

# **COBOL pro IBM i**

Úvod do programování

# Obsah

<b>Předmluva .....</b>	<b>5</b>
<b>Úvod.....</b>	<b>5</b>
<b>Celková struktura programu .....</b>	<b>5</b>
<i>Struktura prostého programu.....</i>	<i>6</i>
Identification Division .....	6
Environment Division .....	6
Data Division .....	6
Procedure Division .....	6
<i>Podrobnější struktura prostého programu .....</i>	<i>7</i>
<i>Obecná struktura zdrojového programu .....</i>	<i>8</i>
Vnější program .....	8
Vnořený program .....	8
<i>Formulář .....</i>	<i>8</i>
<b>Stručný Přehled některých příkazů .....</b>	<b>9</b>
<i>Výpočetní příkazy .....</i>	<i>9</i>
<i>Příkaz PERFORM – základní tvary .....</i>	<i>9</i>
In-line .....	9
Out-of-line .....	9
<i>Provedení cyklu – varianta out-of-line .....</i>	<i>9</i>
Cyklus s podmínkou .....	9
Čítaný cyklus .....	9
<i>Provedení cyklu – varianta in-line .....</i>	<i>9</i>
<i>Příkazy pro vstup a výstup databázových souborů .....</i>	<i>9</i>
<i>Příkaz pro ladicí výpis.....</i>	<i>10</i>
<i>Volání programu .....</i>	<i>10</i>
<i>Ukázka složitějšího příkazu ve formě věty (sentence).....</i>	<i>10</i>
<b>Definice a zobrazení dat (data description entry).....</b>	<b>11</b>
<i>Běžný popis dat .....</i>	<i>11</i>
PICTURE .....	11
USAGE .....	11
Program PICTURE1.....	12
Program PICTURE2.....	12
Program PICTURE3.....	13
<i>Redefinice paměti.....</i>	<i>13</i>
<i>Definice analogií .....</i>	<i>14</i>
<i>Definice vlastního typu.....</i>	<i>14</i>
<i>Popis indikátorů .....</i>	<i>15</i>
Program INDIC1.....	15
<i>Popis podmínkových jmen.....</i>	<i>15</i>
Program COND1 .....	15
<i>Popis pro jméno konstanty .....</i>	<i>16</i>
<b>Manipulace s daty.....</b>	<b>16</b>
<i>Obrazné (figurativní) konstanty.....</i>	<i>16</i>

<i>Odkazy na data</i> .....	16
Kvalifikace .....	16
Indexování.....	16
Modifikace pozice a délky dat .....	16
Odkaz na funkci .....	17
<i>Transformace dat příkazem MOVE</i> .....	17
Program MOVE1 .....	17
Program MOVE2 .....	18
<i>Příkazy STRING a UNSTRING</i> .....	19
Program STRING .....	19
Program UNSTRING .....	20
<b>Tabulky</b> .....	<b>21</b>
Program TABULKY1 .....	21
Program TABULKY2 .....	22
Program TABULKY3 .....	22
<b>Datum a čas</b> .....	<b>23</b>
Program DATETIME1 .....	23
Program DATETIME2 .....	24
Program DATETIME3 .....	24
<b>Aktualizace stavového souboru obrátovým souborem</b> .....	<b>26</b>
<i>Referenční soubor REF</i> .....	26
<i>Popis souboru STAVY</i> .....	26
<i>Popis souboru OBRATY</i> .....	26
<i>Program STAOBR</i> .....	27
Vysvětlivky k programu .....	28
<i>Program STAOBR_2</i> .....	31
<b>Kontrola obrátů a tisk</b> .....	<b>33</b>
<i>Program STAOBRTISK</i> .....	33
<b>Jednoduché obrazkové soubory</b> .....	<b>35</b>
<i>Popis obrazkového souboru STAVYW</i> .....	36
<i>Program STAPOR pro pořízení stavů</i> .....	38
Vysvětlivky k programu .....	39
<b>Obrazkové soubory s podsoubory (subfiles)</b> .....	<b>43</b>
<i>Popis obrazkového souboru STAVYW1</i> .....	44
<i>Program STADSP – zobrazení stavů podsouborem</i> .....	46
Vysvětlivky k programu .....	48
<b>Opravy databázového souboru s použitím podsouboru</b> .....	<b>50</b>
<i>Obrazkový soubor STAVYW2</i> .....	51
<i>Program STAUPD pro opravy souboru STAVY</i> .....	51
<b>Údržba databázového souboru s podsouborem a okny</b> .....	<b>54</b>
<i>Obrazkový soubor STAVYW3I</i> .....	54
<i>Program STAUDRI pro údržbu souboru STAVY</i> .....	55

<b>Volání programu .....</b>	<b>60</b>
Volající program CALLING .....	60
Volaný program CALLED .....	61
Program CALLNESTED .....	61
<b>Prostředky k ladění programů.....</b>	<b>63</b>
<i>Prostředky jazyka .....</i>	<i>63</i>
PROGRAM STATUS .....	63
ON SIZE ERROR .....	64
ON OVERFLOW .....	64
AT END .....	64
INVALID KEY .....	64
NO DATA .....	64
USE AFTER EXCEPTION/ERROR .....	64
Program FILESTATUS .....	64
<i>Prostředky operačního systému .....</i>	<i>66</i>
Výpis paměti (QInDumpCobol API) .....	66
Ladící program .....	67
Systémový výpis paměti po havárii .....	69
Výpis protokolu úlohy (DSPJOBLOG).....	70

## PŘEDMLUVA

Tato publikace je určena účastníkům kurzu, kteří se ještě nesetkali s jazykem COBOL, ale umí programovat v některém jiném programovacím jazyku. Předpokládá se, že znají základy systému IBM i (dříve AS/400) včetně řídicího jazyka CL, jsou obeznámeni s konceptem integrované databáze DB2 a znají způsob popisu DDS (Data Description Specifications) souborů v databázi a na obrazovkách.

## ÚVOD

Programovací jazyk COBOL vznikl v 50. letech ve Spojených státech amerických jako druhý „vyšší programovací jazyk“ po jazyku FORTRAN. Zatímco jazyk FORTRAN byl určen pro vědeckotechnické výpočty, jazyk COBOL (COMmon Business Oriented Language) byl určen pro obchodní aplikace.

V jazyku FORTRAN mohla být jména proměnných dlouhá nejvýše 6 znaků. Aritmetické operace se psaly např. takto:

$$E=P*Q$$

V jazyku COBOL se vycházelo spíše účetním, takže podobný výpočet se zapsal větou:

**MULTIPLY PRICE BY QUANTITY GIVING EXTENDED-AMOUNT.**

Později, jak se účetní začali zabývat programováním, se některé dlouhé výrazy zkracovaly, aby nebylo tolik práce s psáním programu.

Přestože se již dlouho říká, že COBOL je mrtvý jazyk, není to tak. Existuje nemálo firem, které provozují aplikace v něm napsané, a ty je zapotřebí udržovat, to znamená opravovat a přizpůsobovat novým požadavkům.

Jazyk COBOL má historicky pevně danou programovou strukturu, která vůbec nepřipomíná žádný novodobý programovací jazyk.

Také terminologie pro popis struktury programu je zcela odlišná – division, section, paragraph, entry, sentence, clause, phrase. Jediný termín je podobný – statement.

Označení jednoduché proměnné číslem 77 také není zrovna systémové.

Na tyto a jiné podivnosti si programátor, znalý třeba jazyka Java, musí nějakou dobu zvykat. Potíže však nejsou nepřekonatelné a navíc, jakmile jednou vytvoří jeden nebo několik cobolských programů, může je v podstatě používat jako šablony. To se také v praxi děje.

V tomto kurzu probíráme jen nejnutnější prostředky jazyka tak, aby účastníci vytvořili několik typických programů v omezeném čase.

Kdo se bude chtít dozvědět více, musí se obrátit na dokumentaci IBM:

ILE COBOL Language Reference

ILE COBOL Programmer's Guide

## CELKOVÁ STRUKTURA PROGRAMU

Cobolský zdrojový program může být prostý nebo může obsahovat vnořené zdrojové programy.

## Struktura prostého programu

V následujícím schematu je hrubě znázorněna hierarchická struktura prostého cobolského programu. Čtyři oddíly – divize – byly původně povinné, ale časem se pravidla uvolnila a např. identifikační divizi lze někdy zcela vynechat. Prvky tohoto schématu se do programu zařazují podle potřeby.

### Identification Division

*Paragraphs*  
*Entries*  
*Clauses*

### Environment Division

*Sections*  
*Paragraphs*  
*Entries*  
*Clauses*  
*Phrases*

### Data Division

*Sections*  
*Entries*  
*Clauses*  
*Phrases*

### Procedure Division

*Sections*  
*Paragraphs*  
*Sentences*  
*Statements*  
*Phrases*

- **Sekce a paragrafy** definují program.
- **Entry** (zápis) je série klausulí končící tečkou.
- **Clause** (vedlejší věta – klausule) je uspořádaná množina za sebou následujících platných znakových řetězců jazyka COBOL, která určuje atribut pro Entry.
- **Sentence** (věta) je posloupnost jednoho nebo více příkazů (statements) končící tečkou.
- **Statement** (příkaz) je platná kombinace slovesa (verb) jazyka COBOL a jeho operandů. Určuje akci prováděnou programem.
- **Phrase** (fráze). Každá klausule nebo příkaz v programu může být rozdělena na menší jednotky zvané fráze.

## Podrobnější struktura prostého programu

### IDENTIFICATION DIVISION.

PROGRAM-ID.  
program-name ... .

### ENVIRONMENT DIVISION.

CONFIGURATION SECTION.  
SOURCE-COMPUTER. computer-name ... .  
OBJECT-COMPUTER. computer-name ... .  
SPECIAL-NAMES. ... .  
INPUT-OUTPUT SECTION.  
FILE-CONTROL. entries .  
...  
IO-CONTROL.  
SELECT. file-name ASSIGN external-medium ... .  
...

### DATA DIVISION.

FILE SECTION.  
FD file-name ... .  
level-number [data-name | FILLER] ... .  
WORKING-STORAGE SECTION.  
level-number [data-name | FILLER] ... .  
LOCAL-STORAGE SECTION.  
level-number [data-name | FILLER] ... .  
LINKAGE SECTION.  
level-number [data-name | FILLER] ... .

### PROCEDURE DIVISION USING parameters ... .

DECLARATIVES.  
section-name SECTION.  
USE statement.  
paragraph-name.  
sentence  
statement  
phrase  
END-DECLARATIVES.

section-name SECTION.  
paragraph-name.  
sentence  
statement  
phrase

## Obecná struktura zdrojového programu

Na nejvyšší úrovni se program nazývá kompilační jednotka. Ta může začínat příkazem PROCESS s volbami pro kompilaci. Dále pokračuje vnějším zdrojovým programem s případnými vnořenými programy.

Vnější program může obsahovat několik vnořených programů. Vnořený program může také obsahovat několik vnořených programů.

### Vnější program

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. jméno-vnějšího-programu.  
...  
ENVIRONMENT DIVISION.  
...  
DATA DIVISION.  
...  
PROCEDURE DIVISION ... .  
...  
vnořený program  
vnořený program  
...  
END PROGRAM jméno-vnějšího-programu.
```

### Vnořený program

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. jméno-vnořeného-programu.  
...  
ENVIRONMENT DIVISION.  
...  
DATA DIVISION.  
...  
PROCEDURE DIVISION ... .  
...  
vnořený program  
vnořený program  
...  
END PROGRAM jméno-vnořeného-programu.
```

Vnořené programy se volají příkazem CALL z vyšších úrovní.

## Formulář

Zdrojový program se zapisuje do řádků dlouhých 80 sloupců členěných na rubriky. Pro zápis platí následující pravidla.

	11	2	3	4	5	6	7	77	8											
1...	5	7...	12...	5...	0...	5...	0...	5...	0...	5...	0...	5...	0...	5...	0...	5...	0...	23	5...	0
.....	-A+++B	+++++Pgm-id++																		

- Prvních 6 sloupců je původně určeno pro číslování řádků, ale lze tam zapsat cokoliv.
- Sloupec 7 označený pomlčkou slouží pro příznak komentáře. Zapisuje se tam znak hvězdička (\*) nebo lomítko (/). Slouží také k označení pokračovacího řádku znakem pomlčka (-).
- Oblast A (čtyři sloupce 8 až 11). Začíná v ní  
zahájení divize (IDENTIFICATION DIVISION atd.),  
zahájení sekce,  
paragraf,  
indikátor FD,  
indikátor úrovně (level-indicator).



- Oblast B (sloupce 12 až 72). V ní musí být zapsán  
zápis (entry),  
věta (sentence),  
příkaz (statement),  
klausule (clause).
- Pgm-id (73 až 80). Původně určeno pro identifikaci programu, ale lze zapsat cokoliv.

## STRUČNÝ PŘEHLED NĚKTERÝCH PŘÍKAZŮ

### Výpočetní příkazy

CONTINUE	prázdný příkaz
COMPUTE jméno = aritmetický-výraz ...	
MOVE jméno-1 TO jméno-2 jméno-2 ...	
MOVE CORR[ESPONDING] jméno-1 TO jméno-2	
ADD jméno-1 TO TO jméno-2 [GIVING jméno-3]	
SUBTRACT jméno-1 FROM jméno-2 [GIVING jméno-3]	
MULTIPLY jméno-1 BY jméno-2 [GIVING jméno-3]	
DIVIDE jméno-1 INTO jméno-2 [GIVING jméno-3]	
STRING jméno-1 DELIMITED BY jméno-2 INTO jméno-3	
UNSTRING jméno-1 [DELIMITED BY jméno-2 OR jméno-3]	
INTO jméno-4 DELIMITER IN jméno-5 COUNT IN jméno-6	
GO [TO] sekce   paragraf	
IF podmínka [THEN] příkazy [ELSE příkazy]	

### Příkaz PERFORM – základní tvary

#### In-line

PERFORM příkazy END-PERFORM	provede příkazy
-----------------------------	-----------------

#### Out-of-line

PERFORM section-name ...	provede sekci
PERFORM paragraf-1 TROUGH paragraf-2	provede příkazy mezi paragrafy

### Provedení cyklu – varianta out-of-line

PERFORM ... [n TIMES]	provede příkazy [n krát]
-----------------------	--------------------------

#### Cyklus s podmínkou

PERFORM ... [TEST BEFORE] UNTIL podmínka	odpovídá DO WHILE
AFTER	odpovídá DO UNTIL

#### Čítaný cyklus

PERFORM ... VARYING jméno-1 FROM index-1 TO index-2	
[TEST BEFORE] UNTIL podmínka	
AFTER	

### Provedení cyklu – varianta in-line

PERFORM [n TIMES] příkazy END-PERFORM	provede příkazy [n krát]
---------------------------------------	--------------------------

PERFORM [TEST BEFORE] příkazy UNTIL podmínka	odpovídá DO WHILE
AFTER	odpovídá DO UNTIL

příkazy	
END-PERFORM	cyklus s podmínkou

PERFORM VARYING jméno-1 FROM index-1 TO index-2	
[TEST BEFORE] UNTIL podmínka	
AFTER	
příkazy	
END-PERFORM	čítaný cyklus

### Příkazy pro vstup a výstup databázových souborů

OPEN INPUT file-name	
OUTPUT	
I-O	

## Příkaz pro ladicí výpis

výpis programových dat

CALL "jméno-programu" USING parametry ...

Všimněme si, že příkazy lze psát velkými nebo malými písmeny v libovolné kombinaci.

```

Move ZAVOD of OBRATYF0 to ZAVOD of STAVYF0
Move SKLAD of OBRATYF0 to SKLAD of STAVYF0
Move MATER of OBRATYF0 to MATER of STAVYF0

Read STAVY
    invalid key
        Continue
    not invalid key
        Add MNOBR of OBRATYF0 to MNOZ of STAVYF0
        Rewrite STAVYR invalid key Continue End-rewrite
End-read
Read OBRATY at end set END-FILE to TRUE End-read
end-perform.

```

- ©Vladimír Župka 2016

## DEFINICE A ZOBRAZENÍ DAT (DATA DESCRIPTION ENTRY)

Data se definují zápisem zvaným *data description entry*, který má několik formátů. Formáty uvedeme v silně zjednodušené a zestručněné podobě s ohledem na srozumitelnost. Podrobnosti je třeba hledat v dokumentaci IBM.

### Běžný popis dat

*level name occurs-clause picture-clause value-clause usage-clause.*

např.

```
05 MNOZ OCCURS 10 TIMES PICTURE IS S9(8) VALUE IS 152 USAGE IS COMP-3.
```

*level* je číslo od 01 do 49 nebo 77. Level 01 a 77 musí začínat v oblasti A formuláře, ostatní i v oblasti B.

*name* je buď jméno proměnné nebo FILLER (výplňková položka bez proměnné).

*occurs-clause* má tvar OCCURS number (TIMES), kde number je počet výskytů položky.

*picture-clause* má tvar PIC(TURE IS) string, kde string je posloupnost definičních znaků.

*value-clause* má tvar VALUE (IS) literal, kde literal je konstanta, kterou položka získává na začátku výpočtu.

*usage-clause* má tvar USAGE (IS) usage, kde usage označuje použití, např. COMP k výpočtu, DISPLAY k zobrazení apod.

Pořadí klausulí je libovolné. Data se popisují obecně jako hierarchické struktury. Kořen je označen úrovní 01 (nebo 1). Další úrovně mohou mít libovolná čísla do 49, ale každá další úroveň musí mít vyšší číslo. Úroveň 77 není součástí hierarchické struktury, označuje samostatnou elementární položku.

### PICTURE

Znakový řetězec pro needitovaná čísla se skládá ze znaků

9, P, S, V a případně údaje o počtu číslic v závorkách, např.

PIC S99999V99

pro číslo se znaménkem (S), pěti číslicemi celků (99999), dvěma desetinnými místy (99) za znakem určujícím polohu myšlené desetinné čárky (V).

PIC 99999999PP

pro číslo bez znaménka s myšlenou desetinnou čárkou posunutou o dvě místa vpravo od jednotek.

PIC SPP9999999

pro číslo se znaménkem s myšlenou desetinnou čárkou posunutou o dvě místa vlevo od nejvyšší číslice.

### USAGE

Binární čísla: BINARY, COMP-4

Dekadická pakovaná čísla: PACKED-DECIMAL, COMP-3

Čísla s pohyblivou čárkou: COMP-1 (4 bajty), COMP-2 (8 bajtů)

Číselná a znaková data: DISPLAY

Index: INDEX

Ukazatel na data: POINTER

Ukazatel na proceduru: PROCEDURE-POINTER

## Program PICTURE1

Ilustruje dekadická čísla bez editace.

```
PROCESS OPTIONS NOMONOPRC.
WORKING-STORAGE SECTION.
    COPY DUMP_FULL.
77 CISLO                PIC 99999V99.
77 CISLO_DISP           PIC 99999V99  USAGE DISPLAY.
77 CISLO_COMP-3         PIC 99999V99  USAGE COMP-3.
77 CISLO_S              PIC S9(5)V9(2) DISPLAY.
77 CISLO_S_PACK         PIC S99999V99  PACKED-DECIMAL.
77 CISLO_P_UP           PIC 9999999PP  PACKED-DECIMAL.
77 CISLO_P_DWN          PIC SPP9999999.

PROCEDURE DIVISION.
PAR.
    MOVE 12345.67      TO  CISLO
    MOVE 12345.67      TO  CISLO_DISP
    MOVE 12345.67      TO  CISLO_COMP-3
    MOVE -12345.67     TO  CISLO_S
    MOVE -12345.67     TO  CISLO_S_PACK
    MOVE 123456700     TO  CISLO_P_UP
    MOVE .001234567    TO  CISLO_P_DWN

    CALL LINKAGE IS PROCEDURE "QlnDumpCobol" USING
                                OMITTED, OMITTED,
                                OMITTED, PROGRAM-TYPE,
                                DUMP-TYPE, ERROR-CODE
```

Výpis paměti:

CISLO	ZONED(7 2)	12345.67 "F1F2F3F4F5F6F7"X
CISLO_COMP-3	PACKED(7 2)	12345.67 "1234567F"X
CISLO_DISP	ZONED(7 2)	12345.67 "F1F2F3F4F5F6F7"X
CISLO_P_DWN	ZONED(7 0)	.001234567 "F1F2F3F4F5F6F7"X
CISLO_P_UP	PACKED(7 0)	123456700. "1234567F"X
CISLO_S	ZONED(7 2)	-12345.67 "F1F2F3F4F5F6D7"X
CISLO_S_PACK	PACKED(7 2)	-12345.67 "1234567D"X

## Program PICTURE2

Ilustruje binární čísla bez editace. Binární čísla od 1 do 4 číslic (dekadických) jsou 2bajtová, od 5 do 9 číslic jsou 4bajtová, od 10 do 18 číslic jsou 8bajtová. Binární čísla mají vždy znaménko v bitu nejvyššího řádu.

Slovo NOSTDTRUNC dovoluje rozšířit zmíněné rozsahy (dekadických) číslic na větší. Například do 4bajtového čísla se vejde od - 32768 do 32767 (2 na 15 minus 1) místo -9999 až 9999.

```
PROCESS OPTIONS NOMONOPRC NOSTDTRUNC.

WORKING-STORAGE SECTION.
    COPY DUMP_FULL.
77 CISLO_B2             PIC 9(4)      BINARY.
77 CISLO_B4             PIC S9(9)     BINARY.
77 CISLO_B8             PIC 9(18)     COMP-4.

PROCEDURE DIVISION.
    MOVE 32767          TO  CISLO_B2
```

```

MOVE 2147483647          TO CISLO_B4
MOVE 9223372036854775807 TO CISLO_B8

CALL LINKAGE IS PROCEDURE "QlnDumpCobol" USING
                                OMITTED, OMITTED,
                                OMITTED, PROGRAM-TYPE,
                                DUMP-TYPE, ERROR-CODE

```

### Výpis paměti:

```

CISLO_B2      BIN(2)          32767.
                                "7FFF"X
CISLO_B4      BIN(4)          2147483647.
                                "7FFFFFFF"X
CISLO_B8      BIN(8)          0922337203685477580.
                                "0CCCCCCCCCCCCC"X

```

Znakový řetězec pro editovaná čísla může obsahovat další znaky:

Z ... číslice, je-li nulová je nahrazena mezerou,

B ... na dané místo se vloží mezera,

0 / , . + - CR DB \$ ... vloží se na dané místo.

### Program PICTURE3

Ilustruje editovaná čísla, vesměs s USAGE DISPLAY. Místo dolaru si volíme měnový symbol Kč pomocí klausule CURRENCY SIGN IS v paragrafu SPECIAL-NAMES. V editovaném čísle bude text Kč, v klausuli PIC bude písmeno K.

```

CONFIGURATION SECTION.
SPECIAL-NAMES.
    CURRENCY SIGN IS "Kč" WITH PICTURE SYMBOL "K".

WORKING-STORAGE SECTION.
77 CISLO_ED1      PIC 9999.9          DISPLAY.
77 CISLO_ED2      PIC KBZZZ99.99-.
77 CISLO_ED3      PIC ZZZZ.9-.

PROCEDURE DIVISION.
    MOVE -123.4    TO CISLO_ED1
    MOVE -12.34    TO CISLO_ED2
    MOVE -123.4    TO CISLO_ED3

```

### Výpis paměti:

```

CISLO_ED1      CHAR(6)          "0123.4"
                                "F0F1F2F34BF4"X
CISLO_ED2      CHAR(12)         "Kč    12.34-"
                                "D24740404040F1F24BF3F460"X
CISLO_ED3      CHAR(7)          " 123.4-"
                                "40F1F2F34BF460"X

```

## Redefinice paměti

Každá datová položka zabírá určitou paměť. Tato paměť může být strukturována jen jedním způsobem nebo několika. Ke strukturování stejné paměti různými způsoby se používá klausule REDEFINES, která dovoluje definovat položku jinak než je její původní členění.

*level name-1 REDEFINES name-2 ...*

např.

```

01 FORMATY.
05 CENPORW-RECORD PIC X(44).
05 CENPORW1-I      REDEFINES CENPORW-RECORD.
06 CENPORW1-I-INDIC.
07 IN03            PIC 1    INDIC 03.
07 IN12            PIC 1    INDIC 12.

```

```

07 IN06 PIC 1 INDIC 06.
06 MATER PIC X(5).
05 CENPORW1-O REDEFINES CENPORW-RECORD.
06 MATER PIC X(5).
05 CENPORW2-I REDEFINES CENPORW-RECORD.
06 CENPORW2-I-INDIC.
07 IN03 PIC 1 INDIC 03.
07 IN12 PIC 1 INDIC 12.
07 IN05 PIC 1 INDIC 05.
07 IN23 PIC 1 INDIC 23.
06 CENAJ PIC S9(7)V9(2).
06 NAZEV PIC X(30).
05 CENPORW2-O REDEFINES CENPORW-RECORD.
06 MATER PIC X(5).
06 CENAJ PIC S9(7)V9(2).
06 NAZEV PIC X(30).

```

V této struktuře je na úrovni 05 nejprve definována jednoduchá položka CENPORW-RECORD dlouhá 44 znaků. Ta je potom redefinována (překryta) čtyřmi dalšími definicemi, které mají různé členění a délku. Jejich délka však nepřekračuje 44 znaků. Například položka CENPORW1-O je dlouhá jen 5 znaků a zbytek redefinované paměti není definován.

## Definice analogií

Chceme-li definovat položku, která má stejné vlastnosti jako jiná položka, použijeme klausuli LIKE.

```
level name-1 LIKE name-2 [integer]
```

Položka *name-1* získává stejné vlastnosti jako *name-2* případně s modifikovanou délkou *integer* (kladné nebo záporné celé číslo).

```

01 KEY-STAVY.
05 ZAVOD LIKE ZAVOD of STAVYF0.
05 SKLAD LIKE SKLAD of STAVYF0.
05 MATER LIKE MATER of STAVYF0.

```

```
01 KEY-OBRTY LIKE KEY-STAVY.
```

Struktura KEY-OBRTY je definována stejně jako struktura KEY-STAVY, zatímco struktura KEY-STAVY je složena z jednoduchých položek definovaných také klausulí LIKE podle položek ZAVOD, SKLAD, MATER struktury STAVYF0.

## Definice vlastního typu

Pomocí klausule TYPEDEF si můžeme definovat vlastní typ dat, ale jen tak, že pojmenujeme existující položku.

```
level name-1 IS TYPEDEF
```

Položka *name-1* se stává jménem typu a může se použít v jiné definici jako zkratka.

```
level name-1 ... TYPE typedef-name ...
```

V následujícím úryvku programu jsou definice tří typů. INV-TYPE a ITEM-PRICE jsou jednoduché typy, BOOK-ITEM je složitější typ. První dva jsou použity v definici třetího typu a třetí typ je použit v definici nových položek BOOK-001 a BOOK-002.

```

01 INV-TYPE IS TYPEDEF PIC S9(3) VALUE 0.
01 ITEM-PRICE TYPEDEF PIC S9(4)V9(2) VALUE 0.
01 BOOK-ITEM IS TYPEDEF.
05 BOOK-TYPE TYPE INV-TYPE.
05 PRICE TYPE ITEM-PRICE.
05 BOOK-TITLE PIC X(40).

01 BOOK-001 TYPE BOOK-ITEM.
01 BOOK-002 TYPE BOOK-ITEM.

```

## Popis indikátorů

Indikátory jsou boolovské položky nabývající jen dvou hodnot (zapnuto, vypnuto). Jde o rozšíření jazyka pro IBM i potřebné pro práci s obrazovkovými a tiskovými soubory popsány v DDS (Data Description Specifications).

*level name picture-clause indicator-clause value-clause usage-clause.*

např.

```
77  BOOL-1  PICTURE IS 1  INDICATOR 61  VALUE IS B"0"  USAGE IS DISPLAY.
08  BOOL-2  OCCURS 10  PIC 1  VALUE IS B"1".
```

V klausuli PICTURE musí být uvedeno číslo 1. V klausuli INDIC(ATOR(S)) lze zapsat jen literál B"0" (vypnuto) nebo B"1" (zapnuto). Klausule USAGE je vždy DISPLAY, takže se nemusí zapisovat.

### Program INDIC1

Ilustruje definici indikátorů.

```
WORKING-STORAGE SECTION.
77  BOOL-1  PIC 1  INDICATOR 61  VALUE B"0"  USAGE IS DISPLAY.
01  BOOL-ARRAY.
    08  BOOL-2  OCCURS 10  PIC 1  VALUE B"1".

PROCEDURE DIVISION.
    MOVE B"1" to BOOL-1
    MOVE B"0" to  BOOL-2(2)
```

Výpis paměti:

```
BOOL-1      CHAR(1)          "1"
              "F1"X
BOOL-2      OF BOOL-ARRAY
DIM(1)      (1 10)
BOOL-2      OF BOOL-ARRAY
CHAR(1)
(1)          "1"
              "F1"X
(2)          "0"
              "F0"X
(3)          "1"
              "F1"X
...          ...
```

## Popis podmínkových jmen

*Podmínková jména* slouží jako zkratka k testování relací v podmínkových příkazech, např. v příkazu IF. Definují se jako položky v úrovni 88, která je podřízena položce vyšší úrovně (01 až 49 nebo 77), tzv. *podmínkové proměnné*.

### Program COND1

Ilustruje popis a použití podmínkových jmen s podmínkovou proměnnou.

```
WORKING-STORAGE SECTION.
77  CISLO PIC 99.
    88  JEDNA-AZ-99 VALUE 1 THRU 99.
    88  STO          VALUE 100.
    88  TISIC        VALUE 1000.

PROCEDURE DIVISION.
    Move 5      to CISLO
    If JEDNA-AZ-99 display "JEDNA-AZ-99 ANO"
    Move 100    to CISLO
    If STO      display "STO ANO"
    Move 101    to CISLO
    If not STO  display "STO NE"
    Move 1000   to CISLO
    If TISIC    display "TISIC ANO"
```

## Popis pro jméno konstanty

Definuje jméno, které může být použito v příkazech místo literálu nebo místo délky dat.

```
1. constant-name CONSTANT AS literal
1. constant-name CONSTANT AS LENGTH-OF data-name
```

kde úroveň 01 je povinná a *data-name* je jméno datové položky, jejíž délku jméno konstanty vyjadřuje.

## MANIPULACE S DATY

### Obrazné (figurativní) konstanty

Obrazné konstanty jsou anglická slova znázorňující některé numerické nebo nenumerické hodnoty.

ZERO/ZEROS/ZEROES	číselná nula nebo znaková nula (jedna nebo více)
SPACE/SPACES	jedna nebo více mezer
HIGH-VALUE/HIGH-VALUES	nejvyšší znaková hodnota (hexadecimální FF) – jeden nebo více bajtů
LOW-VALUE/LOW-VALUES	nejnižší znaková hodnota (hexadecimální 00) – jeden nebo více bajtů
QUOTE/QUOTES	uvozovky uvnitř znakové konstanty
ALL literal	všechny výskyty literálu (znakového)
NULL/NULLS	prázdná hodnota ukazatele (pointer)

### Odkazy na data

V příkazech se operandy vyjadřují buď literály (konstantními hodnotami) nebo jmény dat z definic.

#### Kvalifikace

Není-li prosté jméno dat jednoznačné v daném kontextu, je nutné je kvalifikovat jménem nadřazené úrovně:

```
jméno-1 OF/IN jméno-2 OF/IN ...
```

Např.

```
01 STRUCT1.
  02 SUBSTRUCT PIC ...
    03 FIELD PIC ...
01 STRUCT2.
  02 SUBSTRUCT PIC ...
    03 FIELD PIC ...
```

```
FIELD OF SUBSTRUCT IN STRUCT1
```

#### Indexování

Data definovaná s klauzulí OCCURS se používají s jedním nebo několika indexy v závorkách:

```
jméno (index index ...)
```

kde *jméno* může být kvalifikované a *index* je buď celé číslo (od 1) nebo celočíselná proměnná.

#### Modifikace pozice a délky dat

Ve znakových operandech lze vyjmout z obsahu proměnné část danou začáteční pozicí a délkou podřetězce (podobně jako u funkce substring v jiných jazycích).

```
jméno ( pozice : [ délka ] )
```

kde

*jméno* může být kvalifikované nebo odkaz na funkci,



*pozice* je celé číslo (od 1) nebo celočíselný aritmetický výraz,

*délka* je celé kladné číslo nebo celočíselný aritmetický výraz; jestliže chybí, vybere se zbytek řetězce.

### Program SUBSTR1

Ilustruje použití modifikace operandu pozicí a délkou.

```
WORKING-STORAGE SECTION.  
77 OPERAND-1 PIC X(10) VALUE "ABCDEFGHIJK".  
77 OPERAND-2 PIC X(5)  VALUE "12345".  
  
77 POS-1  PIC 99      VALUE 5.  
77 POS-2  LIKE POS-1  VALUE 1.  
  
77 LEN-1  PIC 99      VALUE 2.  
77 LEN-2  LIKE LEN-1  VALUE 4.  
  
PROCEDURE DIVISION.  
    Display "OPERAND-1: " OPERAND-1  
    Display "OPERAND-2: " OPERAND-2  
    Move OPERAND-1 (POS-1 : LEN-1) to OPERAND-2 (POS-2 : LEN-2)  
    Display "OPERAND-2: " OPERAND-2
```

### Odkaz na funkci

V jazyku COBOL jsou definovány funkce zvané intrinsic functions s předepsanými jmény a argumenty. Hodnota funkce se stává operandem v příkazu.

**FUNCTION** jméno-funkce [ ( argument argument . . . ) ] [ modifikace ]

kde

*argument* má předepsané vlastnosti podle typu funkce,

*modifikace* může u znakových funkcí vybrat podřetězec.

## Transformace dat příkazem MOVE

Příkaz MOVE kopíruje data z výchozí položky do jedné nebo několika cílových položek. Příkaz MOVE může pracovat ve třech režimech: elementární (elementary), skupinový (group) a odpovídající (corresponding).

*Elementární* režim kopíruje obsah jedné jednoduché položky do jedné nebo několika jednoduchých položek:

```
MOVE data1 TO data2 data3 ...
```

kde data1 je jednoduchá proměnná a data2, data3 atd. jsou také jednoduché proměnné.

*Skupinový* režim kopíruje skupinovou položku do skupinové nebo jednoduché položky jako by to byla jednoduchá znaková položka, tedy bez ohledu na to, jakého typu jsou její složky.

*Odpovídající* režim kopíruje z jedné struktury do druhé struktury jen ty složky, které mají stejné jméno v obou skupinových položkách. Ostatní zůstávají nezměněny.

### Program MOVE1

Ilustruje příkaz MOVE ve všech třech režimech. V elementárním režimu probíhá konverze dat, je-li možná. V případě, že cílem je numerická položka a měla by přijmout nenumerická data, dojde k chybě.

```
WORKING-STORAGE SECTION.  
01 STRUCT_01.  
    02 CHAR_01 PIC X(10) VALUE "ABCD".  
    02 CHAR_02 PIC X(10) VALUE "CDEF".  
    02 NUM_03  PIC S9(6)  DISPLAY VALUE 123456.
```

```

02 NUM_04 PIC S9(3)V9(3) DISPLAY VALUE 777.888.
01 TARGET_01.
02 NUM_04 PIC S9(3)V9(3).
02 CHAR_02 PIC X(10).
02 NUM_03 PIC S9(6).
02 CHAR_01 PIC X(10).
01 TARGET_02.
02 NUM_04 PIC S9(3)V9(3).
02 CHAR_02 PIC X(10).
02 NUM_03 PIC S9(6).
02 CHAR_01 PIC X(10).
01 TARGET_03.
02 CHAR_01 PIC X(10).
02 CHAR_02 PIC X(10).
02 NUM_03 PIC S9(6).
02 NUM_04 PIC S9(3)V9(3).

PROCEDURE DIVISION.

    DISPLAY "STRUCT_01          :" STRUCT_01.
*   Group move
    MOVE STRUCT_01 TO TARGET_01.
    DISPLAY "TARGET_01 GROUP MOVE  :" TARGET_01.
*   Corresponding move
    MOVE CORRESPONDING STRUCT_01 TO TARGET_02.
    DISPLAY "TARGET_02 CORRESPONDING:" TARGET_02.
*   Elementary moves
*       No conversion - char to char
    MOVE CHAR_01 IN STRUCT_01 TO CHAR_02 IN TARGET_03
    DISPLAY "TARGET_03 CHAR_01 to CHAR_02:" CHAR_02 IN TARGET_03
*       Num to char
    MOVE NUM_03 IN STRUCT_01 TO CHAR_01 IN TARGET_03
    DISPLAY "TARGET_03 NUM_03 TO CHAR_01 :" CHAR_01 IN TARGET_03
*       Num to num - OK
    MOVE NUM_03 IN STRUCT_01 TO NUM_04 IN TARGET_03
    DISPLAY "TARGET_03 NUM_03 TO NUM_04  :" NUM_04 IN TARGET_03
*       Char to num - error
    MOVE CHAR_01 IN STRUCT_01 TO NUM_04 IN TARGET_03
    DISPLAY "TARGET_03 CHAR_01 TO NUM_04  :" NUM_04 IN TARGET_03

```

### Display:

```

STRUCT_01          :ABCD      CDEF      123456777888
TARGET_01 GROUP MOVE  :ABCD      CDEF      123456777888
TARGET_02 CORRESPONDING:777888CDEF      123456ABCD
TARGET_03 CHAR_01 to CHAR_02:ABCD
TARGET_03 NUM_03 TO CHAR_01 :123456
TARGET_03 NUM_03 TO NUM_04  :456000
Message 'MCH1202' in program object 'MOVE1' in library 'VZCOBOL' (C D F G).

```

## Program MOVE2

Ilustruje příkaz MOVE použitý v cyklu s přesuny dat z jedné položky do několika položek struktury v cyklu přes pole struktur.

```

WORKING-STORAGE SECTION.
77  IDX PIC 999.
77  NEW-VALUE PIC 9999.

01  STRUCT_01.
05  ARRAY OCCURS 3.
08  CHAR_01 PIC X(10) VALUE "ABCD".
08  CHAR_02 PIC X(10) VALUE "CDEF".
08  NUM_03 PIC S9(6) DISPLAY VALUE 123456.
08  NUM_04 PIC S9(3)V9(3) DISPLAY VALUE 123.456.

PROCEDURE DIVISION.
*   Výpis tabulky
    PERFORM VARYING IDX FROM 1 BY 1 UNTIL IDX > 3
    DISPLAY "ARRAY "
    " : " CHAR_01 IN ARRAY (IDX)
    " : " CHAR_02 IN ARRAY (IDX)

```

```

        ":" NUM_03  IN ARRAY(IDX)
        ":" NUM_04  IN ARRAY(IDX) ":"
    END-PERFORM
* Tabulka se plní hodnotami indexu ve všech složkách struktury
    PERFORM VARYING IDX FROM 1 BY 1 UNTIL IDX > 3
        MOVE  IDX TO NEW-VALUE
* -----One to many
        MOVE  NEW-VALUE TO
            CHAR_01 IN ARRAY(IDX)
            CHAR_02 IN ARRAY(IDX)
            NUM_03  IN ARRAY(IDX)
            NUM_04  IN ARRAY(IDX)
* -----
    END-PERFORM
* Výpis výsledné tabulky s hodnotami indexu
    PERFORM VARYING IDX FROM 1 BY 1 UNTIL IDX > 3
        DISPLAY "ARRAY "
            ":" CHAR_01 IN ARRAY(IDX)
            ":" CHAR_02 IN ARRAY(IDX)
            ":" NUM_03  IN ARRAY(IDX)
            ":" NUM_04  IN ARRAY(IDX) ":"
    END-PERFORM

```

Display:

```

ARRAY :ABCD      :CDEF      :123456:123456:
ARRAY :ABCD      :CDEF      :123456:123456:
ARRAY :ABCD      :CDEF      :123456:123456:
ARRAY :0001      :0001      :000001:001000:
ARRAY :0002      :0002      :000002:002000:
ARRAY :0003      :0003      :000003:003000:

```

## Příkazy STRING a UNSTRING

K manipulaci se znakovými řetězci jsou určeny dva příkazy. Příkaz STRING sestavuje znakový řetězec z několika proměnných a UNSTRING naopak rozděluje úseky řetězce (např. slova) oddělené stanoveným znakem (např. mezerou) do jednoduchých znakových proměnných.

### Program STRING

Ilustruje sestavení textu řetězením z jednoduchých a skupinových proměnných. Výchozí data mohou být znaková a numerická celočíselná (bez klausule USAGE IS nebo jen s DISPLAY).

```

    WORKING-STORAGE SECTION.
* Zdroj
    77 Napis-1          pic X(20)      value "Položka 1".
    77 POLOZKA-1        pic S9(4)      value 1234.
    77 Napis-2          pic X(20)      value "Pol 2      ".
    77 POLOZKA-2        pic S9(2)      value      56.

    01 ZAZNAM.
        02 Item1        pic X(25)      value "  Závod  Sklad  Mater
-      "iál".
        02 Item2        pic 99         value 78.

* Cíle
    77 CIL1            pic X(30).
    77 CIL2            pic X(130).

    PROCEDURE DIVISION.
* jednoduché položky zbavené okolních mezer oddělené jednou mezerou
        STRING Napis-1 delimited by space
            space delimited by SIZE
            POLOZKA-1 delimited by space
            space delimited by SIZE
            Napis-2 delimited by space
            space delimited by SIZE
            POLOZKA-2 delimited by space
        INTO CIL1
    END-STRING.

```



```

POLOZKA-2                                "E945A596844040404040"X
CHAR(10)                                "      "
                                           "40404040404040404040"X
POLOZKA-3                                "Sklad      "
CHAR(10)                                "E2929381844040404040"X
...
TEXT1                                     "  Závod,  Sklad  Materiál      "
CHAR(45)                                "4040E945A596846B4040E292938184404040D481A3859989459340404040404040404040"X
                                           "40404040404040"X

```

**Poznámka:** Zatímco příkaz STRING je dosti přímočarý, příkaz UNSTRING je málo názorný a není tak pružný jako příkazy v jiných jazycích, např. funkce strtok() – string tokenizer v jazyku C nebo třída java.util.StringTokenizer (i když ty se provádějí vícenásobně). Obsahuje-li proměnná TEXT1 například za slovem Závod čárku následovanou mezerami, a příkaz obsahuje dodatečnou frázi OR ALL “,”

```
UNSTRING TEXT1  DELIMITED BY ALL SPACES OR ALL “,”
```

dojde v přenosu do proměnné POLOZKA-2 k přetečení (OVERFLOW), slovo Sklad se dostane do proměnné POLOZKA-3 a slovo Materiál se ztratí.

Display:

```

OVERFLOW
  Závod,  Sklad  Materiál
Závod    ,      ,Sklad    .

```

Výpis paměti:

```

POLOZKA-1                                "Závod      "
CHAR(10)                                "E945A596844040404040"X
POLOZKA-2                                "      "
CHAR(10)                                "40404040404040404040"X
POLOZKA-3                                "Sklad      "
CHAR(10)                                "E2929381844040404040"X

```

Kdyby však mezi slovem Závod a Sklad byla jen *čárka bez mezer*, dopadl by výsledek podle očekávání:

```

  Závod,Sklad  Materiál
Závod    ,Sklad    ,Materiál  .

```

## TABULKY

Tabulky (arrays, pole) jsou elementární nebo skupinové položky s několika výskyty, které lze indexovat, tedy takové, jejichž definice obsahují klausuli OCCURS. Klausule OCCURS však nesmí být použita na úrovních 1, 66, 77 a 88. Vícerozměrné tabulky se definují jako vnořené.

### Program TABULKY1

Ilustruje jednorozměrnou tabulku jednotlivých znaků vzniklou překrytím textu.

```

WORKING-STORAGE SECTION.
77 IDX  PIC 999.

01 STRUCT.
05 TEXT1          PIC X(5) VALUE "ABCDE".
05 ARRAY OCCURS 5 REDEFINES TEXT1.
08 CHAR          PIC X.

PROCEDURE DIVISION.
*   Cyklus přes všechny položky tabulky podle indexu
  PERFORM VARYING IDX FROM 1 BY 1 UNTIL IDX > 5
    DISPLAY "ARRAY "
      ":" CHAR IN ARRAY(IDX)
  END-PERFORM

```

Display:

```
ARRAY :A
ARRAY :B
ARRAY :C
ARRAY :D
ARRAY :E
```

## Program TABULKY2

Ilustruje dvourozměrnou tabulku čísel.

```
WORKING-STORAGE SECTION.
77 IDX1 PIC 999.
77 IDX2 PIC 999.

01 STRUCT.
10 TAB1 OCCURS 2.
20 TAB2 OCCURS 3.
30 NUM PIC 999 VALUE ZERO.

PROCEDURE DIVISION.
*   Cyklus přes všechny položky první tabulky
PERFORM VARYING IDX1 FROM 1 BY 1 UNTIL IDX1 > 2
*   Cyklus přes všechny položky druhé tabulky
PERFORM VARYING IDX2 FROM 1 BY 1 UNTIL IDX2 > 3
    COMPUTE
        NUM (IDX1 IDX2) = NUM (IDX1 IDX2) + IDX1 + IDX2
    DISPLAY "ARRAY NUM (" IDX1 ", " IDX2 "): "
        NUM (IDX1, IDX2)
END-PERFORM
END-PERFORM
```

Display:

```
ARRAY NUM (001, 001): 002
ARRAY NUM (001, 002): 003
ARRAY NUM (001, 003): 004
ARRAY NUM (002, 001): 003
ARRAY NUM (002, 002): 004
ARRAY NUM (002, 003): 005
```

## Program TABULKY3

Ilustruje tabulku proměnné délky. Tabulka má ve skutečnosti pevnou maximální délku, ale horní mez lze při výpočtu zmenšit. Ve zmenšené délce zpracujeme tabulku použitím funkcí MIN, MAX a SUM a hromadném indexu ALL označujícím všechny položky tabulky.

```
WORKING-STORAGE SECTION.
77 IDX PIC 999.
77 ELEM-COUNT PIC 999.
77 SUM-OF-ELEMENTS PIC 9999.
77 MAX-OF-ELEMENTS PIC 9999.
77 MIN-OF-ELEMENTS PIC 9999.

01 STRUCT.
05 ARRAY OCCURS 1 TO 5 TIMES DEPENDING ON ELEM-COUNT.
08 NUM PIC 999.

PROCEDURE DIVISION.
*   Naplním všechny položky tabulky hodnotou indexu.
MOVE 5 TO ELEM-COUNT
PERFORM VARYING IDX FROM 1 BY 1 UNTIL IDX > ELEM-COUNT
    MOVE IDX TO NUM (IDX)
END-PERFORM
*   Sečtu méně položek tabulky
MOVE 3 TO ELEM-COUNT
COMPUTE MIN-OF-ELEMENTS = FUNCTION MIN (NUM (ALL) )
COMPUTE MAX-OF-ELEMENTS = FUNCTION MAX (NUM (ALL) )
COMPUTE SUM-OF-ELEMENTS = FUNCTION SUM (NUM (ALL) )
DISPLAY "MIN-OF-ELEMENTS: "
```

```

MIN-OF-ELEMENTS
DISPLAY "MAX-OF-ELEMENTS: "
MAX-OF-ELEMENTS
DISPLAY "SUM-OF-ELEMENTS: "
SUM-OF-ELEMENTS

```

Display:

```

MIN-OF-ELEMENTS: 0001
MAX-OF-ELEMENTS: 0003
SUM-OF-ELEMENTS: 0006

```

## DATUM A ČAS

Datum a čas jsou speciální typy dat. Definují se zápisem (data description entry) s klausulí **FORMAT (IS)** místo **PICTURE (IS)**.

```

FORMAT IS DATE           datum
FORMAT IS TIME           čas
FORMAT IS TIMESTAMP      časové razítko

```

Datum, čas a časové razítko lze přesouvat příkazem **MOVE** nebo při čtení a zápisu dat v souborech a lze je použít v podmínkách.

### Program DATETIME1

Ilustruje použití data a času v programu.

```

CONFIGURATION SECTION.
*SPECIAL-NAMES.  FORMAT OF DATE IS "@Y-%m-%d",
*                FORMAT OF TIME IS "%H:%M:%S:@Sm".

WORKING-STORAGE SECTION.
01 TimestampT IS TYPEDEF
    FORMAT TIMESTAMP VALUE "1997-12-01-05.52.50.000000".
01 group-item.
05 date1 FORMAT DATE OCCURS 3 TIMES VALUE "1997-05-08".
* 05 date2 FORMAT DATE "@Y-%m-%d" VALUE "2001-09-08".
05 date2 FORMAT DATE VALUE "2001-09-08".
05 date3 REDEFINES date2 FORMAT DATE.
    88 date3-key-dates VALUE "1997-05-01" THRU "2002-05-01".
* 05 time1 FORMAT TIME "%H:%M" VALUE "14:10".
05 time1 FORMAT TIME VALUE "14.10".
05 time2 LIKE time1 VALUE "15:30".
05 timestamp1 TYPE TimestampT.

PROCEDURE DIVISION.
    display date1(1) space date1(2) space date1(3)
    move "2016-02-01" to date2
    display date2
    if date3-key-dates
        display "date2 is OK"
    else
        display "date2 is out of limits"
    end-if
    display time1
    display time2 " like time1"
    display timestamp1

```

V paragrafu **SPECIAL-NAMES** lze definovat tvar data a času (ale ne časového razítka) platný pro celý program, není-li u jednotlivých položek zadán jiný. Jinak platí formát ISO. Jako příklad jsme si definovali položku *TimestampT* jako datový typ **TIMESTAMP** s počáteční hodnotou. Tento typ pak použijeme v definici položky

```
05 timestamp1 TYPE TimestampT.
```

S datumem a časem lze použít následující funkce.

```
ADD-DURATION           přičtení časového intervalu
```

SUBTRACT-DURATION	odečtení časového intervalu
FIND-DURATION	zjištění časového intervalu
CURRENT-DATE	zjištění současného data a času
EXTRACT-DATE-TIME	výběr části údaje
TEST-DATE-TIME	testování, zda jde o časový typ
a další.	

## Program DATETIME2

Ilustruje použití některých funkcí pro datum a časové razítko.

```

WORKING-STORAGE SECTION.
77  date1 FORMAT DATE VALUE "2016-01-01".
77  date2 FORMAT DATE VALUE "2016-12-31".
77  timestamp1 FORMAT TIMESTAMP
      VALUE "2016-02-28-23.59.59.999999".
77  number-of-days PIC 999999.

PROCEDURE DIVISION.
    display date2
    move function ADD-DURATION(date2 YEARS 1 MONTHS 1 DAYS 1)
      to date2
    display "plus 1 year, 1 month, 1 day is " date2
    compute number-of-days =
      function FIND-DURATION(date2 date1 DAYS)
    move function
      SUBTRACT-DURATION(date2 YEARS 1 MONTHS 1 DAYS 1)
      to date2

    if function TEST-DATE-TIME(date2 DATE) = B"1"
      display "YES, item " date2 " is DATE"
    else display "NO, item " date2 " is not DATE"
    end-if

    if function TEST-DATE-TIME(timestamp1 TIMESTAMP) = B"1"
      display "YES, item " timestamp1 " is TIMESTAMP"
    else display "NO, item " timestamp1 " is not TIMESTAMP"
    end-if

```

## Program DATETIME3

Ilustruje získání současného data a času pro časové razítko.

```

WORKING-STORAGE SECTION.
77  timestamp1 FORMAT TIMESTAMP
      VALUE "2016-02-01-23.59.59.000000".
77  timestamp2 FORMAT TIMESTAMP
      VALUE "2016-02-28-23.59.59.000000".
77  number-of-days PIC 999999.

01  WS-CURRENT-DATE-FIELDS.
05  WS-CURRENT-DATE.
10  WS-CURRENT-YEAR      PIC X(4).
10  WS-CURRENT-MONTH    PIC X(2).
10  WS-CURRENT-DAY      PIC X(2).
05  WS-CURRENT-TIME.
10  WS-CURRENT-HOUR     PIC X(2).
10  WS-CURRENT-MINUTE   PIC X(2).
10  WS-CURRENT-SECOND   PIC X(2).
10  WS-CURRENT-MS       PIC X(2).
05  WS-DIFF-FROM-GMT     PIC X(3).
* exact length of the time stamp
01  TEXT1 PIC X(25).

PROCEDURE DIVISION.

    compute number-of-days =
      function FIND-DURATION(timestamp1 timestamp2 DAYS)
    display number-of-days

    move function CURRENT-DATE to WS-CURRENT-DATE-FIELDS
* totéž na dvakrát (v závorce pozice a počet znaků)
    move function CURRENT-DATE (1:8) to WS-CURRENT-DATE

```



```

move function CURRENT-DATE (9:12) to WS-CURRENT-TIME
display WS-CURRENT-YEAR "-" WS-CURRENT-MONTH "-"
        WS-CURRENT-DAY "-" WS-CURRENT-HOUR "."
        WS-CURRENT-MINUTE "." WS-CURRENT-SECOND "."
        WS-CURRENT-MS          WS-DIFF-FROM-GMT
string
        WS-CURRENT-YEAR "-"
        WS-CURRENT-MONTH "-"
        WS-CURRENT-DAY "-"
        WS-CURRENT-HOUR  "."
        WS-CURRENT-MINUTE "."
        WS-CURRENT-SECOND "."
        WS-CURRENT-MS
        WS-DIFF-FROM-GMT delimited by size
into TEXT1
on overflow display "??? overflow"
end-string

display TEXT1

```

## AKTUALIZACE STAVOVÉHO SOUBORU OBRATOVÝM SOUBOREM

Ukážeme aktualizaci stavu zásob podle obrátů (příjmů a výdejů) ve skladu. Program bude pracovat se dvěma databázovými soubory. Další soubor je jen pomocný – referenční, jmenuje se REF a neobsahuje skutečná data, ale jen definice datových polí používaných v programech aplikace. Na něj se odvolávají definice ostatních souborů.

- První soubor je stavový, jmenuje se STAVY a obsahuje  
identifikační údaje skladu – číslo závodu a číslo skladu,  
údaje o materiálu – číslo materiálu a množství materiálu na skladě.
- Druhý soubor je obrátový, jmenuje se OBRATY a obsahuje  
identifikační údaje skladu – číslo závodu a číslo skladu,  
údaje o obrátu – číslo materiálu a množství obrátu.

### Referenční soubor REF

```
*****
*   Referenční soubor                               *
*   (neměl by obsahovat žádný datový člen)         *
*****
A       R REFR
*       MMMMMMMMMM
A       MATER          5          COLHDG('Číslo' 'mater.')
A       MNOBR          8P 0       COLHDG('Množství' 'obratu')
A       MNOZ           8P 0       COLHDG('Množství' 've skladu')
*       NNNNNNNNNN
A       NAZEV          30          COLHDG('Název zboží')
*       SSSSSSSSSS
A       SKLAD          2          COLHDG('Sk1')
*       ZZZZZZZZZZ
A       ZAVOD          2          COLHDG('Zav')
```

### Popis souboru STAVY

```
*****
*   Soubor STAVY - Stav materiálových zásob         *
*****
*   Jednoznačný klíč (zákaz duplicit)
A       UNIQUE
*   Referenční soubor definic polí
A       REF(REF)
*   Jméno věty (formátu, záznamu)
A       R STAVYF0
*   Jména datových polí (s odkazem na referenční soubor)
A       ZAVOD          R
A       SKLAD          R
A       MATER          R
A       MNOZ           R
*   Definice klíče (má tři složky)
A       K ZAVOD
A       K SKLAD
A       K MATER
```

### Popis souboru OBRATY

```
*****
*   Soubor OBRATY - Obrat materiálu                 *
*****
*   Referenční soubor definic polí
A       REF(REF)
*   Jméno věty (formátu, záznamu)
A       R OBRATYF0
*   Jména datových polí (s odkazem na referenční soubor)
A       ZAVOD          R
A       SKLAD          R
A       MATER          R
A       MNOBR          R
```

```

*   Definice klíče
A       K ZAVOD
A       K SKLAD
A       K MATER

```

## Program STAOBR

Program čte pořadově záznamy souboru OBRATY od začátku do konce a u každého vyhledá párový záznam souboru STAVY. Přitom přičte množství obrátu k množství ve skladu a přepíše je. Stavový záznam se vyhledává podle klíče tvořeného čísly závodu, skladu a materiálu.

```

.....-A+++B+++++Pgm-id++
/*****
/
/ Program aktualizuje množství materiálu ve stavech podle obrátů
/
/*****

```

### IDENTIFICATION DIVISION.

```

PROGRAM-ID. STAOBR.
AUTHOR. Já.
INSTALLATION. Zde.

```

### ENVIRONMENT DIVISION.

```

CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-i.
OBJECT-COMPUTER. IBM-i.
INPUT-OUTPUT SECTION.
FILE-CONTROL.

```

```

*   Fyzický soubor STAVY
    SELECT STAVY ASSIGN TO DATABASE-STAVY,
           ACCESS IS DYNAMIC,
           ORGANIZATION IS INDEXED,
           RECORD KEY IS EXTERNALLY-DESCRIBED-KEY.

*   Fyzický soubor OBRATY
    SELECT OBRATY ASSIGN TO DATABASE-OBRATY,
           ACCESS IS SEQUENTIAL,
           ORGANIZATION IS SEQUENTIAL.

```

### DATA DIVISION.

#### FILE SECTION.

```

*   Fyzický soubor STAVY
FD STAVY.
01 STAVYR.
   COPY DDS-ALL-FORMATS OF STAVY.

*   Fyzický soubor OBRATY
FD OBRATY.
01 OBRATY-RECORD.
   COPY DDS-ALL-FORMATS OF OBRATY.

```

#### WORKING-STORAGE SECTION.

```

01 INPUT-END          PIC X.
   88 END-FILE         VALUE "E".

```

### PROCEDURE DIVISION.

```

/-----
Hlavni-sekce SECTION.
/-----

```

```

*   Otevřu soubory

    Open i-o STAVY.
    Open input OBRATY.

```

```

*   Přečtu první záznam obrátů
    Read OBRATY at end set END-FILE to TRUE End-read.

*   Cyklus zpracuje obraty s aktualizací stavů
    Perform until END-FILE

*       Naplním klíč pro čtení stavů z proměnných obrátů
        Move ZAVOD of OBRATYF0 to ZAVOD of STAVYF0
        Move SKLAD of OBRATYF0 to SKLAD of STAVYF0
        Move MATER of OBRATYF0 to MATER of STAVYF0

*       Čtu záznam stavů odpovídající danému obratu
        Read STAVY
            invalid key
                Continue
            not invalid key
*           Přičtu množství obratu k množství ze stavového záznamu
                Add MNOBR of OBRATYF0 to MNOZ of STAVYF0
*           a přepíšu záznam stavů
                Rewrite STAVYR
        End-read

*       Čtu další záznam obrátů
        Read OBRATY at end set END-FILE to TRUE End-read
    End-perform.

    Close OBRATY.
    Close STAVY.

```

## Vysvětlivky k programu

Popisy vstupů a výstupů jsou obsaženy ve dvou divizích. V divizi ENVIRONMENT-DIVISION v sekci INPUT-OUTPUT SECTION je paragraf FILE-CONTROL a v něm jsou popsány vlastnosti souborů jako celků pomocí zápisu SELECT. Popisují se vlastnosti jako zařízení, na němž se nachází soubor (ASSIGN), způsob přístupu k souboru (ACCESS), organizace dat (ORGANIZATION), případně další vlastnosti v závislosti na organizaci souboru, např. popis klíče (RECORD KEY) aj.

```

SELECT STAVY ASSIGN TO DATABASE-STAVY,
       ACCESS IS DYNAMIC,
       ORGANIZATION IS INDEXED,
       RECORD KEY IS EXTERNALLY-DESCRIBED-KEY.

SELECT OBRATY ASSIGN TO DATABASE-OBRATY,
       ACCESS IS SEQUENTIAL,
       ORGANIZATION IS SEQUENTIAL.

```

V divizi DATA DIVISION v sekci FILE-SECTION se popisuje struktura dat v jednotlivých souborech pomocí zápisů *file-description-entry* a *record-description-entry*.

```

FD STAVY.
01 STAVYR.
   COPY DDS-ALL-FORMATS OF STAVY.

FD OBRATY.
01 OBRATY-RECORD.
   COPY DDS-ALL-FORMATS OF OBRATY.

```

Zápis *file-description-entry* je uvozen indikátorem FD a obsahuje jméno souboru uvedené v zápisu SELECT, tj. v našem příkladu STAVY popř. OBRATY.

Zápis *record-description-entry* je uvozen číslem úrovně a jménem dat. Číslo úrovně může být 01 až 49 nebo 77. Jméno může být libovolné. V našem příkladu máme 01 STAVYR a 01 OBRATY-RECORD. U souborů externě popsaných (DDS) je nutné stanovit strukturu tohoto jména dat, a to pomocí příkazu COPY pro kompilátor. Zápis DDS-ALL-FORMATS kopíruje nebo spíše

transformuje formát databázového záznamu z objektu souboru do tvaru vhodného k popisu dat v jazyku COBOL. Kopíruje se jen jeden formát, protože jich u databázového souboru víc není.

Pro názornost se podíváme na výsledek kopírování ze souboru STAVY v protokolu o kompilaci.

```
FD STAVY.
  01 STAVYR.
    COPY DDS-ALL-FORMATS OF STAVY.
      05 STAVY-RECORD PIC X(14).
        * I-O FORMAT:STAVYF0 FROM FILE STAVY OF LIBRARY VZCOBOL
        *
        *THE KEY DEFINITIONS FOR RECORD FORMAT STAVYF0
        * NUMBER NAME RETRIEVAL ALTSEQ
        * 0001 ZAVOD ASCENDING YES
        * 0002 SKLAD ASCENDING YES
        * 0003 MATER ASCENDING YES
        05 STAVYF0 REDEFINES STAVY-RECORD.
          06 ZAVOD PIC X(2).
            * Zav
          06 SKLAD PIC X(2).
            * Skl
          06 MATER PIC X(5).
            * Cislo mater.
          06 MNOZ PIC S9(8) COMP-3.
            * Mnozství ve skladu
```

Kompilátor vytvořil interně datovou položku na další úrovni popisu dat (zvolil 05), nazval ji STAVY-RECORD a dosadil délku záznamu do klausule PIC X(14). Na stejné úrovni 05 vytvořil datovou položku STAVYF0 a překryl ji předchozí položku pomocí fráze REDEFINES, takže obě sdílejí stejnou paměť. Tuto položku dále rozdrobil na další úrovni číslo 06 na jednotlivá datová pole souboru STAVY s uvedením klausulí PIC (picture), kde písmeno X znamená znakový typ, písmeno S numerický typ. Číslice 9 označuje číslici (dekadickou) a číslo v závorce počet číslic. Zápis COMP-3 značí totéž co PACKED-DECIMAL. Konečně jako komentář jsou uvedena klíčová pole. Všimněme si, že na konci každé větve hierarchické struktury musí být zadán typ a velikost datové položky v klausuli PIC(TURE).

Pracovní data jsou popsána v sekci WORKING-STORAGE-SECTION.

```
WORKING-STORAGE SECTION.
  01 INPUT-END PIC X.
  88 END-FILE VALUE "E".
```

Zde je položka INPUT-END překryta položkou END-FILE, která má úroveň 88. Toto číslo určuje END-FILE jako podmínkové jméno odpovídající hodnotě "E". To umožňuje použít to jméno v příkazu SET a testovat jím, zda je podmínka splněna. Například příkaz

SET END-FILE TO TRUE zapíná podmínku jako splněnou

a příkaz

PERFORM UNTIL END-FILE testuje podmínku, zda je splněna.

Divize PROCEDURE DIVISION začíná otevřením souborů.

```
Open i-o STAVY.
Open input OBRATY.
```

Soubor STAVY je otevřen pro vstup i výstup, protože v zadání je řečeno, že z něj program bude číst záznam a přepisovat jej. Soubor OBRATY je otevřen jen pro výstup.

Program přečte první záznam souboru OBRATY příkazem

```
Read OBRATY at end set END-FILE to TRUE End-read.
```

Fráze AT END určuje, co se má stát, když příkaz READ v souboru OBRATY nepřečte žádný záznam, tedy je-li soubor prázdný. V tom případě se zapne podmínka END-FILE a příkaz READ končí. Jestliže přečte první záznam, uloží jeho data do paměti definované výše v položce OBRATY-RECORD.

Příkaz `set END-FILE to TRUE` provede totéž co příkaz `MOVE "E" TO INPUT-END`. První varianta se volí, když chceme mít text programu názornější.

Následuje in-line varianta příkazu PERFORM.

```
Perform until END-FILE
  Move ZAVOD of OBRATYF0 to ZAVOD of STAVYF0
  Move SKLAD of OBRATYF0 to SKLAD of STAVYF0
  Move MATER of OBRATYF0 to MATER of STAVYF0
  Read STAVY
    invalid key
      Continue
    not invalid key
      Add MNOBR of OBRATYF0 to MNOZ of STAVYF0
      Rewrite STAVYR
  End-read
  Read OBRATY at end set END-FILE to TRUE End-read
End-perform.
```

Příkaz ve formě věty (sentence) začíná PERFORM UNTIL END-FILE a končí zápisem END-PERFORM a tečkou. Podmínka END-FILE bude zpočátku splněná, jestliže soubor OBRATY je prázdný. V tom případě se příkaz neprovede a program pokračuje za koncovou tečkou. Jestliže první příkaz READ přečetl záznam, podmínka END-FILE není splněná a program provede příkazy, tzv. tělo cyklu.

Poznámka: Úvod příkazu by mohl být zapsán také v rozšířené podobě jako

```
PERFORM WITH TEST BEFORE UNTIL INPUT-END = "E"
```

V těle cyklu příkazy MOVE kopírují klíčová data z paměti souboru OBRATY do paměti souboru STAVY. Jména datových polí jsou kvalifikována jménem formátu s předložkou OF nebo IN. Tím jsou nastaveny hodnoty klíče pro vyhledávání záznamu v souboru STAVY.

Příkaz READ STAVY obsahuje fráze INVALID KEY a NOT INVALID KEY. Fráze INVALID KEY určuje, že když se hledaný záznam souboru STAVY nenajde, provede se příkaz CONTINUE, tedy prázdný příkaz. Fráze NOT INVALID KEY naopak určí postup při nalezení záznamu souboru STAVY. To se provedou dva příkazy:

ADD - přičtení množství z paměti obrátů k množství v paměti stavů,

REWRITE – přepis záznamu nalezeného v souboru STAVY novými hodnotami (množstvím).

Příkaz REWRITE odkazuje na jméno položky STAVYR definované na úrovni 01 v sekci FILE-SECTION, nikoliv na jméno formátu STAVYF0.

Na konci těla cyklu příkaz READ čte další záznam souboru OBRATY a opět testuje, zda nějaký přečetl. Jestliže ano, přenesou se data ze záznamu do paměti a příkaz končí. Jestliže ne, zapne se podmínka END-FILE a příkaz končí. Výpočet se pak vrací na začátek příkazu PERFORM a testuje podmínku END-FILE. Není-li podmínka splněná, provede tělo cyklu znovu, je-li splněná, končí příkaz PERFORM.

Po skončení cyklického výpočtu, jsou-li zpracovány všechny záznamy souboru OBRATY, provedou se příkazy CLOSE uzavírající oba soubory a výpočet končí.

## Program STAOBR\_2

Tento program je variantou programu STAOBR, ale přistupuje k aktualizaci stavů obráceně. Čte pořadově soubor STAVY a u každého záznamu se pokusí zjistit příkazem START, zda v souboru OBRATY existují záznamy se stejným klíčem. Jestliže ano, přečte skupinu obrátových záznamů se stejným klíčem a z každého přičte množství obrátu MNOBR k množství stavu MNOZ. Sečtené množství pak promítne do stavového záznamu příkazem REWRITE.

```
INPUT-OUTPUT SECTION.
FILE-CONTROL.

*   Fyzický soubor STAVY
*   -----
      SELECT STAVY ASSIGN TO DATABASE-STAVY,
             ACCESS MODE IS DYNAMIC,
             ORGANIZATION IS INDEXED,
             RECORD KEY IS EXTERNALLY-DESCRIBED-KEY.

*   Fyzický soubor OBRATY
*   -----
      SELECT OBRATY ASSIGN TO DATABASE-OBRATY,
             ACCESS IS DYNAMIC,
             ORGANIZATION IS INDEXED,
             RECORD KEY IS EXTERNALLY-DESCRIBED-KEY
                               WITH DUPLICATES.

DATA DIVISION.
FILE SECTION.

*   Fyzický soubor STAVY - Popis dat
*   -----
FD   STAVY.
01   STAVYR.
      COPY DDS-ALL-FORMATS OF STAVY.

*   Fyzický soubor OBRATY - Popis dat
*   -----
FD   OBRATY.
01   OBRATYR.
      COPY DDS-ALL-FORMATS OF OBRATY.

WORKING-STORAGE SECTION.

01   END-OF-STAVY                      PIC X VALUE SPACE.
      88 EOF-STAVY                      VALUE "E".
      88 NOT-EOF-STAVY                  VALUE " ".
01   END-OF-OBRATY                     PIC X VALUE SPACE.
      88 EOF-OBRATY                      VALUE "F".
      88 NOT-EOF-OBRATY                  VALUE " ".

01   KEY-STAVY.
      05 ZAVOD          like ZAVOD of STAVYF0.
      05 SKLAD          like SKLAD of STAVYF0.
      05 MATER          like MATER of STAVYF0.

01   KEY-OBRATY.
      05 ZAVOD          like ZAVOD of STAVYF0.
      05 SKLAD          like SKLAD of STAVYF0.
      05 MATER          like MATER of STAVYF0.

PROCEDURE DIVISION.

Uvodni-akce SECTION.
*   Otevřu soubory
*   -----
      Open I-O   STAVY.
      Open Input OBRATY.

      Set NOT-EOF-STAVY to TRUE.
```

```

*      Čtu PRVNÍ záznam souboru STAVY
      Read STAVY FIRST at end set EOF-STAVY to TRUE End-read.

*      Cyklus do konce souboru STAVY
*      -----
      Perform until EOF-STAVY,

*          Pamatuji přečtený klíč souboru STAVY
          Move corr STAVYF0 to KEY-STAVY,

*          Naplním klíč souboru OBRATY klíčem ze stavů
          Move corr KEY-STAVY to OBRATYF0

*          Nastavím ukazatel v souboru OBRATY na záznam
*          odpovídající klíči ze stavů
          Start OBRATY key is equal to KEY-STAVY
              invalid key Continue
              not invalid key
*              Zpracuji skupinu obrátů se stejným klíčem
              Perform Process-obraty
          End-start

*      Čtu DALŠÍ záznam se souboru STAVY
      Read STAVY NEXT at end set EOF-STAVY to TRUE End-read,
      End-perform.
      Goback.

* -----
* Zpracování skupiny obrátů se stejným klíčem
* -----
      Process-obraty SECTION.

      Read OBRATY next at end
          Set EOF-OBRATY to TRUE End-read
      Perform until EOF-OBRATY
          Move corr OBRATYF0 to KEY-OBRATY
          If KEY-OBRATY is equal to KEY-STAVY
              Add MNOBR of OBRATYR to MNOZ of STAVYF0
          Else
              Rewrite STAVYR
              Go to End-loop
          End-if
          Read OBRATY next at end
              Set EOF-OBRATY to TRUE End-read
      End-perform.
      End-loop.

```



# KONTROLA OBRATŮ A TISK

## Program STAOBRTISK

Program kontroluje, zda obraty párují se stavy. Nepárové obraty zjistí a vytiskne upozornění.

```
INPUT-OUTPUT SECTION.
FILE-CONTROL.

*   Fyzický soubor STAVY
    SELECT STAVY ASSIGN TO DATABASE-STAVY,
           ACCESS IS DYNAMIC,
           ORGANIZATION IS INDEXED,
           RECORD KEY IS EXTERNALLY-DESCRIBED-KEY.

*   Fyzický soubor OBRATY
    SELECT OBRATY ASSIGN TO DATABASE-OBRATY,
           ACCESS IS SEQUENTIAL,
           ORGANIZATION IS SEQUENTIAL.

*   Tiskový soubor TISK - systémový soubor QSYSPRT
    SELECT TISK ASSIGN TO PRINTER-QSYSPRT
           ACCESS SEQUENTIAL.

DATA DIVISION.

FILE SECTION.

*   Fyzický soubor STAVY
FD   STAVY.
01   STAVYR.
     COPY DDS-ALL-FORMATS OF STAVY.

*   Fyzický soubor OBRATY
FD   OBRATY.
01   OBRATY-RECORD.
     COPY DDS-ALL-FORMATS OF OBRATY.

*   Tiskový soubor TISK - délka řádku 120 znaků
FD   TISK.
01   PRINTER-RECORD  PIC X(120).

WORKING-STORAGE SECTION.
01   INPUT-END          PIC XXX VALUE SPACES.
     88  END-FILE       VALUE "END".

*   Hlavičkový řádek
01   HEADER-LINE        PICTURE  X(120).
     05  ZAVOD          PIC X(5) VALUE "Závod".
     05  FILLER         PIC X(3).
     05  SKLAD          PIC X(5) VALUE "Sklad".
     05  FILLER         PIC X(3).
     05  MATER          PIC X(8) VALUE "Materiál".
     05  FILLER         PIC X(4).
     05  TEXT-CHYBY     PIC A(30) VALUE "Text upozornění".

*   Detailní řádek
01   DETAIL-LINE        PICTURE  X(120).
     05  ZAVOD          like ZAVOD OF STAVYF0.
     05  FILLER         PIC X(6).
     05  SKLAD          like SKLAD OF STAVYF0.
     05  FILLER         PIC X(6).
     05  MATER          like MATER OF STAVYF0.
     05  FILLER         PIC X(7).
     05  TEXT-CHYBY     PIC A(30).

PROCEDURE DIVISION.

*   Otevřu soubory
    Open i-o STAVY.
```

```

Open input OBRATY.
Open output TISK.

*   Přečtu první záznam obrátů
    Read OBRATY at end Set END-FILE to TRUE End-read.

*   Zapišu hlavičku do tiskového souboru
        WRITE  PRINTER-RECORD FROM HEADER-LINE
                AFTER ADVANCING 3 LINES END-WRITE

*   Cyklus zpracuje obraty a stavy
    Perform until END-FILE

*       Naplním klíč pro čtení stavů z proměnných obrátů
        Move corr OBRATYF0 to STAVYF0
*       Čtu záznam stavů odpovídající danému obratu
        Read STAVY record key is externally-described-key,
            invalid key
            Move corr OBRATYF0 to DETAIL-LINE
            Move "Obrat nepáruje se stavem"
                to TEXT-CHYBY of DETAIL-LINE

*       Zapišu upozornění do detailního řádku
        Write  PRINTER-RECORD from DETAIL-LINE
                after advancing 1 line End-write
    End-read

*       Čtu další záznam obrátů
        Read OBRATY at end Set END-FILE to TRUE End-read
    End-perform.

```

## JEDNODUCHÉ OBRAZOVKOVÉ SOUBORY

Následující příklad slouží k ilustraci interakčního způsobu programování s pomocí obrazovkového souboru. Úkolem je umožnit uživateli vytvářet, opravovat, popř. rušit záznamy databázového souboru.

Program zobrazí obrazovkový formát k zadání klíče zpracovávaného záznamu souboru STAVY.

Uživatel zadá klíč (číslo závodu, číslo skladu a číslo materiálu) záznamu, který chce vytvořit nebo opravit, popř. zrušit.

The screenshot shows a terminal window titled 'tn5250j <2>' with a sub-header 'IASIST x'. The main area displays the prompt 'Zadejte udaje:' followed by three lines of input data: 'Zavod: 01', 'Sklad: 01', and 'Material: 00001'. Below the input area, there are two function key labels: 'F3=Konec' and 'F12=Navrat'. A horizontal line separates the main area from the bottom panel. The bottom panel contains a row of function keys (PF1 to PF24) and system keys (Enter, PgUp, Clear, SysReq, PgDn, Next Pad). The text 'MW' is displayed in blue in the center of the bottom panel, and '3/20' is displayed in red on the right side.

Program odpoví zobrazením formátu pro zadání všech dat. V zobrazeném formátu se zobrazí klíč a ostatní údaje (v našem případě množství materiálu). Program pozná sám, zda uživatel pořizuje nový záznam, nebo zpracovává starý, a to podle toho, zda v databázovém souboru nenalezl nebo našel záznam se zadaným klíčem. Podle toho zobrazí buď prázdné (nulové) množství materiálu nebo to, které je obsaženo v nalezeném záznamu.

3/18/16 17.29.08

Zavod: 01  
 Sklad: 01  
 Material: 00001  
 Mnozstvi: 1

F3=Konec F12=Navrat F5=Obnova F23=Zrusit

MW 6/20

PF1 PF2 PF3 PF4 PF5 PF6 PF7 PF8 PF9 PF10 PF11 PF12 Enter PgUp Clear  
 PF13 PF14 PF15 PF16 PF17 PF18 PF19 PF20 PF21 PF22 PF23 PF24 SysReq PgDn Next Pad

Uživatel zapíše nové množství a stiskne klávesu Enter. Program pak buď zapíše nový záznam do souboru STAVY nebo přepíše právě nalezený záznam novými údaji, a to opět podle toho, zda nenalezl nebo našel zadaný klíč.

Uživatel může v každém okamžiku ukončit výpočet tím, že stiskne klávesu F3 a v prvním formátu také F12. V druhém formátu může také stisknout klávesu F23, chce-li zobrazený záznam ze souboru vymazat (zrušit). Kromě toho může klávesou F5 obnovit zobrazení původních dat ještě předtím, než potvrdí změny klávesou Enter. Stisk klávesy F12 v druhém formátu způsobí opětovné zobrazení prvního formátu.

## Popis obrazovkového souboru STAVYW

AAN01N02N03T.Name+++++RLen++TDpBLinPosFunctions+++++

```

A          DSPSI2 (*DS3)
A          REF (REFMZP)
A          PRINT
A          CA03 (03 'Konec')
A          CA12 (12 'Navrat')
*   První formát - zadání klíče
A       R STAVYW1
A          1 35'Zadejte udaje:'
A          3 2'Zavod:'
A          ZAVOD      R      B 3 20
A          4 2'Sklad:'
A          SKLAD      R      B 4 20
A          5 2'Material:'
A          MATER      R      B 5 20
*
A          23 2'F3=Konec'
A          23 15'F12=Navrat'
*   Druhý formát - stavová data
A       R STAVYW2
A          CF05 (05 'Obnova')
A          CF23 (23 'Zrusit')
A          1 62DATE EDTCDE (Y)
A          1 72TIME EDTWRD (' . . ')
A          3 2'Zavod:'
A          ZAVOD      R      O 3 20
A          4 2'Sklad:'

```

A	SKLAD	R	O	4	20
A				5	2'Material:'
A	MATER	R	O	5	20
A				6	2'Mnozstvi:'
A	MNOZ	R	B	6	20
*					
A				23	2'F3=Konec'
A				23	15'F12=Navrat'
A				23	28'F5=Obnova'
A				23	41'F23=Zrusit'

V popisu DDS obrazkového souboru STAVYW jsou definovány dva formáty: STAVYW1 a STAVYW2. První slouží k zadání klíče a druhý k zobrazení obsahu záznamu.

Na začátku jsou uvedeny zápisy platné pro celý obrazkový soubor, tj. pro oba formáty společně.

Klíčové slovo DSPSIZ určuje obrazovku standardní velikosti, REF určuje referenční soubor REF, z něhož se čerpají definice polí, PRINT umožňuje použít klávesu Print k získání otisku obrazovky.

Klíčové slovo CA03 říká, že uživatel může stisknout klávesu F3 a program může testovat, zda byla stisknuta. Po stisku klávesy F3 se automaticky zapne *indikátor* 03 a výpočet se vrátí do programu. Program při zapnutém indikátoru 03 rozhodne, že výpočet skončí. Stejně vysvětlení platí pro klíčové slovo CA12 a indikátor 12. Čísla funkčních kláves se nemusí shodovat s čísly indikátorů, ale shodují-li se, je program lépe čitelný.

Poznámka: Indikátory obrazkového souboru jsou speciální proměnné označené dvoumístným číslem (01 až 99), které mohou nabývat hodnot „zapnuto“ nebo „vypnuto“. V jazyku CL se nazývají &IN01 až &IN99, mají typ \*LGL a nabývají hodnot '1' a '0'. V jazyku COBOL se jmenují IN01 až IN99, mají typ PIC 1 a nabývají hodnot B'1' a B'0'.

V prvním formátu jsou popsány konstanty a datová pole. Konstanty jsou uvedeny vpravo od pozic udávajících číslo řádku a sloupce na obrazovce. Např. konstanta 'Závod:' je umístěna v 3. řádku v 2. sloupci. Číslo sloupce se vztahuje k začátku konstanty, tj. k písmenu Z. Datové pole ZAVOD přebírá definici z referenčního souboru, což je určeno písmenem R, je obousměrné (tj. výstupní i vstupní), což je určeno písmenem B, a je umístěno v 3. řádku ve 20. sloupci. Všechna datová pole prvního formátu jsou obousměrná, tzn., že jejich obsah je zobrazován programem při výstupu na obrazovku a mohou být měněna uživatelem při vstupu z klávesnice.

Druhý formát obsahuje kromě konstant a datových polí ještě klíčová slova na úrovni popisu formátu (záznam, record): CF05 a CF23. Klíčová slova CFxx umožňují použití funkčních kláves podobně jako klíčová slova CAXx, s tím podstatným rozdílem, že po jejich stisknutí se do programu přenesou nejen hodnota zapnutého indikátoru, ale přenesou se i všechna vstupní data vložená z klávesnice do datových polí obrazovky. Rozdíl mezi klíčovými slovy CAXx a CFxx je třeba vždy bedlivě uvážit. Po stisku klávesy Enter se rovněž přenášejí vstupní data, ale žádný klávesový indikátor se nezapíná, takže po všech testech na klávesové indikátory v programu zbývá už jen možnost, že byla stisknuta klávesa Enter.

Klíčové slovo DATE představuje systémové datum (šestimístné číslo ve tvaru ddmmrr, rmmdd nebo mmddrr, podle zadání v popisu úlohy), klíčové slovo EDTCDE s parametrem Y znamená ediční kód, který vkládá do data oddělovací znaky mezi dnem, měsícem a rokem. Klíčové slovo TIME představuje systémový čas (šestimístné číslo ve tvaru hhmmss) a klíčové slovo EDTWRD určuje ediční slovo (masku), která vkládá mezi dvojice číslic tečky.

Datová pole představující klíč věty jsou v druhém formátu označena písmenem O jako výstupní, takže je uživatel nemůže měnit. Jediné pole, které lze měnit, je množství MNOZ označené jako obousměrné.

## Program STAPOR pro pořízení stavů

INPUT-OUTPUT SECTION.  
FILE-CONTROL.

- \*   Obrazovkový soubor STAVYW  
      SELECT STAVYW  
          ASSIGN TO WORKSTATION-STAVYW  
          ACCESS MODE IS SEQUENTIAL  
          ORGANIZATION IS TRANSACTION.
- \*   Fyzický soubor STAVY  
      SELECT STAVY ASSIGN TO DATABASE-STAVY,  
          ACCESS IS DYNAMIC,  
          ORGANIZATION IS INDEXED,  
          RECORD KEY IS EXTERNALLY-DESCRIBED-KEY.

DATA DIVISION.  
FILE SECTION.

- \*   Obrazovkový soubor STAVYW  
      FD   STAVYW.  
      01   FORMATY.  
          Copy DDS-ALL-FORMATS of STAVYW.
- \*   Fyzický soubor STAVY  
      FD   STAVY.  
      01   STAVY-RECORD.  
          Copy DDS-STAVYF0 of STAVY.

WORKING-STORAGE SECTION.

01	IND-NENALEZENI	PICTURE IS X(1).
88	NENALEZEN	VALUE IS "1".
88	NALEZEN	VALUE IS "0".
01	INDIKATORY-OBRAZOVKY.	
05	IN03	PICTURE 1 INDIC 03.
88	KONEC	VALUE B"1".
88	NENI-KONEC	VALUE B"0".
05	IN12	PICTURE 1 INDIC 12.
88	NAVRAT	VALUE B"1".
88	NENI-NAVRAT	VALUE B"0".
05	IN05	PICTURE 1 INDIC 05.
88	OBNOVA	VALUE B"1".
88	NENI-OBNOVA	VALUE B"0".
05	IN23	PICTURE 1 INDIC 23.
88	ZRUSIT	VALUE B"1".
88	NEZRUSIT	VALUE B"0".

PROCEDURE DIVISION.

Uvodni-akce.

Open I-O STAVYW.  
Open I-O STAVY.

Zobrazit-zadani.

Write FORMATY from STAVYW1-O, format is "STAVYW1".  
Read STAVYW record into STAVYW1-I.  
Move corr STAVYW1-I-INDIC  
      to INDIKATORY-OBRAZOVKY.  
If KONEC go to Konec-programu.  
If NAVRAT go to Konec-programu.

- \*   Klíč pro hledání v souboru STAVY  
      Move corr STAVYW1-I to STAVYF0.
- \*   Čtu záznam souboru STAVY  
      Read STAVY record into STAVYF0 no lock  
          invalid key Set NENALEZEN to TRUE  
          not invalid key Set NALEZEN to TRUE

```

End-read
* Připravím data do druhé obrazovky pro výstup
  If NALEZEN
*       Když se záznam našel, přesunu data
*       z databáze do obrazovky:
    Move corr STAVYF0 to STAVYW2-O
  Else
*       Když se záznam nenalezl, přesunu do databáze
*       klíčová data:
    Move ZAVOD of STAVYW1-I to ZAVOD of STAVYW2-O
    Move SKLAD of STAVYW1-I to SKLAD of STAVYW2-O
    Move MATER of STAVYW1-I to MATER of STAVYW2-O
*       a zbývající data (MNOZ) vynuluji:
    Move 0 to MNOZ of STAVYW2-O
  End-if.
*   Za End-if musí být tečka, protože následuje paragraf:

Znovu-druha.
  Write FORMATY from STAVYW2-O, format is "STAVYW2".
  Read STAVYW record into STAVYW2-I.
  Move corr STAVYW2-I-INDIC
    to INDIKATORY-OBRAZOVKY.
  If KONEC
    Go to Konec-programu.
  If NAVRAT
    Move corr STAVYF0 to STAVYW1-O
    Go to Zobrazit-zadani.
  If OBNOVA
    Move corr STAVYF0 to STAVYW2-O
    Go to Znovu-druha.

*   Vstupní data (MNOZ) přesunu do záznamu stavů
  Move MNOZ of STAVYW2-I to MNOZ of STAVYF0.

  If NENALEZEN
*       Když se záznam nenašel, zapišu nový:
    Write STAVY-RECORD from STAVYF0
*       a obnovím data prvního formátu:
    Move corr STAVYF0 to STAVYW1-O
*       a přejdu na zobrazení prvního formátu:
    Go to Zobrazit-zadani
  End-if.

  If ZRUSIT
    Delete STAVY
    Move corr STAVYF0 to STAVYW1-O
    Go to Zobrazit-zadani
  End-if.

*   Po stisku Enter přepíšu záznam v databázi
  Rewrite STAVY-RECORD from STAVYF0.
  Move corr STAVYF0 to STAVYW1-O.
  Go to Zobrazit-zadani.

Konec-programu.
  Close STAVY.
  Close STAVYW.

```

## Vysvětlivky k programu

Obrazovkový soubor STAVYW je umístěn na zařízení WORKSTATION, přístup k němu je SEQUENTIAL a jeho organizace je TRANSACTION.

```

SELECT STAVYW
  ASSIGN TO WORKSTATION-STAVYW
  ACCESS MODE IS SEQUENTIAL
  ORGANIZATION IS TRANSACTION.

```

Jeho popis dat je nazván FORMATY na úrovni 01 a realizován příkazem kompilátoru COPY, který generuje z objektu STAVYW datové struktury pro všechny formáty.

```
FD STAVYW.
```

```
1.  FORMATY.
    COPY DDS-ALL-FORMATS OF STAVYW.
```

Jména datových položek ve strukturách zjistíme, když zkompilujeme program a podíváme se na protokol o kompilaci. Uvidíme jména definovaná na několika úrovních. Na úrovni 05 kompilátor vygeneroval položku se jménem STAVYW-RECORD typu X s délkou 17 bajtů, což je délka nejdelšího formátu. Pod ním na stejné úrovni další položky, které ji překrývají (REDEFINES). Tyto položky odpovídají vstupním a výstupním oblastem prvního a druhého formátu. Např. položka STAVYW1-I je paměť prvního formátu pro vstupní data.

```
FD  STAVYW.
01  FORMATY.
    COPY DDS-ALL-FORMATS OF STAVYW.
      05  STAVYW-RECORD PIC X(17) .
      05  STAVYW1-I      REDEFINES STAVYW-RECORD.
      06  STAVYW1-I-INDIC.
            07  IN03          PIC 1  INDIC 03.
            07  IN12          PIC 1  INDIC 12.
      06  ZAVOD           PIC X(2) .
      06  SKLAD           PIC X(2) .
      06  MATER           PIC X(5) .
      05  STAVYW1-O      REDEFINES STAVYW-RECORD.
      06  ZAVOD           PIC X(2) .
      06  SKLAD           PIC X(2) .
      06  MATER           PIC X(5) .
      05  STAVYW2-I      REDEFINES STAVYW-RECORD.
      06  STAVYW2-I-INDIC.
            07  IN03          PIC 1  INDIC 03.
            07  IN12          PIC 1  INDIC 12.
            07  IN05          PIC 1  INDIC 05.
            07  IN23          PIC 1  INDIC 23.
      06  MNOZ            PIC S9(8) .
      05  STAVYW2-O      REDEFINES STAVYW-RECORD.
      06  ZAVOD           PIC X(2) .
      06  SKLAD           PIC X(2) .
      06  MATER           PIC X(5) .
      06  MNOZ            PIC S9(8) .
```

Na úrovni 06 jsou pak jména jednotlivých datových polí a u vstupních oblastí také paměti pro indikátory

```
06  STAVYW1-I-INDIC.
06  STAVYW2-I-INDIC.
```

Samotné indikátory jsou na úrovni 07 se jmény IN03, IN12, atd. Jejich typ je PICTURE IS 1 a jsou označeny klausulí INDICATOR s číslem indikátoru, např.

```
07  IN03          PIC 1  INDIC 03.
```

Je důležité si uvědomit, že všechny datové struktury se překrývají v jedné paměti dlouhé 17 bajtů, takže nelze spoléhat na trvání informací v určité struktuře po provedení dalších příkazů.

V sekci WORKING-STORAGE SECTION definujeme podmínkové proměnné, abychom si indikátory pojmenovali svými jmény. První z nich není vlastně indikátor obrazovky, ale jen podmínková proměnná odrážející stav po čtení záznamu ze souboru STAVY (nalezen, nenalezen).

```
01  IND-NENALEZENI.          PICTURE IS X(1) .
    88  NENALEZEN            VALUE IS "1" .
    88  NALEZEN              VALUE IS "0" .
```



Zde jsou uvedeny dvě položky v úrovni 88, protože bude třeba proměnnou zapínat i vypínat. Jazyk COBOL nezná hodnotu FALSE, je tedy nutné použít hodnotu TRUE pro příslušnou podmínku:

SET NENALEZEN TO TRUE nebo SET NALEZEN TO TRUE. Počáteční hodnota není stanovena.

Můžeme pochopitelně použít i příkaz MOVE:

MOVE "1" to IND-NENALEZENI nebo MOVE "0" to IND-NENALEZENI, což je ovšem méně názorné.

Poznámka: Neobsahuje-li definice koncové složky klausuli VALUE IS, není její hodnota určena. Při kompilaci však může být zadána volba \*STDINZ (standardní inicializace), která je předvolená, a pak je hodnota určena podle typu (např. mezera u typu PIC X, nula u typu PIC 9 atd.)

Obdobně jsou sestaveny obrazkové indikátory, které jsou sice definovány v paměti obrazkových formátů, ale pro účel názorného zacházení jsou definovány ještě jednou s výstižnými jmény (KONEC, NENI-KONEC, atd.). Všechny jsou soustředěny do společné proměnné INDIKATORY-OBRAZOVKY, aby se jejich hodnoty daly snadno hromadně inicializovat příkazem INITIALIZE a kopírovat příkazem MOVE CORRESPONDING.

Výpočet začíná otevřením obou souborů pro vstup i výstup a inicializací obrazkových indikátorů. Příkaz INITIALIZE dosadí do všech koncových složek datové struktury standardní hodnoty podle jejich typu, tj. B"0" v případě indikátorů.

Program pak zobrazí první formát obrazovky dvěma příkazy: WRITE a READ. Příkaz

```
Write FORMATY from STAVYW1-O, format is "STAVYW1".
```

zobrazí formát s konstantami a datovými poli (zprvu prázdnými).

**Příkaz**

```
Read STAVYW record into STAVYW1-I.
```

čeká na vstup z klávesnice. Poté co uživatel zapíše vstupní údaje a stiskne některou povolenou funkční klávesu nebo Enter, přečte tyto vstupní informace do paměti obrazovky a končí.

Poznámka: Výstupní paměť STAVYW-O není nutné předem inicializovat mezerami, je-li kompilace prováděna s předvolenou volbou \*STDINZ.

Po ukončení příkazu READ je nutné testovat všechny funkční klávesy, tj. jejich indikátory. Přípravou k tomu je příkaz

```
Move corr STAVYW1-I-INDIC  
to INDIKATORY-OBRAZOVKY.
```

který kopíruje odpovídající (corresponding) položky první struktury do druhé struktury. Teď můžeme testovat zapnutí klávesových indikátorů podle názorných jmen:

```
If KONEC go to Konec-programu.  
If NAVRAT go to Konec-programu.
```

Jestliže ani jeden z testů nevyjde, je jisté, že uživatel stiskl klávesu Enter. V tom případě se program pokusí přečíst záznam souboru STAVY podle hodnot zadaných z klávesnice. K tomu nejprve zkopíruje odpovídající data ze vstupní obrazkové paměti do paměti souboru STAVY příkazem

```
Move corr STAVYW1-I of FORMATY of STAVYW to STAVYF0.
```

Víme, že jde o klíčové složky ZÁVOD, SKLAD, MATER (viz výše). Čtení záznamu podle právě získaného klíče provede příkazem READ:

```
Read STAVY record into STAVYF0 no lock
      invalid key Set NENALEZEN to TRUE
      not invalid key Set NALEZEN to TRUE
End-read.
```

Fráze NO LOCK způsobí, že se přečtený záznam nezamkne.

Následující příkazy připraví výstupní paměť druhého obrazovkového formátu pro příkaz WRITE hodnotami přečtenými ze souboru STAVY nebo nulovým množstvím:

```
If NALEZEN Move corr STAVYF0 to STAVYW2-O
Else Move corr STAVYW1-I to STAVYW2-O
      Move 0 to MNOZ of STAVYW2-O
End-if.
```

Stejným způsobem jako první formát se příkazem WRITE zobrazí druhý formát a příkaz READ čeká na vstup z klávesnice, načež přečte data a končí.

```
Write FORMATY from STAVYW2-O, format is "STAVYW2".
Read STAVYW record into STAVYW2-I.
```

Stejně jako u prvního formátu testujeme funkční klávesy.

Zkopírujeme přečtené hodnoty do paměti souboru STAVY:

```
Move corr STAVYW2-I to STAVYF0.
```

Jestliže nebyl nalezen záznam v souboru STAVY, zapíšeme do něj nový záznam s novými hodnotami, obnovíme paměť pro výstup prvního formátu a opakujeme zobrazení prvního formátu.

```
If NENALEZEN WRITE STAVY-RECORD from STAVYF0
      Move corr STAVYF0 to STAVYW1-O
      Go to Zobrazit-zadani
End-if.
```

Jestliže se záznam v souboru STAVY našel, přepíšeme jej novými hodnotami příkazem REWRITE a vrátíme se na první formát.

```
Rewrite STAVY-RECORD from STAVYF0.
Move corr STAVYF0 to STAVYW1-O.
Go to Zobrazit-zadani.
```

Program uzavře soubory a skončí, když výpočet dojde na paragraf Konec-programu.

```
Konec-programu.
      Close STAVY.
      Close STAVYW.
```

## OBRAZOVKOVÉ SOUBORY S PODSOUBORY (SUBFILES)

Podsoubory představují speciální mechanismus pro obrazkové soubory a dovolují poměrně snadným způsobem zobrazovat data ve formě seznamů a listovat v nich. Jde o "nádrž" pro data, která se zpravidla získávají výběrem vět z databázového souboru (proto "podsoubor"). Protože podsoubor patří neoddělitelně k obrazkovému souboru, tj. jednomu uživateli, jsou jeho data nedostupná jiným uživatelům. Každý uživatel si tedy vytváří vlastní exemplář podsouboru.

Mechanismu podsouborů používá v hojné míře operační systém. Příkladem může být třeba seznam tiskových souborů ve výstupní frontě (získaný příkazem WRKSPLF). Obsahuje pevné záhlaví a seznam, ve kterém lze listovat.

K realizaci podsouboru je nutné popsat *dva obrazkové formáty: datový a řídicí*, zde STAVYSF a STAVYCTL. Datový formát popisuje strukturu věty v podsouboru a řídicí formát obsahuje údaje nezbytné k manipulaci s podsouborem. Řídicí formát také popisuje pevnou část obrazovky, zpravidla záhlaví umístěné nad stránkou dat podsouboru.

Program zapisuje data do podsouboru pomocí příkazu WRITE se jménem datového formátu. Věty podsouboru jsou číslovány pořadovými čísly, proto příkaz WRITE zapisuje nový záznam na místo určené pořadovým číslem (nikoliv automaticky za konec dosavadních dat). Předpokladem je, že na těchto místech nejsou umístěny žádné záznamy.

Jakmile jsou potřebné věty v podsouboru uloženy, lze je zobrazit ve formě obrazkových stránek, zpravidla příkazem WRITE se jménem řídicího formátu následovaným příkazem READ se jménem souboru. Příkaz READ čeká na vstup z klávesnice. Podsoubor obsahuje zpravidla více záznamů, než se jich vejde na jednu stránku obrazovky, proto je třeba umožnit listování v záznamech. Uživatel listuje v podsouboru pomocí stránkovacích kláves (Page down, Page up). Výměnu stránek obstarává počítač v rámci příkazu READ automaticky, takže se jím nemusíme v programu zabývat. Stránkování lze však řídit programem, zadáme-li speciální klíčová slova v popisu řídicího formátu.

Datový formát musí být označen klíčovým slovem SFL.

Řídicí formát je zařazen až po datovém formátu, musí obsahovat několik klíčových slov a může popisovat záhlaví obrazkové stránky. Nejdůležitější klíčová slova řídicího formátu jsou tato:

SFLCTL - odkazuje na datový formát STAVYSF,

SFLSIZ - určuje velikost podsouboru v počtu záznamů (max. 9999),

SFLPAG - určuje velikost obrazkové stránky v počtu řádků,

SFLDSP - umožňuje zobrazit stránku podsouboru, je-li zapnut podmínkový indikátor 50,

SFLDSPCTL - umožňuje zobrazit záhlaví stránky,

SFLEND - umožňuje zobrazit znaménko + na konci stránky, není-li to stránka poslední, je-li zapnut indikátor 50,

SFLCLR - umožňuje vymazat podsoubor, je-li zapnut podmínkový indikátor 60.

## Popis obrazkového souboru STAVYW1

```

A          DSPSIZ(24 80 *DS3)
A          REF(*LIBL/REF)
*   Datový formát
A          R STAVYSF          SFL
A          ZAVOD      R      O 9
A          SKLAD      R      O 9
A          MATER      R      O 9
A          MNOZ      R      O 9 EDTCDE(P)
A          CENAJ      R      O 9 28EDTCDE(P)
A          CELKEM      15Y 2O 9 40EDTCDE(P)
*   Řídicí formát
A          R STAVYCTL          SFLCTL(STAVYSF)
A          SFLSIZ(0100)
A          SFLPAG(0005)
A          CA03(03)
A          OVERLAY
A 50          SFLDSP
A 50          SFLEND
A          SFLDSPCTL
A 60          SFLCLR
A          3 19'Přehled'
A          3 27'stavů'
A          6 19'Množství'
A          6 35'Cena'
A          6 53'Cena'
A          7 2'Zav'
A          7 7'Skl'
A          7 12'Mater'
A          7 18'na skladě'
A          7 29'jednotková'
A          7 51'celkem'
*   Výstupní formát s nápovědou
A          R FUNCKEYS
A          23 3'F3=Konec'

```

Obrazkový soubor při zpracování programem je znázorněn na následujícím obrázku.

Přehled stavů					
Zav	Skl	Mater	Množství na skladě	Cena jednotková	Cena celkem
01	01	00001	1	10.00	10.00
01	01	00002	2	20.00	40.00
01	01	00003	3	30.00	90.00
01	01	00005	5	50.00	250.00
01	01	00006	6	.00	.00
					+

F3=Konec

MW 1/1

PF1	PF2	PF3	PF4	PF5	PF6	PF7	PF8	PF9	PF10	PF11	PF12	Enter	PgUp	Clear
PF13	PF14	PF15	PF16	PF17	PF18	PF19	PF20	PF21	PF22	PF23	PF24	SysReq	PgDn	Next Pad

Podmínkový indikátor 50 u slova SFLDSP dovoluje zobrazit stránku subfajlu a u slova SFLEND případný křížek na konci stránky, následuje-li ještě další stránka. Podmínkový indikátor 60 u slova SFLCLR dovoluje smazat veškerý obsah subfajlu.

Slovo OVERLAY zabraňuje smazání návodu k použití klávesy F3=Konec předem zobrazeného formátem FUNCKEYS, poté co se zobrazí záhlaví a stránka dat.

## Program STADSP – zobrazení stavů podsouborem

```
*****
*
*   Zobrazení stavů s cenami
*
*****

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.

*   Obrazovkový soubor STAVYW1
*   -----
      SELECT STAVYW1 ,
            ASSIGN TO WORKSTATION-STAVYW1,
            ACCESS MODE IS DYNAMIC,
            RELATIVE KEY IS RRN,
            ORGANIZATION IS TRANSACTION.

*   Fyzický soubor CENY
*   -----
      SELECT CENY  ASSIGN TO DATABASE-CENY,
            ACCESS IS RANDOM,
            ORGANIZATION IS INDEXED,
            RECORD KEY IS EXTERNALLY-DESCRIBED-KEY.

*   Fyzický soubor STAVY
*   -----
      SELECT STAVY ASSIGN TO DATABASE-STAVY,
            ACCESS IS DYNAMIC,
            ORGANIZATION IS INDEXED,
            RECORD KEY IS EXTERNALLY-DESCRIBED-KEY.

DATA DIVISION.

FILE SECTION.

*   Obrazovkový soubor STAVYW1 - Popis dat
*   -----
FD   STAVYW1.
01   FORMATY.
      COPY DDS-ALL-FORMATS OF STAVYW1.

*   Fyzický soubor STAVY - Popis dat
*   -----
FD   STAVY.
01   STAVYR.
      COPY DDS-ALL-FORMATS OF STAVY.

*   Fyzický soubor CENY - Popis dat
*   -----
FD   CENY.
01   CENYR.
      COPY DDS-ALL-FORMATS OF CENY.

WORKING-STORAGE SECTION.

01   END-OF-FILE                                PIC X value spaces.
      88 EOF                                     value is "E".
      88 NOT-EOF                                value is " ".

01   INDIKATORY-OBRAZOVKY.
      05   IN03                                PICTURE 1 INDIC 03.
            88   KONEC                         VALUE IS B"1".
            88   NENI-KONEC                     VALUE IS B"0".
      05   IN50                                PICTURE 1 INDIC 50.
            88   ZOBRAZIM                       VALUE IS B"1".
            88   NEZOBRAZIM                     VALUE IS B"0".
      05   IN60                                PICTURE 1 INDIC 60.
            88   CLEAR                         VALUE IS B"1".
```

\* Pořadové číslo pro záznamy subfajlu  
 01 RRN PIC 9(05).

# PROCEDURE DIVISION.

## Uvodni-akce SECTION.

\* Otevřu soubory  
 \* -----  
 Open I-O STAVYW1.  
 Open Input STAVY.  
 Open Input CENY.

## Zobrazit-znovu.

Initialize INDIKATORY-OBRAZOVKY.

\* Provedu sekci, kde se naplní subfajl  
 \* -----  
 Perform Plneni-subfajlu.

\* Zobrazit záhlaví a subfajl a čekat na vstup z klávesnice  
 \* -----  
 Write FORMATY, format is "FUNCKEYS".  
**Write** FORMATY, format is "**STAVYCTL**".  
**Read** STAVYW1 record format is "**STAVYCTL**".

\* Přečíst vstupní indikátory do indikátorové struktury  
 \* ze vstupní datové struktury řídicího formátu  
 \* -----

Move corresponding STAVYCTL-I-INDIC  
 to INDIKATORY-OBRAZOVKY.

\* Po stisku klávesy F3 končím  
 \* -----  
 If KONEC stop run.

\* Vrátím se na paragraf Zobrazit-znovu  
 \* -----  
 Go to Zobrazit-znovu.

\* -----  
 \* Sekce, kde se plní subfajl  
 \* -----

## Plneni-subfajlu SECTION.

\* Vymažu data ze subfajlu  
 Set CLEAR to TRUE.  
 Move corr INDIKATORY-OBRAZOVKY to STAVYCTL-O-INDIC  
**Write** FORMATY format is "**STAVYCTL**".  
 Set NOT-CLEAR to TRUE.  
 Move corr INDIKATORY-OBRAZOVKY to STAVYCTL-O-INDIC

Move 0 to RRN.  
 Set NOT-EOF to TRUE.

\* Čtu PRVNÍ záznam souboru STAVY  
 Read STAVY FIRST record at end set EOF to TRUE End-read.

\* Cyklus do konce souboru STAVY  
 \* -----

Perform until EOF,  
 \* Nastavím vyhledávací klíč pro CENY  
 Move MATER of STAVYF0 to MATER of CENY,

\* Čtu CENY podle klíče MATER  
 Read CENY record  
 invalid key  
 Move 0 to CENAJ of STAVYSF,  
 not invalid key  
 Move CENAJ of CENY to **CENAJ of STAVYSF**,  
 End-read,

```

*      Vynásobím množství jednotkovou cenou a výsledek
*      dám do proměnné CELKEM v subfajlu
      Compute CELKEM of STAVYSF =
          MNOZ of STAVYF0 * CENAJ of STAVYSF

*      Zvýším pořadové číslo pro zápis do subfajlu
      Add 1 to RRN,

*      Data ze souboru STAVY přesunu do subfajlu
      Move corr STAVYF0 to STAVYSF

*      Zapišu záznam do subfajlu podle pořadového čísla RRN
      Write subfile FORMATY format is "STAVYSF",

*      Čtu DALŠÍ záznam se souboru STAVY
      Read STAVY NEXT record at end set EOF to TRUE End-read,
      End-perform.

*      Zapnu indikátor povolující zobrazení subfajlu
      If RRN > 0
          Set ZOBRAZIM to TRUE,
          Move corr INDIKATORY-OBRAZOVKY
              to STAVYCTL-O-INDIC,
      End-if.

```

## Vysvětlivky k programu

Obrazkový soubor je nyní definován jako dynamický s relativním klíčem.

```

SELECT STAVYW1 ,
      ASSIGN TO WORKSTATION-STAVYW1,
      ACCESS MODE IS DYNAMIC,
      RELATIVE KEY IS RRN,
      ORGANIZATION IS TRANSACTION.

```

Znamená to, že se pracuje s podsouborem (subfile), který je ovládán prostřednictvím pořadového čísla záznamu (relative key), s hodnotami od 1 výše.

Nyní jsou kromě vstupního indikátoru IN03 v položce STAVYCTL-I-INDIC definovány ještě výstupní indikátory IN50 a IN60 definované v položce STAVYCTL-O-INDIC.

Ve formátech STAVYSF a FUNCKEYS žádné indikátory nejsou.

Výpočet začíná vyvoláním sekce Plneni-subfajlu, kde se naplní subfajl záznamy ze souboru STAVY a dalšími údaji (ze souboru CENY).

Perform Plneni-subfajlu.

Nato se zobrazí příkazem WRITE nejprve formát FUNCKEYS s návodem F3=Konec a pak řídicí formát STAVYCTL zobrazující záhlaví a stránku dat.

```

Write FORMATY, format is "FUNCKEYS".
Write FORMATY, format is "STAVYCTL".
Read STAVYW1 record format is "STAVYCTL".

```

Následující příkaz READ přečte z obrazkového souboru vstupní data do vstupní paměti řídicího formátu STAVYCTL-I a končí. Stiskl-li uživatel klávesu F3, výpočet končí příkazem STOP RUN. Stiskl-li klávesu Enter, výpočet přejde zpět na paragraf Zobrazit-znovu, kde se znovu zobrazí všechny formáty.

V sekci Plneni-subfajlu SECTION se nejprve smaže případný obsah subfajlu sekvencí příkazů



```
Set CLEAR to TRUE.  
Move corr INDIKATORY-OBRAZOVKY to STAVYCTL-O-INDIC.  
Write FORMATY format is "STAVYCTL".  
Set NOT-CLEAR to TRUE.  
Move corr INDIKATORY-OBRAZOVKY to STAVYCTL-O-INDIC.
```

Příkaz WRITE s řídicím formátem provede výmaz všech záznamů subfajlu, když je zapnutý indikátor IN60.

Pak se v cyklu čte soubor STAVY a ke každému jeho záznamu se přečte odpovídající záznam souboru CENY (podle klíče MATER), množství MNOZ se vynásobí jednotkovou cenou CENAJ a výsledek se umístí do proměnné CELKEM. Všechny získané hodnoty se nakopírují do paměti datového formátu subfajlu STAVYSF, pořadové číslo RRN se zvýší o 1 a záznam se zapíše do subfajlu příkazem

```
Write subfile FORMATY format is "STAVYSF"
```

a cyklus se opakuje do konce souboru STAVY.

Po ukončení cyklu se zapne indikátor IN50, jestliže číslo RRN je větší než nula, a tím umožní zobrazit stránku dat.

Pak se výpočet vrací za vyvolání sekce a pokračuje v hlavním programovém cyklu.

## OPRAVY DATABÁZOVÉHO SOUBORU S POUŽITÍM PODSOUBORU

S pomocí podsouborů lze data nejen zobrazovat, ale i opravovat a doplňovat. V tomto příkladu se budeme věnovat jen opravám existujících vět. Datový formát podsouboru (SFL) obsahuje obousměrná pole odpovídající těm polím v databázovém záznamu, která dovolíme opravovat. V našem příkladu to bude pouze pole MNOZ. K tomu budeme potřebovat nejen zapsat data souboru STAVY do záznamů podsouboru a zobrazit je, ale pak i přecházet ty záznamy podsouboru, které uživatel změnil (tím, že zapsal do jejich pole MNOZ nějakou hodnotu). Nezměněné záznamy podsouboru nebudeme zpracovávat.

K usnadnění tohoto úkolu slouží příkaz

**READ SUBFILE ... MODIFIED** - čtení dalšího změněného záznamu podsouboru.

Když je totiž podsoubor zobrazen a uživatel vkládá data z klávesnice do vstupních polí, zapíše počítač do každého změněného záznamu podsouboru speciální *příznak změny* (**MDT** – modified data tag). Podle toho příznaku pak příkaz **READ SUBFILE ... MODIFIED** rozeznává změněné záznamy od nezměněných.

Náš program bude číst změněný záznam podsouboru, zkontroluje správnost vložených dat (v našem příkladu jenom pole MNOZ) a buď ohlásí chybu, nebo opraví odpovídající větu databázového souboru. Jestliže jsou vložená data správná, program musí nalézt odpovídající databázovou větu, a opravit ji. Proto datový záznam podsouboru musí obsahovat jednoznačný vyhledávací argument (klíč) pro hledání v souboru STAVY. Ten byl získán při plnění podsouboru - je to klíč přečteného souboru složený z polí ZAVOD, SKLAD, MATER.

Příkaz **READ SUBFILE ... MODIFIED** přečte všechna datová pole podsouborového záznamu včetně indikátorů, dokonce i pole označená písmenem O jako výstupní.

Za chybu v datech budeme považovat hodnotu větší než 1000 v poli MNOZ. Jestliže takovou hodnotu vloží uživatel z klávesnice, zapneme indikátor 80 a indikátor 81 (pro stálou chybu). V popisu řídicího formátu bude zapsáno klíčové slovo

**SFLMSG** - chybová zpráva pro záznam podsouboru - které bude podmíněno indikátorem 81. Tato zpráva bude stále zobrazená a indikátor 81 stále zapnutý, bude-li po opravě a stisku klávesy Enter chybná hodnota ještě na jiném záznamu než na právě zpracovávaném.

Indikátorem 80 podmíníme ještě další klíčové slovo,

**SFLNXTCHG** – „příště změna“

zapsané tentokrát v *datovém formátu*. Umožňuje zapsat do záznamu podsouboru příznak změny (MDT), i když uživatel do něj nevložil žádná data z klávesnice. Klíčové slovo SFLNXTCHG se používá v kombinaci s příkazem

**REWRITE SUBFILE**, který přepíše záznam podsouboru. Operandem příkazu bude jméno datového formátu podsouboru DATA. Do záznamu podsouboru se tak dostanou kromě změněných dat také hodnoty indikátoru 80 a příznaku změny (ať už zapnuté nebo vypnuté). Při příštím zobrazení se takový záznam objeví s inverzně zvýrazněným množstvím a příkaz **READ SUBFILE ... MODIFIED** jej pak přečte, i když uživatel do něj nic nezapsal.

U pole MNOZ bude indikátorem 80 podmíněno nastavení kurzoru a inverzní zobrazení, aby se zvýraznilo místo, kde je chyba. Bude-li chybných záznamů více, nastaví se kurzor na poslední z nich.

K volbě stránky podsouboru, kterou chceme zobrazit, použijeme klíčového slova

## SFLRCDNBR(CURS0R)

To představuje číslo záznamu podsouboru, který se má přístě zobrazit na stránce obrazovky. Toto klíčové slovo zapíšeme u skrytého (H) pole KURZOR, které si definujeme v řídicím formátu pro volbu čísla záznamu, na který chceme nastavit kurzor při přístím zobrazení.

## Obrazovkový soubor STAVYW2

Obsahuje tři formáty: DATA, RIDICI, NAVOD. V datovém formátu je nyní pole MNOZ označeno kódem B, což znamená obousměrný tok dat – vstup i výstup.

```
*****
*   Soubor STAVYW2 - Obrazovkový soubor pro opravu stavu
*****
A                                     DSPSIZ(*DS3)
A                                     REF(REF)
A                                     PRINT
A                                     CA03(03 'Konec')
*   Datový formát podsouboru
A       R DATA                       SFL
A 80                                SFLNXTCHG
A       ZAVOD      R      O 7 3
A       SKLAD      R      O 7 10
A       MATER      R      O 7 17
A       MNOZ       R      B 7 30EDTCDE(K)
A 80                                DSPATR(RI)
*   Ridici formát podsouboru - zhlavi
A       R RIDICI                      SFLCTL(DATA)
A                                     SFLSIZ(100)
A                                     SFLPAG(5)
A 51                                SFLDSP
A                                     SFLDSPCTL
A 51                                SFLEND
A 81                                SFLMSG('Chyba v datech')
A                                     OVERLAY
A       KURZOR      4 0H              SFLRCDNBR(CURS0R)
A                                     2 18'Prehled stavu'
A                                     4 3'Zavod'
A                                     4 10'Sklad'
A                                     4 17'Material'
A                                     4 33'Mnozstvi'
*   Navod k pouziti funkcnich klaves
A       R NAVOD
A                                     23 3'F3=Konec'
```

## Program STAUPD pro opravy souboru STAVY

```
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.

*   Obrazovkový soubor STAVYW2
*   -----
      SELECT STAVYW2,
            ASSIGN TO WORKSTATION-STAVYW2,
            ACCESS MODE IS DYNAMIC,
            RELATIVE KEY IS RRN,
            ORGANIZATION IS TRANSACTION.

*   Fyzický soubor STAVY
*   -----
      SELECT STAVY ASSIGN TO DATABASE-STAVY,
            ACCESS IS DYNAMIC,
            ORGANIZATION IS INDEXED,
            RECORD KEY IS EXTERNALLY-DESCRIBED-KEY.

DATA DIVISION.
```

FILE SECTION.

```
*   Obrazovkový soubor STAVYW2 - Popis dat
*   -----
FD   STAVYW2.
01   FORMATY.
      COPY DDS-ALL-FORMATS OF STAVYW2.
```

```
*   Fyzický soubor STAVY - Popis dat
*   -----
FD   STAVY.
01   STAVYR.
      COPY DDS-ALL-FORMATS OF STAVY.
```

WORKING-STORAGE SECTION.

```
01   END-OF-FILE                PICTURE IS X VALUE IS SPACE.
      88 EOF                    VALUE IS "E".
      88 NOT-EOF                VALUE IS " ".

01   INDIKATORY-OBRAZOVKY.
      05   IN03                 PICTURE IS 1 INDICATOR 03.
            88   KONEC          VALUE IS B"1".
            88   NENI-KONEC     VALUE IS B"0".
      05   IN51                 PICTURE IS 1 INDICATOR 51.
            88   ZOBRAZIM       VALUE IS B"1".
            88   NEZOBRAZIM     VALUE IS B"0".
      05   IN80                 PICTURE IS 1 INDICATOR 80.
            88   CHYBA-MNOZSTVI VALUE IS B"1".
            88   NENI-CHYBA-MNOZSTVI VALUE IS B"0".
      05   IN81                 PICTURE 1 INDIC 81.
            88   STALA-CHYBA    VALUE IS B"1".
            88   NENI-STALA-CHYBA VALUE IS B"0".
```

```
*   Pořadové číslo pro záznamy subfajlu - čítač
77   RRN                        PIC 9(05)    VALUE ZERO.
```

PROCEDURE DIVISION.

Uvodni-akce SECTION.

```
*   Otevřu soubory
*   -----
      Open I-O STAVYW2.
      Open I-O STAVY.

*   Provedu sekci, kde se naplní subfajl
*   -----
      Perform Plneni-subfajlu.

*   Nastavím kurzor pro subfajl. Nesmí být 0.
      Move 1 to KURZOR of RIDICI-O.

Zobrazit-znovu.
*   Zobrazím záhlaví a subfajl a čekám na vstup z klávesnice
*   -----
      Write FORMATY, format is "NAVOD".
      Write FORMATY, format is "RIDICI".
      Read STAVYW2, format is "RIDICI".

*   Zruším stálou chybu v subfajlu
      Set NENI-STALA-CHYBA to TRUE.

*   Přečtu vstupní indikátory do indikátorové struktury
*   ze vstupní datové struktury řídicího formátu
*   -----
      Move corresponding RIDICI-I-INDIC
        to INDIKATORY-OBRAZOVKY.

*   Po stisku klávesy F3 končím
*   -----
      If KONEC
```

```

        Close STAVY
        Close STAVYW2
        Goback
    End-if.

*   Po stisku klávesy ENTER zpracuji změněné záznamy subfajlu
*   -----
    Set NOT-EOF to TRUE.

*   Čtu první změněný záznam subfajlu
    Read subfile STAVYW2 modified format "DATA"
        at end set EOF to TRUE.

*   Cyklus do konce změněných záznamů subfajlu
    Perform until EOF

*       Zjistím, zda množství z obrazovky je chybné - větší než 1000
    Perform Kontrola-mnozstvi

*       Je-li množství bez chyby, opravím STAVY
    If NENI-CHYBA-MNOZSTVI
*       Nastavím klíč a čtu stavový záznam
        Move corresponding DATA-I to STAVYF0,
        Read STAVY invalid key continue End-read,
*       Přepíšu množství ve stavech množstvím z obrazovky
        Move MNOZ of DATA-I to MNOZ of STAVYF0,
*       Přepíšu záznam v souboru STAVY
        Rewrite STAVYR,
    End-if

*       Dosadím chybový indikátor do paměti subfajlu
    Move corr INDIKATORY-OBRAZOVKY to DATA-O-INDIC,

*       Přepíšu záznam v subfajlu chyba nechýba
    Rewrite subfile FORMATY format is "DATA" end-rewrite,

*       Čtu další změněný záznam subfajlu
    Read subfile STAVYW2 modified format "DATA"
        at end set EOF to TRUE End-read,
    End-perform.

*   Pořadové číslo posledního změněného záznamu v subfajlu
*   si pamatuji v proměnné KURZOR
    If RRN > 0
        Move RRN to KURZOR in RIDICI-O
        Set ZOBRAZIM to TRUE
        Move corr INDIKATORY-OBRAZOVKY to RIDICI-O-INDIC
    End-if.

*   Vrátím se na paragraf Zobrazit-znovu
*   -----
    Go to Zobrazit-znovu.

*   -----
*   Sekce, kde se plní subfajl
*   -----
    Plneni-subfajlu SECTION.

        Move 0 to RRN.
        Set NOT-EOF to TRUE.

*   Čtu PRVNÍ záznam souboru STAVY
    Read STAVY FIRST record NO LOCK at end set EOF to TRUE.

*   Cyklus do konce souboru STAVY
*   -----
    Perform until EOF,
*       Zvýším pořadové číslo pro zápis do subfajlu
        Add 1 to RRN,

*       Data ze souboru STAVY přesunu do subfajlu
        Move corr STAVYF0 to DATA-O,

```

```

*      Zapišu záznam do subfajlu podle pořadového čísla RRN
      Write subfile FORMATY format is "DATA",

*      Čtu DALŠÍ záznam se souboru STAVY
      Read STAVY NEXT record NO LOCK at end set EOF to TRUE
      End-read,
      End-perform.

*      Zapnu indikátor povolující zobrazení subfajlu
      If RRN > 0
        Set ZOBRAZIM to TRUE,
        Move corr INDIKATORY-OBRAZOVKY to RIDICI-O-INDIC,
      End-if.

*      -----
*      Sekce kontroly dat z obrazovky
*      -----
Kontrola-mnozstvi SECTION.
      If MNOZ in DATA-I > 1000
        Set CHYBA-MNOZSTVI to TRUE
        Set STALA-CHYBA to TRUE
      Else
        Set NENI-CHYBA-MNOZSTVI to TRUE
      End-if.

```

## ÚDRŽBA DATABÁZOVÉHO SOUBORU S PODSOUBOREM A OKNY

Následující příklad představuje jeden z možných způsobů, jak použít obrazovkový podsoubor k údržbě databázového souboru, tj. ke změnám, přidávání a rušení záznamů.

Nejprve se zobrazí seznam databázových záznamů, tak aby v každém řádku byly vidět jen jeho rozpoznávací údaje. Na začátku každého řádku se navíc zobrazí jednomístné pole (VOLBA) sloužící k zadání činnosti, kterou uživatel chce se zvoleným záznamem provést. Činnost zadává ve formě znakového kódu (volby): 2 - prohlížení nebo změna, 4 - zrušení.

Po zadání kódů ke zvoleným záznamům stiskne uživatel klávesu Enter. Jednotlivé volby se pak postupně provedou, tzn., že záznam se zadanou volbou 4 se zruší, zatímco záznam s volbou 2 se zobrazí a uživatel jej může změnit. Přitom se zkontroluje, zda uživatel nezadal chybný kód volby. Jako správný kód se připouští mezera, která neurčuje žádnou činnost, ale je nutná pro případný výmaz nežádoucí nebo chybné volby.

Chce-li uživatel vložit nový záznam do databáze, stiskne klávesu F6. Zobrazí se prázdný formát záznamu a uživatel zapíše potřebné údaje včetně klíčových.

Chce-li uživatel obnovit zobrazení podsouboru podle skutečného stavu databáze, např. po zrušení záznamu, stiskne klávesu F12.

Formát, v němž se zobrazuje databázový záznam, bude mít tentokrát formu „okna“, tzn., že bude orámován a zbytek obrazovky zůstane viditelný (i bez klíčového slova OVERLAY). Rozhodující je v něm klíčové slovo

**WINDOW** - poloha a rozměry okna.

## Obrazovkový souboru STAVYW3I

A	DSPSIZ(24 80 *DS3)
A	REF(*LIBL/REF)
A	<b>INDARA</b>
A	CA03(03 'Konec')
A*	Datový formát podsouboru
A	R DATASF
	SFL

```

A 80          SFLNXTCHG
A          VOLBA          1A B 8 3
A 80          DSPATR(RI)
A          ZAVOD          R          O 8 9
A          SKLAD          R          O 8 15
A          MATER          R          O 8 21
A          MNOZ          R          O 8 30EDTCDE(P)
A* Ridici formát podsouboru
A          R RIDICI          SFLCTL(DATASF)
A          SFLSIZ(0100)
A          SFLPAG(0005)
A          CF05(05 'Obnova')
A          CF06(06 'Nový záznam')
A          OVERLAY
A 51          SFLDSP
A          SFLDSPCTL
A 56          SFLCLR
A 51          SFLEND
A 81          SFLMSG('Chybná volba')
A          KURZOR          4S 0H          SFLRCDNBR(CURSOR)
A          2 9'Přehled stavu zásob'
A          4 3'Volba: 2=Změna 4=Zrušení'
A          6 2'Volba'
A          6 8'Závod'
A          6 14'Sklad'
A          6 21'Materiál'
A          6 31'Množství'
A* Formát okna pro opravu a rušení
A          R OKNOOPR          WINDOW(*DFT 6 40)
A          CF12(12 'Navrat')
A*          (Pozice v okně jsou vztaženy k hornímu levému rohu)
A          1 4'Závod'
A          1 11'Sklad'
A          1 18'Materiál'
A          1 28'Množství'
A          ZAVOD          R          O 2 4
A          SKLAD          R          O 2 11
A          MATER          R          O 2 18
A          MNOZ          R          B 2 26EDTCDE(P)
A          4 2'F3=Konec'
A          4 14'F12=Navrat'
A* Formát okna pro nový záznam
A          R OKNONOVA          WINDOW(7 30 6 40)
A          CF12(12 'Návrat')
A*          (Pozice v okně jsou vztaženy k hornímu levému rohu)
A          1 4'Závod'
A          1 11'Sklad'
A          1 18'Materiál'
A          1 28'Množství'
A          ZAVOD          R          B 2 4
A 82          DSPATR(RI)
A          SKLAD          R          B 2 11
A 82          DSPATR(RI)
A          MATER          R          B 2 18
A 82          DSPATR(RI)
A          MNOZ          R          B 2 26EDTCDE(P)
A          4 2'F3=Konec'
A          4 14'F12=Návrat'
A 82          5 2'Materiál je už ve stavu'
A          R NAVOD
A          23 3'F3=Konec'
A          23 15'F6=Nový záznam'

```

## Program STAUDRI pro údržbu souboru STAVY

```

*****
*
*   Údržba stavového souboru s volbami a oknem
*   - v DDS obrazovky je INDARA (indicator area)
*

```

```

*      ASSIGN ... přidá se -SI.
*      Indikátory obrazovky jsou ve zvláštní struktuře mimo data:
*      01 STAVYW3I-INDICS.
*      COPY DDS-ALL-FORMATS-INDIC OF STAVYW3I.
*      Indikátory se nastavují a testují přímo v této struktuře.
*
*      Příkazy WRITE a READ mají přidanou frázi INDICATORS ARE ...
*
*****

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.

*      Obrazovkový soubor STAVYW3I
*      -----
- SI      SELECT STAVYW3I,
          ASSIGN TO WORKSTATION-STAVYW3I-SI,
          ACCESS MODE IS DYNAMIC,
          RELATIVE KEY IS RRN,
          ORGANIZATION IS TRANSACTION.

*      Fyzický soubor STAVY
*      -----
          SELECT STAVY ASSIGN TO DATABASE-STAVY,
          ACCESS IS DYNAMIC,
          ORGANIZATION IS INDEXED,
          RECORD KEY IS EXTERNALLY-DESCRIBED-KEY.

DATA DIVISION.

FILE SECTION.

*      Obrazovkový soubor STAVYW3I - Popis dat
*      -----
FD STAVYW3I.
01 FORMATY.
   COPY DDS-ALL-FORMATS OF STAVYW3I.

*      Fyzický soubor STAVY - Popis dat
*      -----
FD STAVY.
01 STAVYR.
   COPY DDS-ALL-FORMATS OF STAVY.

WORKING-STORAGE SECTION.

01  END-OF-FILE                      PIC X VALUE SPACE.
    88 EOF                          VALUE "E".
    88 NOT-EOF                       VALUE SPACE.

*      Pořadové číslo pro záznamy subfajlu - čítač
77  RRN                             PIC 9(05)    VALUE ZERO.

*      Hodnoty indikátorů - zapnuto/vypnuto
77  IND-ON                          PIC 1         VALUE B"1".
77  IND-OFF                         PIC 1         VALUE B"0".

*      Indikátory obrazovky ve zvláštních strukturách oddělených od dat
01  STAVYW3I-INDICS.
    COPY DDS-ALL-FORMATS-INDIC OF STAVYW3I.

*      Příznak zrušení stavového záznamu
77  PRIZNAK-ZRUSENI                 PIC 1         VALUE IS B"0".
    88 ZRUSENO                      VALUE IS B"1".
    88 NEZRUSENO                    VALUE IS B"0".

PROCEDURE DIVISION.

*-----

```



```

*       Hlavní programová sekce
*-----

Hlavni-sekce SECTION.
*       Otevřu soubory
*       -----
      Open I-O STAVYW3I.
      Open I-O STAVY.

*       Provedu sekci, kde se naplní subfajl
      Perform PLNENI-SUBFAJLU.

*       Kurzor nastavím na 1 (musí být 1 až počet záznamů v subfajlu)
      Move 1 to KURZOR of RIDICI-O.

*       Zobrazím záhlaví a subfajl a čekám na vstup z klávesnice
*       -----
      ZOBRAZIT-ZNOVU.

*       Jestliže byl zrušen aspoň jeden záznam, naplním znovu subfajl
      If ZRUSENO
        Perform PLNENI-SUBFAJLU
*       Kurzor nastavím na 1 po zrušení záznamu stavů
        Move 1 to KURZOR of RIDICI-O
      End-if.

*       Zapnu indikátor povolující zobrazení subfajlu
      If RRN > 0
        Move IND-ON to IN51 of RIDICI-O-INDIC,
      End-if.

*       Zobrazím záhlaví a subfajl a čekám na vstup z klávesnice
      Write FORMATY format is "NAVOD".
      Write FORMATY from RIDICI-O of STAVYW3I format is "RIDICI"
          indicators are RIDICI-O-INDIC.
      Read STAVYW3I record into RIDICI-I of STAVYW3I
          format is "RIDICI"
          indicators are RIDICI-I-INDIC.

*       Po stisku klávesy F3 končím program
      If IN03 in RIDICI-I-INDIC = IND-ON Go to KONEC-PROGRAMU.

*       Po stisku klávesy F6 vytvořím nový záznam stavů
      If IN06 in RIDICI-I-INDIC = IND-ON
        Perform NOVA
        Perform PLNENI-SUBFAJLU
        Move 1 to KURZOR of RIDICI-O
        Go to ZOBRAZIT-ZNOVU
      End-if.

*       Po stisku klávesy ENTER zpracuji změněné záznamy subfajlu
      Set NOT-EOF, NEZRUSENO to TRUE.
      Move IND-OFF to IN81 in RIDICI-O-INDIC.
      Move IND-OFF to IN80 in DATASF-O-INDIC.
      Move 1 to KURZOR of RIDICI-O

*       Čtu první změněný záznam subfajlu
      Read subfile STAVYW3I modified into DATASF-I of STAVYW3I
          format "DATASF"
          indicators are DATASF-I-INDIC
          at end set EOF to TRUE.

*       Cyklus do konce změněných záznamů subfajlu
      Perform until EOF
*       Kontrola pole VOLBA na 2, 4 nebo mezera
      Perform KONTROLA,
*       Když je VOLBA správná, provedu akci 2 (oprava) nebo 4 (zrušení)
      If IN80 in DATASF-O-INDIC = IND-OFF Then
        If VOLBA of DATASF-I = "2" Then
          Perform OPRAVA
          Move SPACE to VOLBA of DATASF-O
*       Přepíšu záznam subfajlu - promítnou se změny

```

```

        Rewrite subfile FORMATY from DATASF-O of STAVYW3I
            format is "DATASF" End-rewrite
    Else
    If VOLBA of DATASF-I = "4" Then
        Perform ZRUSENI,
        Set ZRUSENO to TRUE,
    Else
    If VOLBA of DATASF-I = " " Then
        Move corr DATASF-I of STAVYW3I
            to DATASF-O of STAVYW3I
        Rewrite subfile FORMATY from DATASF-O of STAVYW3I
            format is "DATASF" End-rewrite
    End-if
    End-if
    End-if
Else
*       Chybné pole VOLBA - přepíšu záznam subfajlu
    If IN80 in DATASF-O-INDIC = IND-ON Then
*       Přepíšu záznam subfajlu - promítnou se změny
        Rewrite subfile FORMATY from DATASF-O of STAVYW3I
            format is "DATASF" End-rewrite
    End-if
    End-if

*       Určím pořadové číslo naposled změněného záznamu subfajlu
    Move RRN to KURZOR of RIDICI-O

*       Čtu další změněný záznam subfajlu
    Read subfile STAVYW3I modified into DATASF-I of STAVYW3I
        format "DATASF"
        indicators are DATASF-O-INDIC
        at end set EOF to TRUE End-read,
*       Konec cyklu přes změněné záznamy subfajlu
    End-perform.

*       Vrátím se na nové zobrazení subfajlu s řídícím formátem
    Go to ZOBRAZIT-ZNOVU.

KONEC-PROGRAMU.
    Close STAVY.
    Close STAVYW3I.
Exit program and continue run unit.

*-----
*       Sekce PLNENI-SUBFAJLU, kde se plní subfajl
*-----
    PLNENI-SUBFAJLU SECTION.
*       Vymažu data ze subfajlu
    Move IND-ON to IN56 in RIDICI-O-INDIC.
    Write FORMATY from RIDICI-O of STAVYW3I format is "RIDICI"
        indicators are RIDICI-O-INDIC.
    Move IND-OFF to IN56 in RIDICI-O-INDIC.

*       Vypnu indikátory chyb v subfajlu
    Move IND-OFF to IN80 in DATASF-O-INDIC.
*       Smažu pole VOLBA
    Move SPACE to VOLBA of DATASF-O.
*       Vynuluji čítač záznamů subfajlu
    Move 0 to RRN.
*       Vypnu příznak konce souboru (subfajlu)
    Set NOT-EOF to TRUE.

*       Čtu PRVNÍ záznam souboru STAVY
    Read STAVY FIRST record NO LOCK at end set EOF to TRUE.

*       Cyklus do konce souboru STAVY
    Perform until EOF,
*       Zvýším pořadové číslo pro zápis do subfajlu
    Add 1 to RRN,
*       Data ze souboru STAVY přesunu do subfajlu
    Move corr STAVYF0 to DATASF-O of STAVYW3I,
*       Zapišu záznam do subfajlu podle pořadového čísla RRN

```

```

        Write subfile FORMATY from DATASF-O of STAVYW3I
                format is "DATASF"
                indicators are DATASF-O-INDIC
*       Čtu DALŠÍ záznam se souboru STAVY
        Read STAVY NEXT record NO LOCK at end set EOF to TRUE
        End-read,
End-perform.

*-----
*       OPRAVA - sekce, kde se přes okno opraví záznam souboru STAVY
*-----
OPRAVA SECTION.
*       Nastavím klíč pro stavový záznam
        Move corresponding DATASF-I of STAVYW3I to STAVYF0.
*       Přesunu data ze záznamu stavů do okna
        Move corresponding STAVYF0 to OKNOOPR-O.
*       Zobrazím okno a čtu vstupní data
        Write FORMATY from OKNOOPR-O of STAVYW3I format "OKNOOPR".
        Read STAVYW3I into OKNOOPR-I of STAVYW3I format "OKNOOPR"
                indicators are OKNOOPR-I-INDIC.
*       Po klávese F3 končím program
        If IN03 in OKNOOPR-I-INDIC = IND-ON Go to KONEC-PROGRAMU.
*       Po klávese F12 končím sekci
        If IN12 in OKNOOPR-I-INDIC = IND-ON Go to KONEC-OPRAVA.

*       Po klávese ENTER přepíšu záznam stavů
        Read STAVY invalid key continue End-read.
*       Přepíšu množství ve stavech
        Move corr OKNOOPR-I of STAVYW3I to STAVYF0.
*       Přepíšu záznam v souboru STAVY
        Rewrite STAVYR invalid key continue End-rewrite.
        KONEC-OPRAVA.
        Move corr STAVYF0 to DATASF-O of STAVYW3I.

*-----
*       NOVA - sekce, kde se přes okno vytvoří nový záznam souboru STAVY
*-----
NOVA SECTION.
        Initialize OKNONOVA-O of FORMATY.
        ZNOVU-NOVA.
        Write FORMATY from OKNONOVA-O of STAVYW3I format "OKNONOVA"
                indicators are OKNONOVA-O-INDIC.
        Read STAVYW3I into OKNONOVA-I of STAVYW3I format "OKNONOVA"
                indicators are OKNONOVA-I-INDIC.
*       Po klávese F3 končím program
        If IN03 in OKNONOVA-I-INDIC = IND-ON Go to KONEC-PROGRAMU.
*       Po klávese F12 končím sekci
        If IN12 in OKNONOVA-I-INDIC = IND-ON Go to KONEC-NOVA.

*       Po klávese ENTER zapíšu nový záznam stavů
*       Nastavím klíč a čtu stavový záznam
        Move corr OKNONOVA-I of STAVYW3I to STAVYF0,
*       Zapíšu nový záznam do souboru STAVY
        Write STAVYR invalid key
        Existuje-li vkládaný záznam, zapnu chybové indikátory
*       a opakuji zobrazení okna
        Move corr STAVYF0 to OKNONOVA-O of STAVYW3I
        Move IND-ON to IN82 in OKNONOVA-O-INDIC
        Go to ZNOVU-NOVA
        End-write.
        KONEC-NOVA.

*-----
*       ZRUSENI - sekce, kde se zruší záznam databáze STAVY
*-----
ZRUSENI SECTION.
        Move corresponding DATASF-I of STAVYW3I to STAVYF0.
        Delete STAVY invalid key Continue.

*-----
*       KONTROLA - sekce, kde se nastaví indikátory pro pole VOLBA
*-----

```

#### KONTROLA SECTION.

```
If VOLBA of DATASF-I not equal to "2" and
VOLBA of DATASF-I not equal to "4" and
VOLBA of DATASF-I not equal to " "
Move IND-ON to IN80 in DATASF-O-INDIC
Move IND-ON to IN81 in RIDICI-O-INDIC
Else
Move IND-OFF to IN80 in DATASF-O-INDIC
End-if.
```

## VOLÁNÍ PROGRAMU

V jazyku COBOL se programy volají příkazem CALL.

Příkaz CALL má tvar

```
CALL [ PROGRAM ] program [IN LIBRARY knihovna] [USING parametr parametr ...]
[ON EXCEPTION příkaz] END-CALL
```

kde

*program* je buď jméno volaného programu uzavřené v uvozovkách nebo znaková proměnná, jejíž prvních 10 znaků je jméno volaného programu,

*knihovna* je buď jméno knihovny uzavřené v uvozovkách nebo znaková proměnná, jejíž prvních 10 znaků je jméno knihovny,

*parametr* má tvar

```
[BY REFERENCE] jméno-dat
[BY CONTENT] jméno-dat
```

*jméno-dat* je definováno na úrovni 01, 77 nebo je to elementární položka v sekci WORKING-STORAGE, LOCAL-STORAGE nebo LINKAGE. REFERENCE znamená, že volanému programu se předává odkaz na data volajícího programu. CONTENT znamená, že volanému programu se předává kopie dat z volajícího programu.

Volající program vydá příkaz CALL, který specifikuje volaný program a případné parametry. Volaný program může být buď samostatný nebo vnořený (nested). Předává-li volající program v příkazu CALL parametry, musí volaný program specifikovat tyto parametry v záhlaví PROCEDURE DIVISION a v sekci LINKAGE.

### Volající program CALLING

Ilustruje příkaz CALL s parametry, který volá program CALLED. Po návratu z volaného programu získá z posledního parametru obsah trojúhelníka (spočteného z prvních tří parametrů) a zobrazí jej v editovaném tvaru.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.      CALLING.
```

```
WORKING-STORAGE SECTION.
```

```
01  a          PACKED-DECIMAL PICTURE S9(10)V9(5) VALUE 3.0.
01  b          PACKED-DECIMAL PICTURE S9(10)V9(5) VALUE 4.0.
01  c          PACKED-DECIMAL PICTURE S9(10)V9(5) VALUE 5.0.
01  plocha     PACKED-DECIMAL PICTURE S9(10)V9(5) .
```

```
77  plocha-displayed PICTURE Z(10).9(5) .
```

```
PROCEDURE DIVISION.
```

```
CALL PROGRAM "CALLED" USING a b c plocha.
MOVE plocha to plocha-displayed.
DISPLAY "plocha = " plocha-displayed.
```

## Volaný program CALLED

Ilustruje volaný program přijímající parametry, které specifikuje v sekci LINKAGE a v záhlaví PROCEDURE DIVISION s klausulí USING. Program vypočítá obsah trojúhelníka podle Heronova vzorce z prvních tří parametrů a vrátí jej v posledním parametru.

```
IDENTIFICATION DIVISION. PROGRAM-ID.    CALLED.
WORKING-STORAGE SECTION.
01  s                PACKED-DECIMAL PICTURE S9(10)V9(5) .
01  soucin            PACKED-DECIMAL PICTURE S9(10)V9(5) .

LINKAGE SECTION.
01  a                PACKED-DECIMAL PICTURE S9(10)V9(5) .
01  b                PACKED-DECIMAL PICTURE S9(10)V9(5) .
01  c                PACKED-DECIMAL PICTURE S9(10)V9(5) .
01  plocha           PACKED-DECIMAL PICTURE S9(10)V9(5) .

PROCEDURE DIVISION USING a  b  c  plocha.
    IF a >= b + c OR b >= a + c OR c >= a + b
        MOVE -1.0 TO plocha
        GOBACK
    END-IF.
    COMPUTE  s = (a + b + c) / 2
    COMPUTE  soucin = s * (s - a) * (s - b) * (s - c)
    COMPUTE  plocha = FUNCTION SQRT ( soucin )
    GOBACK
```

## Program CALLNESTED

Ilustruje volání vnořeného programu OBSAH z vnějšího programu CALLNESTED. Vnořený program začíná divizí IDENTIFICATION, končí zápisem END PROGRAM OBSAH a je obsažen ve stejném zdrojovém textu mezi koncem procedurové divize vnějšího programu a zápisem END PROGRAM CALLNESTED.

```
*  -----
*  hlavní program
*  -----

IDENTIFICATION DIVISION. PROGRAM-ID.    CALLNESTED.

WORKING-STORAGE SECTION.
01  a                PACKED-DECIMAL PICTURE S9(10)V9(5) VALUE 3.0.
01  b                PACKED-DECIMAL PICTURE S9(10)V9(5) VALUE 4.0.
01  c                PACKED-DECIMAL PICTURE S9(10)V9(5) VALUE 5.0.
01  plocha           PACKED-DECIMAL PICTURE S9(10)V9(5) .
77  plocha-displayed PICTURE  Z(10).9(5) .
PROCEDURE DIVISION.
    CALL "OBSAH",
        USING a b c plocha
        MOVE plocha TO plocha-displayed
        DISPLAY "plocha = " plocha-displayed.

*  -----
*  Vnořený program OBSAH
*  -----

IDENTIFICATION DIVISION. PROGRAM-ID.    OBSAH.
WORKING-STORAGE SECTION.
01  s                PACKED-DECIMAL PICTURE S9(10)V9(5) .
01  soucin            PACKED-DECIMAL PICTURE S9(10)V9(5) .

LINKAGE SECTION.
01  a                PACKED-DECIMAL PICTURE S9(10)V9(5) .
01  b                PACKED-DECIMAL PICTURE S9(10)V9(5) .
01  c                PACKED-DECIMAL PICTURE S9(10)V9(5) .
01  plocha           PACKED-DECIMAL PICTURE S9(10)V9(5) .

PROCEDURE DIVISION USING a b c  plocha.
    IF a >= b + c OR b >= a + c OR c >= a + b
        MOVE -1.0 TO plocha
        GOBACK
    END-IF.
```

```
      COMPUTE  s = (a + b + c) / 2
      COMPUTE  soucin = s * (s - a) * (s - b) * (s - c)
      COMPUTE  plocha = FUNCTION SQRT ( soucin )
      GOBACK.
END PROGRAM OBSAH.
END PROGRAM CALLNESTED.
```

## PROSTŘEDKY K LADĚNÍ PROGRAMŮ

K ladění programů a zachycení chyb můžeme použít prostředky vlastního jazyka COBOL nebo prostředky operačního systému.

### Prostředky jazyka

Je pochopitelné, že se snažíme nejprve chybám zabránit především použitím prostředků jazyka. Jsou to následující jazykové konstrukce.

#### *Chyby v programu obecně*

klausule PROGRAM STATUS

#### *Chyby aritmetických operací*

fráze ON SIZE ERROR

#### *Chyby operací se znakovými řetězci*

fráze ON OVERFLOW

#### *Chyby u operací vstupu a výstupu*

fráze AT END

fráze INVALID KEY

fráze NO DATA

deklarativní procedura USE AFTER EXCEPTION/ERROR

klausule FILE STATUS

#### *Chyby operace CALL*

fráze ON EXCEPTION

### PROGRAM STATUS

Tato fráze se zapisuje v divizi ENVIRONMENT DIVISION do paragrafu SPECIAL-NAMES.

**PROGRAM STATUS (IS)** *jméno-dat* **START POSITION (IS)** *integer*

Zde je jméno-dat název dostatečně dlouhé znakové položky v sekci WORKING STORAGE a integer je celé číslo určující pozici (počínaje nulou) ve struktuře *program data structure*:

Pozice	Délka	Formát	Popis
0	10	znaky	Program name
10	10	znaky	Program library name
20	10	znaky	Module name
30	10	znaky	Statement number
40	6	znaky	Optimization level
46	7	znaky	Exception message ID
53	10	znaky	Job name
63	6	znaky	Job number
69	1	znaky	Job type
70	10	znaky	User profile

80	14	znaky	Timestamp YYYYMMDDHHMMSS
----	----	-------	-----------------------------

## ON SIZE ERROR

Tato fráze se uvádí na konci příkazu u operací ADD, SUBTRACT, MULTIPLY, DIVIDE a COMPUTE. Např.

```
ADD ... TO ... ON SIZE ERROR příkaz END-ADD
```

Kde *příkaz* reaguje na výjimečný stav při operaci.

## ON OVERFLOW

Tato fráze se uvádí na konci příkazu u operací STRING a UNSTRING. Např.

```
UNSTRING TEXT1 DELIMITED BY ALL SPACES
      INTO EMPTY-FIELD, POLOZKA-1, POLOZKA-2, POLOZKA-3
ON OVERFLOW DISPLAY "OVERFLOW"
END-UNSTRING
```

## AT END

Tato fráze se uvádí v příkazu READ při sekvenčním zpracování souboru (ACCESS MODE IS SEQUENTIAL v klausuli SELECT). Může být zapsána jednou nebo dvakrát, a to pozitivně AT END nebo negativně NOT AT END. Např.

```
READ STAVY FIRST record NO LOCK AT END set EOF to TRUE
```

## INVALID KEY

Tato fráze se uvádí v příkazech READ, START, WRITE, REWRITE a DELETE u souborů s indexovou a relativní organizací (ORGANIZATION IS INDEXED/RELATIVE v klausuli SELECT). Může být zapsána jednou nebo dvakrát, a to pozitivně INVALID (KEY) nebo negativně NOT INVALID (KEY). Např.

```
READ STAVY record into STAVYF0 no lock
      INVALID KEY Set NENALEZEN to TRUE
      NOT INVALID KEY Set NALEZEN to TRUE
End-read
```

## NO DATA

Tato fráze se uvádí v příkazu READ u obrazovkových souborů (ORGANIZATION IS TRANSACTION v klausuli SELECT). Použije s v případě, že příkaz nemá čekat na vstup z klávesnice.

## USE AFTER EXCEPTION/ERROR

Procedura uvozená příkazem USE ve speciální sekci (jedné nebo více) mezi slovy DECLARATIVES a END DECLARATIVES umožňuje provést příkazy reagující na výjimečnou událost (exception, error).

```
USE AFTER STANDARD EXCEPTION PROCEDURE ON kontext .
```

kde *kontext* je buď jedno nebo několik jmen souborů, anebo jedno ze slov INPUT, OUTPUT, I-O, EXTEND.

## Program FILESTATUS

Ilustruje deklaraci chybové procedury pro vstupní soubor a pro výstupní soubor. Zároveň ilustruje využití klausule FILE-STATUS v paragrafu FILE-CONTROL v sekci INPUT-OUTPUT. Tento program chybně otevírá výstupní soubor jako vstupní, takže po otevření obou souborů zhavaruje



příkaz WRITE, v důsledku čehož se provede výpočet v deklarované sekci OUTPUT-ERROR SECTION v úseku mezi DECLARATIVES a END DECLARATIVES.

```
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT INPUT-FILE ASSIGN TO DISK-FILEA
        FILE STATUS IS INPUT-FILE-STATUS.
    SELECT OUTPUT-FILE ASSIGN TO DISK-FILEB
        FILE STATUS IS OUTPUT-FILE-STATUS.

DATA DIVISION.
FILE SECTION.
FD INPUT-FILE.
01 INPUT-RECORD.
    05 INPUT-EMPLOYEE-NUMBER PICTURE 9(6).
    05 INPUT-EMPLOYEE-NAME    PICTURE X(28).
    05 INPUT-EMPLOYEE-CODE    PICTURE 9.
    05 INPUT-EMPLOYEE-SALARY  PICTURE 9(6)V99.
FD OUTPUT-FILE.
01 OUTPUT-RECORD LIKE INPUT-RECORD.

WORKING-STORAGE SECTION.
77 INPUT-FILE-STATUS          PICTURE XX.
77 OUTPUT-FILE-STATUS        PICTURE XX.
77 OP-NAME                    PICTURE X(7).
01 INPUT-END                  PICTURE X VALUE SPACE
    88 THE-END-OF-INPUT       VALUE "E".

PROCEDURE DIVISION.
DECLARATIVES.
INPUT-ERROR SECTION.
    USE AFTER STANDARD ERROR PROCEDURE ON INPUT-FILE.
INPUT-ERROR-PARA.
    DISPLAY "UNEXPECTED ERROR ON ", OP-NAME, "FOR INPUT-FILE".
    DISPLAY "FILE STATUS IS ", INPUT-FILE-STATUS.
    DISPLAY "PROCESSING ENDED".
    STOP RUN.
OUTPUT-ERROR SECTION.
    USE AFTER STANDARD ERROR PROCEDURE ON OUTPUT-FILE.
OUTPUT-ERROR-PARA.
    DISPLAY "UNEXPECTED ERROR ON ", OP-NAME, "FOR OUTPUT-FILE".
    DISPLAY "FILE STATUS IS ", OUTPUT-FILE-STATUS.
    DISPLAY "PROCESSING ENDED".
    STOP RUN.
END DECLARATIVES.

MAIN-PROGRAM SECTION.
MAINLINE.
    MOVE "OPEN" TO OP-NAME.
    OPEN INPUT INPUT-FILE OUTPUT-FILE.
    MOVE "READ" TO OP-NAME.
    READ INPUT-FILE INTO OUTPUT-RECORD
        AT END SET THE-END-OF-INPUT TO TRUE END-READ.
    PERFORM UNTIL THE-END-OF-INPUT
        MOVE "WRITE" TO OP-NAME
        WRITE OUTPUT-RECORD
        MOVE "READ" TO OP-NAME
        READ INPUT-FILE INTO OUTPUT-RECORD
        AT END SET THE-END-OF-INPUT TO TRUE END-READ
    END-PERFORM.
    MOVE "CLOSE" TO OP-NAME.
    CLOSE INPUT-FILE
        OUTPUT-FILE.
    STOP RUN.
```

## Prostředky operačního systému

### Výpis paměti (QlnDumpCobol API)

Výpis paměti je program volaný jako tzv. API (Application Program Interface) s názvem *QlnDumpCobol*. Jeho volání vyžaduje příkaz pro kompilátor PROCESS OPTIONS NOMONOPRC, který nedovolí převést jméno programu v příkazu CALL na velká písmena. K volání toho programu musíme zadat data pro parametry.

```
PROCESS OPTIONS NOMONOPRC.

WORKING-STORAGE SECTION.
01  ERROR-CODE.
    05  BYTES-PROVIDED      PIC S9(6)  BINARY VALUE ZERO.
    05  BYTES-AVAILABLE    PIC S9(6)  BINARY VALUE ZERO.
    05  EXCEPTION-ID       PIC X(7) .
    05  RESERVED-X        PIC X.
    05  EXCEPTION-DATA     PIC X(64) .
01  PROGRAM-NAME          PIC X(10) .
01  LIBRARY-NAME          PIC X(10) .
01  MODULE-NAME           PIC X(10) .
01  PROGRAM-TYPE          PIC X(10) .
01  DUMP-TYPE             PIC X.

PROCEDURE DIVISION.
    MOVE LENGTH OF ERROR-CODE  TO BYTES-PROVIDED.
    MOVE "DUMP"                TO PROGRAM-NAME.
    MOVE "VZCOBOL"             TO LIBRARY-NAME.
    MOVE "DUMP"                TO MODULE-NAME.
    MOVE "*PGM"                TO PROGRAM-TYPE.
    MOVE "D"                   TO DUMP-TYPE.
    CALL LINKAGE IS PROCEDURE "QlnDumpCobol" USING
        PROGRAM-NAME, LIBRARY-NAME,
        MODULE-NAME, PROGRAM-TYPE,
        DUMP-TYPE, ERROR-CODE.
```

Struktura ERROR-CODE je předepsaná, BYTES-PROVIDED určují počet bajtů, které dáváme k dispozici programu, BYTES-AVAILABLE bude obsahovat počet bajtů získaných po skončení programu. EXCEPTION-ID je kód zprávy, který dosadí program a EXCEPTION-DATA je paměť vyhrazená pro zprávu z programu.

Údaj PROGRAM-NAME může být \*PGM nebo \*SRVPGM, údaj DUMP-TYPE může být D (data) nebo F (full).

První tři parametry lze vynechat tak, že místo jejich jmen dosadíme slovo OMITTED. Tím se do parametrů dostanou údaje vztahující se k právě běžícímu programu.

```
MOVE "*PGM"          TO PROGRAM-TYPE.
MOVE "F"             TO DUMP-TYPE.
CALL LINKAGE IS PROCEDURE "QlnDumpCobol" USING
    OMITTED, OMITTED,
    OMITTED, PROGRAM-TYPE,
    DUMP-TYPE, ERROR-CODE.
```

Datovou strukturu pro výpis paměti lze s výhodou zapsat do zvláštního zdrojového členu, např. DUMP\_FULL

```
01  ERROR-CODE.
    05  BYTES-PROVIDED      PIC S9(6)  BINARY VALUE 100.
    05  BYTES-AVAILABLE    PIC S9(6)  BINARY VALUE ZERO.
    05  EXCEPTION-ID       PIC X(7) .
    05  RESERVED-X        PIC X.
    05  EXCEPTION-DATA     PIC X(64) .
01  PROGRAM-NAME          PIC X(10) .
01  LIBRARY-NAME          PIC X(10) .
01  MODULE-NAME           PIC X(10) .
01  PROGRAM-TYPE          PIC X(10)  VALUE "*PGM".
01  DUMP-TYPE             PIC X      VALUE "F".
```

s konstantními parametry pro typ programu a typ výpisu a ten pak začlenit příkazem COPY do programu, který chceme ladit:

```
PROCESS OPTIONS NOMONOPRC.
...

WORKING-STORAGE SECTION.
...
    COPY DUMP_FULL.
...

PROCEDURE DIVISION.
...
    CALL LINKAGE IS PROCEDURE "QlnDumpCobol" USING
        OMITTED, OMITTED,
        OMITTED, PROGRAM-TYPE,
        DUMP-TYPE, ERROR-CODE.
...
```

## Ladicí program

V některých situacích je vhodné sledovat průběh výpočtu v programu a proměnné hodnoty. Ladění se zahajuje CL příkazem STRDBG. Ladění probíhá ve zvláštním režimu, který umožňuje zadávat příkazy a využívat funkčních kláves. Po ukončení ladění je žádoucí ukončit ladicí režim příkazem ENDDBG. Bez toho nelze spustit příkaz STRDBG znovu.

### Postup ladění

- Zkompilujeme moduly, které chceme v programu ladit, s parametrem **DBGVIEW**, nejlépe s hodnotou **\*LIST** nebo **\*SOURCE**. **\*LIST** zařadí do modulu protokol o kompilaci, takže se velikost modulu značně zvětší. **\*SOURCE** znamená, že ladicí program při své činnosti bude potřebovat zdrojový text. Hodnota **\*STMT** nepracuje s obrazem zdrojového textu nebo protokolu, ale jen s pořadovými čísly příkazů, a to podle protokolu o kompilaci.
- Vydáme příkaz **STRDBG** s příslušnými parametry.
- Zobrazí se zdrojový text programu.
- Zadáme alespoň jeden bod zastavení (breakpoint) nastavením kurzoru na některý řádek zobrazeného zdrojového textu a stiskem klávesy F6.
- Pomocí klávesy F21 vyvoláme příkazový řádek CL a vyvoláme program příkazem CALL.
- Používáme funkce ladicího programu (vydáváme příkazy na příkazovém řádku, nebo nastavujeme kurzor a tiskneme klávesové funkce).
- **Ukončíme** ladicí režim příkazem **ENDDBG**.

### Příklad ladění

Ladicí program si předvedeme na programu STAPOR. Napíšeme příkaz STRDBG a stiskneme klávesu F4, pak F10 a uvidíme tuto obrazovku:

```
Start Debug (STRDBG)

Type choices, press Enter.

Program . . . . . > STAPOR          Name, *NONE
Library . . . . . *LIBL            Name, *LIBL, *CURLIB
      + for more values
      *LIBL
Default program . . . . . *PGM      Name, *PGM, *NONE
Maximum trace statements . . . . . 200      Number
Trace full . . . . . *STOPTRC      *STOPTRC, *WRAP
```

Update production files . . . .	*NO	*NO, *YES
OPM source level debug . . . .	*NO	*NO, *YES
Service program . . . . .	*NONE	Name, *NONE
Library . . . . .		Name, *LIBL, *CURLIB

+ for more values

Vyplníme jméno programu STAPOR, jméno knihovny a \*YES do parametru *Update production files*, protože v programu pracujeme s přepisovaným databázovým souborem. Stiskneme Enter.

V zobrazeném zdrojovém textu programu najdeme výkonný příkaz (Open I-O STAVYW), kde chceme zahájit ladění. Na něj nastavíme kurzor a stiskneme klávesu F6:

```

Display Module Source

Program:  STAPOR      Library:  VZCOBOL      Module:  STAPOR
121      005500  Uvodni-akce.
122      66      005600  Open I-O STAVYW.
123      67      005700  Open I-O STAVY.
124      005800
125      68      005900  Initialize INDIKATORY-OBRAZOVKY.
126      006000
127      006100  Zobrazit-zadani.
128      69      006200  Write FORMATY from STAVYW1-O, format is "STAVYW1
129      70      006300  Read STAVYW record into STAVYW1-I.
130      71      006400  Move corr STAVYW1-I-INDIC
131      006500  to INDIKATORY-OBRAZOVKY.
132      72      006600  If KONEC go to Konec-programu.
133      74      006700  If NAVRAT go to Konec-programu.
134      006800
135      006900

More...

Debug . . .

F3=End program  F6=Add/Clear breakpoint  F10=Step  F11=Display variable
F12=Resume      F17=Watch variable      F18=Work with watch  F24=More keys
Breakpoint added to line 122.

```

Zpráva říká, že jsme nastavili bod zastavení (příkaz s ladicím číslem 122 a kompilačním číslem 66). Pak stiskneme klávesu F21 (vyvolání příkazového řádku CL) a na příkazový řádek napíšeme příkaz CALL STAPOR:

```

Display Module Source

Program:  STAPOR      Library:  VZCOBOL      Module:  STAPOR
121      005500  Uvodni-akce.
122      66      005600  Open I-O STAVYW.
123      67      005700  Open I-O STAVY.
124      005800
125      68      005900  Initialize INDIKATORY-OBRAZOVKY.
126      006000
127      006100  Zobrazit-zadani.
128      69      006200  Write FORMATY from STAVYW1-O, format is "STAVYW1
129      70      006300  Read STAVYW record into STAVYW1-I.
130      71      006400  Move corr STAVYW1-I-INDIC
131      006500  to INDIKATORY-OBRAZOVKY.
132      72      006600  If KONEC go to Konec-programu.
133      74      006700  If NAVRAT go to Konec-programu.
134      006800

.....
:                                     Command                                     :
:                                                                              :
:  ==> call stapor                                                            :
:  F4=Prompt  F9=Retrieve  F12=Cancel                                         :
:                                                                              :
:                                                                              :
:.....

```

Po stisku Enter se program spustí v ladicím režimu a výpočet se zastaví na bodu zastavení (breakpoint). Pak několikrát stiskneme klávesu F10 (step) a zobrazí se náš první formát:

**Zadejte údaje:**

```
Zavod:      01
Sklad:      01
Material:    00001
...
```

Vyplníme-li údaje jak je naznačeno a stiskneme Enter, skončí příkaz READ a výpočet se zastaví před příkazem Move corr.

Display Module Source

```
Program:  STAPOR      Library:  VZCOBOL      Module:  STAPOR
126      006000
127      006100  Zobrazit-zadani.
128      69      006200      Write FORMATY from STAVYW1-O, format is "STAVYW1
129      70      006300      Read STAVYW record into STAVYW1-I.
130      71      006400      Move corr STAVYW1-I-INDIC
131      006500                        to INDIKATORY-OBRAZOVKY.
132      72      006600      If KONEC go to Konec-programu.
133      74      006700      If NAVRAT go to Konec-programu.
134      006800
135      006900
136      007000*      Klíč pro hledání v souboru STAVY
137      76      007100      Move corr STAVYW1-I of FORMATY of STAVYW to STAV
138      007200
139      007300*      Čtu záznam souboru STAVY
140      77      007400      Read STAVY record into STAVYF0 no lock
More...
```

Debug . . .

```
F3=End program      F6=Add/Clear breakpoint      F10=Step      F11=Display variable
F12=Resume          F17=Watch variable      F18=Work with watch      F24=More keys
Step completed at line 130
```

Nastavíme kurzor na název STAVYW1-I a stiskneme klávesu F11. Objeví se informace s obsahem této struktury:

Evaluate Expression

Previous debug expressions

```
> EVAL STAVYW1-I
IN03 OF STAVYW1-I-INDIC OF STAVYW1-I OF FORMATY OF STAVYW = '0'
IN12 OF STAVYW1-I-INDIC OF STAVYW1-I OF FORMATY OF STAVYW = '0'
ZAVOD OF STAVYW1-I OF FORMATY OF STAVYW = '01'
SKLAD OF STAVYW1-I OF FORMATY OF STAVYW = '01'
MATER OF STAVYW1-I OF FORMATY OF STAVYW = '00001'
```

Můžeme nastavit jiný bod zastavení (breakpoint) klávesou F6 a stisknout klávesu F12=Resume (pokračovat bez krokování).

Až program skončí nebo až stiskneme klávesu F3 (konec programu), objeví se opět příkazový řádek CL, kde můžeme buď zopakovat volání programu, nebo můžeme ladění skončit (stiskem F12=Cancel).

Nezapomeňme vydat příkaz ENDDBG k ukončení ladicího režimu.

## Systemový výpis paměti po havárii

Dojde-li k nepředvídanému ukončení programu, objeví se na obrazovce zpráva vyžadující odpověď. Nabízí čtyři možnosti: C – Cancel, D – Data dump, F – Full dump, G – Go. Odpověď C ukončí výpočet bez další informace, D vypíše obsah proměnných do tiskové fronty jako spool file, F vypíše obsah proměnných a ještě informace o stavu souborů.

## Display Program Messages

```

Job 334002/VZUPKA/QPADEV0002 started on 02/19/16 at 12:44:41 in subsystem QI
STRUCT_01                :ABCD          CDEF          123456777888
TARGET_01 GROUP MOVE     :ABCD          CDEF          123456777888
TARGET_02 CORRESPONDING:777888CDEF          123456ABCD
TARGET_03 CHAR_01 to CHAR_02:ABCD
TARGET_03 NUM_03 TO CHAR_01 :123456
TARGET_03 NUM_03 TO NUM_04  :456000
Message 'MCH1202' in program object 'MOVE1' in library 'VZCOBOL' (C D F G).

```

Type reply, press Enter.  
Reply . . . **F**

F3=Exit      F12=Cancel

Výpis označuje číslo příkazu, v němž došlo k chybě (v ukázce číslo 35). Není to číslo z editoru, ale z protokolu o kompilaci.

[illegible]

Pohledem do kompilačního protokolu zjistíme, že chyba nastala v příkazu číslo 35, což je operace MOVE, která posílá znaková data do číselné proměnné.

```

File . . . . . : MOVE1
Control . . . . .
Find . . . . . : 35
*...+...1...+...2...+...3...+...4...+...5...+...6...+...7...+...8.
004200* Char to num - error
35 004300 MOVE CHAR_01 IN STRUCT_01 TO NUM_04 IN TARGET_03
36 004400 DISPLAY "TARGET_03 CHAR_01 TO NUM_04 : " NUM_04 IN TARGET_03

```

## Výpis protokolu úlohy (DSPJOBLOG)

Vydáme-li z příkazového řádku příkaz DspJobLog, pak po stisku klávesy F10 a Page Up uvidíme záznamy protokolu úlohy (jobu), z nichž můžeme někdy vyčíst podrobněji příčinu havárie programu:

```

                                Display All Messages
                                System:      IASSIST2
Job . . . :   QPADEV0002      User . . . :   VZUPKA      Number . . . :   334002

TARGET_02 CORRESPONDING:777888CDEF      123456ABCD
TARGET_03 CHAR_01 to CHAR_02:ABCD
TARGET_03 NUM_03 TO CHAR_01 :123456
TARGET_03 NUM_03 TO NUM_04  :456000
Decimal data error.

```

```
Function check. MCH1202 unmonitored by MOVE1 at statement 0000000035,  
instruction X'0000'.  
Message 'MCH1202' in program object 'MOVE1' in library 'VZCOBOL' (C D F  
G).  
? F  
Application error. *N unmonitored by *N at statement *N, instruction  
X'4000'.  
Job 334015/VZUPKA/MOVE1 submitted to job queue QBATCH in library QGPL.  
3 > WRKSPLF  
  
More...  
Press Enter to continue.  
  
F3=Exit   F5=Refresh   F12=Cancel   F17=Top   F18=Bottom
```