2      ALSO 2
                MOVE      8      TO  COMISSAO
  WHEN        2      ALSO 1
                MOVE      12     TO  COMISSAO
  WHEN        OTHER
                MOVE      0      TO  COMISSAO
END-EVALUATE.
```

Observe que quando a frase WHEN não possui um comando imperativo, o próximo procedimento é executado. Assim, se TIPO-PRODUTO = 1 e CATEGORIA = 1, 10 será movimentado COMISSAO.

## 13.4 Comandos Básicos

### 13.4.1 Accept

Executa uma operação de entrada.

#### Formato:

```
ACCEPT (IDENTIFICADOR) FROM SYSIN.  
CONSOLE.  
(NOME-MNEMÔNICO) .
```

#### Exemplos:

```
ACCEPT WS-DADOS FROM SYSIN.  
ACCEPT WS-DIA-SYS FROM CONSOLE.  
ACCEPT WS-DATE-SYS FROM DATE.
```

**NOME-MNEMÔNICO** = É um campo que executa uma operação de entrada do computador para um campo definido na Working-Storage.

#### Exemplo para solicitar uma data pela console:

```
DISPLAY `*PGM FUTU0010ABC* TECLE A DATA ` UPON CONSOLE.  
DISPLAY `DESEJADA NO FORMATO DD/MM/AAAA = ` UPON CONSOLE.
```

```
ACCEPT WS-DATE-SYS FROM CONSOLE.
```

#### Definição de WS-DATE-SYS

```
01 WS-DATE-SYS.  
05 WS-DIA-SYS PIC 9(02).  
05 FILLER PIC X(01).  
05 WS-MES-SYS PIC 9(02).  
05 FILLER PIC X(01).  
05 WS-ANO-SYS PIC 9(04).  
ou  
01 WS-DATE-SYS PIC X(10).
```

### 13.4.2 Alter

Utilizado para modificar um comando simples GO TO em qualquer lugar da PROCEDURE DIVISION, mudando assim a seqüência da execução dos comandos do programa.

#### Formato:

```
ALTER [parágrafo] TO [PROCEED TO] [nome do procedimento]
```

#### Observação:

No parágrafo especificado deve ter apenas um comando GO TO simples.

#### Exemplo:

```
001-00-TRATAMENTO-ALTER          SECTION.  
001-01-EXEMPLO-ALTER.  
    GO TO 001-02-ABERTURA.  
  
001-02-ABERTURA.  
    OPEN    INPUT  ARQUIVOE  
           OUTPUT ARQUIVOS.  
    ALTER  001-02-ABERTURA      TO PROCEED 001-03-LEITURA.  
  
001-03-LEITURA.  
    READ   ARQUIVOE  
    .  
    .  
    .
```

#### Explicação das linhas acima:

Na primeira vez em que o programa passar pelo parágrafo 001-01-EXEMPLO-ALTER, o programa irá fazer o tratamento de abertura dos arquivos, nas demais vezes irá direto para o tratamento de leitura.

### 13.4.3 Continue

Indica a ausência de um comando executável.

**Formato:**

CONTINUE

Pode ser usado em qualquer procedimento condicional ou imperativo.

**Exemplo:**

```
READ    ARQUIVO-1    AT    END    CONTINUE.
```

É usado quando o fim condicional do arquivo ocorrer durante a execução do comando READ.

O comando CONTINUE é funcionalmente similar ao comando EXIT, só que os mesmos têm objetivos diferentes. O EXIT deve ser usado para se ter um ponto final comum dentro de um parágrafo ou seção, o comando CONTINUE pode ser usado em qualquer parte quando um passo nulo for requerido. Pode também ser usado nos procedimentos IF na troca pela frase NEXT SENTENCE.

### 13.4.4 Display

Escrever dados em um dispositivo de saída.

**Formato:**

```
DISPLAY (LITERAL-1)          (LITERAL-2)          UPON    CONSOLE.  
        (IDENTIFICADOR-1) (IDENTIFICADOR-2)          SYSPUNCH.
```

**Observação:**

Quantidade de caracteres (bytes) para seu respectivo dispositivo:

- Console = 100 BYTES.
- Syspunch = 72 BYTES.
- Sysout = 120 BYTES.

**Exemplos:**

```
DISPLAY 'TOTAL DE REGISTROS = ' TOT-REG.  
DISPLAY ACUM-LIDOS ' = CODIGOS LIDOS'          UPON CONSOLE.
```

**13.4.5 End program**

Indica o fim de um programa fonte, definido no parágrafo PROGRAM-ID (IDENTIFICATION DIVISION).

**Formato:**

END       PROGRAM.

## 13.4.6 Examine

### Formato 1:

```

EXAMINE <campo1> TALLYING UNTIL FIRST
                  ALL
                  LEADING
                  [REPLACING BY } literal1
                              <literal2>]
    
```

### Formato 2:

```

EXAMINE <campo1> REPLACING ALL
                  LEADING
                  FIRST
                  UNTIL FIRST } <literal1>
                  BY           <literal2>
    
```

### Funções:

Função	Significado
TALLYING UNTIL FIRST	Conta os caracteres existentes no <campo1> até ser encontrado o primeiro caractere especificado no <literal1> e os substitui pelo <literal2> se a opção REPLACING tiver sido usada.
TALLYING ALL	Conta às ocorrências do <literal1> existentes em <campo1> e as substitui pelo <literal2>, se a opção REPLACING tiver sido usada.
TALLYING LEADING	Conta às ocorrências do <literal1> nas posições sucessivas e adjacentes do <campo1>, a partir da posição mais à esquerda deste, e as substitui pelo <literal2>, se a opção REPLACING tiver sido usada.
REPLACING ALL	Substitui as ocorrências do <literal1> em <campo1> pelo <literal2>.
REPLACING LEADING	Substitui as ocorrências do <literal1> nas posições sucessivas e adjacentes do <campo1> , a partir da posição mais à esquerda deste, pelo <literal2>.
REPLACING FIRST	Substitui apenas a primeira ocorrência do <literal1> existente no <campo1> pelo <literal2>.
REPLACING UNTIL FIRST	Substitui os caracteres existentes no <campo1> pelo <literal2>, até ser encontrado o primeiro caractere de <literal1>.

**Exemplos:**

- 1) EXAMINE CAMPO-01 TALLYING UNTIL FIRST 'X'.
  - Antes  
CAMPO-01 IDAXXXBXXX  
TALLY 00000
  - Depois  
CAMPO-01 **IDAXXXBXXX**  
TALLY 00003
- 2) EXAMINE CAMPO-02 TALLYING ALL 'X' REPLACING BY 'Y'.
  - Antes  
CAMPO-02 IDAXXXBXXX  
TALLY 00000
  - Depois  
CAMPO-02 **IDAYYYBYYY**  
TALLY 00006
- 3) EXAMINE CAMPO-03 TALLYING LEADING ZEROS.
  - Antes  
CAMPO-03 0000000844  
TALLY 00000
  - Depois  
CAMPO-03 **000000**844  
TALLY 00007
- 4) EXAMINE CAMPO-04 TALLYING ALL ',','.
  - Antes  
CAMPO-04 00,000,008,44  
TALLY 00000
  - Depois  
CAMPO-04 00,000,008,44  
TALLY 00003
- 5) EXAMINE CAMPO-05 REPLACING ALL 'C' BY 'A'.
  - Antes  
CAMPO-05 BCNCNC
  - Depois  
CAMPO-05 **BANANA**
- 6) EXAMINE CAMPO-06 REPLACING LEADING ZEROS BY 5.
  - Antes  
CAMPO-06 0000840000
  - Depois  
CAMPO-06 **5555**840000
- 7) EXAMINE CAMPO-07 REPLACING FIRST 6 BY ZERO.
  - Antes  
CAMPO-07 0000765632
  - Depois  
CAMPO-07 00007**0**5632
- 8) EXAMINE CAMPO-08 REPLACING UNTIL FIRST 8 BY ZERO.
  - Antes  
CAMPO-08 0123487878
  - Depois  
CAMPO-08 **00000**87878

**Regras para o uso do EXAMINE:**

- a) Utilizado apenas para campos numéricos zonados, alfabéticos ou alfanuméricos;
- b) A opção 'TALLYING' gera um número inteiro, onde o valor é armazenado em um item binário denominado 'TALLY', que representa:
  - ✓ Quantas vezes ocorre uma determinada literal com o uso da opção 'ALL';
  - ✓ Quantas vezes ocorre uma determinada literal, antes de encontrar um campo diferente desta literal, com o uso da opção 'LEADING';
  - ✓ Número de caracteres diferentes de uma determinada literal, até o primeiro caractere igual à literal a ser encontrada, com o uso da opção 'UNTIL FIRST'.
- c) No caso de ser usada à opção 'REPLACING' (alterar):
  - ✓ Com a opção 'ALL', cada literal será substituído pelo respectivo literal de alteração;
  - ✓ Com a opção 'LEADING', a substituição pelo respectivo caractere de alteração, termina no momento em que é encontrado um caractere diferente da literal a ser substituída;
  - ✓ Com a opção 'UNTIL FIRST' a substituição pelo respectivo caractere de alteração termina no momento em que é encontrado o literal a ser substituído; e;
  - ✓ Com a opção 'FIRST', a primeira literal que aparecer será substituída pela respectiva literal de alteração.

#### 13.4.7 Exhibit

Tem a finalidade de mostrar o conteúdo dos campos.

##### Formato:

```
EXHIBIT NAMED          (identificador 1) (identificador 2)
          CHANGED NAMED (literal não numérica)
          CHANGED
```

##### Definições quanto ao formato:

- NAMED - Mostra o conteúdo do campo todas as vezes que passa pelo comando ;
- CHANGED NAMED - Mostra o conteúdo do campo somente na troca conteúdo;
- CHANGED – Mostra o conteúdo dos campos só na troca de valores em forma de colunas.

##### Observações:

- O EXHIBIT não pode ser utilizado para contadores especiais;
- O EXHIBIT mostra o nome do campo e em seguida o seu respectivo conteúdo.

##### Exemplos:

```
EXHIBIT NAMED CAMPO-A.
EXHIBIT CHANGED NAMED CAMPO-A.
EXHIBIT CHANGED CAMPO-A.
EXHIBIT CHANGED CAMPO-1 CAMPO-2 CAMPO-3.
```

**13.4.8 Exit**

Ponto comum de finalização de uma série de procedimentos (comandos)..

O comando 'EXIT' deve ser precedido por um nome de parágrafo e deve ser único.

O programa pode ter vários EXITS associados com o comando 'PERFORMS'.

**Formato:**

nome do parágrafo. EXIT.

**Exemplo:**

```
001-00-INICIO.  
    PERFORM 002-00-AAAA THRU 002-99-CCCC.  
    .....  
002-00-AAAA.  
    .....  
    .....  
002-99-AAAA. EXIT.  
  
002-00-BBBB.  
    .....  
    .....  
002-99-BBBB. EXIT.  
  
002-00-CCCC.  
    .....  
    .....  
002-99-CCCC. EXIT.
```

**13.4.9 Go to**

Permite a transferência de uma parte do programa para outra.

**Formato:**

- Desvio incondicional.  
GO TO (nome do endereço).
- Desvio sob condição.  
GO TO (nome do endereço 1)  
(nome do endereço 2)  
DEPENDING ON (identificador).

**Exemplo 1:**

```
LEITURA.  
.....  
.....  
MOVE CAMPO TO DADOS.  
WRITE FITA.  
GO TO LEITURA.
```

**Exemplo 2:**

```
IF CODIGO EQUAL 10  
GO TO ROTINA-10.  
IF CODIGO EQUAL 20  
GO TO ROTINA-20.  
IF CODIGO EQUAL 30  
GO TO ROTINA-30.  
ou  
GO TO ROTINA-10  
ROTINA-20  
ROTINA-30 DEPENDING ON CODIGO.  
WRITE FITA.
```

**Explicação do exemplo 2:**

O processamento será desviado para ROTINA-10 se CÓDIGO for 10, ROTINA-20 se o CÓDIGO FOR 20 e ROTINA-30 se o CÓDIGO for 30, e se for um CODIGO diferente destes, passará para a próxima instrução.

### 13.4.10 Goback

Termina o processamento de um programa ou o processamento de uma ligação entre programas. Pode substituir o comando 'STOP RUN'.

#### Exemplo 1:

```
IDENTIFICATION DIVISION.  
    ...  
    ...  
    ...  
PROCEDURE      DIVISION.  
    ...  
    ...  
    ...  
    CALL 'PROGB' USING DADOS.  
    ...  
    ...  
    ...  
GOBACK.
```

#### Exemplo 2:

```
IDENTIFICATION DIVISION.  
    ...  
    ...  
    ...  
LINKAGE        SECTION.  
    ...  
    ...  
    ...  
PROCEDURE      DIVISION.  
    ...  
    ...  
    ...  
GOBACK.
```

**13.4.11 Initialize**

O propósito deste comando é inicializar determinado campo, que possa ser item de grupo ou item elementar.

Opção	Significado
ALPHABETIC	Inicializa campos da categoria alfabética.
ALPHANUMERIC	Inicializa campos da categoria alfanumérica.
NUMERIC	Inicializa campos da categoria numérica.
ALPHANUMERIC-EDITED	Inicializa campos da categoria alfanumérica-editada.
NUMERIC-EDITED	Inicializa campos da categoria numérica-editada.

**Formato:**

```
INITIALIZE <identificador 1>...  
  [REPLACING ALPHABETIC  
    ALPHANUMERIC  
    NUMERIC  
    ALPHANUMERIC-EDITED  
    NUMERIC-EDITED]  
  DATA BY <literal1> | <campo1>
```

Quando identificador 1 especificar um item de grupo, apenas os itens elementares que pertencem à categoria indicada pela frase REPLACING serão inicializados pelo valor indicado no identificador 2 ou literal 2.

**Exemplo 1:**

```
01  ITEM1.  
   05  CAMPO1          PIC  9(05).  
   05  CAMPO2          PIC  X(04).  
   05  CAMPO3          PIC  9(03).  
   05  CAMPO4          PIC  ZZZ9V99.  
PROCEDURE  DIVISION.  
  INITIALIZE ITEM1 REPLACING NUMERIC          DATA BY 50.
```

No exemplo acima somente os campos CAMPO1, CAMPO3 e CAMPO4 serão inicializados, pois são numéricos.

**Exemplo 2:**

```
01 ITEM1.  
05 CAMPO1          PIC 9(05).  
05 CAMPO2          PIC X(04).  
05 CAMPO3          PIC 9(03).  
05 CAMPO4          PIC ZZZ9,99.  
PROCEDURE DIVISION.  
INITIALIZE ITEM1 REPLACING NUMERIC          DATA BY 50.  
INITIALIZE ITEM1 REPLACING ALPHANUMERIC     DATA BY 'A'.  
INITIALIZE ITEM1 REPLACING NUMERIC-EDITED   DATA BY 54,2.
```

No exemplo acima todos os campos serão inicializados. O processo de inicialização é equivalente à execução de seqüência de comandos MOVE, onde são transferidos valores do identificador 2 ou literal 1 para os itens elementares do identificador 1. Os campos são inicializados na seqüência que os mesmos aparecem dentro do item de grupo do identificador 1.

Assim neste exemplo os campos serão inicializados assim:

Campo	Inicialização
CAMPO1	00050
CAMPO2	A
CAMPO3	050
CAMPO4	054,20

Quando o identificador 1 for um item elementar, então a inicialização será efetuada se a categoria mencionada na frase REPLACING se igualar ao do identificador 1.

Note que a frase REPLACING é opcional, sendo que se for omitida, todos os campos numéricos serão inicializados com zeros e todos os demais campos com brancos.

**Exemplo 3:**

```
01 ITEM1.  
05 CAMPO1          PIC 9(05).  
05 CAMPO2          PIC X(04).  
05 CAMPO3          PIC 9(03).  
05 CAMPO4          PIC ZZZ9,99.  
PROCEDURE DIVISION.  
INITIALIZE ITEM1.
```

Neste exemplo os campos CAMPO1 e CAMPO3 serão inicializados com ZEROS, enquanto CAMPO2 e CAMPO3 com brancos.

**Exemplo 4:**

```
01  ITEM1.  
   05  CAMPO1          PIC 9(05).  
   05  CAMPO2          PIC X(10).  
   05  CAMPO2-R REDEFINES CAMPO2.  
      10  CAMPOA        PIC 9(04).  
      10  CAMPOB        PIC 9(04)V99.  
PROCEDURE  DIVISION.  
  INITIALIZE ITEM1.
```

No exemplo acima mostramos um item de dado com a cláusula REDEFINES, então o item de dado ou qualquer item de dados subordinado não serão inicializados, ou seja, CAMPO1 será inicializado com zeros enquanto CAMPO2 será inicializado com brancos. Haverá dificuldade para que CAMPO2 seja preenchido com zeros, porque CAMPOA e CAMPOB não estão inicializados.

**Exemplo 5:**

```
01  ITEM1.  
   05  CAMPO1          PIC 9(05).  
   05  CAMPO2          PIC X(10).  
   05  CAMPO2-R REDEFINES CAMPO2.  
      10  CAMPOA        PIC 9(04).  
      10  CAMPOB        PIC 9(04)V99.  
PROCEDURE  DIVISION.  
  INITIALIZE CAMPO2-R.
```

Neste caso, identificador 1, por ter a cláusula REDEFINES ou por ser um item contendo a cláusula REDEFINES, CAMPOA e CAMPOB serão inicializados com zeros.

**Exemplo 6:**

```
01  TABELA.  
   05  CAMPO1          PIC 9(05) OCCURS 20 TIMES.  
PROCEDURE  DIVISION.  
  INITIALIZE TABELA.
```

Neste exemplo, todas as 20 ocorrências serão inicializadas com zeros.

**Observação:**

- O item FILLER (ou sem nome) e nome de índice não são afetados pelo comando INITIALIZE;

### 13.4.12 Inspect

Permite ao programador o exame de uma determinada seqüência de caracteres, podendo-se combinar as seguintes ações:

- Contar as ocorrências de um dados caractere;
- Substituir certo caractere por um alternativo; e;
- Qualificar e limitar as operações acima, condicionando-as à ocorrência de caracteres específicos.

#### Formato 1 (TALLYING):

```
INSPECT nome-de-dado-1 TALLYING nome-de-dado-2
FOR [CHARACTERS [ALL / LEADING] operando-3]
[[BEFORE / AFTER] INITIAL operando-4].
```

#### Formato 2 (TALLYING):

```
REPLACING [CHARACTERS [ALL / LEADING / FIRST] operando-5]
BY [[BEFORE / AFTER] INITIAL operando-6].
```

Nos formatos descritos **operando-n** pode ser:

- Um literal entre apóstrofos de um caractere;
- Uma constante figurativa que signifique um caractere; e;
- O nome-de-dado de um item de tamanho unitário.

A cláusula TALLYING, a cláusula REPLACING ou ambas, devem constar sempre de um INSPECT. Quando ambas ocorrerem TALLYING deve vir primeiro.

A cláusula TALLYING determina comparação caractere a caractere a partir da esquerda de nome-de-dado-1 com operando-3.

Se a cláusula AFTER INITIAL operando-4 estiver presente, então as comparações iniciarão apenas depois do ponto em que ocorrer, pela primeira vez, operando-4.

Se a cláusula BEFORE INITIAL operando-4 tiver sido especificado, as comparações terminam quando pela primeira vez for encontrado operando-4 (se não ocorrer, a comparação prosseguirá até o último caractere de nome-de-dado-1).

A cláusula REPLACING permite a substituição de caracteres sob condições especificadas.

Se AFTER INITIAL operando-6 estiver presente, então as substituições só serão efetuadas após a primeira ocorrência de operando-6; se, a cláusula BEFORE INITIAL operando-6 estiver presente, as substituições serão efetuadas até a primeira ocorrência de operando-6.

Se um INSPECT contiver TALLYING e REPLACING então tudo se passa como se dois comandos INSPECT, um contendo TALLYING e outro contendo REPLACING, tivessem sido declarados.

Quando TALLYING for usado o resultado da contagem é adicionado ao valor do nome-de-dado-2. O programador deve, portanto, mover zero para nome-de-dado-2 se quiser o valor absoluto da contagem.

Abaixo, descreveremos as funções de cada uma das palavras reservadas que aparecem no comando INSPECT:

- TALLYING – conta às ocorrências de um determinado caracteres;
- REPLACING – substitui determinado caractere;
- CHARACTERS – qualquer caractere do código EBCDIC (Extend Binary Coded Decimal Interchange Code);
- AFTER ou BEFORE INITIAL operando-n – contagem de caracteres do campo examinado que precedem ou sucedem operando-n para TALLYING;
- ALL operando-p – Conta ou substitui todas as ocorrências de operando-n, até operando-p ou após operando-p, se BEFORE INITIAL operando-p estiver presente. Na ausência de BEFORE / AFTER INITIAL, examina todo o campo reservado para nome-de-dado-2;
- LEADING operando-n – Conta ou substitui todas as ocorrências de operando-n, que aparecem no começo do campo sob exame, consecutivas, sem interrupção; e;
- FIRST operando-5 – Especifica que somente o primeiro caractere encontrado igual ao operando-5 participa da substituição pelo operando-6.

### Exemplo 1:

```
INSPECT  NOME      TALLYING  CONTADOR  FOR CHARACTERS
FOR      INITIAL 'P' .
```

Campos	Antes	Depois
NOME	AB5PS134)7(9	AB5P <b>S134</b> )7(9
CONTADOR	0	<b>8</b>

### Explicação do exemplo 1:

Foi efetuada uma contagem de caracteres que precedem o primeiro caractere 'P' encontrado, no caso, 8.

### Exemplo 2:

```
INSPECT  NOME      REPLACING  LEADING  ZEROS
BY       '9'      AFTER      INITIAL  'P' .
```

Campo	Antes	Depois
NOME	BA004P0020AB	BA004P <b>99</b> 20AB
	ABCPD5004P0	ABCPDP5004P0

### Explicação do exemplo 2:

Na primeira execução houve uma substituição dos caracteres '00' que estão após o caractere 'P', pelos caracteres '99', enquanto que na segunda execução, como não existe nenhum caractere '0' após o caractere 'P', não foi efetuada nenhuma substituição.

### Exemplo 3:

```
INSPECT      ITEM          TALLYING      CONTADOR FOR LEADING 'L'
BEFORE      INITIAL 'A' .
```

	Campos	Conteúdo		
ANTES	ITEM	LAGOA	ANALISTA	LANCA
	CONTADOR	0	0	0
DEPOIS	ITEM	LAGOA	ANALISTA	LANCA
	CONTADOR	1	0	1

### Explicação do exemplo 3:

Foi efetuada uma contagem de caracteres 'L' que precedem o primeiro caractere 'A'.

### Exemplo 4:

```
MOVE        ZEROS          TO          CONTADOR.
INSPECT     ITENS          TALLYING     CONTADOR FOR ALL 'L'
REPLACING   LEADING 'E' BY 'A' AFTER   INITIAL 'L' .
```

	Campos	Conteúdo		
ANTES	ITENS	SALARIO	PARALELA	SALADA
	CONTADOR	0	0	0
DEPOIS	ITEM	SALERIO	PARALELA	SALEDA
	CONTADOR	1	2	1

### Explicação do exemplo 4:

Foi efetuada uma contagem de caracteres 'L' que aparecem nos conteúdos do campo ITENS e ao mesmo tempo, houve uma substituição dos caracteres 'A' que precedam o primeiro caractere 'L' por 'E'.

**Exemplo 5:**

```
INSPECT      SENTIDO      REPLACING    ALL
             SEPARADORES  BY          TRANSFORMACAO.
```

	Campos	Conteúdo
Antes	SENTIDO	SAO[PAULO]-[RIO][DE][JANEIRO][
	SEPARADORES	'['
	TRANSFORMACAO	''
Depois	SENTIDO	SAO PAULO - RIO DE JANEIRO
	SEPARADORES	'['
	TRANSFORMACAO	''

**Explicação do exemplo 5:**

Foi efetuada uma verificação no campo SENTIDO, onde os caracteres iguais aos informados no campo SEPARACAO, foi substituído pelos caracteres encontrados no campo TRANSFORMACAO.

**13.4.13 Move**

Efetua a movimentação de dados dentro de um programa, ou seja, transfere o conteúdo de um determinado campo, para outro, podendo ser um ou mais.

**Formato 1:**

```
MOVE identificador-1 / literal-1 TO identificador-2
                                     identificador-3
                                     literal-2

MOVE CORRESPONDING identificador-1 TO identificador-2
                                     ou
MOVE CORR          identificador-1 TO identificador-2
```

**Observação:**

MOVE CORRESPONDING – movimenta dados entre itens com o mesmo nome.

**Exemplo:**

```
WORKING-STORAGE SECTION.
01 AREA-1                PIC X(06) VALUE 'FUTURE'.
01 AREA-2                PIC X(06).
01 AREA-3                PIC X(06).

PROCEDURE                DIVISION.

MOVE AREA-1              TO AREA-2.
MOVE AREA-1              TO AREA-2
                          AREA-3.
```

**Exemplos com literais figurativas:**

```
MOVE SPACES              TO AREA-1
                          AREA-4.

MOVE 'FUTURE'            TO AREA-1
                          AREA-4.

MOVE ZEROS                TO CAMPO-5
                          CAMPO-8.

MOVE 13579                TO CAMPO-5
                          CAMPO-8.
```

**Exemplos com MOVE CORRESPONDING (CORR):**

```
01 AREA-1.
   05 NUMERO              PIC 9.
   05 NOME                PIC X(30).
01 AREA-2.
   05 NOME                PIC X(30).
   05 NUMERO              PIC 9.

PROCEDURE                DIVISION.

MOVE CORRESPONDING AREA-1 TO AREA-2.
                                     ou
MOVE CORR          AREA-1 TO AREA-2.
```

**Regras para utilização do comando MOVE:**

1. Campos numéricos para campos numéricos ou de edição;
  - ✓ Os itens são alinhados pelo ponto decimal com geração de zeros ou truncamento em ambas as extremidades dependendo do número de significativos no campo-fonte;
  - ✓ Quando os tipos dos campos fonte e receptor diferem, a conversão para o tipo do receptor é feita; e;
  - ✓ Os itens podem receber tratamento de edição como supressão de zeros não significativos, inclusão do cifrão ou de um ponto decimal explícito de acordo com a PICTURE do campo receptor.
  
2. Campos não numéricos para campos não numéricos
  - ✓ Os caracteres são gravados no campo receptor da esquerda para a direita;
  - ✓ Se o campo receptor for mais longo que o campo fonte então será completado com brancos.
  
3. Se o campo receptor for o menor, o comando MOVE termina quando este estiver totalmente preenchido.

**Observação:**

Se o campo fonte e o campo receptor forem de algum modo superpostos (uso do REDEFINES) o resultado do comando MOVE será imprevisível.

**Tratamento do comando MOVE:**

Antes				Depois
Campo Emissor		Campo Receptor		
Picture	Valor	Picture	Valor	Valor
99V99	1234	S99V99	9876 -	1234 +
99V99	1234	99V9	987	123
S9V9	12 -	99V999	98765	01200
XXX	A2B	XXXXX	Y9X8W	A2Bbb **
9V99	123	99,99	87,65	01,23

\*\* b = espaço.

**Tabela de movimentação**

Emissor \ Receptor	GR	AL	NA	BI	NE	ANE	DI
grupo (GR)	S	S	S	S1	S1	S1	S1
alfabético (AL)	S	S	S	N	N	S	N
alfanumérico (AN)	S	S	S	S4	S4	S	S4
binário (BI)	S1	N	S2	S	S	S2	S
numérico editado (NE)	S	N	S	N	N	S	N
alfanumérico editado (ANE)	S	S	S	N	N	S	N
zeros	S	N	S	S3	S3	S3	S3
brancos (spaces)	S	S	S	N	N	S	N
high-values / low-values	S	N	S	N	N	S	N
all literal	S	S	S	S5	S5	S	S5
literal numérico	S1	N	S2	S	S	S2	S
literal não numérico	S	S	S	S5	S5	S	S5
decimal interno (DI)	S1	N	S2	S	S	S1	S

**S1** – o movimento é efetuado sem conversão;

**S2** – efetuado somente se o ponto decimal estiver colocado à direita do último dígito significativo;

**S3** – movimento numérico;

**S4** – o campo alfanumérico é tido como se fosse um campo numérico inteiro;

**S5** – o literal deve ter apenas caracteres numéricos e ser tratado como se fosse um campo numérico.

**13.4.14 On**

Este comando não está sendo mais utilizado.

Formato:

```
ON inteiro-1 AND EVERY inteiro-2
```

Para cada sentença 'ON' o compilador gera e associa um contador, inicializando com zero, cada vez que passa pelo 'ON' o contador é incrementado de um (1) e a condição de contagem (AND EVERY) é testada.

**Exemplo 1:**

```
ON 50  
GO TO FIM-PROCES  
ELSE  
MOVE A TO B.
```

Neste caso em que somente o inteiro-1 é declarado, a condição é satisfeita uma única vez, quando o contador atingir o número 50 a sentença 'GO TO FIM-PROCESS' será executada, em outros casos será feita a movimentação de 'A' para 'B'.

**Exemplo 2:**

```
ON 1  
MOVE '*' TO XAVE.
```

Neste caso só a primeira vez o campo XAVE será atualizado com '\*'.

**Exemplo 3:**

```
ON 1 AND EVERY 20  
WRITE RELATO FROM CABEC1 AFTER CANAL-1.
```

Neste caso na primeira vez e a cada 20 vezes a linha 'CABEC1' será impressa.

**Observação:**

A comando 'ON' não aceita o comando 'IF' ou um comando 'READ' que tenha a condição 'AT END' ou 'INVALID KEY'.

### 13.4.15 Perform

Transfere o controle para uma ou mais rotinas dentro de um programa e retorna após a execução.

#### Observações gerais:

Existem 2 (dois) tipos de denominação para o comando PERFORM. Ao usarmos a procedure-name1 e/ou a procedure-name2 denominamos 'OUT-OF-LINE PERFORM'; se não usarmos nenhuma procedure-name denominamos 'IN-LINE PERFORM'.

Se a procedure-name1 for omitida, o comando e o delimitador END-PERFORM deverão ser especificados. Se a procedure-name1 for usada, o comando1 e o delimitador END-PERFORM não deverão ser especificados.

#### Formato 1:

Executa uma rotina ou comandos no seu próprio escopo.

```
PERFORM [<procedure-name1>
        [THRU | THROUGH <procedure-name2>]] [<comando1>
[END-PERFORM]]
```

#### Exemplo de 'out-of-line' PERFORM:

```
PROCEDURE      DIVISION.
  PERFORM      ROTINA1.
  STOP        RUN.

ROTINA1.
  COMANDO1.
  COMANDO2.
  COMANDO3.
```

#### Exemplo de 'IN-LINE PERFORM':

```
PROCEDURE      DIVISION.
  PERFORM
    COMANDO1
    COMANDO2
    COMANDO3
  END-PERFORM.
```

#### Formato 2:

Executa uma rotina, ou comandos pertencentes ao seu próprio escopo, um determinado número de vezes.

```
PERFORM [<procedure-name1> [THRU | THROUGH <procedure-name2>]]
        [<literaln1> | <campon1> TIMES
        [<comando1>
[END-PERFORM]]
```

Opção	Significado
-------	-------------

<b>TIMES</b>	Número de vezes que uma rotina ou os comandos do seu escopo serão executados.
--------------	---

### Exemplo de 'IN-LINE PERFORM':

```
PROCEDURE DIVISION.
  PERFORM 10 TIMES.
  COMANDO1
  COMANDO2
  COMANDO3
  END-PERFORM.
```

### Formato 3:

Executa uma rotina, ou comandos pertencentes ao seu próprio escopo, enquanto uma condição for falsa.

```
PERFORM[<procedure-name1> [THRU | THROUGH <procedure2>]]
  [WITH TEST BEFORE | TEST AFTER | UNTIL <condição2>]
  [<comando1>]
[END-PERFORM]]
```

Opção	Significado
WITH TEST BEFORE	Faz com que o teste da <condição2> seja feito antes da tentativa de execução do PERFORM (opção default).
WITH TEST AFTER	Faz com que o teste da <condição2> seja feito após a primeira execução do PERFORM.
UNTIL	Especifica a condição que irá governar o PERFORM.

### Exemplo:

```
77 ACUM-VEZES PIC 9(01) VALUE 0.
PROCEDURE DIVISION.
  PERFORM EXIBICAO UNTIL ACUM-VEZES > 4.
  STOP RUN.
EXIBICAO.
  DISPLAY 'FUTURE SCHOOL' NO ADVANCING.
  ADD 1 TO ACUM-VEZES.
```

**Formato 4:**

Executa uma rotina, ou comandos conforme a variação de campos ou índices de uma tabela..

```
PERFORM [<procedure-name1> [THRU | THROUGH <procedure2>]]  
  [WITH TEST BEFORE | TEST AFTER  
  VARYING <campon1> | <indice1>  
  FROM <campon2> | <indice2> | <literaln1>  
  BY <campon3> | <literaln2> UNTIL <condicao1>  
  AFTER <campon4> | <indice3>  
  FROM <campon5> | <indice4> | <literaln3>  
  BY <campon6> | <literaln4> UNTIL <condicao2>]  
  [<comando1>  
[END-PERFORM]
```

Opção	Significado
VARYING	Índice o item a ter o seu valor modificado.
FROM	Valor inicial a ser atribuído ao item da opção VARYING.
BY	Valor a ser adicionado ou subtraído da opção VARYING.
UNTIL	Especifica a condição que irá governar o PERFORM.

### 13.4.16 Ready / Reset

O comando 'READY' tem a finalidade de mostrar os passos do programa assinalando os parágrafos pelo qual passou, ou seja, imprimir a seqüência de parágrafos ou Section pelo qual o programa passar, enquanto o comando 'RESET' encerra a operação do 'READY'.

#### Formato:

```
READY TRACE.  
RESET TRACE.
```

#### Exemplo:

```
PROCEDURE DIVISION.  
000-00-COMECO SECTION.  
    READY TRACE.  
    .  
    .  
    .  
000-00-EXIT.  
001-00-PRIM-ROTINA SECTION.  
    .  
    .  
    .  
001-00-EXIT.  
999-99-ULTIMA-ROTINA SECTION.  
    .  
    .  
    .  
    RESET TRACE.  
999-99-EXIT.
```

#### Observação:

Quando ocorrer um 'ABEND', os parágrafos após o mesmo, podem não aparecer.

**13.4.17 Stop run**

Provoca a suspensão temporária ou definitiva da execução de um programa .

**Formato:**

```
STOP    RUN.  
      ou  
STOP    RUN | <literal> ...
```

Opção	Significado
RUN	Encerra definitivamente o programa.
<literal>	Suspende a execução até que seja pressionada a tecla ENTER.

**Exemplo:**

```
999-00-ULTIMA-ROTINA  SECTION.  
CLOSE  ARQUIVO  
        RELATO.  
STOP   RUN.
```

## 13.4.18 String

Concatena, parcial ou totalmente, o conteúdo de dois ou mais itens em um único item.

### Formato:

```
STRING{<campo1> | <literalnn1>
  DELIMITED BY <campo2> | <literalnn2> | SIZE}
INTO <campo3>
[WITH POINTER <campon4>]
[ON OVERFLOW <comando1>]
[NOT ON OVERFLOW <comando2>]
[END-STRING
```

Opção	Significado
<campo1>/<literalnn1>	Itens a serem concatenados.
DELIMITED BY	Com a opção SIZE, os itens a serem concatenados serão transferidos para o <campo3> até ocorrer o fim destes ou até o final do <campo3>. Sem a opção SIZE, os itens a serem concatenados serão transferidos para o <campo3> até o seu final ou até ser encontrado o caractere no <campo2>/<literalnn2>.
WITH POINTER	Define um contador, cujo valor indicará a posição inicial no <campo3> onde começará a transferência dos dados.
INTO	Define o campo resultante da concatenação.
ON OVERFLOW	Condição que ocorrerá quando: o <campon4> for zero, o <campon4> exercer o comprimento do <campo3> ou o tamanho do <campo3> for insuficiente para conter o resultado.
NOT ON OVERFLOW	Ocorrerá como execução bem-sucedida..

### Exemplo 1:

```
77 CAMPO1          PIC X(06)  VALUE 'FUTURE' .
77 CAMPO2          PIC X(06)  VALUE 'SCHOOL' .
77 RESULTADO      PIC X(13) .

PROCEDURE DIVISION.
  STRING CAMPO1 '-' CAMPO2 DELIMITED BY SIZE INTO RESULTADO.
```

Campos	Antes	Depois
CAMPO1	FUTURE	FUTURE
CAMPO2	SCHOOL	SCHOOL
RESULTADO		<b>FUTURE-SCHOOL</b>

### Exemplo 2:

```

FD  ARQENTRA
   03  DATA-ATUAL.
       05  DIA-ATUAL          PIC  99.
       05  MES-ATUAL         PIC  99.
       05  ANO-ATUAL         PIC  99.

WORKING-STORAGE SECTION.

01  LIN-IMPRESSAO.
   03  FILLER                  PIC  X(20) VALUE SPACES.
   03  LIN-DATA.
       05  LIN-DIA            PIC  X(02)B.
       05  LIN-DATA-EXT      PIC  X(40).

01  TABELA-MESEXT.
   03  FILLER                  PIC  X(20) VALUE `DE JANEIRO   DE 20*'.
   03  FILLER                  PIC  X(20) VALUE `DE FEVEREIRO DE 20*'.
   03  FILLER                  PIC  X(20) VALUE `DE MARCO     DE 20*'.
   03  FILLER                  PIC  X(20) VALUE `DE ABRIL     DE 20*'.
   03  FILLER                  PIC  X(20) VALUE `DE MAIO      DE 20*'.
   03  FILLER                  PIC  X(20) VALUE `DE JUNHO     DE 20*'.
   03  FILLER                  PIC  X(20) VALUE `DE JULHO     DE 20*'.
   03  FILLER                  PIC  X(20) VALUE `DE AGOSTO    DE 20*'.
   03  FILLER                  PIC  X(20) VALUE `DE SETEMBRO  DE 20*'.
   03  FILLER                  PIC  X(20) VALUE `DE OUTUBRO   DE 20*'.
   03  FILLER                  PIC  X(20) VALUE `DE NOVEMBRO  DE 20*'.
   03  FILLER                  PIC  X(20) VALUE `DE DEZEMBRO  DE 20*'.

01  FILLER REDEFINES TABELA-MESEXT.
   03  MES-EXT                PIC  X(20) OCCURS 12 TIMES.

PROCEDURE          DIVISION.

MOVE  DIA-ATUAL          TO  LIN-DIA
STRING MES-EXT (MES-ATUAL) ANO-ATUAL
DELIMITED BY `*' INTO LIN-DATA-EXT.

```

Campos	Antes	Depois
DIA-ATUAL	15	15
MES-ATUAL	02	02
ANO-ATUAL	05	05
LIN-DATA		<b>15 DE FEVEREIRO DE 2005</b>

**13.4.19 Synchronized**

Utilizado para obter alinhamento de um item elementar em uma das limitações próprias da memória (half-word) - (full-word).

**Formato:**

(SYNCHRONIZED) (LEFT)  
ou  
(SYNC) (RIGHT)

Assegura a eficiência das operações aritméticas das cláusulas comp, comp-1 e comp-2. Para as demais cláusulas é interpretada como comentário. A necessidade do 'SYNC' é pelo fato de não existir alinhamento em tempo de compilação para descrição de itens binários para nível superior '01'.

**Exemplo 01:**

Os campos AREA-A, AREA-B e AREA-C necessitam ser alinhados.

```
01 REGISTRO.  
05 NOME PIC X(15).  
05 CODIGO PIC 9(06).  
05 FILLER PIC X(01).  
05 AREA-A PIC S9(04) COMP.  
05 FILLER PIC X(02).  
05 AREA-B PIC S9(03) COMP.  
05 AREA-C PIC S9(07) COMP.
```

Se usarmos o comando 'SYNC' não precisaremos nos preocupar com o problema do alinhamento exemplo.

```
01 REGISTRO.  
05 NOME PIC X(15).  
05 CODIGO PIC 9(06).  
05 FILLER PIC X(01).  
05 AREA-A PIC S9(04) COMP SYNC.  
05 FILLER PIC X(02).  
05 AREA-B PIC S9(03) COMP SYNC.  
05 AREA-C PIC S9(07) COMP SYNC.  
  
PROCEDURE DIVISION.  
ADD 10 TO AREA-A  
AREA-B  
AREA-C.  
  
SUBTRACT 20 FROM AREA-A  
AREA-C.
```

## 13.4.20 Transform

Altera caracteres, de acordo com uma regra de transformação

### Formato:

```

TRANSFORM nome-do-dado-3  CHARACTERS
FROM  constante-figurativa-1
TO    constante-figurativa-2 /
      literal não numérica-1 /
      literal não numérica-2 /
      nome de dado-1          /
      nome de dado-2
    
```

### Regras:

1. nome-do-dado-3 – tem que ser um item elementar alfabético, alfanumérico ou um item de grupo com um comprimento fixo até 256 bytes;
2. A regra de transformação é estabelecida por combinação da opção 'FROM' e 'TO'; e;
3. Para os operandos da opção 'FROM' e 'TO' valem as regras abaixo:
  - a) Literais não numéricas devem estar sempre entre apostrofes;
  - b) Na literal não numérica ou nome-de-dado-1, o mesmo caractere não pode figurar mais de uma vez, se for repetido o resultado não é previsível; e;
  - c) São permitidos como constantes figurativas: ZEROS, SPACES, QUOTES, HIGH-VALUES, LOW-VALUES.

### Exemplos:

```

TRANSFORM CAMPO-A  CHARACTERS  FROM  ZEROS  TO  QUOTE .
TRANSFORM CAMPO-B  CHARACTERS  FROM  '17CB'  TO  'QRST'
TRANSFORM CAMPO-1  CHARACTERS  FROM  CAMPO-X  TO  CAMPO-Y .
    
```

Campos	Antes	Depois
CAMPO-A	10700ABC	<b>'17' 'ABC</b>
CAMPO-B	1X7XXABC	<b>QXRXXATS</b>
CAMPO-1	1X7XXABC	<b>BCACC71X</b>
CAMPO-X	X17ABC	
CAMPO-Y	CBA71X	

**13.4.21 Unstring**

Faz com que os dados de um único campo sejam separados, pelo delimitadores, em sub-campos.

**Formato:**

```
UNSTRING identificador-1 [DELIMITED BY
  [ALL] operando-1 [OR [ALL] operando-2]] INTO
  {identificador-2 [DELIMITER IN identificador-3
    [COUNT IN identificador-4]] ...
  [WITH POINTER identificador-5]]
  [TALLYING IN identificador-6]
  [ON OVERFLOW comando imperativo].
FROM constante-figurativa-1
TO constante-fifurativa-2 /
  literal não numérica-1 /
  literal não numérica-2 /
  nome de dado-1 /
  nome de dado-2
```

Opção	Significado
Identificador-1	Campo a ter os caracteres extraídos.
DELIMITED BY	Especifica um delimitador.
ALL	Indica que uma ou mais ocorrências contíguas do delimitador serão tratadas como uma única.
INTO	Especifica o(s) campo(s) receptor(es) dos caracteres extraídos do identificador-1.
DELIMITER IN	Especifica o(s) campo(s) que conterà(ao) os caracteres delimitadores. Usada apenas se a opção DELIMITED BY for especificada.
COUNT IN	Especifica o(s) campo(s) que conterà(ao) as quantidades de caracteres verificadas no identificador-1. Usada apenas se a opção DELIMITED BY for especificada.
WITH POINTER	Indica a posição de início no identificador-1 a partir da qual começará a verificação.
TALLYING N	Especifica o campo que conterà um valor que é igual ao seu valor inicial mais a quantidade de campo ativada durante a execução.
OVERFLOW	Ocorrerá: 1) Quando o valor do identificador-5 for menor que 1 ou maior que o comprimento do identificador-1 ou, 2) Se durante a execução do UNSTRING todos os campos receptores tiverem sido ativados e o identificador-1 ainda contiver caracteres não verificados.
NOT OVERFLOW	Ocorrerá em situação contrária à de OVERFLOW.

**Exemplo 1:**

```
UNSTRING RECEPTOR-1 DELIMITED BY '/'  
INTO CAMPO-01, CAMPO-02, CAMPO-03, CAMPO-04
```

Campos	Antes	Depois
RECEPTOR-1	FUTURE/SCHOOL/CURSOS/COMPUTACAO	
CAMPO-01		<b>FUTURE</b>
CAMPO-02		<b>SCHOOL</b>
CAMPO-03		<b>CURSOS</b>
CAMPO-04		<b>COMPUTACAO</b>

### Exemplo 2:

```

UNSTRING RECEPTOR-1 DELIMITED BY '/' SPACE OR '*' INTO
RECEPT(1) DELIMITTER DELM(1) COUNT CONT(1)
RECEPT(2) DELIMITTER DELM(2) COUNT CONT(2)
RECEPT(3) DELIMITTER DELM(3) COUNT CONT(3)
RECEPT(4) DELIMITTER DELM(4) COUNT CONT(4)
        POINTER CONT-POINTER
        TALLYING IN CONT-REG.
    
```

Campos	Antes	Depois
RECEPTOR-1	FUTURE SCHOOL*CURSOS/COMPUTACAO	
CAMPO-01		<b>FUTURE</b>
DELM(1)	0	
CONT(1)	0	<b>6</b>
CAMPO-02		<b>SCHOOL</b>
DELM(2)	0	*
CONT(2)	0	<b>6</b>
CAMPO-03		<b>CURSOS</b>
DELM(3)	0	/
CONT(3)	0	<b>6</b>
CAMPO-04		<b>COMPUTACAO</b>
DELM(4)	0	
CONT(4)	0	<b>10</b>
CONT-POINTER	0	<b>32</b>
CONT-REG	0	<b>4</b>

## 13.5 Comandos para comunicação entre programas

O módulo de comunicação entre programas permite que um programa se comunique com outro programa. Quando um programa, por exemplo PROG0001, chama um outro, por exemplo PROG0002, diz-se que: PROG0001 é o programa chamador e PROG0002 é o programa chamado. A chamada é feita através dos comandos CALL ou CHAIN. Na chamada pode haver passagem de parâmetros ou não.

### 13.5.1 Call

Transfere o controle de um programa para outro, sendo usado no programa chamador. Ao término do programa chamado, o controle volta automaticamente para o programa chamador.

#### Formatos:

```
CALL nome do programa/módulo USING parâmetros
```

```
CALL nome do programa/módulo ON OVERFLOW comando imperativo
```

Parâmetros que constam do USING tornam-se disponíveis ao programa chamado pela passagem de endereços. Estes endereços são, então, associados aos itens descritos na LINKAGE SECTION e arrolados no USING do cabeçalho da PROCEDURE DIVISION do programa chamado. Por isso, o número de itens no USING do programa chamador de ser o mesmo do USING do programa chamado.

#### Observação:

Caso a memória seja insuficiente para acomodar o programa chamado e a cláusula ON OVERFLOW for mencionada, não ocorrerá a transferência de comando para o programa chamado e a instrução imperativa da cláusula ON OVERFLOW será executada. Se a cláusula ON OVERFLOW não estiver presente e a memória for insuficiente para carregar o programam chamado, o programa terminará anormalmente.

**13.5.2 Cancel**

Faz com que o programa chamado volte ao seu estado inicial em uma próxima chamada.

**Formato:**

CANCEL nome do programa/módulo

**Observação:**

É conveniente usar o comando CANCEL após um comando CALL.

### 13.5.3 Chain

Transfere o controle de um programa para outro, sendo usado no programa chamador. Ao término do programa chamado, o controle não retorna para o programa chamador.

#### Formatos:

CHAIN nome do programa/módulo USING parâmetros

CHAIN nome do programa/módulo ON OVERFLOW comando imperativo

Parâmetros que constam do USING tornam-se disponíveis ao programa chamado pela passagem de endereços. Estes endereços são, então, associados aos itens descritos na LINKAGE SECTION e arrolados no USING do cabeçalho da PROCEDURE DIVISION do programa chamado. Por isso, o número de itens no USING do programa chamador de ser o mesmo do USING do programa chamado.

#### Observação:

Caso a memória seja insuficiente para acomodar o programa chamado e a cláusula ON OVERFLOW for mencionada, não ocorrerá a transferência de comando para o programa chamado e a instrução imperativa da cláusula ON OVERFLOW será executada. Se a cláusula ON OVERFLOW não estiver presente e a memória for insuficiente para carregar o programam chamado, o programa terminará anormalmente.

**13.5.4 Exit Program**

Retorna o controle ao programa que chamou o módulo em que aparece.

O retorno se dá no primeiro comando executável após o CALL.

**Formato:**

```
EXIT PROGRAM.
```

### 13.5.5 Exemplo de comunicação entre programas

#### Programa Chamador:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.     FUTURE01.
AUTHOR.        FUTURE SCHOOL CURSOS DE COMPUTACAO.

ENVIRONMENT     DIVISION.
CONFIGURATION   SECTION.

SPECIAL-NAMES.
    DECIMAL-POINT IS COMMA.

INPUT-OUTPUT SECTION.
FILE-CONTROL.
.
.
DATA            DIVISION.
FILE           SECTION.
.
.
01 REG-ARQENTRA.
   05 CODCLI-ENTRA          PIC 9(05).
   05 VLR-EMPRESTIMO       PIC S9(13)V99      COMP-3.
   05 PERIODO              PIC 9(03).
   05 TAXA                 PIC S9(03)V9(06)   COMP-3.
   05 TIPO-JUROS          PIC X.
   05 FILLER              PIC X(03).

FD ARQSAIDA
.
.

WORKING-STORAGE SECTION.
01 AC-COMUNICACAO.
   05 AC-EMPRESTIMO       PIC S9(13)V99.
   05 AC-PERIODO         PIC 9(03).
   05 AC-TAXA           PIC S9(03)V9(06).
   05 AC-TIPO-JUROS     PIC X.
   05 AC-VLR-JUROS      PIC S9(13)V99.

PROCEDURE      DIVISION.
.
.
MOVE          VLR-EMPRESTIMO          TO          AC-EMPRESTIMO.
MOVE          PERIODO                 TO          AC-PERIODO.
MOVE          TAXA                    TO          AC-TAXA.
MOVE          TIPO-JUROS              TO          AC-TIPO-JUROS.

CALL 'FUTURE02' USING AC-COMUNICACAO.
.
.
STOP        RUN.
```

**Programa Chamado:**

```
IDENTIFICATION DIVISION.
PROGRAM-ID.      FUTURE02.
AUTHOR.          FUTURE SCHOOL CURSOS DE COMPUTACAO.

ENVIRONMENT      DIVISION.
CONFIGURATION    SECTION.

SPECIAL-NAMES.
    DECIMAL-POINT IS COMMA.

DATA             DIVISION.
FILE             SECTION.
WORKING-STORAGE SECTION.
01 AREAS-AUXILIARES.
    05 WS-TAXA          PIC 9(03)V9(06).
    05 WS-JUROS         PIC 9(12)V99.

LINKAGE          SECTION.
01 LK-COMUNICACAO.
    05 LK-TAMANHO      PIC S9(04)          COMP.
    05 LK-EMPRESTIMO   PIC S9(13)V99.
    05 LK-PERODO       PIC 9(03).
    05 LK-TAXA         PIC S9(03)V9(06).
    05 LK-TIPO-JUROS   PIC X.
    05 LK-VLR-JUROS    PIC S9(13)V99.

PROCEDURE        DIVISION USING LK-COMUNICACAO.
.
.
COMPUTE WS-TAXA = LK-TAXA / 100.

IF LK-TIPO-JUROS EQUAL 'S'
    COMPUTE WS-JUROS ROUNDED = (LK-EMPRESTIMO *
                                (LK-PERODO-TELA * WS-TAXA))
    MOVE WS-JUROS TO LK-VLR-JUROS
ELSE
    IF LK-TIPO-JUROS EQUAL 'C'
        COMPUTE WS-JUROS ROUNDED = (LK-EMPRESTIMO *
                                    ((1 + WS-TAXA) ** LK-PERODO))
        COMPUTE WS-JUROS = WS-JUROS - LK-EMPRESTIMO
        MOVE WS-JUROS TO LK-VLR-JUROS
    ELSE
        MOVE ZEROS TO LK-VLR-JUROS
    END-IF
END-IF.

GOBACK.
```

**14 Processamento de arquivos de organização indexada**

A organização indexada é capaz de recuperar / gravar registros de um arquivo de dados em disco, a partir de um diretório de ponteiros, chamado de índice de controle. Tais ponteiros permitem uma localização direta de registros que tenham valores únicos de uma determinada chave.

O acesso a tais chaves pode ser seqüencial, aleatório ou dinâmico.

- **Seqüencial** ocorre quando é feito em ordem crescente de valores da RECORD KEY;
- **Aleatório** ocorre quando a ordem de acesso aos registros é controlada pelo programador. A mecânica consiste em colocar o valor de RECORD KEY desejado e depois realizar o acesso; e;
- **Dinâmico** ocorre quando a lógica do programa pode alterar o modo de acesso de seqüencial para aleatório e vice-versa, tantas vezes quantas for convenientes.

**Formato**

```
ENVIRONMENT      DIVISION.
INPUT-OUTPUT     SECTION.
FILE-CONTROL.
  SELECT  ARQUIVO      ASSIGN      TO  DA-I-ARQUIVO
          ACCESS      MODE        IS   RANDOM      /
          RECORD      KEY         IS   SEQUENTIAL  /
          NOMINAL     KEY         IS   DYNAMIC
          NOMINAL     KEY         IS   CHAVE-ARQUIVO
          NOMINAL     KEY         IS   WS-CHAV-ARQ.

WORKING-STORAGE SECTION.
FD  ARQUIVO
   LABEL      RECORD      STANDARD
   RECODING   MODE        F
   RECORD     CONTAINS    130 CHARACTERS
   BLOCK      CONTAINS    10 RECORDS.
01  REG-ARQUIVO.
    05  CHAVE-ARQUIVO     PIC  9(06) .
    05  NOME-ARQ         PIC  X(40) .
    .
    .
    .

WORKING-STORAGE SECTION.
01  WS-CHAV-ARQ         PIC  9(06) VALUE 2456.
```

**Explicação das cláusulas marcadas no formato acima:**

Cláusulas	Descrição
DA-I	Indica que o arquivo terá um acesso direto e que sua organização é indexada.
ACCESS MODE IS RANDOM	Indica que o acesso à chave de pesquisa será de forma aleatória (randômica)
RECORD KEY IS	Define o nome da chave de pesquisa no registro.
NOMINAL KEY IS	Define o nome da chave, que será movimentada a chave de pesquisa.
CHAVE-ARQUIVO	Especifica as propriedades da chave de pesquisa no registro, definida pela RECORD KEY e montada na FD do arquivo,
WS-CHAV-ARQ	Especifica as propriedades da chave que será movimentada a chave de pesquisa, definida pela NOMINAL KEY e montada na WORKING-STORAGE SECTION.

**Observação:** Hoje em dia a NOMINAL KEY não está sendo mais utilizada, e a cláusula ACCESS MODE, geralmente é DYNAMIC.

## 15 Tabelas internas

Denominamos tabela interna, um determinado número de ocorrências de um mesmo dado, dentro do programa. Podemos criar vários tipos de tabelas, por exemplo, tabela de meses, dias da semana, estados etc. Sua criação, visa à simplificação e redução das linhas de comando em um programa.

Supondo-se que um programa necessite gerar um relatório com o nome do estado, porém no arquivo exista apenas a sigla; não utilizando a tabela, seria necessário efetuar 27 perguntas (total de estados do Brasil), que relacionam a sigla ao seu respectivo estado. Ao utilizarmos a tabela, não serão necessárias as 27 perguntas, e sim apenas uma pergunta que ficará em um LOOP 27 vezes.

### 15.1 Tipo de tabelas

Existem 2 (dois) tipos de tabelas: Identidade e Não Identidade, que serão descritas abaixo:

#### 15.1.1 Identidade

Possuem um conteúdo pré definido e correspondência entre um determinado código e a posição das ocorrências na tabela

**Exemplo:**

Código da Semana	Dia da Semana
01	DOMINGO
02	SEGUNDA-FEIRA
03	TERÇA-FEIRA
04	QUARTA-FEIRA
05	QUINTA-FEIRA
06	SEXTA-FEIRA
07	SÁBADO

A ocorrência equivale à posição do dia da semana, quando o código lido for igual a 06, automaticamente é a sexta ocorrência que será relacionada, no caso, SEXTA-FEIRA (esta tabela tem uma ocorrência de 7 vezes.)

**15.1.2 Não identidade**

Possuem um conteúdo pré definido e não possuem correspondência entre um determinado código e a posição das ocorrências na tabela:

Código da Linguagem	Linguagem
AA	RPG II
AB	CLIPPER
AC	FORTRAN
AD	COBOL
AE	PL 1
AF	VISUAL BASIC
AG	JAVA

O código da linguagem, não equivale à posição da ocorrência na tabela, ou seja, quando o código for AD, a ocorrência da tabela será a de número 4, que está relacionada com a linguagem COBOL. Esta tabela ocorre 7 vezes.

**15.2 Dimensões de tabelas**

Existem vários níveis de ocorrências de tabelas. Até agora nos referimos a tabelas de apenas um nível, e estas tabelas são denominadas Unidimensionais. Existem também as tabelas denominadas Bidimensionais (dois níveis de ocorrências) e Tridimensionais (três níveis de ocorrências).

**15.2.1 Unidimensional**

Os níveis correspondem somente à quantidade de linhas da tabela.

**Exemplo 1:**

```
01 TABELA.
  05 TAB-DIAS-SEMANA.
    10 FILLER          PIC X(13) VALUE 'DOMINGO      ' .
    10 FILLER          PIC X(13) VALUE 'SEGUNDA-FEIRA' .
    10 FILLER          PIC X(13) VALUE 'TERCA-FEIRA  ' .
    10 FILLER          PIC X(13) VALUE 'QUARTA-FEIRA' .
    10 FILLER          PIC X(13) VALUE 'QUINTA-FEIRA' .
    10 FILLER          PIC X(13) VALUE 'SEXTA-FEIRA  ' .
    10 FILLER          PIC X(13) VALUE 'SABADO      ' .
  05 TAB-DIAS-SEMANA-R REDEFINES TAB-DIAS-SEMANA.
    10 DIA-DA-SEMANA  PIC X(13) OCCURS 7 TIMES.
```

**Exemplo02:**

```
01 TABELA.
```

```

05  TAB-SIGLAS-ESTADOS.
    10  FILLER          PIC  X(21) VALUE  `ACACRE          ' .
    10  FILLER          PIC  X(21) VALUE  `ALALAGOAS       ' .
    10  FILLER          PIC  X(21) VALUE  `AMAMAZONAS      ' .
    10  FILLER          PIC  X(21) VALUE  `APAMAPA         ' .
    10  FILLER          PIC  X(21) VALUE  `BABAHIA         ' .
    10  FILLER          PIC  X(21) VALUE  `CECEARA         ' .
    10  FILLER          PIC  X(21) VALUE  `DFDISTRITO FEDERAL ' .
    10  FILLER          PIC  X(21) VALUE  `ESESPIRITO SANTO  ' .
    10  FILLER          PIC  X(21) VALUE  `GOGOIAS         ' .
    10  FILLER          PIC  X(21) VALUE  `MAMARANHAO       ' .
    10  FILLER          PIC  X(21) VALUE  `MGMINAS GERAIS   ' .
    10  FILLER          PIC  X(21) VALUE  `MSMATO GROSSO DO SUL ' .
    10  FILLER          PIC  X(21) VALUE  `MTMATO GROSSO    ' .
    10  FILLER          PIC  X(21) VALUE  `PAPARA          ' .
    10  FILLER          PIC  X(21) VALUE  `PBPARAIBA       ' .
    10  FILLER          PIC  X(21) VALUE  `PEPERNAMBUCO    ' .
    10  FILLER          PIC  X(21) VALUE  `PIPIAUI         ' .
    10  FILLER          PIC  X(21) VALUE  `PRPARANA        ' .
    10  FILLER          PIC  X(21) VALUE  `RJRIO DE JANEIRO  ' .
    10  FILLER          PIC  X(21) VALUE  `RNRIO GRANDE DO NORTE ' .
    10  FILLER          PIC  X(21) VALUE  `RORONDONIA      ' .
    10  FILLER          PIC  X(21) VALUE  `RRRORAIMA       ' .
    10  FILLER          PIC  X(21) VALUE  `RSRIO GRANDE DO SUL ' .
    10  FILLER          PIC  X(21) VALUE  `SCSNATA CATARINA ' .
    10  FILLER          PIC  X(21) VALUE  `SESERGIPE       ' .
    10  FILLER          PIC  X(21) VALUE  `SPSAO PAULO     ' .
    10  FILLER          PIC  X(21) VALUE  `TOTOCANTINS     ' .
05  TAB-ESTADOS      REDEFINES  TAB-SIGLA-ESTADOS OCCURS 27.
    10  SIGLA          PIC  X(02) .
    10  ESTADO        PIC  X(19) .

```

**15.2.2 Bidimensional**

Os níveis de ocorrências correspondem à quantidade de linhas e colunas da tabela.

Para definição destas tabela, usamos dois subscritores ou indexadores.

Suponhamos que a tabela possua 4 linhas e 3 colunas. Quando a linha estiver com o valor 1, variamos todas as 3 colunas e quando a coluna for maior que 3, acrescentamos 1 na linha e esta passa a valer 2, e assim por diante. A pesquisa só termina quando a coluna for maior que 3 e linha maior que 4.

**Exemplo :**

```
WORKING-STORAGE SECTION.
77 LIN                PIC 9          VALUE 0.
77 COL                PIC 9          VALUE 4.
01 TABELA.
   05 LINHAS          OCCURS        4.
      10 COLUNAS      OCCURS        3.
          15 SIGLA    PIC X(02) .
          15 ESTADO   PIC X(19) .

PROCEDURE            DIVISION.
.
.
PERFORM PESQUISA VARYING LIN FROM 1 BY 1 UNTIL LIN GREATER 4
                AFTER COL FROM 1 BY 1 UNTIL COL GREATER 3
PERFORM LER.

PESQUISA.
PERFORM CARREGAR.
PERFORM IMPRIMIR.
```

### 15.2.3 Tridimensional

Os níveis de ocorrências correspondem à quantidade de folhas, linhas e colunas da tabela.

Para definição destas tabela, usamos três subscritores ou indexadores.

Suponhamos que a tabela possua 4 folhas, 4 linhas e 7 colunas. Quando folha estiver com o valor 1, a linha estiver com o valor 1, variamos todas as 7 colunas e quando a coluna for maior que 7, acrescentamos 1 na linha e esta passa a valer 2, e assim por diante até que a linhas seja maior que 4, daí acrescentamos 1 na folha e passa valer 2, e assim por diante. A pesquisa só termina quando a coluna for maior que 7, linha maior que 4 e folha maior que 4.

#### Exemplo :

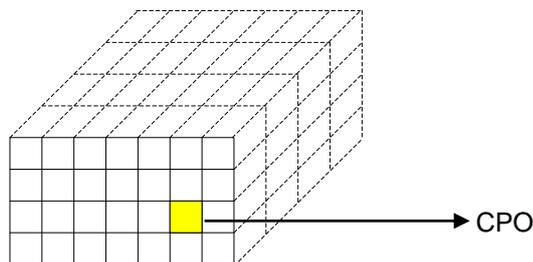
```

WORKING-STORAGE SECTION.
77 FOL          PIC 9          VALUE 0.
77 LIN          PIC 9          VALUE 0.
77 COL          PIC 9          VALUE 0.
01 TABELA.
   05 FOLHAS    OCCURS        4.
     10 LINHAS  OCCURS        4.
       15 COLUNAS OCCURS      7.
         20 CPO  PIC X(03) .

PROCEDURE      DIVISION.
.
.
PERFORM PESQUISA VARYING FOL FROM 1 BY 1 UNTIL FOL GREATER 4
                    AFTER    LIN FROM 1 BY 1 UNTIL LIN GREATER 4
                    AFTER    COL FROM 1 BY 1 UNTIL COL GREATER 7

PERFORM LER.

PESQUISA.
PERFORM CARREGAR.
PERFORM IMPRIMIR.
  
```



**15.3 Comandos de tabelas****15.3.1 Occurs**

Permite substancial redução de trabalho quando se trata de especificar itens que se repetem. Especifica o número de vezes que um item (grupo ou elementar) se repete com o mesmo formato. As cláusulas associadas a um item cuja descrição inclua o OCCURS são extensivas a todas as repetições deste mesmo item. O nome de dado que contém a cláusula OCCURS deverá ser subscrito (ou indexado) toda vez que for referenciado na PROCEDURE DIVISION. Se este nome de dado corresponder a um item de grupo, então toda ocorrência de item subordinado a ele (na PROCEDURE DIVISION) deverá estar subscrita (ou indexada).

No máximo três cláusulas OCCURS, podem governar um item, conseqüentemente, são necessários um, dois ou três subscritores.

É proibido o uso da cláusula OCCURS nos níveis 01 e 77.

**Formatos:**

```
Nível nome de dado OCCURS inteiro [TIME(S)]  
[INDEXED BY nome de índice]
```

```
Nível nome de dado OCCURS inteiro-1 inteiro-2  
DEPENDING ON campo
```

**Exemplo 1:**

```
WORKING-STORAGE SECTION.  
01 DIA-EXTENSO PIC X(13) VALUE SPACES.  
77 INDICE PIC 9 VALUE 5.  
01 TABELA.  
05 TAB-DIAS-SEMANA.  
10 FILLER PIC X(13) VALUE 'DOMINGO ' .  
10 FILLER PIC X(13) VALUE 'SEGUNDA-FEIRA' .  
10 FILLER PIC X(13) VALUE 'TERCA-FEIRA ' .  
10 FILLER PIC X(13) VALUE 'QUARTA-FEIRA ' .  
10 FILLER PIC X(13) VALUE 'QUINTA-FEIRA ' .  
10 FILLER PIC X(13) VALUE 'SEXTA-FEIRA ' .  
10 FILLER PIC X(13) VALUE 'SABADO ' .  
05 TAB-DIAS-SEMANA-R REDEFINES TAB-DIAS-SEMANA.  
10 DIA-DA-SEMANA PIC X(13) OCCURS 7 TIMES.  
PROCEDURE DIVISION.  
.  
MOVE DIA-DA-SEMANA (INDICE) TO DIA-EXTENSO.  
.
```

**Explicação do exemplo acima:**

Sendo 5 o valor do campo INDICE, então, o 5º elemento da tabela será selecionado e movimentado para o campo DIA-EXTENSO, no caso, QUINTA-FEIRA.

### Exemplo 2:

```

77 IND PIC 9(02) COMP SYNC VALUE 0.
01 LINHA-DISPLAY.
   05 FILLER PIC X(19) VALUE.
      'TOTAL DO ESTADO DE \.
   05 NOME-ESTADO PIC X(19).
01 TABELA.
   05 TAB-SIGLAS-ESTADOS.
      10 FILLER PIC X(63) VALUE
      'ACACRE ALALAGOAS AMAMAZONAS '.
      10 FILLER PIC X(63) VALUE
      'APAMAPA BABAHIA CECEARA '.
      10 FILLER PIC X(63) VALUE
      'DFDISTRITO FEDERAL ESESPIRITO SANTO GOGOIAS '.
      10 FILLER PIC X(63) VALUE
      'MAMARANHAO MGINAS GERAIS MSMATO GROSSO DO SUL '.
      10 FILLER PIC X(63) VALUE
      'MTMATO GROSSO PAPARA PBPARAIBA '.
      10 FILLER PIC X(63) VALUE
      'PEPERNAMBUCO PIPIAUI PRPARANA '.
      10 FILLER PIC X(63) VALUE
      'RJRIO DE JANEIRO RNRIO GRANDE DO NORTERORONDONIA '.
      10 FILLER PIC X(63) VALUE
      'RRRORAIMA RSRIO GRANDE DO SUL SCSNATA CATARINA '.
      10 FILLER PIC X(63) VALUE
      SESERGIPE SPSAO PAULO TOTOCANTINS '.
   05 TAB-ESTADOS REDEFINES TAB-SIGLA-ESTADOS OCCURS 27.
      10 SIGLA PIC X(02).
      10 ESTADO PIC X(19).
PROCEDURE DIVISION.
.
.
PERFORM ACHA-ESTADO VARYING IND FROM 1 BY 1
UNTIL IND GREATER 27.
.
.
ACHA-ESTADO.
IF SIGLA(IND) EQUAL EST-ARQUIVO
MOVE ESTADO(IND) TO NOME-ESTADO
DISPLAY LINHA-DISPLAY
MOVE 30 TO IND
ELSE
IF IND EQUAL 27
MOVE 'ESTADO INVALIDO' TO NOME ESTADO.

```

### Explicação do exemplo acima:

Na leitura no arquivo, será obtido a sigla do estado, mas, por ter que mostrar o nome do estado por extenso, então, será feito uma leitura seqüencial em todos os itens da tabela interna até encontrar a sigla lida. Ao encontrar a sigla na tabela, será movimentado o respectivo nome por extenso para a área de saída. Se não for encontrado será mostrada uma mensagem de estado inválido.

### 15.3.2 Search

É utilizado para pesquisar uma tabela, procurando um elemento que satisfaça certas condições e determinando o valor do nome indexado associado ao índice correspondente do elemento da tabela. Somente um único identificador da tabela pode ser referenciado.

#### Formato 1:

Pesquisa uma tabela de forma a encontrar um elemento que satisfaça uma condição.

```
SEARCH tabela
      [AT END comando1]
          WHEN condição1 comando2 | NEXT SENTENCE
          WHEN condição2 comando3 | NEXT SENTENCE]
END-SEARCH
```

Opção	Significado
AT END	Ação a ser tomada se a condição da opção WHEN não tiver sido satisfeita.
WHEN	Especifica a condição relativa à pesquisa na tabela e conseqüente ação através do comando ou da frase NEXT SENTENCE.
NEXT SENTENCE	Desvia para o comando após o primeiro “.”(ponto) encontrado.

#### Formato 2:

Pesquisa uma tabela de forma a encontrar um elemento que satisfaça uma condição. Pode ser usado apenas com as opções ASCENDING KEY ou DESCENDING KEY.

```
SEARCH ALL tabela
      [AT END comando1]
          WHEN condição1 comando2 | NEXT SENTENCE
END-SEARCH
```

Opção	Significado
WHEN	Especifica a condição relativa à pesquisa na tabela e conseqüente ação através do comando2 ou da frase NEXT SENTENCE.

## Exemplo de tabela indexada – Search tabela

```

WORKING-STORAGE SECTION.
01  TABELA1.
    05  TAB-NOMES.
        05  FILLER          PIC X(11) VALUE 'ANA LUCIA' .
        05  FILLER          PIC X(11) VALUE 'VINICIUS' .
        05  FILLER          PIC X(11) VALUE 'VITOR' .
        05  FILLER          PIC X(11) VALUE 'LUIS' .
        05  FILLER          PIC X(11) VALUE 'NATHALIA' .
        05  FILLER          PIC X(11) VALUE 'FELIPE' .
        05  FILLER          PIC X(11) VALUE 'CAROLINA' .
        05  FILLER          PIC X(11) VALUE 'MARISTELA' .
        05  FILLER          PIC X(11) VALUE 'ZILDA' .
        05  FILLER          PIC X(11) VALUE 'ANDERSON' .
        05  FILLER          PIC X(11) VALUE 'MEYRE' .
        05  FILLER          PIC X(11) VALUE 'ANA PAULA' .
        05  FILLER          PIC X(11) VALUE 'SAMANTHA' .
        05  FILLER          PIC X(11) VALUE 'RODOLFO' .
    05  TAB-NOMES-R        REDEFINE    TAB-NOMES
                            OCCURS 14 INDEXED BY IND.
        10  TAB-NOM        PIC X(11) .
    *
01  AUX-NOME              PIC X(11) VALUE SPACES.
01  IND                   PIC 9.

PROCEDURE      DIVISION.
000-00-INICIO          SECTION.
*
000-01-RECEBER-NOME.
*
    DISPLAY 'FAVOR DIGITAR O NOME A SER PESQUISADO, PIC X(11) '
            UPON CONSOLE.
*
    ACCEPT  AUX-NOME          FROM          CONSOLE.
*
    SET    IND    TO    1
*
    SEARCH TAB-NOMES
            AT END          GO TO 999-99-FIM
            WHEN TAB-NOM (IND) EQUAL AUX-NOME
            PERFORM 005-00-ACHOU-NOME.
.
.
    
```

### Exemplo de tabela indexada – Search all

```

77 IND                                PIC 9(02) VALUE 0.
01 LINHA-DISPLAY.
05 FILLER                             PIC X(19) VALUE.
   'TOTAL DO ESTADO DE '.
05 NOME-ESTADO                         PIC X(19).
01 TABELA.
05 TAB-SIGLAS-ESTADOS.
   10 FILLER                           PIC X(63) VALUE
   'ACACRE                             ALALAGOAS                AMAMAZONAS          '.
   10 FILLER                           PIC X(63) VALUE
   'APAMAPA                             BABAHIA                  CECEARA             '.
   10 FILLER                           PIC X(63) VALUE
   'DFDISTRITO FEDERAL                 ESESPRITO SANTO        GOGOIAS             '.
   10 FILLER                           PIC X(63) VALUE
   'MAMARANHAO                         MGINAS GERAIS          MSMATO GROSSO DO SUL '.
   10 FILLER                           PIC X(63) VALUE
   'MTMATO GROSSO                       PAPARA                  PBPARAIBA           '.
   10 FILLER                           PIC X(63) VALUE
   'PEPERNAMBUCO                       PIPIAUI                 PRPARANA            '.
   10 FILLER                           PIC X(63) VALUE
   'RJRIO DE JANEIRO                    RNRIO GRANDE DO NORTERORONDONIA '.
   10 FILLER                           PIC X(63) VALUE
   'RRRORAIMA                           RSRIO GRANDE DO SUL   SCSNATA CATARINA   '.
   10 FILLER                           PIC X(63) VALUE
   SESERGIPE                            SPSAO PAULO            TOTOCANTINS        '.
05 FILLER                             REDEFINES TAB-SIGLA-ESTADOS.
   10 TAB-ESTADOS OCCURS 27
      ASCENDING KEY IS SIGLA
      INDEXED BY IND.
      15 SIGLA PIC X(02).
      15 ESTADO PIC X(19).
PROCEDURE DIVISION.
.
.
SEARCH ALL TAB-ESTADOS
      AT END
          MOVE 'ESTADO INVALIDO' TO NOME ESTADO
          GO TO 999-99-FIM
      WHEN
          MOVE ESTADO(IND)          TO NOME ESTADO.

```

#### Observação:

- Não é necessário apontar o indexador quando utilizar SEARCH ALL'.
- A ordem de seqüência das opções abaixo do SEARCH ALL devem ser respeitadas:
  1. OCCURS;
  2. ASCENDING KEY IS ou DESCENDING KEY IS; e;
  3. INDEXED BY.

### 15.3.3 Set

Esta cláusula estabelece pontos de referência na pesquisa de tabelas, colocando determinados valores nos indexadores associados com os elementos das tabelas. A cláusula Set deve ser utilizada quando quisermos inicializar um indexador antes da execução de uma cláusula Search. Pode também ser utilizada na transferência dos conteúdos dos indexadores para outros itens de dados elementares ou ainda, somar ou subtrair o conteúdo do indexador.

#### Formato 1:

Atribui valores aos índices.

```
SET <índice1> | <item-ind1> TO <índice2> | <literaln1>  
    <item-ind2>
```

#### Formato 2:

Incrementa ou subtrai valores dos índices.

```
SET <índice1> ... UP BY | DOWN BY <índice2> | <literaln1>
```

Opção	Significado
UP BY	Incrementa o valor do <índice1> com o <índice2> ou <literaln1>.
DOWN BY	Subtrai do <índice1> o valor do <índice2> ou <literaln1>.

#### Exemplos:

```
SET      I3          TO          25.  
SET      I2          I1         TO          I3.  
SET      I3          UP        BY      1.  
SET      I4          DOWN      BY      1.
```

**16 Tratamento de Impressão****16.1 Advancing**

C01 ATÉ C09	SALTO DO CANAL 1 ATÉ 9
C10 ATÉ C12	SALTO DO CANAL 10,11 E 12

Se um número inteiro for usado (1 a 100) serão saltadas tantas linhas quantas forem o valor do inteiro.

**Exemplos:**

```
WRITE REG-RELATO FROM CAB001 AFTER ADVANCING CANAL-1.  
WRITE REG-RELATO FROM DET001 AFTER ADVANCING 3 LINES.
```

**16.2 Positioning**

Deve ser declarada como caractere alfanumérico (PIC X).

CARACTERE	SIGNIFICADO
' ' BRANCO	ESPACEJAMENTO SIMPLES
'0' ZEROS	ESPACEJAMENTO DUPLO
'-' MENOS	ESPACEJAMENTO TRIPLO
'+' MAIS	SUPRESSÃO DO ESPACEJAMENTO
'1' A '9'	SALTO DO CANAL 1 A 9
'A', 'B', 'C'	SALTO DO CANAL 10, 11, 12

**Exemplos:**

```
FD  RELATO
   LABEL OMITTED
   RECORDING F
   BLOCK 0.
01  REG-RELATO.
   05  CARRO          PIC X(01) .
   05  FILLER         PIC X(132) .
*
01  CAB001.
   05  FILLER         PIC X(01) VALUE '1' .
   .
   .
01  CAB002.
   05  FILLER         PIC X(01) VALUE '-' .
   .
   .
*
01  DET001.
   05  FILLER         PIC X(01) VALUE '0' .
   .
   .
PROCEDURE DIVISION.
.
.
.
WRITE REG-RELATO FROM CAB001  AFTER POSITIONING CARRO.
WRITE REG-RELATO FROM CAB002  AFTER POSITIONING CARRO.
WRITE REG-RELATO FROM DET001  AFTER POSITIONING 2 LINES.
```

**Observações:**

- No POSITIONING o máximo de linhas que se pode pular são três (3);
- Se omitido os comandos POSITIONING ou ADVANCING, o default é ADVANCING;
- AFTER – salta primeiro, depois efetua a impressão;
- BEFORE – imprime primeiro, depois efetua o salto; e;
- Não utilizar 'AFTER' e 'BEFORE' no mesmo programa.

**17 File status**

Verificar se a operação de I/O (OPEN, START, WRITE, READ, REWRITE, CLOSE etc) foi efetuada com sucesso ou não.

O tratamento de FILE STATUS permite ao programador monitorar a execução das operações de I/O.

Após cada operação, o sistema movimenta um valor para a STATUS KEY (campo alfanumérico, com 2 (dois) caracteres definido na ENVIRONMENT DIVISION através da cláusula SELECT e posteriormente montado na DATA DIVISION) que acusa se a execução foi efetuada com sucesso ou não.

Qualquer valor movido para a STATUS KEY diferente de zero, revela que a execução não foi efetuada com sucesso, porém, no caso da STATUS KEY receber o código 10 na primeira leitura (READ), isso quer dizer que o arquivo está vazio, neste caso, pode acontecer do programa poder continuar sem problemas.

**Exemplo**

```
IDENTIFICATION DIVISION.
PROGRAM-ID.     FUTURE01.
AUTHOR.        FUTURE SCHOOL CURSOS DE COMPUTACAO.
*
* USANDO FILE STATUS
*
ENVIRONMENT     DIVISION.
CONFIGURATION   SECTION.
SPECIAL-NAMES.
                DECIMAL-POINT IS COMMA.
*
INPUT-OUTPUT    SECTION.
FILE-CONTROL.
    SELECT CADPECA      ASSIGN TO UT-S-CADPECA
                FILE STATUS IS FS-CADPECA.
*
    SELECT CADATU       ASSIGN TO UT-S-CADATU
                FILE STATUS IS FS-CADATU.
*
DATA            DIVISION.
FILE            SECTION.
*
FD CADPECA
RECORD CONTAINS 70 CHARACTERS
LABEL RECORD IS STANDARD
DATA RECORD IS REG-CADPECA.
01 REG-CADPECA.
    05 COD-PECA          PIC 9(05).
    05 NOME-PECA        PIC X(30).
    05 QTD-PECA         PIC S9(05) COMP-3.
    05 LOCAL-PECA.
        10 PRAT-PECA    PIC X(02).
        10 ALTU-PECA    PIC X.
        10 GAVE-PECA    PIC X(03).
    05 PUNIT-PECA      PIC S9(07)V99 COMP-3.
    05 CONTA-PECA     PIC X(10).
    05 FILLER          PIC X(11).
*
FD CADATU
RECORD CONTAINS 75 CHARACTERS
LABEL RECORD IS STANDARD
DATA RECORD IS REG-CADATU.
    05 COD-ATUAL        PIC 9(05).
    05 NOME-ATUAL      PIC X(30).
    05 QTD-ATUAL       PIC S9(05) COMP-3.
    05 LOCAL-ATUAL.
        10 PRAT-ATUAL   PIC X(02).
        10 ALTU-ATUAL   PIC X.
        10 GAVE-ATUAL   PIC X(03).
    05 PUNIT-ATUAL     PIC S9(07)V99 COMP-3.
    05 PTOTAL-ATUAL    PIC S9(09)V99 COMP-3.
    05 CONTA-ATUAL     PIC X(10).
*
WORKING-STORAGE SECTION.
*
* FILE STATUS
```

```

*
77  FS-CADPECA          PIC  X(02) VALUE SPACES.
77  FS-CADATU           PIC  X(02) VALUE SPACES.
77  FS-COD-STATUS      PIC  X(02) VALUE SPACES.
77  FS-ARQUIVO         PIC  X(08) VALUE SPACES.
77  FS-OPERACAO        PIC  X(13) VALUE SPACES.
77  FS-ABERTURA       PIC  X(13) VALUE 'NA ABERTURA'.
77  FS-LEITURA        PIC  X(13) VALUE 'NA LEITURA'.
77  FS-GRAVACAO        PIC  X(13) VALUE 'NA GRAVACAO'.
77  FS-FECHAMENTO      PIC  X(13) VALUE 'NO FECHAMENTO'.
*
PROCEDURE      DIVISION.
*=====
000-00-INICIO          SECTION.
*=====
    PERFORM      001-00-ABRIR-ARQUIVOS.
    PERFORM      002-00-VER-ARQ-VAZIO.
    PERFORM      003-00-TRATAR-CADPECA
                UNTIL FS-CADPECA EQUAL '10'.
    PERFORM      004-00-FECHAR-ARQUIVOS.
    STOP         RUN.
*=====
001-00-ABRIR-ARQUIVOS SECTION.
*=====
    MOVE  FS-ABERTURA      TO  FS-OPERACAO.
    OPEN  INPUT  CADPECA
          OUTPUT CADATU.
    PERFORM 001-01-TESTAR-FS.
001-00-FIM.      EXIT.
*=====
001-01-TESTAR-FS SECTION.
*=====
    PERFORM 001-02-FS-CADPECA.
    PERFORM 001-03-FS-CADATU.
001-01-FIM.      EXIT.
*=====
001-02-FS-CADPECA SECTION.
*=====
    MOVE  'CADPECA'      TO  FS-ARQUIVO.
    MOVE  FS-CADPECA     TO  FS-COD-STATUS.
    IF  FS-CADPECA      NOT EQUAL '00' AND '10'
        PERFORM 900-00-ERRO.
001-02-FIM.      EXIT.
*=====
001-03-FS-CADATU SECTION.
*=====
    MOVE  'CADATU'      TO  FS-ARQUIVO.
    MOVE  FS-CADATU     TO  FS-COD-STATUS.
    IF  FS-CADATU      NOT EQUAL '00' AND '10'
        PERFORM 900-00-ERRO.
001-03-FIM.      EXIT.
*=====
002-00-VER-ARQ-VAZIO SECTION.
*=====
    PERFORM 002-01-LER-CADPECA.
    IF  FS-CADPECA      EQUAL '10'
        DISPLAY '*****'
        DISPLAY '* * * * *'

```

```

DISPLAY '*                ARQUIVO CADPECA VAZIO                *'
DISPLAY '*                                                        *'
DISPLAY '*                PROGRAMA CANCELADO                    *'
DISPLAY '*                                                        *'
DISPLAY '*****'
PERFORM 004-00-FECHAR-ARQUIVOS
STOP    RUN.

002-00-FIM.          EXIT.
*=====
002-01-LER-CADPECA          SECTION.
*=====
MOVE    FS-LEITURA          TO    FS-OPERACAO.
READ    CADPECA.
IF FS-CADPECA          NOT EQUAL '10'
PERFORM 001-02-FS-CADPECA.
002-01-FIM.          EXIT.
*=====
003-00-TRATAR-CADPECA          SECTION.
*=====
MOVE    FS-GRAVACAO          TO    FS-OPERACAO.
MOVE    COD-PECA          TO    COD-ATUAL.
MOVE    NOME-PECA          TO    NOME-ATUAL.
MOVE    QTD-PECA          TO    QTD-ATUAL.
MOVE    LOCAL-PECA          TO    LOCAL-ATUAL.
MOVE    PUNIT-PECA          TO    PUNIT-ATUAL.
COMPUTE PTOTAL-ATUAL = QTD-PECA * PUNIT-PECA.
MOVE    CONTA-PECA          TO    CONTA-ATUAL.
WRITE    REG-CADATU          FROM    REG-CADPECA.
PERFORM    001-03-FS-CADATU.
PERFORM    002-01-LER-CADPECA.
003-00-FIM.          EXIT.
*=====
004-00-FECHAR-ARQUIVOS          SECTION.
*=====
MOVE    FS-FECHAMENTO          TO    FS-OPERACAO.

CLOSE    CADPECA
          CADATU.
PERFORM    001-01-TESTAR-FS.
004-00-FIM.          EXIT.
*=====
900-00-ERRO          SECTION.
*=====
DISPLAY '*****'
DISPLAY '*                *'
DISPLAY '* ERRO ' FS-OPERACAO ' DO ARQUIVO ' FS-ARQUIVO ' *'
DISPLAY '*                *'
DISPLAY '* FILE STATUS = ' FS-COD-STATUS
          ' *'

DISPLAY '*                *'
DISPLAY '* PROGRAMA ENCERRADO'
          '*

DISPLAY '*                *'
DISPLAY '*****'
STOP    RUN.
900-00-FIM.          EXIT.

```

**18 Comandos do Sort / Merge**

O módulo SORT-MERGE permite a classificação de arquivos ou a intercalação de dois ou mais arquivos previamente classificados por uma chave comum.

**18.1 Definição na ENVIRONMENT DIVISION.**

**Formato :**

```
SELECT <arquivo-sort> ASSIGN TO <literalnn>
      EXTERNAL <variável> |
      DYNAMIC <data-name1>
      [SORT STATUS IS |
      FILE STATUS IS <data-name2>]
```

Opção	Significado
<arquivo-sort>	Nome (interno) do arquivo sort-merge, no programa.
ASSIGN TO <literalnn>	Identifica o nome externo do arquivo através do <literalnn>.
ASSIGN TO DYNAMIC	Identifica o nome externo do arquivo através do <data-name1>, que deve ser definido na DATA DIVISION.
ASSIGN TO EXTERNAL	Identifica o nome externo do arquivo através de uma variável de ambiente do DOS, definida através do comando SET.
SORT STATUS IS	Indica, através de um valor no <data-name2>, o resultado das operações de classificação ou intercalação. <data-name2> deve ser definido na WORKING-STORAGE ou LINKAGE SECTION.
FILE STATUS IS	Equivalente ao "SORT STATUS IS".

**18.2 Definição na DATA DIVISION.****Formato :**

```
SD <arquivo-sort>  
[RECORD CONTAINS <literal> CHARACTERS  
 DATA RECORD IS | RECORDS ARE <data-name1> | <data-name2>] ...]  
<descrição-registro>]
```

Opção	Significado
<arquivo-sort>	Nome (interno) do arquivo sort-merge declarado na cláusula SELECT.
RECORD	Comprimento do registro lógico em bytes.
DATA RECORD	Nome(s) do(s) registro(s), através dos <data-names>.
<descrição-registro>	Estrutura do registro.

## 18.3 Sort

### Formato 1:

- Classifica arquivos.

```

SORT <arquivo-sort> ON ASCENDING KEY |
                        DESCENDING KEY <data-name1> ...
                        [ON ASCENDING | DESCENDING KEY <data-name2>]
                        [WITH DUPLICATES IN ORDER]
                        {INPUT PROCEDURE IS <procedure1>
                        [THRU | THROUGH <procedure2>]} {USING <arquivo2> ...}
                        {OUTPUT PROCEDURE IS <procedure3>
                        [THRU | THROUGH <procedure4>]} {GIVING <arquivo3>}
    
```

Opção	Significado
<arquivo-sort>	Arquivo-sort definido em nível SD
ASCENDING KEY	A classificação será em ordem ascendente.
DESCENDING KEY	A classificação será em ordem descendente.
WITH DUPLICATES	Prevê registros com chaves de classificação duplicadas.
INPUT PROCEDURE	Define uma rotina que realiza um processamento antes dos registros serem liberados para a classificação.
USING	Indica o nome do arquivo a ser classificado, definido em nível FD.
OUTPUT PROCEDURE	Define uma rotina para um processamento após a classificação.
GIVING	Indica o nome do arquivo resultante da classificação (o arquivo classificado). Deve ser definido em nível FD.

### Formato 2:

- Classifica tabelas.

```

SORT <tabela> [ON ASCENDING KEY |
              DESCENDING KEY <data-name>] ... ]
              [WITH DUPLICATES IN ORDER]
    
```

Opção	Significado
<tabela>	A tabela a ser classificada. Deve conter a cláusula OCCURS.
<data-name>	Chave de classificação. Pode ser a própria <tabela> ou um item subordinado a ela. Se for omitido, a <tabela> será a chave de classificação.
KEY	Indica o tipo de classificação (ascendente ou descendente). Poderá ser omitida apenas se a descrição da <tabela> contiver a opção KEY. Se for especificada, substituirá a opção KEY da <tabela>, caso exista.

#### 18.4 Merge

Intercala arquivos previamente classificados por uma chave comum.

**Formato :**

```
MERGE <arquivo-merge> [ON ASCENDING KEY |
                        DESCENDING KEY <data-namel>...
                        {USING <arquivo1> <arquivo2> ...
                        {OUTPUT PROCEDURE IS <procedure-namel>
                        [THRU | THROUGH <procedure-name2>]}] {GIVING <arquivo3>}
```

Opção	Significado
<arquivo merge>	Arquivo-merge definido em nível SD.
ASCENDING KEY	A intercalação será em ordem ascendente.
DESCENDING KEY	A intercalação será em ordem descendente.
USING	Indica os nomes dos arquivos a serem intercalados. Devem ser definidas em nível FD.
OUTPUT PROCEDURE	Define uma rotina para um processamento após a intercalação.
GIVING	Indica o nome do arquivo resultante da intercalação (o arquivo intercalado). Deve ser definido em nível FD.

#### 18.5 Release

Libera registros para classificação. Usado na INPUT PROCEDURE do SORT.

**Formato :**

```
RELEASE <registro> FROM <área>...
```

Opção	Significado
FROM	Conteúdo (registro) a ser liberado para classificação

**18.6 Return**

Lê registros classificados / intercalados pelos comandos SORT ou MERGE, respectivamente. Usado na OUTPUT PROCEDURE do SORT ou MERGE.

**Formato :**

```
RETURN <arquivo-sort> RECORD [INTO <área>]
      AT END <comando1>
      [NOT AT END <comando2>]
[END-RETURN]
```

Opção	Significado
<arquivo-sort>	Arquivo sort ou merge definido em nível SD.
INTO	Torna o registro lido disponível também na <área> especificada.
AT END	Condição de fim-de-arquivo.
NOT AT END	Condição contrária à de AT END.

**Exemplo de programa com SORT.**

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. FUTURE02.  
AUTHOR. FUTURE SCHOOL CURSOS DE COMPUTACAO.
```

```
*  
* EMITIR RELATORIO DO CADASTRO DE ALUNOS  
* CLASSIFICADO POR NOME DE ALUNO
```

```
*  
ENVIRONMENT DIVISION.  
CONFIGURATION SECTION.  
SPECIAL-NAMES.  
  
DECIMAL-POINT IS COMMA.
```

```
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
SELECT CADALUNO ASSIGN TO UT-S-CADALUNO  
FILE STATUS IS FS-CADALUNO.
```

```
*  
SELECT SORALUNO ASSIGN TO UT-S-SORALUNO.
```

```
*  
SELECT RELATO ASSIGN TO PRINTER  
FILE STATUS IS FS-RELATO.
```

```
*  
DATA DIVISION.  
FILE SECTION.
```

```
*  
FD CADALUNO  
RECORD CONTAINS 60 CHARACTERS  
LABEL RECORD IS STANDARD  
DATA RECORD IS REG-CADALUNO.
```

```
01 REG-CADALUNO.  
05 CODALU PIC 9(04).  
05 NOMEALU PIC X(30).  
05 TURMAALU PIC 9(03).  
05 NOTA1ALU PIC 9(02)V99.  
05 NOTA2ALU PIC 9(02)V99.  
05 NOTA3ALU PIC 9(02)V99.  
05 NOTA4ALU PIC 9(02)V99.  
05 FILLER PIC X(07).
```

```
*  
FD RELATO  
LABEL RECORD IS OMITTED  
RECORD CONTAINS 77 CHARACTERS  
DATA RECORD IS REG-RELATO.  
01 REG-RELATO PIC X(77).
```

```
*  
SD SORALUNO  
DATA RECORD IS REG-SORALUNO.  
01 REG-SORALUNO.  
05 CODSOR PIC 9(04).  
05 NOMESOR PIC X(30).  
05 TURMASOR PIC 9(03).  
05 NOTA1SOR PIC 9(02)V99.  
05 NOTA2SOR PIC 9(02)V99.  
05 NOTA3SOR PIC 9(02)V99.  
05 NOTA4SOR PIC 9(02)V99.  
05 FILLER PIC X(07).
```

```

*
WORKING-STORAGE SECTION.
*
* AREAS DE MEMORIA
*
01  WS-FIM-ARQ           PIC X(03)    VALUE SPACES.
01  WS-DATA-SYS.
    05  WS-ANO-SYS       PIC 9(02)    VALUE ZEROS.
    05  WS-MES-SYS      PIC 9(02)    VALUE ZEROS.
    05  WS-DIA-SYS      PIC 9(02)    VALUE ZEROS.
01  WS-HORARIO-SYS.
    05  WS-HORA-SYS     PIC 9(02)    VALUE ZEROS.
    05  WS-MINU-SYS     PIC 9(02)    VALUE ZEROS.
*
* ACUMULADORES
*
77  ACUM-LINHAS         PIC 9(02)    VALUE 99.
77  ACUM-PAG           PIC 9(05)    VALUE ZEROS.
*
* FILE STATUS
*
77  FS-CADALUNO        PIC X(02)    VALUE SPACES.
77  FS-RELATO          PIC X(02)    VALUE SPACES.
77  FS-OPERACAO        PIC X(13)    VALUE SPACES.
77  FS-ABERTURA       PIC X(13)    VALUE 'NA ABERTURA'.
77  FS-LEITURA        PIC X(13)    VALUE 'NA LEITURA'.
77  FS-GRAVACAO        PIC X(13)    VALUE 'NA GRAVACAO'.
77  FS-FECHAMENTO      PIC X(13)    VALUE 'NO FECHAMENTO'.
*
* LINHAS DO RELATORIO
*
01  CAB001.
    05  FILLER           PIC X(66)    VALUE
        'FUTURE SCHOOL CURSOS DE COMPUTACAO'.
    05  FILLER           PIC X(06)    VALUE 'PAG.: '.
    05  CAB001-PAG      PIC Z.ZZ9.
*
01  CAB002.
    05  FILLER           PIC X(51)    VALUE
        'RELATORIO DO CADASTRO DE ALUNOS'.
    05  CAB002-HORA     PIC 99.
    05  FILLER           PIC X(01)    VALUE ':'.
    05  CAB002-MINU     PIC 99.
    05  FILLER           PIC X(11)    VALUE SPACES.
    05  CAB002-DIA     PIC 99/.
    05  CAB002-MES     PIC 99/.
    05  FILLER           PIC X(02)    VALUE '20'.
    05  CAB002-ANO     PIC 99.
*
01  CAB003.
    05  FILLER           PIC X(14)    VALUE 'CODIGO'.
    05  FILLER           PIC X(26)    VALUE
        'NOME DO ALUNO'.
    05  FILLER           PIC X(09)    VALUE 'TURMA'.
    05  FILLER           PIC X(08)    VALUE 'NOTA'.
    05  FILLER           PIC X(08)    VALUE 'NOTA'.
    05  FILLER           PIC X(08)    VALUE 'NOTA'.
    05  FILLER           PIC X(04)    VALUE 'NOTA'.

```

```

*
01 CAB004.
   05 FILLER                PIC X(48)    VALUE SPACES.
   05 FILLER                PIC X(08)    VALUE '1.BIM'.
   05 FILLER                PIC X(08)    VALUE '2.BIM'.
   05 FILLER                PIC X(08)    VALUE '3.BIM'.
   05 FILLER                PIC X(05)    VALUE '4.BIM'.
*
01 DET001.
   05 FILLER                PIC X(01)    VALUE ' '.
   05 DET001-CODIGO        PIC 9(04) .
   05 FILLER                PIC X(03)    VALUE SPACES.
   05 DET001-NOME         PIC X(30) .
   05 FILLER                PIC X(03)    VALUE SPACES.
   05 DET001-TURMA        PIC 9(03) .
   05 FILLER                PIC X(04)    VALUE SPACES.
   05 DET001-1BIM         PIC Z9,99.
   05 FILLER                PIC X(03)    VALUE SPACES.
   05 DET001-2BIM         PIC Z9,99.
   05 FILLER                PIC X(03)    VALUE SPACES.
   05 DET001-3BIM         PIC Z9,99.
   05 FILLER                PIC X(03)    VALUE SPACES.
   05 DET001-4BIM         PIC Z9,99.
*
PROCEDURE          DIVISION.
*=====
000-00-INICIO                SECTION.
*=====
      SORT      SORALUNO      ASCENDING  KEY      NOMESOR
              INPUT          PROCEDURE   001-00-CLASSIFICAR-ARQUIVO
              OUTPUT         PROCEDURE   002-00-LISTAR-ARQUIVO.
      PERFORM   003-00-FECHAR-ARQUIVOS.
      STOP     RUN.
001-00-FIM.                  EXIT.
*=====
001-00-CLASSIFICAR-ARQUIVO  SECTION.
*=====
      PERFORM   001-01-ABRIR-ARQUIVOS.
      PERFORM   001-06-VER-CADALUNO-VAZIO.
      PERFORM   001-07-CLASSIFICACAO      UNTIL
              FS-CADALUNO      EQUAL    '10'.
001-00-FIM.                  EXIT.
*=====
001-01-ABRIR-ARQUIVOS      SECTION.
*=====
      MOVE     FS-ABERTURA      TO      FS-OPERACAO.
      OPEN    INPUT  CADALUNO
              OUTPUT  RELATO.
      PERFORM   001-02-TESTAR-FS.
001-01-FIM.                  EXIT.
*=====
001-02-TESTAR-FS          SECTION.
*=====
      PERFORM   001-03-TESTAR-FS-CADALUNO.
      PERFORM   001-04-TESTAR-FS-RELATO.
001-02-FIM.                  EXIT.
*=====
001-03-TESTAR-FS-CADALUNO  SECTION.

```

```

*=====
      MOVE      'CADALUNO'                TO      FS-ARQUIVO.
      MOVE      FS-CADALUNO                TO      FS-COD-STATUS.
      IF  FS-CADALUNO      NOT EQUAL  '00' AND '10'
          PERFORM 900-00-ERRO.
001-03-FIM.                                EXIT.

*=====
001-04-TESTAR-FS-RELATO                    SECTION.
*=====
      MOVE      'RELATO'                  TO      FS-ARQUIVO.
      MOVE      FS-RELATO                 TO      FS-COD-STATUS.
      IF  FS-RELATO      NOT EQUAL  '00' AND '10'
          PERFORM 900-00-ERRO.
001-04-FIM.                                EXIT.

*=====
001-06-VER-CADALUNO-VAZIO                 SECTION.
*=====
      PERFORM      001-08-LER-CADALUNO.
      IF  FS-CADALUNO      EQUAL      '10'
          DISPLAY '*****'
          DISPLAY '*      ARQUIVO CADALUNO VAZIO      *'
          DISPLAY '*      PROGRAMA CANCELADO      *'
          DISPLAY '*****'
          PERFORM 003-00-FECHAR-ARQUIVOS
          STOP  RUN.
001-06-FIM.                                EXIT.

*=====
001-07-CLASSIFICACAO                     SECTION.
*=====
      MOVE      REG-CADALUNO              TO      REG-SORALUNO.
      RELEASE   REG-SORALUNO.
      PERFORM   001-08-LER-CADALUNO.
001-07-FIM.                                EXIT.

*=====
001-08-LER-CADALUNO                      SECTION.
*=====
      MOVE      FS-ABERTURA              TO      FS-OPERACAO
      READ      CADALUNO.
      IF  FS-CADALUNO      NOT EQUAL  '10'
          PERFORM 001-03-TESTAR-FS-CADALUNO.
001-08-FIM.                                EXIT.

*=====
002-00-LISTAR-ARQUIVO                    SECTION.
*=====
      PERFORM      002-01-OBTER-DATA-HORA.
      PERFORM      002-02-LER-SORALUNO.
      PERFORM      002-03-IMPRIMIR-SORALUNO UNTIL
          WS-FIM-ARQ      EQUAL      'FIM'.
002-00-FIM.                                EXIT.

*=====
002-01-OBTER-DATA-HORA                   SECTION.
*=====
      ACCEPT     WS-DATA-SYS              FROM    DATE.
      MOVE      WS-DIA-SYS                TO      CAB002-DIA.
      MOVE      WS-MES-SYS                TO      CAB002-MES.
      MOVE      WS-ANO-SYS                TO      CAB002-ANO.

```

```

ACCEPT      WS-HORARIO-SYS      FROM      TIME.
MOVE        WS-HORA-SYS       TO        CAB002-HORA.
MOVE        WS-MINU-SYS      TO        CAB002-MINU.
002-01-FIM.          EXIT.
*=====
002-02-LER-SORALUNO          SECTION.
*=====
MOVE        FS-LEITURA      TO        FS-OPERACAO.
RETURN     SORALUNO
           AT      END
           MOVE  'FIM'      TO        WS-FIM-ARQ.
002-02-FIM.          EXIT.
*=====
002-03-IMPRIMIR-SORALUNO    SECTION.
*=====
MOVE        FS-GRAVACAO      TO        FS-OPERACAO.
IF  ACUM-LINHAS      GREATER 59
   PERFORM 002-04-CABECALHOS.
MOVE        CODSOR           TO        DET001-CODIGO.
MOVE        NOMESOR          TO        DET001-NOME.
MOVE        TURMASOR         TO        DET001-TURMA.
MOVE        NOTA1SOR         TO        DET001-1BIM.
MOVE        NOTA2SOR         TO        DET001-2BIM.
MOVE        NOTA3SOR         TO        DET001-3BIM.
MOVE        NOTA4SOR         TO        DET001-4BIM.
WRITE      REG-RELATO        FROM      DET001  AFTER  1.
PERFORM    001-04-TESTAR-FS-RELATO.
ADD        1                 TO        ACUM-LINHAS.
PERFORM    002-02-LER-SORALUNO.
002-03-FIM.          EXIT.
*=====
002-04-CABECALHOS          SECTION.
*=====
ADD        1                 TO        ACUM-PAG.
MOVE      ACUM-PAG          TO        CAB001-PAG.
WRITE     REG-RELATO        FROM      CAB001  AFTER  PAGE.
PERFORM   001-04-TESTAR-FS-RELATO.
WRITE     REG-RELATO        FROM      CAB002  AFTER  1.
PERFORM   001-04-TESTAR-FS-RELATO.
WRITE     REG-RELATO        FROM      CAB003  AFTER  2.
PERFORM   001-04-TESTAR-FS-RELATO.
WRITE     REG-RELATO        FROM      CAB004  AFTER  1.
PERFORM   001-04-TESTAR-FS-RELATO.
MOVE      SPACES            TO        REG-RELATO.
WRITE     REG-RELATO        AFTER    1.
PERFORM   001-04-TESTAR-FS-RELATO.
MOVE      06                TO        ACUM-LINHAS.
002-04-FIM.          EXIT.
*=====
003-00-FECHAR-ARQUIVOS     SECTION.
*=====
MOVE      FS-FECHAMENTO      TO        FS-OPERACAO.
CLOSE    CADALUNO
         RELATO.
PERFORM   001-02-TESTAR-FS.
003-00-FIM.          EXIT.
*=====
900-00-ERRO              SECTION.

```

```
*=====*
DISPLAY '*****'
DISPLAY '* *'
DISPLAY '* ERRO ' FS-OPERACAO ' DO ARQUIVO ' FS-ARQUIVO ' *'
DISPLAY '* *'
DISPLAY '* FILE STATUS = ' FS-COD-STATUS
' *'
DISPLAY '* *'
DISPLAY '* PROGRAMA ENCERRADO'
' *'
DISPLAY '* *'
DISPLAY '*****'
STOP RUN.
900-00-FIM. EXIT.
```