Programování v jazyku RPG IV

Vladimír Župka

Obsah

Úvod

Jazyk RPG byl vytvořen na začátku šedesátých let 20. století ve firmě IBM ke specifickému zpracování dat na děrných štítcích, zejména tisku výstupních sestav. Odtud pochází jeho název: *Report Program Generator* - generátor sestav.

Jazyk RPG se stále vyvíjí. Prodělal dlouhý vývoj od začátečního neprocedurálního jazyka RPG (1959) přes jazyk RPG II (1970) a RPG III (1978) k jazyku RPG IV (1994) a k jeho posledním modifikacím podle verzí operačního systému s oficiálním názvem ILE/RPG (stále také nazývané RPG IV). Poslední verze, zejména ve srovnání s verzí jazyka RPG III, umožňuje mnohem efektivnější vývoj programů, a to nejen zavedením volného formátu zápisu programového textu, ale i dalšími možnostmi. *Referenční příručka* je na adrese https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_73/rzasd/rzasdmain.htm. *Příručka programátora* s příklady je na adrese https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_73/rzasc/rzascmain.htm.

Umístění zdrojového textu programu, editace a kompilace

Kompilaci lze provádět CL příkazem CRTBNDRPG (Create Bound RPG Program) nebo příkazem CRTRPGMOD (Create RPG Module). Obsahuje-li program příkazy SQL, kompiluje se příkazem CRTSQLRPGI (Create SQL ILE RPG Object).

Například modul TESTPROG, jehož zdroj je umístěn v adresáři /home/vzrpg72, se kompiluje do knihovny VZRPG72 příkazem

```
CRTRPGMOD MODULE(VZRPG72/TESTPROG) SRCSTMF('/home/vzrpg72/
testprog.rpgle')
```

Zdrojový program ve členu zdrojového souboru se tradičně pořizuje a edituje pomocí programu SEU (Source Entry Utility). Ten však u programu obsahujícího popisy *souborů, dat a procedur* ve *volném tvaru* na nich hlásí chyby, protože takové příkazy nezná. Editor SEU není již firmou IBM nadále vylepšován. Tuto potíž lze řešit několika způsoby.

V samotném editoru SEU lze zvýrazněné chyby ignorovat a ukládat zdrojový text s volbou N ve volbě Return to editing.

Editaci a kompilaci ze zdrojového fyzického souboru a z IFS souboru v PC lze provádět pomocí komerčního programu IBM RDi (IBM Rational Developer for i), který je ke stažení na 60 dní zdarma na adrese https://www.ibm.com/products/rational-developer-for-i?mhsrc=ibmsearch_a&mhq=RDi.

Podobně lze editaci (bez kontoly syntaxe) a kompilaci provádět také pomocí bezplatné aplikace *IBMiProgTool*, která je ke stažení na adrese https://github.com/vzupka/IBMiProgTool.

K vytvoření zdrojového souboru a členu v knihovně, jakož i adresáře a souboru v IFS se zvolenou kódovou stránkou lze použít také aplikaci *IBMiProgTool*. Zdrojový text z PC lze zkopírovat do IFS souboru a tam jej kompilovat. Také je možné vytvářet, editovat a kompilovat zdrojový text přímo v IFS souboru.

Zaměření kursu

Kurs je koncipován jako praktický, neklade si za cíl seznámit účastníky se všemi možnostmi jazyka. Místo toho uvádí ucelené příklady, na nichž demonstruje nejdůležitější programovací prostředky.

Začáteční fáze kursu je zaměřena k praktickému procvičení práce s databázovými, tiskovými a obrazovkovými soubory na připravených příkladech. Účastníci se postupně seznámí s potřebnými příkazy a vždy po objasnění příkladu si sami pořídí jeho zdrojový text doslovně podle předlohy, pak ho přeloží a spustí. Když při tom udělají chyby, je to příležitost k dotazům a jejich objasnění.

Teprve v další fázi se na příkladech ukazují další prostředky jazyka RPG: manipulace s textem, operace s časovými údaji, datové struktury a vektory, obsluha mimořádných stavů a ladění programu. Tyto příklady již z časových důvodů nemusí účastníci sami pořizovat, ale mohou s nimi experimentovat, jak to časové poměry v kursu dovolí.

Další témata, která s jazykem ILE/RPG souvisí, tj. podprocedury a principy ILE (Integrated Language Envionment), zde nenajdeme, protože na to v základním kursu nezbývá čas. Jsou předmětem samostatného kursu.

Databázové soubory použité v příkladech

V kurzu budeme používat jednoduchý *referenční soubor* pro agendu materiálového zásobování:

```
     ******************************************************************
     *   Referenční soubor REFMZP
     ******************************************************************
    A..........T.Name++++++RLen++TDpB......Functions++++++++++++++++++++++++++
    A          R REFMZPF0
     *   CCCCCC
    A            CENA          10P 2      COLHDG('Cena')
     *   MMMMMM
    A            MATER          5         COLHDG('Číslo' 'mater.')
    A            MNOZ          10P 2      COLHDG('Množství')
    A            MNOBR         10P 2      COLHDG('Množství' 'obratu')
     *   NNNNNN
    A            NAZEV         50         COLHDG('Název' 'materiálu')
     *   SSSSSS
    A            SKLAD          2         COLHDG('Skl')
     *   ZZZZZZ
    A            ZAVOD          2         COLHDG('Zav')
```

První řádek popisu určuje jméno záznamu (formátu), které je povinné (zvolili jsme jméno REFMZPF0). Další řádky obsahují jména polí, která se budou používat v datových souborech, a zároveň jejich definice. Např. pole CENA je dlouhé 10 číslic, je uloženo ve staženém tvaru (P - packed) a má dvě desetinná místa. Pole MATER je znakové (nemá vyplněn údaj o desetinných místech) a je dlouhé 5 znaků. Popisy polí v referenčním souboru s výhodou řadíme abecedně.

Fyzický soubor STAVYP

```
     ******************************************************************
     *   Soubor STAVYP - Stavy materiálových zásob
     ******************************************************************
    A..........T.Name++++++RLen++TDpB......Functions++++++++++++++++++++++++++
     *                  Unikátní klíč (zákaz duplicit)
    A                                     UNIQUE
     *                  Referenční soubor definic polí
    A                                     REF(REFMZP)
     *   Jméno záznamu (formátu)
    A          R STAVYPF0
     *   Písmeno R u jmen polí označuje odkaz na referenční soubor REFMZP
    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
```

Fyzický soubor CENIKP

```
      ******************************************************************
      *   Soubor CENIKP - Ceník materiálu                              *
      ******************************************************************
    A                                     UNIQUE
    A                                     REF(REFMZP) CCSID(870)
    A          R CENIKPF0
     *   Číslo materiálu
    A            MATER     R
     *   Cena za jednotku materiálu
    A            CENA      R
     *   Název materiálu
    A            NAZEV     R
     *   Definice klíče - Číslo materiálu
    A          K MATER
```

Fyzický soubor OBRATP

```
     ****************************************************************
     *   Soubor OBRATP - Obraty materiálu                           *
     ****************************************************************
     *                                Referenční soubor definic polí
    A                                    REF(REFMZP)
     *   Jméno vety (formátu, záznamu)
    A          R OBRATPF0
     *   Jmena datových poli (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
```

Pořízení zdrojového programu a kompilace

Zdrojový program lze pořizovat a kompilovat dvěma způsoby:

- v textovém souboru IFS (Integrated File System).
- ve zdrojovém členu (member) zdrojového souboru (source file),

Člen či soubor musí být označen tzv. zdrojovým typem RPGLE nebo v případě použití předkompilátoru SQL typem SQLRPGLE.

Postup při použití zdrojového členu

Použijeme obslužný program SEU - Source Entry Utility, který je určen k pořizování a údržbě zdrojových textů různých typů. SEU rozeznává typ zdrojového textu. Používá se hlavně v rámci obslužného programu PDM (Programming Development Manager).

Při pořizování nového zdrojového programu postupujeme v následujících krocích:

- Založíme nový zdrojový soubor **QRPGLESRC** ve své knihovně příkazem **CRTSRCPF** (Create Source Physical File) s délkou řádku 112.
- Vydáme příkaz **WRKMBRPDM** (práce s členy v PDM) a vyplníme jméno zdrojového souboru **QRPGLESRC**, jméno knihovny, typ zdrojového textu **RPGLE**, jméno zdrojového členu (member) **PRVNI** a jeho textový popis.
- Stiskneme klávesu Enter a dostaneme prázdnou stránku, do níž napíšeme zdrojový text programu ve volném formátu.
- Program uložíme příkazem SAVE nebo pomocí klávesy F3. Na obrazovce se pak v seznamu zdrojvých členů objeví nový řádek s názvem programu. U tohoto řádku je vstupní pole pod nadpisem Opt. Do něj zapíšeme volbu 14 a stiskneme Enter. Spustí se kompilace, jejímž výsledkem je programový objekt PRVNI (typu *PGM). Kromě toho se vytiskne protokol o kompilaci.
- Kompilace je vlastně spuštění příkazu **CRTBNDRPG** s parametry určujícími zdrojový soubor a člen, např.:

```
CRTBNDRPG s parametry SRCFILE(VZRPG_FREE/QRPGLESRC) SRCMBR(PRVNI).
```

Postup při použití textového souboru

- Text programu zapíšeme v libovolném textovém editoru osobního počítače a přeneseme do IFS souboru vytvořeného se zvoleným atributem CCSID.
- Programy PDM a SEU nejsou v tomto případě použitelné. Kompilace se musí provést příkazem **CRTBNDRPG**, který musí určit cestu k textovému souboru, např.:

```
CRTBNDRPG s parametrem SRCSTMF('/home/user01/PRVNI.RPGLE').
```

Zápis textu programu

Text programu se zapisuje jako série příkazů, z nichž každý začíná na novém řádku. Příkaz může být zapsán na jednom nebo několika řádcích (prvním a pokračovacím), a to buď v pevném formátu (podle příslušného formuláře s pevně členěnými rubrikami) nebo ve volném formátu (bez formuláře). V programu mohou být kombinovány příkazy zapsané ve volném formátu a příkazy zapsané v pevném formátu. Příkaz volného formátu musí končit středníkem za posledním pokračovacím řádkem.

Jsou-li v programu použity příkazy s pevným formátem, musí všechny příkazy začínat od 8. sloupce v řádku a končit v 80. sloupci.

Upozornění: Editor SEU *nekontroluje* správnost zápisu příkazů ve volném formátu! Ve volném formátu *nejsou* k dispozici příkazy skoku a návěští – GOTO a TAG. Tuto potíž lze řešit např. pomocí bezplatné aplikace *IBMiProgTool*, která je ke stažení na adrese https://github.com/vzupka/IBMiProgTool. Tato aplikace umožňuje kopírování textů mezi PC do zdrojového členu nebo do IFS souboru a zpět a také dovoluje editovat a kompilovat programy.

Komentáře

Poznámkový řádek začíná *dvěma lomítky* // ve volném formátu kdekoliv. V polovolném a formulářovém formátu lze zapsat komentář začínající od od 8. sloupce nebo komentář obsahující *hvězdičku* * v 7. sloupci. Za středníkem volného příkazu může být zapsán komentář začínající dvěma lomítky //. Za komentář lze v pevném i volném formátu považovat i zcela *prázdný řádek*.

Jednotlivé formuláře v polovolném formátu mají pro komentář vyhrazeny *sloupce 80 – 100*.

Příklady zápisu programu

Starší verze jazyka RPG IV vyžadovaly tento formulářový zápis:

```
1234567. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 0
.....H*eywords++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++Comments++++++++++++
     h debug

.....F*ilename++IPEASFRlen+LKlen+AIDevice+.Keywords++++++++++++++++++++++++++++Comments++++++++++++
     fSTAVYP    if   e             disk

.....D*ame++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++Comments++++++++++++
     d soucet          s              10  2

.....C*0N01Factor1+++++++Opcode&ExtExtended-factor2++++++++++++++++++++++++++++Comments++++++++++++
     c                   read      STAVYP
     c                   dow       not %eof
     c                   eval      soucet = soucet + MNOZ
     c                   read      STAVYP
     c                   enddo
     c     soucet        dsply
     c                   eval      *inlr = *on
```

Další verze jazyka umožňuje volný zápis příkazů kombinovaný s formulářovým zápisem, v tzv. *polovolném* formátu. V tomto formátu bylo dříve nutné označovat úseky volného zápisu příkazy / free a /end-free, což již bylo zrušeno. Příkazy se musí zapisovat od 8. do 80. sloupce řádku:

```
1234567. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 0
     h debug

     fSTAVYP    if   e             disk

     d soucet          s              10  2

       read STAVYP;

       dow  not %eof;
          soucet += MNOZ;
          read STAVYP;
       enddo;

       dsply soucet;
       *inlr = *on;
```

Prozatím poslední verze jazyka (7.3) umožňuje zcela volný zápis bez možnosti kombinace s formulářovým zápisem. Symbol **FREE v prvním řádku zapsaný od 1. sloupce znamená, že program bude zapisován ve *zcela volném* formátu v řádcích libovolné délky.

```
1234567. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 0
**FREE
ctl-opt debug(*yes);

dcl-f STAVYP;

dcl-s soucet packed(10: 2);

read STAVYP;
dow  not %eof;
   soucet += MNOZ;
   read STAVYP;
enddo;

dsply soucet;
*inlr = *on;
```

Uvádíme protokol o kompilaci programu PRVNI (verze zcela volného formátu). V protokolu o kompilaci jsou kromě textu programu vyznačeny části, které se přebírají z externího popisu souboru STAVYP, což je objekt typu *FILE.

```
 5770WDS V7R4M0  190419 RN       IBM ILE RPG            VZRPG_FREE/PRVNI        PASSIST4   05/13/24 19:30:07       Page      1

 Command  . . . . . . . . . . . :   CRTBNDRPG
   Issued by  . . . . . . . . . :     VZUPKA
 Program  . . . . . . . . . . . :   PRVNI
   Library  . . . . . . . . . . :     VZRPG_FREE
 Text 'description' . . . . . . :   *SRCMBRTXT

 Source Member  . . . . . . . . :   PRVNI
 Source File  . . . . . . . . . :   QRPGLESRC
   Library  . . . . . . . . . . :     VZRPG_FREE
   CCSID  . . . . . . . . . . . :     870
 Text 'description' . . . . . . :   Source member PRVNI
 Last Change  . . . . . . . . . :   05/13/24  19:29:52

 Generation severity level  . . :   10
 Default activation group . . . :   *YES
 Compiler options . . . . . . . :   *XREF      *GEN       *NOSECLVL  *SHOWCPY
                                    *EXPDDS    *EXT       *NOSHOWSKP *NOSRCSTMT
                                    *DEBUGIO   *UNREF     *NOEVENTF
 Debugging views  . . . . . . . :   *ALL
 Debug encryption key . . . . . :   *NONE
 Output . . . . . . . . . . . . :   *PRINT
 Optimization level . . . . . . :   *NONE
 Source listing indentation . . :   *NONE
 Type conversion options  . . . :   *NONE
 Sort sequence  . . . . . . . . :   *HEX
 Language identifier  . . . . . :   *JOBRUN
 Replace program  . . . . . . . :   *YES
 User profile . . . . . . . . . :   *USER
```

```
  Authority  . . . . . :   *LIBCRTAUT
  Truncate numeric . . . . . :   *YES
  Fix numeric  . . . . . :   *NONE
  Target release . . . . . :   *CURRENT
  Allow null values  . . . . . :   *NO
  Define condition names . . . . . :   *NONE
  Enable performance collection  . :   *PEP
  Profiling data . . . . . :   *NOCOL
  Licensed Internal Code options . :
  Generate program interface . . . :   *NO
  Include directory  . . . . . :
  Preprocessor options . . . . . :   *NONE
 5770WDS V7R4M0  190419 RN       IBM ILE RPG            VZRPG_FREE/PRVNI        PASSIST4   05/13/24 19:30:07       Page      2

Line   <---------------------- Source Specifications ---------------------------------------------------->  Do  Change Src Seq
Number ....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....+....9....+...10  Num Date   Id  Number

                      S o u r c e   L i s t i n g

     1 **FREE                                                                                                 000000     000100
     2 ctl-opt debug(*yes);                                                                                   000000     000200
     3                                                                                                        000000     000300


     4 dcl-f STAVYP;                                                                                          000000     000400
        *------------------------------------------------------------------------------------------*
        *                                RPG name         External name                          *
        * File name. . . . . :  STAVYP           VZRPG_FREE/STAVYP                                *
        * Record format(s) . . . . . :  STAVYPF0         STAVYPF0                                 *
        *------------------------------------------------------------------------------------------*
     5                                                                                                        000000     000500


     6 dcl-s soucet packed(10: 2);                                                                            000000     000600
     7                                                                                                        000000     000700


     8=ISTAVYPF0                                                                                                         1000001
        *------------------------------------------------------------------------------------------*                     1
        * RPG record format  . . . . :  STAVYPF0                                                  *                     1
        * External format  . . . . . :  STAVYPF0 : VZRPG_FREE/STAVYP                              *                     1
        *------------------------------------------------------------------------------------------*                     1
     9=I                           A    1    2  ZAVOD                          Zav                                  1000002
    10=I                           A    3    4  SKLAD                          Skl                                  1000003
    11=I                           A    5    9  MATER                          Cislo mater.                         1000004
    12=I                           P   10   15 2MNOZ                          Mnozstvi                             1000005


    13 read STAVYP;                                                                                           000000     000800
    14                                                                                                        000000     000900
    15 dow  not %eof;                                                                                  B01    000000     001000
    16    soucet += MNOZ;                                                                               01    000000     001100
    17    read STAVYP;                                                                                  01    000000     001200
    18 enddo;                                                                                          E01    000000     001300
    19                                                                                                        000000     001400
    20 dsply soucet;                                                                                          000000     001500
    21 *inlr = *on;                                                                                           000000     001600

       * * * * *   E N D   O F   S O U R C E   * * * * *
 5770WDS V7R4M0  190419 RN       IBM ILE RPG            VZRPG_FREE/PRVNI        PASSIST4   05/13/24 19:30:07       Page      3

         A d d i t i o n a l   D i a g n o s t i c   M e s s a g e s

 Msg id  Sv Number Seq     Message text
*RNF7066 00      4 000400  Record-Format STAVYPF0 not used for input or output.
*RNF7086 00      4 000400  RPG handles blocking for file STAVYP. INFDS is updated only
                           when blocks of data are transferred.

 * * * * *   E N D   O F   A D D I T I O N A L   D I A G N O S T I C   M E S S A G E S   * * * * *
 5770WDS V7R4M0  190419 RN       IBM ILE RPG            VZRPG_FREE/PRVNI        PASSIST4   05/13/24 19:30:07       Page      4

                      C r o s s   R e f e r e n c e

      File and Record References:

         File              Device            References (D=Defined)
           Record
         STAVYP            DISK                   4D     13      17
           STAVYPF0                             4D      8

      Global Field References:

         Field             Attributes        References (D=Defined M=Modified)
         *INLR             N(1)                  21M
*RNF7031 MATER             A(5)                  11D
         MNOZ              P(10,2)               12D     16
*RNF7031 SKLAD             A(2)                  10D
         SOUCET            P(10,2)                6D     16M     20
*RNF7031 ZAVOD             A(2)                   9D

      Indicator References:

         Indicator                           References (D=Defined M=Modified)
         LR                                     21M

       * * * * *   E N D   O F   C R O S S   R E F E R E N C E   * * * * *
 5770WDS V7R4M0  190419 RN       IBM ILE RPG            VZRPG_FREE/PRVNI        PASSIST4   05/13/24 19:30:07       Page      5

                    E x t e r n a l   R e f e r e n c e s

      Statically bound procedures:
         Procedure                           References
```

```
        No references in the source.

      Imported fields:
         Field             Attributes         Defined
         No references in the source.

      Exported fields:
         Field             Attributes         Defined
         No references in the source.

   * * * * *   E N D   O F   E X T E R N A L   R E F E R E N C E S   * * * * *
 5770WDS V7R4M0  190419 RN        IBM ILE RPG            VZRPG_FREE/PRVNI         PASSIST4   05/13/24 19:30:07        Page      6

                         M e s s a g e   S u m m a r y

 Msg id  Sv Number Message text
*RNF7031 00      3 The name or indicator is not referenced.
*RNF7066 00      1 Record-Format name of Externally-Described file is not used.
*RNF7086 00      1 RPG handles blocking for the file. INFDS is updated only when
                   blocks of data are transferred.

       * * * * *   E N D   O F   M E S S A G E   S U M M A R Y   * * * * *
 5770WDS V7R4M0  190419 RN        IBM ILE RPG            VZRPG_FREE/PRVNI         PASSIST4   05/13/24 19:30:07        Page      7

                          F i n a l   S u m m a r y

  Message Totals:

    Information  (00) . . . . . :        5
    Warning      (10) . . . . . :        0
    Error        (20) . . . . . :        0
    Severe Error (30+)  . . . . :        0
    --------------------------------  -------
    Total . . . . . :        5


  Source Totals:

    Records . . . . . :       21
    Specifications  . . . . . :       15
    Data records  . . . . . :        0
    Comments  . . . . . :        5

         * * * * *   E N D   O F   F I N A L   S U M M A R Y   * * * * *
Program PRVNI placed in library VZRPG_FREE. 00 highest severity. Created on 05/13/24 at 19:30:08.
          * * * * *   E N D   O F   C O M P I L A T I O N   * * * * *
```

Příkazy použité v příkladech

Popisy souborů

Databázový soubor

```
DCL-F STAVYP DISK(*EXT) USAGE(*INPUT);
dcl-f stavyp; // totéž
```

Obrazovkový soubor

```
DCL-F STAVYW WORKSTN(*EXT) USAGE(*INPUT: *OUTPUT);
dcl-f stavyw workstn; // totéž
```

Popisy dat

Samostatné proměnné

```
DCL-S SOUCET PACKED(10:2);   // samostatná dekadická proměnná
dcl-s total like(soucet) inz(0);  // definice analogií
dcl-s text char(150) inz('Úvod'); // textová proměnná s inicializací
```

Konstanty

```
DCL-C pocet const(100);  // číselná konstanta
dcl-c pocet 100;         // totéž
dcl-c cmd 'Úvod';        // textová konstanta
```

Datové struktury

```
DCL-DS struktura;   // datová struktura
   znaky char(10);    // znakové podpole
   cislo int(10);     // celočíselné podpole
   datum date;        // datumové podpole
```

```
End-ds struktura;   // konec datové struktury jako příkaz

DCL-DS ds3 LEN(100) END-DS; // END-DS je zde součástí příkazu

// Datová struktura shodná s formátem klíče souboru STAVYP
dcl-ds schovaneKlice likerec(STAVYPF0: *key) end-ds;
```

Vektory

```
dcl-s vektor packed(5: 2) dim(12); // vektor s 12 pakovanými položkami
```

Výpočty

Přiřazovací příkaz

```
proměnná = výraz;
```

Výrazy

```
(a * b)  /  c             aritmetický výraz
a + b  <  c * (d - 100)   aritmeticko logický výraz
text1 +  text2            znakový výraz
```

Podmiňovací příkazy

IF - ELSE (jestliže - jinak)

```
if podmínka;
   ...  když je splněna
else;
   ...       když není splněna
endif;
```

IF - ELSEIF - ELSE (jestliže - jinak jestliže - jinak)

```
if podmínka1;
    ...      když je splněna podmínka1
elseif podmínka2;
    ...      jinak, když je splněna podmínka2
elseif podmínka3;
    ...      jinak, když je splněna podmínka3
else;
    ...      když žádná podmínka není splněna
endif;
```

SELECT (výběr)

```
select;
when podmínka1;
   ...       když je splněna podmínka1
when podmínka2;
   ...       jinak, když je splněna podmínka2
other;
   ...       když žádná podmínka není splněna
endsl;
```

Opakovací příkazy

DOW (do while)

```
dow podmínka;
   ...       když je splněna
enddo;
```

DOU (do until)

```
dou podmínka;
   ...       když není splněna
enddo;
```

FOR

```
for čítač = začátek by krok to konec;
   ...
endfor;
```

```
for idx = 1 by 1 to 10;
   Soucet += idx;
endfor;
```

Podprogramy

EXSR (execute subroutine)

```
exsr subr1; // provedení subrutiny
...

begsr subr1; // začátek subrutiny
...
exsr subr2;
...
endsr;       // konec subrutiny

begsr subr2;
...
endsr;
```

Příkazy pro soubory

```
read   jméno_souboru; // čtení dalšího záznamu

write  jméno_formátu; // zápis nového záznamu

exfmt  jméno_formátu; // zobrazení formátu na obrazovce

chain  (ZAVOD: SKLAD: MATER) STAVYP; // čtení záznamu podle klíče

delete (ZAVOD: SKLAD: MATER) STAVYP; // zrušení záznamu

update STAVYPF0 %fields(MNOZ); // přepsání záznamu v uvedeném poli

setll  (ZAVOD: SKLAD) STAVYP; // nastavení pozice v souboru
```

Ediční kódy pro výstup čísel v RPG a DDS

K výstupu číselných údajů jsou v jazyku RPG a v popisu souborů DDS zavedeny speciální jednoznakové kódy, které způsobují úpravu čísel podle předepsaného vzoru.

Jednoduché ediční kódy

Kód	*Vysvětlení*
X	zachovává vedoucí nuly a nevkládá desetinnou čárku
Z	potlačuje vedoucí nuly a nevkládá desetinnou čárku
Y	upravuje datum, přesněji, tří- až devítimístné číslo; vkládá oddělující znak podle klíčového slova DATEDIT v řídicím příkazu CTL-OPT.

Kombinační ediční kódy

Kód	***Oddělení trojic***	***Záporné znaménko***	***Tisk nuly***
1	ano	ne	ano
2	ano	ne	ne
3	ne	ne	ano
4	ne	ne	ne
A	ano	CR za	ano
B	ano	CR za	ne
C	ne	CR za	ano
D	ne	CR za	ne
J	ano	- za	ano
K	ano	- za	ne
L	ne	- za	ano
M	ne	- za	ne
N	ano	- před	ano
O	ano	- před	ne
P	ne	- před	ano
Q	ne	- před	ne

Tiskové soubory

Tiskové soubory jsou objekty typu *FILE s podtypem (atributem) PRTF. Vytvářejí se příkazem **CRTPRTF**, který buď odkazuje na popis DDS nebo ne.

Poznámka: Tiskové soubory bez popisu DDS se popisují v programu *interně* ve formuláři O, kde se popíše tvar výstupních řádků.

Tvar tiskového souboru s popisem DDS se v programu nepopisuje, soubor se jenom označí jako *externě* popsaný. Popisy výstupních záznamů se do programu převezmou při kompilaci.

Popis DDS tiskového souboru je strukturován stejně jako u databázového souboru, tj. obsahuje zápisy na úrovni souboru, na úrovni záznamu (záznamu) a na úrovni datových polí. Zápisy však mají většinou jiný význam. Popis obsahuje zpravidla několik formátů, podle složitosti výstupní sestavy.

K usnadnění práce s popisy DDS se používá progam RLU (Report Layout Utility), kterým se rozvrhují tvary tiskových záznamů na obrazovce. Spouští se příkazem STRRLU. Programem RLU lze vytvářet nové tiskové formáty i modifikovat již existující formáty. Výsledkem činnosti programu RLU je jak zdrojový popis DDS, tak objekt vzniklý jeho překladem (příkazem CRTPRTF).

Pro příklady použijeme tiskový soubor STAVYT, jehož popis DDS následuje.

```
 ************************************************************************
 *   Soubor STAVYT - Tiskový soubor pro stavy
 ************************************************************************
AAN01N02N03T.Name++++++RLen++TDpBLinPosFunctions++++++++++++++++++++++++++
A                                     REF(REFMZP)
 *   První řádek hlavičky
A          R STAVYH1                    SKIPB(1) SPACEA(2)
A                                      10'Přehled stavu zásob'
A                                      37'Datum:'
A                                     + 1DATE EDTCDE(Y)
A                                      55'Str.:'
A                                     + 1PAGNBR EDTCDE(Z)
 *   Druhý řádek hlavičky
A          R STAVYH2                    SPACEA(2)
A                                       1'Zav.'
A                                     + 2'Skl.'
A                                     + 2'Č.mat.'
A                                     +11'Množství'
 *   Detailní řádek
A          R STAVYD1                    SPACEA(1)
A            ZAVOD     R               1
A            SKLAD     R              + 4
A            MATER     R              + 4
A            MNOZ      R              + 9 EDTCDE(M)
 *   Součtový řádek 1. úrovně
A          R STAVYT1                    SPACEB(1) SPACEA(2)
A            S1             10  2      27 EDTCDE(M)
A                                     + 1'*'
 *   Součtový řádek 2. úrovně
A          R STAVYT2                    SPACEA(2)
A            S2             10  2      27 EDTCDE(M)
A                                     + 1'**'
 *   Součtový řádek "Celkem"
A          R STAVYTC                    SPACEA(0) SKIPA(1)
A            SC             10  2      27 EDTCDE(M)
A                                     + 1'***'
```

Na úrovni souboru je jen jeden zápis - zadání referenčního souboru REFMZP. Za ním následuje popis výstupních záznamů (formátů). První dva - STAVYH1 a STAVYH2 - představují dva řádky hlavičky, další představuje detailní řádek (STAVYD1), a poslední tři jsou součtové řádky (STAVYT1, STAVYT2, STAVYTC).

Pozice datových polí v záznamu lze zadávat jednak absolutně jako počáteční pozici, jednak relativně jako počet posuvů vpravo (+) nebo vlevo (-) od předchozího pole . Lze zadat i číslo řádku a sloupce, jestliže např. rozvrhujeme celou stránku jako jeden záznam.

Klíčové slovo **SKIPB** (skip before) určuje *číslo řádku*, na který se má nastavit papír před tiskem záznamu, **SKIPA** (skip after) po tisku. Klíčové slovo **SPACEB** (space before) určuje *počet řádků*, o něž se má posunout papír před tiskem záznamu, **SPACEA** (space after) po tisku.

Klíčové slovo **PAGNBR** představuje číslo stránky.

K ovládání externě popsaného tiskového souboru v programu slouží příkaz **WRITE** - zápis záznamu do tiskového souboru se zadaným jménem *formátu*.

K indikaci konce stránky je třeba zvolit jeden z obecných indikátorů (*IN01 až *IN99) a zapsat jej do popisu souboru v parametru **OFLIND**(). Tento indikátor se pak zapíná samočinně při naplnění stránky. Jeho vypnutí v pravý okamžik je však třeba zařídit v programu.

Program STATSK pro tisk jednoduché sestavy

```
**FREE

// Popisy souborů
// --------------
dcl-f STAVYP keyed;
dcl-f STAVYT printer oflind(*in55);

// Tisk hlaviček na první stránce
write STAVYH1;
write StavyH2;

// Zpracování databázového souboru
// -------------------------------
// Čtu první záznam souboru
read STAVYP;

// Zpracuji celý soubor
dow not %eof; // Cyklus do konce dat

   // Test na konec stránky. Naplní-li se stránka, vytisknou se hlavičky.
   if *in55;
      write STAVYH1;
      write STAVYH2;
      *in55 = *off;
   endif;

   // Tisknu detailní řádek.
   write STAVYD1;
   SC += MNOZ; // Ke střádači přičtu množství

   read STAVYP; // Čtu další záznam souboru

enddo; // Konec cyklu přes soubor

// Tisknu celkový součet.
write STAVYTC;

// Ukončím program s uzavřením souborů.
*inlr =*on;
```

Vysvětlivky k programu STATSK

V programu STATSK se pro jednoduchost tisknou jen hlavičky, detaily a koncový součet.

První příkaz určuje soubor STAVYP. Soubor je externě popsán a je uložen na disku - **DISK(*EXT)**, je vstupní - **USAGE(*INPUT)** implicitně a pracuje s klíčem - **KEYED**.

Druhý příkaz určuje soubor STAVYT. Říká, že soubor je tiskový a externě popsaný **PRINTER(*EXT**), že soubor je výstupní - **USAGE(*OUTPUT)**, a že pro indikaci plné stránky je použit indikátor 55 - **OFLIND(*in55)** - testuje se když tisk překročí logický konec stránky - overflow line. Indikátor 55 je zde deklarován: kompilátoru je oznámeno, že při každém naplnění stránky se má tento indikátor zapnout. Toho se využívá v programu, zejména k podmínění tisku hlaviček.

Všimněme si, že kdekoliv můžeme psát malá nebo velká písmena. Kompilátor je interně převede na velká.

Tisk do výstupního souboru se uskutečňuje příkazem **WRITE** s operandem představujícím jméno formátu z externího popisu DDS.

Příkaz **READ** přečte *další* záznam ze souboru STAVYP. Na začátku výpočtu je to první záznam (nebo nic, je-li soubor prázdný). Příkaz pro otevření souboru nepotřebujeme.

Příkaz cyklu **DOW** má v operandu podmínku, aby se tělo cyklu provádělo, dokud není konec souboru (**%EOF**). Tělo cyklu je ukončeno příkazem **ENDDO**. Uvnitř cyklu je mj. tisk hlaviček podmíněný zapnutým indikátorem 55. Po zápisu hlaviček do souboru je nutné indikátor 55 vypnout nastavením na hodnotu ***OFF**. (Zapnutí se provede hodnotou ***ON**.)

Podmiňovací příkaz **IF** má v operandu podmínku, aby se tělo příkazu provedlo, jestliže je zapnutý indikátor 55. Tělo příkazu IF je ukončeno příkazem **ENDIF**.

Výpočet se realizuje příkazem SC += MNOZ;.

Program končí zapnutím speciálního indikátoru ***INLR**, což způsobí uzavření obou souborů. Protože je to poslední příkaz, program končí.

Spuštěním programu příkazem **CALL** STATSK bychom měli dostat tiskovou sestavu podobnou této:

```
         Prehled stavu zasob        Datum:  5/13/24   Str.:    1

Zav.  Skl.  C.mat.           Mnozstvi

01    01    00001               10.00
01    01    44000                1.00
01    01    55000             1000.00
01    02    00001              714.00
01    02    00003                1.00
01    02    00004                1.00
01    02    00005                8.00
01    03    00002              350.00
02    01    00003              100.00
02    02    00004              100.00
03    01    00005                5.00
03    03    00002                1.00
03    03    00004                1.00
                               2292.00  ***

                              18059.00  ***
```

Program STATSK2 pro tisk součtované sestavy

Zde je naznačen algoritmus zpracování několikaúrovňových součtů. Počet teček před pseudopříkazy naznačuje hloubku vnoření programových "bloků" neboli hierarchickou úroveň příkazů:

Číst 1. záznam z databáze.

Schovat součtové klíče (pro zjišťování jejich změny).

Cyklus 1 - Zpracovat závod (opakovat, dokud není konec souboru):

.Tisknout hlavičky (každý závod na nové stránce).

.Cyklus 2 - Zpracovat sklad (opakovat, dokud není konec souboru a nezměnil se závod):

..Cyklus 3 - Zpracovat materiál (opakovat, dokud není konec souboru a nezměnil se sklad ani závod):

...Tisknout hlavičky (jen je-li plná stránka).

...Detailní výpočet: přičíst množství ke střádači 1. úrovně a tisknout záznam.

...Číst další záznam z databáze.

..Cyklus 3 - konec.

..Zpracovat součet za sklad (podprogram *soucetSkladu*).

.Cyklus 2 - konec.

.Zpracovat součet za závod (podprogram *soucetZavodu*).

Cyklus 1 - konec.

Tisknout součet celkem.

Ukončit výpočet.

Podprogramy *soucetSkladu* a *soucetZavodu* přičítají obsah střádače dané úrovně ke střádači vyšší úrovně, tisknou součtový řádek, čistí střádač a schovávají obsah součtového klíče pro pozdější porovnání.

```
**FREE
dcl-f STAVYP;
dcl-f STAVYT printer oflind(*in55);

dcl-ds schovaneKlice likerec(STAVYPF0: *KEY);

// Hlavní program
// --------------
read STAVYP;

schovaneKlice.ZAVOD = ZAVOD;
schovaneKlice.SKLAD = SKLAD;

dow not %eof; // Cyklus 1

   write STAVYH1;
   write STAVYH2;

   dow not %eof and ZAVOD = schovaneKlice.ZAVOD; // Cyklus 2

      dow not %eof and ZAVOD = schovaneKlice.ZAVOD
                   and SKLAD = schovaneKlice.SKLAD; // Cyklus 3
         if *in55;
            write STAVYH1;
            write STAVYH2;
            *in55 = *off;
         endif;
         write STAVYD1;
         s1 += MNOZ;

         read STAVYP;
      enddo; // konec cyklu 3
      exsr soucetSkladu;
   enddo;
   exsr soucetZavodu;
enddo;
```

```
write STAVYTC;

*inlr = *on;

// Podprogramy
// -----------

// Zpracování součtu za sklad
begsr soucetSkladu;
   S2 += S1;
   write STAVYT1;
   S1 = 0;
   schovaneKlice.SKLAD = SKLAD;
endsr;

// Zpracování součtu za závod
begsr soucetZavodu;
   SC += S2;
   write STAVYT2;
   S2 = 0;
   schovaneKlice.ZAVOD = ZAVOD;
endsr;
```

Vysvětlivky k programu STATSK2

Popisy souborů zůstávají stejné jako v předchozím příkladu, ale výpočty jsou složitější. Přibývají popisy pracovních dat.

Popisy dat

Datová struktura *schovaneKlice* je definována s klíčovým slovem **LIKEREC** na základě klíčových polí ve formátu STAVYPF0 a označena jako klíč (***KEY**). V ní budou uchovány staré hodnoty součtových klíčů k porovnání s aktuálními klíči, abychom určili, zda se určitá součtová skupina mění nebo ne.

Výpočetní příkazy

Příkaz **READ** čte záznam. Přečtená data umístí do proměnných ZAVOD, SKLAD, MATER, MNOZ, které se používají pro tisk.

Uschování součtových klíčů po přečtení prvního záznamu se uskuteční dvěma příkazy

```
schovaneKlice.ZAVOD = ZAVOD;
schovaneKlice.SKLAD = SKLAD;
```

Třetí klíč MATER nepotřebujeme.

Podobně používáme *kvalifikaci* proměnné jménem datové struktury také vpříkazu DOW k formulaci podmínky, např.

dow ... ZAVOD = schovaneKlice.ZAVOD;

Na konci těla cyklu skladu resp. závodu se vyvolává program *soucetSkladu* resp. *soucetZavodu* příkazem **EXSR** (execute subroutine).

Podprogramy (subrutiny) jsou definovány na konci programu za jeho hlavní částí. Definice podprogramu začíná příkazem **BEGSR** a končí příkazem **ENDSR**. Uvnitř mohou být libovolné příkazy, my zde máme ty, které jsou potřeba ke zpracování příslušné součtové skupiny.

Spuštěním programu příkazem **CALL** STATSK2 bychom měli dostat tiskovou sestavu podobnou této:

```
         PREHLED STAVU ZASOB        DATUM  2/20/17    STR.:    1

ZAV.  SKL.  C.MAT.           MNOZSTVI

01    01    00001              105.00
01    01    00002              400.00
01    01    00003              100.00

                               605.00  *

01    02    00001              300.00
01    02    00002              300.00
01    02    00003              100.00

                               700.00  *

                              1305.00  **

         PREHLED STAVU ZASOB        DATUM  2/20/17    STR.:    2

ZAV.  SKL.  C.MAT.           MNOZSTVI

02    01    00001              300.00
02    01    00002              300.00

                               600.00  *

02    02    00001              300.00
02    02    00002              300.00

                               600.00  *

                              1200.00  **

         PREHLED STAVU ZASOB        DATUM  2/20/17    STR.:    3

ZAV.  SKL.  C.MAT.           MNOZSTVI

01    02    90001             9999.00

                              9999.00  *

01    01    89872             5555.00

                              5555.00  *

                             15554.00  **

                             18059.00  ***


                             24988.00  ***
```

Aktualizace souboru

Program STAOBR – primárně podle obratů

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

```
**FREE
dcl-f STAVYP keyed usage(*update);
dcl-f OBRATP keyed;

read OBRATP;
dow not %eof(OBRATP);
    chain (ZAVOD: SKLAD: MATER) STAVYP;
    if %found;
        MNOZ += MNOBR;
        update STAVYPF0 %fields(MNOZ);
    endif;
    read OBRATP;
enddo;
*inlr = *on;
```

Program STAOBR2 – primárně podle stavů

Program čte pořadově záznamy souboru STAVYP od začátku do konce a u každého vyvolá podprogram, kde přičte všechny obratové položky (existují-li) k množství stavu. Po návratu z podprogramu přepíše příkazem UPDATE množství ve stavu upraveným množstvím a čte další záznam.

```
**FREE
dcl-f STAVYP keyed usage(*update);
dcl-f OBRATP keyed;

read STAVYP;
dow not %eof(STAVYP);
   exsr pricistObraty;
   update STAVYPF0 %fields(MNOZ);
   read STAVYP;
enddo;

*inlr = *on;

begsr pricistObraty;
    setll (ZAVOD: SKLAD: MATER) OBRATP;
    if %equal;
        reade (ZAVOD: SKLAD: MATER) OBRATP;
        dow not %eof(OBRATP);
            MNOZ += MNOBR;
            reade (ZAVOD: SKLAD: MATER) OBRATP;
        enddo;
    endif;
endsr;
```

Obrazovkové soubory

Obrazovkové soubory jsou objekty typu *FILE s podtypem DSPF. Vytvářejí se příkazem **CRTDSPF** s použitím popisů DDS, podobných jako u databázových souborů. Význam popisů je ovšem jiný; vyjadřuje všechny charakteristické vlastnosti obrazovkových formátů, kterých obvykle bývá v souboru několik.

K ovládání obrazovkových souborů v RPG slouží několik příkazů, z nichž nejdůležitější jsou tyto:

- WRITE – zápis formátu na obrazovku,
- READ – čtení dat zadaných z klávesnice,
- **EXFMT** – zápis formátu na obrazovku a čtení dat zadaných z klávesnice; jde o kombinaci příkazů WRITE a READ do jednoho příkazu.

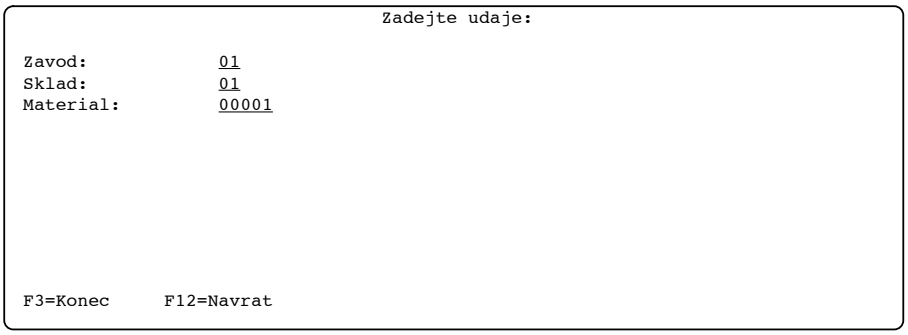
K usnadnění práce s popisy DDS se používá program SDA (Screen Design Aid), kterým se rozvrhují formáty přímo na obrazovce. Spouští se příkazem **STRSDA**. Programem SDA lze vytvářet nové obrazovkové formáty i modifikovat již existující formáty. Výsledkem činnosti programu SDA je jak zdrojový popis DDS, tak objekt vzniklý jeho překladem (příkazem CRTDSPF).

Pořízení souboru jednoduchými obrazovkami

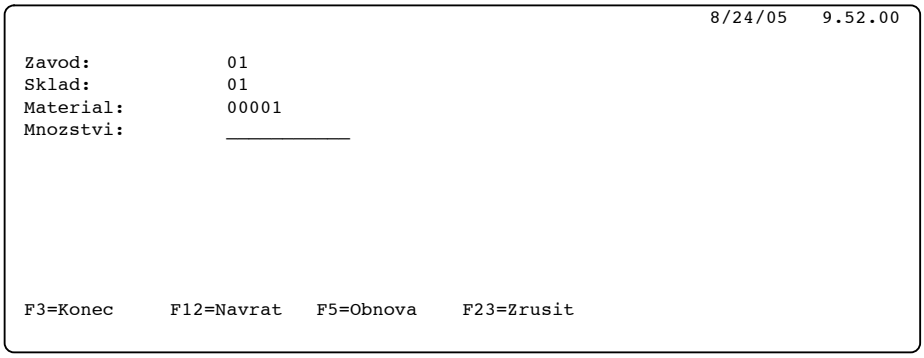
Následující příklad slouží k ilustraci interakčního způsobu programování s pomocí obrazovkového souboru. Úkolem je umožnit uživateli vytvářet, opravovat, popř. rušit záznamy databázového souboru.

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

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



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 nalezl záznam se zadaným klíčem. Podle toho zobrazí buď prázdné (nulové) množství materiálu nebo to, které je obsaženo v nalezeném záznamu.



Uživatel zapíše nové množství a stiskne klávesu Enter. Program pak buď zapíše nový záznam do souboru STAVYP nebo přepíše právě nalezený záznam novými údaji, a to opět podle toho, zda nenalezl nebo nalezl 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                                     DSPSIZ(*DS3)
A                                     REF(REFMZP)
A                                     PRINT
A                                     CA03(03 'Konec')
A                                     CA12(12 'Navrat')
 *   První format - zadani klice
A          R STAVYW01
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 format - stavova data
A          R STAVYW02
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 obrazovkového souboru STAVYW jsou definovány dva formáty: STAVYW01 a STAVYW02. První slouží k zadání klíče a druhý k zobrazení obsahu záznamu.

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

Klíčové slovo **DSPSIZ** určuje obrazovku standardní velikosti, **REF** určuje referenční soubor REFMZP, z něhož se čerpají definice polí, **PRINT** umožňuje použít klávesy 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 (z příkazu EXFMT) do programu. Program při zapnutém indikátoru 03 rozhodne, že výpočet skončí. Stejné 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.

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 (záznamu): 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 přenese 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*, *rrmmdd* 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 záznamu 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 STAPORF pro pořízení stavů

```
**********************************************************************
*   STAPORF - Stavy, porizeni - polo-volny format
**********************************************************************
*   Prepisovany (update) soubor s doplnovanim zaznamu
  dcl-f STAVYP keyed usage(*update: *delete: *output);
*   Obrazovkovy soubor
  dcl-f STAVYW workstn;

*   Datova struktura klice
  dcl-ds klicStavu likerec(STAVYPF0: *key);

*=====================================================================
*   Hlavni program
*=====================================================================

 // zobrazim zadani klice (prvni obrazovka)
 exfmt   STAVYW01;

 // cyklus 1 - dokud neni F3 ani F12
 dow   not *in03 and not *in12;

    // ctu zaznam podle klice zadaneho z obrazovky
    chain   (ZAVOD: SKLAD: MATER) STAVYP;
    // kdyz se nenajde, vymazu vstupni data pro druhou obrazovku
    if   not %found;
       clear   MNOZ;
    endif;

    // zobrazim data precteho zaznamu nebo prazdna data (druha obrazovka)
    exfmt   STAVYW02;

    // cyklus 2 - dokud neni F3 ani F12 - zpracuji druhou obrazovku
    dow   not *in03 and not *in12;
       // po F5 vycistim mnozstvi, zobrazim znovu druhou obrazovku
       if   *in05;
          clear   MNOZ;
          exfmt   STAVYW02;
          iter; // opakuji cyklus 2
       endif;

       // kdyz se zaznam nenasel a nebyl zadan vymaz zaznamu, zapisu novy zaznam
       if not %found and not *in23;
          write   STAVYPF0;
       // kdyz se zaznam nasel a nebyl zadan vymaz, prepisu zaznam
```

```
      elseif %found() and not *in23;
         update   STAVYPF0 %fields (MNOZ);
      // kdyz se zaznam nasel a byl zadan vymaz, vymazu zaznam a vycistim
      // mnozstvi
      elseif   %found(STAVYP) and *in23;
         klicStavu.ZAVOD = ZAVOD ;
         klicStavu.SKLAD = SKLAD ;
         klicStavu.MATER = MATER ;
         delete %kds(klicStavu) STAVYP;
         clear   STAVYW01;
      endif;
      // koncim cyklus 2 - zobrazim prvni obrazovku
      leave;

   // konec cyklu 2
   enddo;

   // neni-li F3 - zobrazim zadani klice (prvni obrazovka)
   if   not *in03;
      exfmt     STAVYW01;
   // jinak koncim cyklus 1 i program
   else;
      leave;
   endif;

// konec cyklu 1
enddo;

// ukoncim vypocet
*inlr = *on;
return;
```

Vysvětlivky k programu STAPORF

Program je napsán v polovolném formátu, který dovoluje zápis hvězdičkových komentářů, ale i formulářových příkazů (které zde ale nepoužíváme).

Popis souborů

První příkaz určuje databázový soubor STAVYP. Říká, že soubor dovoluje přepisovat záznamy (***UPDATE**), zapisovat nové záznamy (***OUTPUT**), rušit záznamy (***DELETE**) a že je zpracováván podle klíče (**KEYED**).

Druhý příkaz určuje obrazovkový soubor STAVYW. Soubor je výstupní a vstupní: formát se nejdříve zobrazí na obrazovce a pak se čtou vstupní data. (Naproti tomu přepisovaný databázový soubor je vstupní a výstupní - záznam je třeba nejprve přečíst a pak jej teprve lze přepsat.). Soubor je realizován obrazovkovým zařízením **WORKSTN** (work station).

Popis dat

Datová struktura *klicStavu* je definována stejně jako klíč souboru STAVYP ve formátu STAVYPF0.

Výpočty

Program je organizován ve dvou úrovních cyklů, který se opakuje tak dlouho, dokud uživatel nestiskne klávesu F3 nebo F12.

První příkaz **EXFMT** (execute format) zobrazuje formát STAVYW01 s výzvou k zadání klíče a čeká na vstup z klávesnice. Jde vlastně o sloučení dvou příkazů WRITE a READ do jednoho. Jakmile uživatel stiskne jednu z povolených kláves (F3, F12, Enter), příkaz EXFMT skončí a výpočet pokračuje testem na indikátory 03 a 12: Je-li jeden z nich zapnut, skončí vnější cyklus a také výpočet. Jinak je zřejmé, že uživatel stiskl klávesu Enter, takže program pokračuje dále.

Příkaz **CHAIN** čte záznam v databázovém souboru podle vyhledávacího argumentu - klíče.

Příkaz **CLEAR** nuluje pole MNOZ. Provádí se jen tehdy, jestliže se záznam v souboru STAVYP nenalezl. Tím připraví pole MNOZ k zobrazení na obrazovce pro zadání nového záznamu.

Následuje zobrazení druhého formátu STAVYW02 příkazem EXFMT . Po skončení příkazu EXFMT se opět nejprve testují funkční klávesy kromě F23 (zrušení záznamu). Heslovitě:

F3 - indikátor 03 - konec výpočtu,

F12 - indikátor 12 - návrat k zobrazení prvního formátu (zadání klíče),

F5 - indikátor 05 - smazat množství a opakovat zobrazení druhého formátu.

Po stisku klávesy F23 nebo Enter se pokračuje dále. Rozliší se tři případy podle kombinace podmínek:

- záznam se v souboru nenalezl a nebylo zadáno zrušení (NOT *IN23): nový záznam se zapíše do souboru příkazem **WRITE**,
- záznam se v souboru nalezl a nebylo zadáno zrušení (NOT *IN23): příkaz **UPDATE** přepíše pole MNOZ v nalezeném záznamu,
- záznam se nalezl a bylo zadáno zrušení (*IN23): záznam se zruší příkazem **DELETE** a pole obrazovky se smažou příkazem CLEAR, v němž je zadáno jméno prvního formátu.

Na konci výpočtu se zapne indikátor *INLR, což má za následek uzavření všech souborů a zrušení všech proměnných při návratu do volajícího programu (na příkazový řádek). Při příštím volání programu STAPOR se znovu otevřou soubory a proměnné se vyčistí. Kdyby při návratu nebyl indikátor LR zapnutý, zůstaly by soubory otevřené a proměnné nezměněné až do příštího volání nebo do skončení úlohy (jobu). Příkaz **RETURN** způsobí okamžitý návrat do volajícího programu.

Cvičení - Obyčejné obrazovky

Napište program pro opravování a doplňování záznamů databázového souboru CENIKP.

Postup:

1. Vytvořte popis DDS databázového souboru CENIKP s datovými poli MATER, CENA, NAZEV, přičemž jméno záznamu je CENIKPF0, klíčem (jednoznačným) je pole MATER a vlastnosti polí se přebírají z referenčního souboru REFMZP.

2. Navrhnete dva obrazovkové formáty a vytvořte popis DDS pro obrazovkový soubor CENIKW. První formát, CENIKW01, bude sloužit k zadání klíče (čísla materiálu), druhý, CENIKW02, k zobrazení databázového záznamu a k zápisu dat z klávesnice.

Klávesa F3 bude ukončovat program v obou formátech.

Klávesa Enter v prvním formátu způsobí vyhledání databázového záznamu podle zadaného klíče a jeho zobrazení druhým formátem. V druhém formátu klávesa Enter potvrdí změny dat.

Klávesa F6 (jen v prvním formátu) bude určena k vyvolání druhého formátu s prázdnými poli, do nichž uživatel zapíše data nového záznamu (a pak stiskne Enter).

Klávesa F12 (jen v druhém formátu) způsobí návrat k prvnímu formátu.

3. Vytvořte program CENPOR, který zobrazením prvního formátu umožní uživateli zadat číslo materiálu nebo stisknout klávesu F6.

Jestliže se po stisku klávesy Enter materiál v databázi nenajde, ohlásí program chybu (tak, že se vrátí na zobrazení prvního formátu se zapnutým chybovým indikátorem). Najde-li se materiál v

databázi, zobrazí program záznam druhým formátem, uživatel změní data a stiskne Enter. Program pak opraví záznam v databázi a zobrazí první formát.

Stiskne-li uživatel v prvním formátu klávesu F6, zobrazí program druhý formát s prázdnými poli a uživatel vyplní všechna data nového záznamu (včetně klíče). Po stisku Enter program zjistí, zda záznam se zadaným klíčem již v databázi je. V tom případě zobrazí znovu druhý formát s indikací chyby a očekává (opravená) vstupní data. Jestliže záznam se zadaným klíčem v databázi není, program jej tam zapíše a vrátí se na zobrazení prvního formátu.

Poznámka 1: Indikací chyby se rozumí zapnutí zvoleného indikátoru a přechod na nové zobrazení toho formátu, kde chyba vznikla. V popisu DDS si proto musíme tímto indikátorem podmínit chybovou zprávu, (např. ve formě obyčejné konstanty) umístěnou ve vhodném řádku a sloupci tak, aby se nepřekrývala s jinými údaji na obrazovce. Například v prvním formátu bychom zařadili konstantu podmíněnou zapnutým indikátorem 81 (nenalezený záznam) a ve druhém formátu konstantu podmíněnou indikátorem 82 (duplicitní záznam). Zapnutí či vypnutí indikátorů musíme ovšem provést v programu.

```
AAN01N02N03T.Name++++++RLen++TDpBLinPosFunctions++++++++++++++++++++++++++
A          R CENIKW01
 ...
A                                    5  2'Materiál:'
A            MATER     R        B  5 20
A  81                                8 20'Materiál není v ceníku'
 ...
```

Poznámka 2: Při zápisu nového záznamu do souboru můžeme postupovat dvěma způsoby:

1. Příkazem

```
setll (MATER) CENIKP;
```

zkusíme vyhledat záznam v souboru (funguje stejně jako příkaz CHAIN, ale nepřečte žádná data) a pak se zeptat příkazem

```
if %equal;
```

zda v souboru již existuje.

2. V příkazu WRITE zapsat modifikátor E (test na chybu) a pak testovat funkci %ERROR:

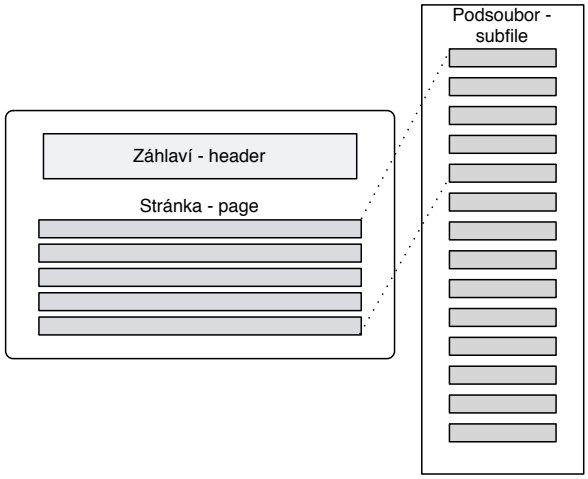
```
write(e) CENIKPF0;
if %error;
   *in82 = *on;
...
```

Podsoubory obrazovkových souborů (subfiles)

Podsoubory představují speciální mechanismus pro obrazovkové 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 záznamů z databázového souboru (proto "podsoubor"). Protože podsoubor patří neoddělitelně k obrazovkovému souboru, nebo spíše k 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 obrazovkové formáty*: *datový a řídicí*. Datový formát popisuje strukturu záznamu 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*. Záznamy podsouboru jsou číslovány pořadovými čísly, proto příkaz WRITE zapisuje novoý 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 záznamy v podsouboru uloženy, lze je zobrazit ve formě obrazovkových stránek příkazem **EXFMT** (ale také příkazem WRITE) se jménem *řídicího formátu*. Podsoubor obsahuje zpravidla více záznamů, než se jich vejde na jednu stránku obrazovky, proto je třeba umožnit listování v podsouboru. Uživatel listuje v podsouboru pomocí stránkovacích kláves (Page down, Page up). Výměnu stránek obstarává počítač v rámci provádění příkazu EXFMT automaticky, takže se jím nemusíme v programu zabývat. Stránkování lze však řídit programem, zadáme-li příslušná klíčová slova v popisu řídicího formátu.

Datový formát musí být označen klíčovým slovem **SFL**.

Řídicí formát musí obsahovat několik klíčových slov a může popisovat záhlaví obrazovkové stránky. Nejdůležitější klíčová slova řídicího formátu jsou tato:

SFLCTL - odkazuje na datový formát,

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

SFLPAG - určuje velikost obrazovkové stránky v počtu řádků,

SFLDSP - umožňuje zobrazit stránku podsouboru (je-li zapnut případný podmínkový indikátor),

SFLDSPCTL - umožňuje zobrazit záhlaví stránky (je-li zapnut případný podmínkový indikátor),

SFLEND - umožňuje zobrazit znaménko + na konci stránky, není-li to stránka poslední,

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

Obrazovkový soubor je v programu označen stejně jako při použití obyčejných obrazovkových formátů, ale do popisu souboru je nutno zadat klíčové slovo **SFILE**, v němž je zadáno, že se používá podsoubor; k tomu se deklaruje jméno použitého datového formátu a jméno číselné proměnné sloužící k čítání záznamů podsouboru.

Zobrazení databázového souboru s použitím podsouboru

Pro první seznámení s programováním podsouboru zvolíme velmi prostou úlohu, a sice zobrazení databázového souboru STAVYP ve formě seznamu bez možnosti cokoliv v něm měnit.

Obrazovkový soubor STAVYW2 obsahuje dva formáty: datový formát DATA a řídicí formát RIDICI.

```
AAN01N02N03T.Name++++++RLen++TDpBLinPosFunctions++++++++++++++++++++++++++
A                                     DSPSIZ(*DS3)
A                                     REF(REFMZP)
A                                     PRINT
A                                     CA03(03 'Konec')
 *   Datovy format podsouboru
A          R DATA                     SFL
A            ZAVOD     R        O  7  3
A            SKLAD     R        O  7 10
A            MATER     R        O  7 17
A            MNOZ      R        O  7 30EDTCDE(K)
 *   Ridici format podsouboru - zahlavi
A          R RIDICI                   SFLCTL(DATA)
A                                     SFLSIZ(0100)
A                                     SFLPAG(0005)
A  51                                 SFLDSP
A                                     SFLDSPCTL
A  51                                 SFLEND
A                                   2 18'Prehled stavu'
A                                   4  3'Zavod'
A                                   4 10'Sklad'
A                                   4 17'Material'
A                                   4 35'Mnozstvi'
```

```
                Prehled stavu

 Zavod  Sklad  Material          Mnozstvi


 01     01     44001                1.00
 01     01     44002                1.00
 01     02     55001                1.00
 01     02     55002                1.00
 02     01     00005              555.00                                  +
```

Na začátku externího popisu DDS jsou uvedeny zápisy platné pro celý obrazovkový soubor, tj. pro oba formáty společně. Klíčové slovo DSPSIZ určuje obrazovku standardní velikosti, REF určuje referenční soubor REFMZP, z něhož se čerpají definice polí, PRINT umožňuje použít klávesu Print k získání otisku obrazovky a CA03 umožňuje použít klávesu F3.

V datovém formátu DATA je zapsáno klíčové slovo SFL na úrovni popisu záznamu. Následují popisy výstupních datových polí, která se mají zobrazit na 7. řádku (a na následujících čtyřech řádcích obrazovkové stránky). Pole MNOZ je na výstupu upraveno edičním kódem K.

V řídicím formátu RIDICI jsou na úrovni záznamu zapsána klíčová slova začínající písmeny SFL:

SFLCTL(DATA) odkazuje na datový formát,

SFLSIZ(0100) vyhrazuje podsouboru prostor pro 100 záznamů,

SFLPAG(0005) říká, že stránka podsouboru má 5 řádků (tj. řádky 7, 8, 9, 10, 11),

SFLDSP zobrazí stránku podsouboru, je-li zapnut indikátor 51,

SFLDSPCTL zobrazí záhlaví podsouboru

SFLEND zobrazí znaménko + na konci stránky, je-li zapnut indikátor 51 a následuje-li ještě další stránka.

Za klíčovými slovy následují popisy konstant tvořících záhlaví podsouboru.

Program STAPSZOB přečte všechny záznamy z databázového souboru STAVYP a zapíše je do podsouboru DATA obrazovkového souboru STAVYW2. Pak zobrazí první stránku podsouboru a čeká na uživatele v příkazu EXFMT. Uživatel může listovat v podsouboru klávesami Page up a Page down. Po stisku klávesy Enter nebo F3 program skončí.

```
**FREE
// ************************************************************************
//  STAPSZOB - Zobrazeni stavu podsouborem
// ************************************************************************
// ========================================================================
//  Popis souboru
// ========================================================================

//  databazovy soubor - jen vstupni
dcl-f STAVYP keyed;

//  obrazovkovy soubor s definici podsouboru
dcl-f STAVYW2 workstn sfile(DATA: idx);

// ========================================================================
//  Definice dat
// ========================================================================

dcl-s idx int(10);

// ========================================================================
//  Hlavni program
// ========================================================================

// naplnim podsoubor zaznamy z databaze
read STAVYP;
dow not %eof;
    idx += 1;
    write DATA; // zapise novy zaznam do podsouboru podle poradoveho cisla
    read STAVYP;
enddo;

// umoznim zobrazeni obrazovkove stranky
if idx > 0;
    *in51 = *on;
else;
    *in51 = *off;
endif;

// zobrazim prvni stranku podsouboru
exfmt RIDICI;

// koncim vypocet
*inlr = *on;
return;
```

Vysvětlivky k programu STAPSZOB

Popis souborů

Soubor STAVYP je vstupní (není zadáno jinak) a je zpracováván podle klíče (KEYED).

Soubor STAVYW2 je realizován obrazovkovým zařízením WORKSTN (work station). Příkaz obsahuje klíčové slovo SFILE, kde slovo DATA oznamuje jméno datového formátu tohoto podsouboru a kde je deklarována proměnná IDX určená k čítání pořadových čísel záznamů podsouboru. Podsouborů by mohlo být v obrazovkovém souboru definováno několik. Pro každý z

nich by ovšem musela být zapsána další dvojice formátů v popisu DDS a další specifikace SFILE v popisu souboru.

Popis dat

Proměnná IDX slouží k čítání záznamů podsouboru a je definována jako celočíselná, typu INTEGER o délce 4 bajty, tj. maximálně 10 dekadických číslic. Počet desetinných míst musí být 0.

Výpočty

Nejprve se přečte první záznam databázového souboru STAVYP příkazem READ. Pak následuje cyklický výpočet (od příkazu DOW do příkazu ENDDO). Příkaz DOW testuje výsledek čtení; dokud není konec vstupního souboru, provádí se tělo cyklu, jinak se končí.

V těle cyklu se nejprve zvýší číslo záznamu o 1, aby se zapsal další záznam do podsouboru, načež se záznam zapíše příkazem WRITE (s fomátem DATA). Pak se čte další záznam z databáze a postup se opakuje.

Když se podsoubor naplní (po vyčerpání databázového souboru), zapneme indikátor 51, který umožňuje zobrazit stránku podsouboru. Je-li soubor STAVYP prázdný, nezapíše se do podsouboru nic a indikátor 51 se vypne. Kdyby v tom případě zůstal zapnutý, program by havaroval na nulovém pořadovém čísle podsouboru.

Příkazem EXFMT (s formátem RIDICI) zobrazíme záhlaví i stránku obrazovky. Uživatel může listovat v obrazovkových stránkách. Nakonec stiskne klávesu Enter nebo F3 a tím ukončí příkaz EXFMT, čímž ukončí výpočet.

Opravy databázového souboru s použitím podsouboru

S pomocí podsouborů lze data nejen zobrazovat, ale i opravovat a doplňovat. Prozatím se budeme věnovat jen *opravám existujících záznamů*. Datový formát podsouboru (SFL) obsahuje obousměrná pole odpovídající těm polím v databázovém záznamju, která dovolíme opravovat. V našem příkladu to bude pouze pole MNOZ. K tomu budeme potřebovat nejen zapsat data do záznamů podsouboru a zobrazit je, ale i přečíst 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

READC - 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 READC rozeznává změněné záznamy od nezměněných a zapisuje jejich pořadové číslo do proměnné (v našem přikladu IDX).

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

Příkaz READC 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. V popisu řídicího formátu bude indikátorem 81 podmíněno klíčové slovo

SFLMSG - chybová zpráva pro záznam podsouboru.

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 **UPDATE** - přepis záznamu podsouboru, který přepíše záznam podsouboru. Operandem příkazu UPDATE bude tedy jméno datového formátu podsouboru (DATA). Do záznamu podsouboru se tak dostanou kromě změněných dat a příznaku změny také hodnoty indikátoru 80 (ať už zapnuté nebo vypnuté). Při příštím zobrazení se takový záznam objeví s inverzně zvýrazněným množstvím a příkaz **READC** 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.

Klíčové slovo
OVERLAY je uvedeno proto, aby při zobrazení řídicího formátu nebyl přepsán formát NAVOD, který byl předtím zobrazen příkazem WRITE.

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

SFLRCDNBR(CURSOR). 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 příštím zobrazení.

Obrazovkový soubor STAVYW3 obsahuje tři formáty: datový formát DATA, řídicí formát RIDICI a obyčejný formát NAVOD.

Obrazovkový soubor STAVYW3 pro opravu stavů

```
 ************************************************************
 *   Soubor STAVYW3 - Obrazovkovy soubor pro opravu stavu
 ************************************************************
AAN01N02N03T.Name++++++RLen++TDpBLinPosFunctions++++++++++++++++++++++++++
A                                       DSPSIZ(*DS3)
A                                       REF(REFMZP)
A                                       PRINT
A                                       CA03(03 'Konec')
 *   Datovy format 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 format podsouboru - zahlavi
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(CURSOR)
A                                  2 18'Prehled stavu'
A                                  4  3'Zavod'
A                                  4 10'Sklad'
A                                  4 17'Material'
A                                  4 35'Mnozstvi'
 *   Navod k pouziti funkcnich klaves
A          R NAVOD
A                                 23  3'F3=Konec'
```

Poznámka: Přestože pole ZAVOD, SKLAD, MATER jsou ve formátu DATA označena jako čistě výstupní (O), příkaz READC jejich hodnoty přečte zároveň s hodnotou pole MNOZ a hodnotou indikátoru 80.

Program STAPSOPR pro opravu stavů

```
**FREE
dcl-f STAVYP keyed usage(*update);
dcl-f STAVYW3 workstn sfile(DATA: idx);

dcl-s idx   packed(3);
dcl-s pocet like(idx);
dcl-s mnoz2 like(MNOZ);

// Naplnim podsoubor
exsr plnitPods;
// Nastavim ukazatel na 1. zaznam podsouboru
KURZOR = 1;
// Cyklus 1 - zpracovat podsoubor (do stisku klavesy F3)
dou *in03;
    // Zobrazit stranku podsouboru
    exsr zobrPods;
```

```
    // Prazdny podsoubor - koncim
    if *in51;
        // Ctu prvni zmeneny zaznam podsouboru
        readc DATA;
        // Cyklus 2 - zpracovat zmenene zaznamy
        dow not %eof;
            // Nastavim ukazatel na prave zpracovavany zaznam podsouboru
            KURZOR = idx;
            // Test dat (pole MNOZ). Pri chybe se zapne ind. 80 a 81
            exsr testDat;
            // Prepat zaznam podsouboru, i kdyz je chyba v datech
            // (zapise se tam i hodnota indikatoru 80 a MDT)
            update DATA;
            // Neni-li chyba, prectu zaznam z databaze a prepisu jej
            if not *in80;
                // Schovam pole MNOZ, ktere se prepise nasledujicim prikazem CHAIN
                mnoz2 = MNOZ;
                // Ctu zaznam z databaze podle klice prevzateho z dat podsouboru
                chain (ZAVOD: SKLAD: MATER) STAVYP;
                // Obnovim pole MNOZ pro prepis zaznamu v databazi
                MNOZ = mnoz2;
                // Prepisu zaznam v databazi
                if %found(STAVYP);
                    update STAVYPF0 %fields(MNOZ);
                endif;
            endif;
            // Ctu dalsi zmeneny zaznam podsoubou
            readc DATA;
        // Cyklus 2 - konec
        enddo;
        // Konec podminky "zobrazeni stranky povoleno"
    endif;
// Cyklus 1 - konec
enddo;
// Ukonceni vypoctu
*inlr = *on;
return;

// Plneni podsouboru
// -----------------
begsr plnitPods;
read STAVYP;
dow not %eof;
    idx += 1;   // citac + 1
    write DATA; // zapis do SFL
    read STAVYP;
enddo;
pocet = idx; // pocet zaznamu v SFL
endsr;

// Zobrazeni podsouboru
// --------------------
begsr zobrPods;
write NAVOD;
if pocet > 0;
   *in51 = *on; // pocet>0, povol SFL
else;
   *in51 = *off;
endif;
exfmt RIDICI;
*in81 = *off;
endsr;

// Test spravnosti dat
// -------------------
begsr testDat;
if MNOZ > 1000;
    *in80 = *on; // chyba
    *in81 = *on;  // zapnu glob. chybu
else;
   *in80 = *off;
```

```
endif;
endsr;
```

Vysvětlivky k programu STAPSOPR

Popis souborů

Soubor STAVYP je nyní označen jako přepisovaný (*UPDATE).

Popis dat

Proměnná POCET je definována pomocí klíčového slova LIKE. Má stejný typ a délku jako proměnná IDX.

Proměnná **mnoz2** je definována pomocí klíčového slova LIKE. Má stejný typ a délku jako proměnná MNOZ. Použije se pro úschovu hodnoty pole MNOZ pro přepis v databázi.

Výpočet

Program je členěn na hlavní část a tři podprogramy:

plnitPods - plnění podsouboru databázovými záznamy,

zobrPods - zobrazení podsouboru (záhlaví a seznamu),

testDat - test na správnost vstupních dat.

Hlavní program začíná naplněním podsouboru (podprogram *plnitPods*) a nastavením proměnné KURZOR na první záznam podsouboru. Pak vstoupí do cyklu zpracování podsouboru, který může být ukončen jen stiskem klávesy F3.

Uvnitř cyklu se provede podprogram *zobrPods*, který příkazem EXFMT zobrazí tu stránku podsouboru, která obsahuje záznam s naposledy nastaveným číslem v proměnné KURZOR. Zde uživatel stránkuje a vyplňuje vstupní údaje z klávesnice. Jakmile stiskne klávesu Enter nebo F3, výpočet se vrátí z podprogramu zpět.

Program pak přečte první změněný záznam podsouboru a jeho pořadové číslo IDX příkazem READC a vstoupí do cyklického zpracování změněných záznamů příkazem DOW, které skončí, když již není k dispozici žádný další změněný záznam a všechny chyby vstupních dat jsou odstraněny (v tom okamžiku funkce %EOF vrátí hodnotu *ON).

Při zpracování změněného záznamu se poloha kurzoru KURZOR nastaví na číslo právě zpracovávaného záznamu, testuje se správnost vstupních dat (podprogram *testDat* nastavuje indikártory 80 a 81) a přepíše se záznam podsouboru (UPDATE). Tím se do záznamu podsouboru dostanou hodnoty chybového indikátoru 80 a příznaku modifikace MDT (jsou-li vstupní data správná, jsou oba vypnuté, jsou-li chybná, jsou oba zapnuté).

Byly-li vstupní údaje v podprogramu *testDat* správné, provede se podprogram pro opravu odpovídající databázového záznamu. Tam se nejprve uschovají údaje, které podléhají změně (u nás pole MNOZ do proměnné *mnoz2*) a přečte se databázový záznam podle klíče příkazem CHAIN. Pole, která byla změněna z klávesnice (jen MNOZ), jsou tak přepsána údaji z databáze, proto je musíme obnovit. Pak teprve příkaz UPDATE přepíše databázový záznam.

V programu není ošetřen případ, že příkaz CHAIN nenalezl databázový záznam odpovídající záznamu podsouboru. K tomu může dojít tehdy, když uživatel prohlížel seznam záznamů podsouboru a jiný uživatel mezitím dotyčnou databázový záznam zrušil. Pak by příkaz UPDATE způsobil abnormální ukončení programu.

V cyklu se přečte další změněný záznam podsouboru a cyklus se opakuje.

Podprogram *zobrPods* povoluje zobrazení dat podsouboru, jen když počet záznamů získaných z databáze není nulový. Jinak by totiž program havaroval. Vypíná také globální indikátor 81 chyby vstupních dat.

Podprogram *testDat* kontroluje údaj množství, zda je větší než 1000; v tom případě zapne indikátory 80 a 81. Indikátor 80 podmiňuje klíčové slovo SFLNXTCHG a zobrazovací atributy pole MNOZ ve formátu DATA. Indikátor 81 podmiňuje klíčové slovo SFLMSG s chybovou zprávou ve formátu RIDICI. Zatímco indikátor 80 se může v podporgramu *testDat* i vypnout, indikátor 81 se může nanejvýš zapnout. Indikátor 81 vyjadřuje skutečnost, že došlo aspoň k jedné chybě v množství MNOZ během analytického cyklu s příkazem READC. Indikátor se vypíná v podprogramu *zobrPods*, ihned po skončení příkazu EXFMT, kdy už není potřeba k aktivaci klíčového slova SFLMSG.

Poznámka: Je-li podsoubor prázdný, nesmí být zapnutý indikátor povolující zobrazit stránku, ani nesmí být vydán příkaz READC, který by havaroval. Příčiny chyb podsouboru se těžko zjišťují, proto je lépe chybám předejít tak, jak to je v příkladu (zapínání a testování indikátoru 51).

Údržba databáze s použitím podsouboru

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 záznamu, 4 - zrušení záznamu.

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 (OKNONOVA) 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 OKNOOPR nebo OKNONOVA, 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 soubor STAVYW5 obsahuje tyto formáty:

- datový formát podsouboru DATA,
- řídicí formát podsouboru RIDICI,
- okenní formát OKNOOPR pro opravu nebo smazání databázového záznamu,
- okenní formát OKNONOVA vložení nového databázového záznamu,
- obyčejný formát NAVOD.

Novým prvkem v řídicím formátu RIDICI je klíčové slovo

SFLCLR - čištění podsouboru,

podmíněné indikátorem 56. Slouží k vyčištění podsouboru před jeho novým naplněním.

Obrazovkový soubor STAVYW5 pro údržbu stavů

```
     *****************************************************************
     *   Soubor STAVYW5 - Obrazovkovy soubor pro udrzbu stavu
     *****************************************************************
A                                       DSPSIZ(*DS3)
A                                       REF(REFMZP)
A                                       CA03(03 'Konec')
     *   Datovy format podsouboru
A          R DATA                       SFL
A  80                                   SFLNXTCHG
A            VOLBA          1A  B  8  3
A  80                                   DSPATR(RI)
A            ZAVOD     R        O  8  8
A            SKLAD     R        O  8 14
A            MATER     R        O  8 21
A            MNOZ      R        H
     *   Ridici format podsouboru
A          R RIDICI                     SFLCTL(DATA)
A                                       SFLSIZ(100)
A                                       SFLPAG(10)
A  51                                   SFLDSP
A                                       SFLDSPCTL
A  51                                   SFLEND
A  56                                   SFLCLR
A                                       CA05(05 'Obnova')
A                                       CA06(06 'Nova veta')
A  81                                   SFLMSG('Chybna volba')
A                                       OVERLAY
A            KURZOR         4  0H      SFLRCDNBR(CURSOR)
A                                  2  9'Prehled stavu'
A                                  4  3'Volba: 2=Zmena 4=Zruseni'
A                                  6  7'Zavod'
A                                  6 13'Sklad'
A                                  6 20'Material'
     *   Format okna pro opravu a ruseni
A          R OKNOOPR
A                                       CF12(12 'Navrat')
     *                           (Horni r. Levy sl. Pocet r. Pocet sl.)
A                                       WINDOW(07 30 06 40)
     *                           (Pozice vztazny k hornimu levemu rohu
A                                  1  4'Zavod'
A                                  1 11'Sklad'
A                                  1 18'Material'
A                                  1 28'Mnozstvi'
A            ZAVOD     R        O  2  4
A            SKLAD     R        O  2 11
A            MATER     R        O  2 18
A  82
AO 83                                   DSPATR(RI)
A            MNOZ      R        B  2 26EDTCDE(K)
A                                  4  2'F3=Konec'
A                                  4 14'F12=Navrat'
A  82                              5  2'Material neni ve stavu'
A  83                              5  2'Material neexistuje'
     *   Format okna pro novou vetu
A          R OKNONOVA
A                                       CF12(12 'Navrat')
     *                           (Horni r. Levy sl. Pocet r. Pocet sl.)
A                                       WINDOW(07 30 06 40)
     *                           (Pozice vztazny k hornimu levemu rohu
A                                  1  4'Zavod'
A                                  1 11'Sklad'
A                                  1 18'Material'
A                                  1 28'Mnozstvi'
A            ZAVOD     R        B  2  4
A            SKLAD     R        B  2 11
A            MATER     R        B  2 18
A  84                                   DSPATR(RI)
A            MNOZ      R        B  2 26EDTCDE(K)
A                                  4  2'F3=Konec'
```

```
A                                4 14'F12=Navrat'
A  84                            5  2'Material je uz ve stavu'
A          R NAVOD
A                               23  3'F3=Konec'
A                               23 15'F6=Nova veta'
A                               23 30'F5=Obnova'
```

```
      Prehled stavu

Volba: 2=Zmena 4=Zruseni

    Zavod Sklad  Material
                          ..........................................
2    01    01     44001    :    Zavod  Sklad  Material  Mnozstvi      :
_    01    01     44002    :    01     01     44001            1.00   :
_    01    02     55001    :                                          :
_    01    02     55002    :  F3=Konec    F12=Navrat                  :
_    02    01     00005    :                                          :
_    02    02     00006    :..........................................:
_    03    01     00001
_    03    02     00001
_    03    02     00002
_    03    02     00003                                               +




F3=Konec    F6=Nova veta   F5=Obnova
```

Program STAPSUDR pro údržbu databázového souboru

```
     ****************************************************************
     *   STAPSUDR - Udrzba stavu
     ****************************************************************
     *================================================================
     *   Popis souboru
     *================================================================
    fSTAVYP    uf a e           k disk
    fSTAVYW5   cf   e             workstn sfile(DATA: idx)

      //================================================================
      //   definice dat
      //================================================================
    d dataStavuI      ds                  likerec(STAVYPF0: *input)
    d dataStavuO      ds                  likerec(STAVYPF0: *output)
    d klicStavu       ds                  likerec(STAVYPF0: *key)
    d idx             s              3p 0
    d pocet           s                   like(idx)
    d oprava          c                   const('2')
    d zruseni         c                   const('4')

     *================================================================
     *   Hlavni program
     *================================================================

      exsr   plnitPods;   // Naplnim podsoubor poprve
      KURZOR = 1;  // Nastavim ukazatel na 1. zaznam podsouboru

      dou   *in03;        // Cyklus 1 - zpracovat podsoubor (do stisku klavesy F3)
         exsr   zobrPods;    // Zobrazit stranku podsouboru
         if   *in05;         // Po F5 obnovim obsah podsouboru a pokracuji
```

```
      exsr   plnitPods;
   elseif   *in06;     // Po F6 zpracuji vlozeni noveho zaznamu do souboru a opakuji cyklus 1
      exsr   novyZaznam;
      iter;
   endif;
   if   *in51;         // Neni-li podsoubor prazdny - zpracuji data podsouboru
      readc   DATA;       // Ctu prvni zmeneny zaznam podsouboru

      dow   not %eof;     // Cyklus 2 - zpracovat zmenene zaznamy
         KURZOR = idx;     // Nastavim ukazatel na prave zpracovavany zaznam podsouboru
         exsr   testVolby;   // Test volby. Pri chybe se zapne ind. 80 a 81
         if   not *in80;     // Neni-li chyba, zpracuji volbu (2, 4)
            if   VOLBA = oprava;
               exsr   opravaZaznamu;
            elseif   VOLBA = zruseni;
               exsr   zruseniZaznamu;
            endif;
            clear   VOLBA;      // Vycistim volbu pro pristi zobrazeni
         endif;
         // Prepisu zaznam podsouboru, i kdyz je chyba
         // (zapise se tam i hodnota indikatoru 80 a MDT)
         update   DATA;
         readc   DATA;    // Ctu dalsi zmeneny zaznam podsoubou
      enddo;            // Cyklus 2 - konec
   endif;            // Konec podminky *in51 "zobrazeni stranky povoleno"
enddo;       // Cyklus 1 - konec

// Ukonceni vypoctu
*inlr = *on;
return;

// ================================================================
//   Podprogramy - subroutines
// ================================================================

//  -----------------
//  Plneni podsouboru
//  -----------------
begsr   plnitPods;
   *in56 = *on;    // Vymazu podsoubor
   write   RIDICI;
   *in56 = *off;

   idx = 0 ;       // Nastavim pocatecni hodnoty
   VOLBA = *blank;
   setll   *start STAVYP;  // Nastavim ukazatel na zacatek souboru
   read(n)   STAVYP;    // Ctu prvni zaznam souboru

   dow   not %eof ;  // Naplnim podsoubor daty ze souboru
      idx += 1 ;      // Zvysim citac zaznamu
      write   DATA ;    // Zapisu zaznam do podsouboru
      read(n)   STAVYP;    // Ctu dalsi zaznam podsouboru
   enddo ;

   pocet = idx;    // Zapamatuji si pocet zaznamu podsouboru
   KURZOR = 1;     // Nastavim cislo 1. zaznamu pro pristi zobrazeni
endsr;

//  --------------------
//  Zobrazeni podsouboru
//  --------------------
begsr   zobrPods ;
   write   NAVOD ;    // Zobrazim navod na pouziti funkcnich klaves
   if   pocet > 0 ;   // Je-li v podsouboru aspon 1 zaznam, povolim zobrazeni stranky
      *in51 = *on ;
   else ;             // jinak to zakazu
      *in51 = *off ;
   endif ;
   exfmt   RIDICI ;   // Zobrazim zahlavi a stranku a cekam na data z klavesnice
   *in81 = *off ;     // Vypnu globalni chybu
endsr ;
```

```
//  -------------------
//  Test spravnosti dat
//  -------------------
begsr   testVolby ;
   if   VOLBA = '2' or VOLBA = '4'
        or VOLBA = ' ' ;
      *in80 = *off ; // spravna volba
   else ;
      *in80 = *on ;  // chybna volba
      *in81 = *on ;  // globalni chyba
   endif ;
endsr ;

//  -------------------------
//  Oprava zaznamu v databazi
//  -------------------------
begsr   opravaZaznamu ;
   dou   not *in82;  // je-li chyba, opakuji
      exfmt   OKNOOPR ; // zobrazim okno s udaji z podsouboru
      if   *in03 or *in12 ; // po F3 nebo F12 koncim podprogram
         *in82 = *off ;
         leavesr ;
      endif ;
      chain   (ZAVOD:  // prectu zaznam z databaze
               SKLAD:  // podle klice ziskaneho z okna
               MATER)
              STAVYP dataStavuI ;
      *in82 = *on ;  // predbezne zapnu chybu nenalezeno
      if   %found ;
         *in82 = *off ;  // vypnu chybu
         dataStavuI.MNOZ = MNOZ ; // dosadim mnozstvi z okna
         update   STAVYPF0 %fields(MNOZ) ;  // prepisu mnozstvi ve zaznam
      endif ;
   enddo ;
endsr ;

//  ------------------------
//  Zruseni zaznamu souboru
//  ------------------------
begsr   zruseniZaznamu ;
   dou   not *in83;  // je-li chyba, opakuji
      klicStavu.ZAVOD = ZAVOD ;
      klicStavu.SKLAD = SKLAD ;
      klicStavu.MATER = MATER ;
      delete   %kds(klicStavu) STAVYP ;
      if   not %found ;
         *in83 = *on ;  // neni-li zaznam v souboru, zapnu chybu
         exfmt   OKNOOPR ; // zobrazim chybu v okne
         if   *in03 or *in12 ; // po F3 nebo F12 koncim podprogram
            *in83 = *off ;
            leavesr ;
         endif ;
      else ;
         *in83 = *off ;
      endif ;
   enddo ;
endsr ;

//  -------------------------------
//  Zapis noveho zaznamu do souboru
//  -------------------------------
begsr   novyZaznam;
   clear   STAVYPF0 ; // NE "clear OKNONOVA" (kvuli indikatorum)
   dou   not *in84 ;
      exfmt   OKNONOVA ;
      if   *in03 or *in12 ;
         *in84 = *off ;
         leavesr ;
      endif ;
      dataStavuO.ZAVOD = ZAVOD;
```

```
            dataStavuO.SKLAD = SKLAD;
            dataStavuO.MATER = MATER;
            dataStavuO.MNOZ  = MNOZ;
            write(e)   STAVYPF0 dataStavuO;
            if   %error ;
               *in84 = *on ;  // duplicita ve stavech
            else ;
               *in84 = *off ;
            endif ;
         enddo ;
         exsr   plnitPods ;
      endsr ;
```

Vysvětlivky k programu STAPSUDR

Program je zapsán v polovolném formátu. Soubory jsou popsány v popisu souborů, pracovní data jsou popsána v popisu dat.

Popis souborů

Soubor STAVYP povoluje kromě přepisování (**U** - update) také přidávání nových záznamů (**A** - add).

Popis dat

Konstanta OPRAVA je určena zápisem DCL-C a klíčovým slovem CONST(‘2‘). Slovo CONST ani závorky nejsou povinné, stačilo by zapsat jen konstantu ‘2‘.

Výpočet

Program STAPSUDR je členěn na hlavní část a podprogramy: Hlavní program nejprve naplní podsoubor a nastaví kurzor. Pak vstoupí do, který může být ukončen jen stiskem klávesy F3 (v podprogramu *zobrPods* nebo *zobrazOkno*).

Program zobrazí stránku podsouboru a uživatel může v podsouboru listovat. Při tom může volit činnosti, které má program provádět:

- zadat kódy voleb do pole VOLBA u jednotlivých záznamů a stisknout klávesu Enter,
- stisknout klávesu F6, chce-li vložit nový záznam do databáze,
- stisknout klávesu F5, chce-li obnovit zobrazení podsouboru podle stavu v databázi,
- stisknout klávesu F3, chce-li ukončit výpočet.

Při cyklickém zpracování podsouboru se v přečteném záznamu testuje (podprogramem *testVolby*) kód volby (smí být jen 2, 4 nebo mezera); je-li chybný, zapne se indikátor 80. Je-li kód volby správný, provede se příslušná činnost:

kód 2 - podprogram *opravaZaznamu* (oprava databázového záznamu),

kód 4 - podprogram *zruseniZaznamu* (zrušení databázového záznamu),

kód mezera - žádná činnost.

Poté se pole VOLBA vyčistí, aby při příštím zobrazení podsouboru bylo prázdné.

Záznam podsouboru se přepíše, ať je kód volby správný nebo chybný (**UPDATE**).

Podprogram *plnitPods* bere nyní ohled na to, že se provádí opakovaně. Nejdříve vyčistí podsoubor příkazem **WRITE** *s řídicím formátem* při zapnutém indikátoru 56 podmiňujícím klíčové slovo **SFLCLR**. Pak nastaví ukazatel záznamu v databázi na začátek (**SETLL**) a přečte první záznam (READ), načež v cyklu naplní podsoubor. Symbol ***LOVAL** (low value) představuje v příkazu SETLL (set lower limit) nejnižší možný klíč. K nastavení na začátek souboru lze použít také symbol ***START**, který není založen na konstrukci klíče.

Podprogram *opravaZaznamu* přečte databázový záznam podle argumentu ze záznamu podsouboru a zobrazí ji v okně formátu OKNO podprogramem *zobrazOkno*. Uživatel může opravit data (včetně klíčových polí) a stisknout klávesu Enter, nebo může stisknout klávesu F12, aby obnovil zobrazení podsouboru, anebo může ukončit výpočet klávesou F3. V obou případech výpočet opustí podprogram příkazem **LEAVESR** (leave subroutine). Po stisku klávesy Enter podprogram *opravaZaznamu* přepíše databázový záznam novými údaji (jestliže jej nalezl).

Podprogram *zruseniZaznamu* zruší databázový záznam podle klíče.

Podprogram *novyZaznam* vyčistí všechna pole formátu OKNO a zobrazí jej prostřednictvím podprogramu *zobrazOkno*. Uživatel vyplní údaje a stiskne klávesu Enter. Pak podprogram *novyZaznam* zapíše nový záznam do databáze a naplní znovu podsoubor. Duplicitní áznam se nezapíše (jen se zapne indikátor 84).

Přehled příkazů pro soubory

Příkazy společné pro více druhů souborů

OPEN otevření souboru

CLOSE uzavření souboru

FEOD vynucený konec dat (u vstupních souborů zapne %EOF, u výstupních ukončí výstup)

U výstupního databázového souboru způsobí příkaz FEOD okamžitý zápis záznamů z vyrovnávací paměti na vnější paměť (disk). Příkaz FEOD(N) dá data posledního zapsaného záznamu k dispozici dalším čtecím příkazům, aniž by je zapisoval na disk.

Příkazy pro databázové soubory

READ čtení dalšího záznamu

READE čtení dalšího záznamu se stejným klíčem

READP čtení předchozího záznamu

READPE čtení předchozího záznamu se stejným klíčem

SETLL nastavení dolní meze

SETGT nastavení horní meze

CHAIN čtení záznamu podle klíče nebo pořadového čísla

UPDATE přepis přečteného záznamu

WRITE zápis nového záznamu

DELETE výmaz přečteného záznamu nebo výmaz přímo podle klíče

UNLOCK odemknutí zamčeného záznamu

Čtení postupně od začátku souboru (slovo USROPN v popisu souboru je nutné, jen když chceme použít příkaz OPEN).

```
dcl-f STAVYP keyed usropn;
if not %open(STAVYP); // není-li již otevřen
    open STAVYP;      // - otevřu soubor
endif;
read STAVYP; // čtu první záznam
dow not %eof; // cyklus do konce dat
    ...
    read STAVYP; // čtu další záznam
enddo; // konec cyklu
close STAVYP; // uzavřu soubor
```

Čtení od začátku souboru s předchozím nastavením

```
setLL *start STAVYP; // nastaví začátek souboru
read STAVYP; // čtu první záznam souboru
dow not %eof;
    ...
```

Zpracování jednoho závodu od začátku

```
setLL ZAVOD STAVYP; // nastavím začátek závodu
if %equal;              // najde-li se přesně,
    readE ZAVOD STAVYP; // čtu 1. záznam závodu
    dow not %eof; // cyklus do konce závodu
        ...
        readE ZAVOD STAVYP; // čtu další záznam závodu
    enddo; // konec cyklu
endif;
```

Čtení od konce souboru pozpátku

```
setGT *end STAVYP; // nastaví pozici za konec souboru
readP STAVYP // čtu poslední záznam souboru
dow not %eof;
    ...
```

Zpracování jednoho závodu od konce pozpátku

```
setGT ZAVOD STAVYP; // nastavím konec závodu
if %found;               // najde-li se vůbec něco,
    readPE ZAVOD STAVYP; // čtu poslední záznam závodu
    dow not %eof; // cyklus do začátku závodu pozpátku
        ...
        readPE ZAVOD STAVYP; // čtu předchozí záznam závodu
    enddo; // konec cyklu
endif;
```

Zpracování jednoho skladu s použitím částečného klíče

```
setLL (ZAVOD: SKLAD) STAVYP; // nastavím pozici začátku skladu v závodu
if %found;       // najde-li se vůbec něco
    read STAVYP; // čtu 1. záznam nějakého skladu
    dow not %eof; // cyklus do konce skladu
        ...
        readE (ZAVOD: SKLAD) STAVYP; // čtu další záznam skladu
    enddo; // konec cyklu
endif;
```

Příkazy pro obrazovkové soubory

Jednoduché obrazovky

READ čtení z klávesnice
WRITE zápis formátu na obrazovku
EXFMT zápis formátu na obrazovku a čtení z klávesnice (WRITE + READ)

```
write RIDICI; // zápis na obrazovku
...
read  RIDICI; // čtení z klávesnice
if *in03;
    ...
```

Obrazovky s podsoubory (subfile)

READC čtení dalšího změněného záznamu podsouboru
WRITE zápis záznamu podsouboru (se zadaným datovým formátem)
WRITE výmaz celého podsouboru (se zadaným řídicím formátem a aktivním slovem SFLCLR)
CHAIN čtení záznamu podle pořadového čísla
UPDATE přepis záznamu přečteného příkazem READC nebo CHAIN

Zpracování celého podsouboru pomocí příkazu CHAIN

```
for idx = 1 to pocet; // cykl přes celý podsoubor
    chain idx DATA;   // čtu podle čísla záznamu
    ...
    update DATA; // přepíšu právě přečtený záznam podsouboru
endfor; // konec cyklu
```

Manipulace se znakovými řetězci

K práci se znakovými řetězci slouží několik prostředků, z nichž zde probereme jen ty nejdůležitější.

Hlavní je operátor + pro *spojení řetězců*.

Další prostředky jsou zejména vestavěné funkce

%SUBST – výběr podřetězce,
%SCAN – vyhledání vzorku,
%REPLACE – náhrada části řetězce jiným řetězcem.

V souvislosti s manipulací řetězci lze s výhodou použít znaková data *proměnné délky* typu VARCHAR.

Ke *zkracování řetězců* slouží funkce

%TRIM (na obou koncích),
%TRIMR (na pravém konci) a
%TRIML (na levém konci).

K *překódování textu* slouží funkce

%XLATE.

K výše jmenovaným vestavěným funkcím existují stejnojmenné příkazy, které lze použít alternativně.

K *proměně znakových dat na číselná a obráceně* slouží funkce %CHAR (proměna čísla na znaky), %DEC (proměna znaků na dekadické číslo).

K *určení délky proměnné* slouží funkce %LEN a %SIZE. Funkce %LEN představuje okamžitou délku, zatímco funkce %SIZE celkovou délku. Funkce **%LEN** představuje délku číselné proměnné

v počtu dekadických číslic, kdežto **%SIZE** délku v počtu bajtů. U znakových proměnných dávají obě funkce počet bajtů.

Program TEXT01 – %SUBST, %SCAN, VARCHAR

```
 *  Užitečné formuláře RPG pro pevný formát
C*0N01Factor1+++++++Opcode&ExtExtended-factor2++++++++++++++++++++++++++++Comments++++++++++++
O*ilename++DF..N01N02N03Excnam++++B++A++Sb+Sa+.........................Comments++++++++++++
O*.............N01N02N03Field+++++++++YB.End++PConstant/editword/DTformat++Comments++++++++++++

   // ===================
   // Manipulace s textem
   // ===================
   dcl-f qprint printer(100);

   dcl-s TextVar  varchar(20) inz('ABCDEFGH');
   dcl-s TextVar2 varchar(30) inz('abcdefgh');

   dcl-s Vzorek varchar(10);
   dcl-s Delka packed(3: 0);

   // Textova promenna s promennou delkou
   // -----------------------------------

   TextVar2 = %subst(TextVar: 2: 3);

   Delka = %len(TextVar2);
   except var;

   TextVar2 = %subst(TextVar: %scan('CDE': TextVar: 1): 3);

   Delka = %len(TextVar2);
   except var;

   Vzorek = 'DEF';
   TextVar2 = %subst(TextVar: %scan(Vzorek: TextVar: 1): %len(Vzorek));

   Delka = %len(TextVar2);
   except var;

   dump(a) 'TEXT01';

   *inlr = *on;

   // Popis vystupu - testovaci vypis

O*ilename++DF..N01N02N03Excnam++++B++A++Sb+Sa+.........................Comments++++++++++++
oqprint    e            var
O*.............N01N02N03Field+++++++++YB.End++PConstant/editword/DTformat++Comments++++++++++++
o                                                 'TextVar2 '
o                      Delka         3     +1
o                      TextVar2            +2
```

Znaková proměnná typu VARCHAR je v paměti uložena tak, že začíná dvoubajtovým binárním údajem o skutečné délce dat a pokračuje paměťovou oblastí ve své zadané délce. Je-li tedy zadaná délka 20, paměťová oblast má 22 bajtů. Ve 20 bajtech dat je od začátku uložen text, jehož okamžitá (použitelná) délka může být 0 až 20 bajtů. Funkce %LEN představuje okamžitou délku, zatímco funkce %SIZE celkovou délku (tedy 22). Délku takové proměnné lze změnit i přímo, třeba příkaz

```
%len(TextVar2) = 0;
```

vlastně vymaže text z proměnné *TextVar2*.

Program TEXT02 – náhrada části textu – %REPLACE

```
// ==============================
// Manipulace s textem - %REPLACE
// ==============================
dcl-f qprint printer(100);

dcl-s TextFix  char(20) inz('ABCDEFGH');
dcl-s TextFix2 char(30);

dcl-s Vzorek   char(3)     inz('BCD');
dcl-s Nahrada  varchar(10) inz('***');
dcl-s Delka    packed(3: 0);

// Textova promenna s pevnou delkou
// --------------------------------

// od 2. pozice nahradim 3 znaky
TextFix2 = %replace(Nahrada: TextFix: 2: 3);
Delka = %len(TextFix2);
except fix;

// od 2. pozice nahradim jen 2 znaky, zbyvajici se posunou vpravo
TextFix2 = %replace(Nahrada: TextFix: %scan('BCD': TextFix: 1): 2);


Delka = %len(TextFix2);
except fix;

// od 2. pozice vlozim vzorek, zbyvajici znaky se posunou vpravo
TextFix2 = %replace(Nahrada: TextFix: %scan(Vzorek: TextFix: 1): 0);

Delka = %len(TextFix2);
except fix;

dump(a) 'TEXT02';

*inlr = *on;
```

```
 *   Popis vystupu - testovaci vypis

O*ilename++DF..N01N02N03Excnam++++B++A++Sb+Sa+.............................Comments++++++++++++
oqprint    e            fix
O*.............N01N02N03Field+++++++++YB.End++PConstant/editword/DTformat++Comments++++++++++++
o                                              'TextFix2 '
o                       Delka         3     +1
o                       TextFix2            +2
```

Program TEXT03 – zkracování, spojování – %TRIM, %TRIMR, %TRIML

Funkce %TRIM zkracuje text o koncové i vedoucí znaky. Funkce %TRIMR jen o koncové (zprava), funkce %TRIML o vedoucí (zleva). Tyto funkce mohou mít druhý operand, který určuje znaky, o něž se má znaková hodnota zkrátit. Není-li druhý operand zadán, zkracuje se text o mezery. Např. výraz

```
%trimr(TextFix2 : ‘ *‘)
```

zkrátí text v proměnné *TextFix2* o všechny mezery a hvězdičky zprava.

Funkce %SUBST vyjme část textu od zadané pozice v zadané délce. Funkce %SCAN vyhledá zadaný vzorek (text) v textové proměnné a dosadí číslo jeho první pozice, jestliž jej našla. Když se vzorek nenašel, dosadí číslo 0.

Funkce %LEN může vystupovat také na levé straně přiřazovacího příkazu a měnit tak délku textu proměnné délky (VARCHAR).

```
// ================================================
// Manipulace s textem - %trim, zkracovani, spojovani
// ================================================
dcl-f qprint printer(100);

dcl-s TextVar1 varchar(20) inz('ABCDEFGH');
dcl-s TextVar2 varchar(30) inz(' ++abcdef**   ');
dcl-s TextVar3 varchar(30);

dcl-s Delka packed(3);

// Spojeni dvou celych retezcu bez zkracovani
// u promenneho se retezce spoji i s mezerami a delka se upravi
TextVar3 = TextVar1 + TextVar2;
Delka = %len(TextVar3);
except var;

// Spojeni dvou celych retezcu se zkracenim druheho na obou koncich
// u fixniho se vysledek spojeni uz vejde cely do promenne TextFix3

TextVar3 = TextVar1 + %trim(TextVar2);
Delka = %len(TextVar3);
except var;

// Spojeni dvou zkracenych retezcu s vlozenim mezery
TextVar3 = %trimr(TextVar1) + ' ' + %triml(TextVar2);
Delka = %len(TextVar3);
except var;

// Zkraceni retezce i o mezery a znaky  '+'  a  '*'

TextVar3 = %trim(TextVar2: '+* ');
Delka = %len(TextVar3);
except var;

// Zkraceni promenneho retezce zmenou delky
%len(TextVar3) = 4;
Delka = %len(TextVar3);
except var;

dump(a) 'TEXT03';
*inlr = *on;

 *   Popis vystupu - testovaci vypis
oqprint    e            var
o                                          'TextVar3 '
o                       Delka         3     +1
o                       TextVar3            +2
```

Program TEXT04 – %XLATE, %CHECK, %CHECKR

Proměnná *Text* je dlouhá 20 bajtů a má stejnou počáteční hodnotu jako konstanta *TextConst*.

Klíčové slovo LIKE v definici dat určuje délku a typ proměnných *Text2* a *Text3* podle proměnné *Text*.

Funkce %XLATE překóduje znakovou proměnnou podle dvou překódovacích tabulek. Znaky zadané první tabulkou, zjištěné v operandu, nahradí odpovídajícími znaky zadanými druhou tabulkou a výsledek dosadí jako funkční hodnotu.

Funkce %CHECK zkoumá operand od zadané pozice zleva doprava, zda najde znak odlišný od zvolených znaků. Jakmile nějaký najde, dosadí jako svou hodnotu jeho pořadové číslo. Jestliže žádný odlišný znak nenajde, dosadí číslo 0. Funkce %CHECKR pracuje stejně, ale zprava doleva.

```
**FREE
//
// Manipulace s textem - %xlate, %check, %checkr
//

dcl-c TextConst        const('  12345678.90 Kč');
dcl-s Text varchar(20) inz(TextConst);
dcl-s Text2            like(Text);
dcl-s Text3            like(Text);
dcl-s Zac              int(10);
dcl-s Kon              uns(10);
dcl-c Tab1             '.Kč';
dcl-c Tab2             ',  ';
dcl-c MEZERA           ' ';

   // Odstranim Kč a nahradim tecku carkou
Text2 = %xlate(Tab1: Tab2: Text);

   // Najdu prvni nemezerovy znak zleva
Zac = %check(' ': Text2: 1);
   /////Zac = %check(MEZERA: Text2: 1);

   // Najdu prvni nemezerovy znak zprava
Kon = %checkr(' ': Text2: %len(Text2));

   // Vyjmu vsechny znaky mezi nimi
Text3 = %subst(Text2: Zac: Kon - Zac + 1);

dump(a) 'TEXT04';

*inlr = *on;
```

Program NUMCHAR – proměna typů dat – %EDITC, %CHAR, %DEC

Klíčové slovo DECEDIT(*JOBRUN) ve řídicím příkazu CTL-OPT určuje vlastnosti editovaného čísla, tj. zda desetinná místa budou oddělena čárkou a tisíce tečkou nebo obráceně. Jak to bude skutečně, můžeme zjistit, když si vypíšeme vlastnosti úlohy CL příkazem DSPJOB a podíváme se na parametr DECFMT. Jinak lze zadat konkrétní vlastnosti; např. DECEDIT('0,') říká, že desetinná místa oddělí čárka, tisíce tečka a číslo menší než 1 bude mít před desetinnou čárkou nulu.

Konstanta *CisloConst* představuje dekadické číslo o 63 číslicích, z čehož 2 tvoří desetinná místa. Oddělovačem může být čárka nebo tečka.

Funkce %DECPOS zjistí počet desetinných míst číselné proměnné nebo konstanty. Zde dosadí číslo 2 pro konstantu *DECPOS*.

Funkce %EDITC upraví číslo do čitelné podoby podle zadaného edičního kódu, který musí být zadán jako prostá nebo pojmenovaná konstanta (zde konstanta *EdicniKod* s hodnotou N).

Číslo na znaky lze převést také funkcí %CHAR. Znaky umístí zprava doleva, před první platnou číslici umístí případné záporné znaménko.

Znaky na číslo lze převést také funkcí %DEC**,** kde musíme zadat kromě znakové proměnné také počet číslic a počet desetinných míst. Znaková proměnná může obsahovat znaménko + nebo - před nebo za číslem, desetinnou čárku nebo tečku a mezery kdekoliv. Kromě toho ale žádné jiné znaky.

```
//*************************************************************
//   Promena cisel na znaky a opacne
//*************************************************************
ctl-opt decedit (*jobrun);

dcl-f qprint printer(120) oflind(*inoa);

dcl-c CisloConst
 -12345678901234567890123456789012345678901234567890123456789012345678901,23 ;
dcl-c LEN     %len(CisloConst);
dcl-c DECPOS  %decpos(CisloConst);

dcl-s Cislo   packed(63: 2) inz(CisloConst);
dcl-s TextFix char(100);

//  Edicni kod musi byt konstanta
dcl-c EdicniKod const('N');

dcl-s Znaky char(20) inz('12345678901234567890');
dcl-s Cislo2 zoned(20: 2);

dcl-s Delka packed(3);

// Editace cisla - %editc
TextFix = %editc(Cislo: EdicniKod);
Delka = %len(TextFix);
except    fix;

// Promena cisla na znaky
TextFix = %char(Cislo);
Delka = %len(TextFix);
except    fix;

// Promena znaku na cislo
Cislo = %dec(Znaky: LEN: DECPOS);
Delka = %len(Cislo);
except    cis;


dump(a) 'NUMCHAR';

*inlr = *on;


 * Popis vystupu - testovaci vypis

oqprint    e            fix            1
o                                              'TextFix  '
o                       Delka          3     +1
o                       TextFix              +2
oqprint    e            cis             1
o                                              'Cislo    '
o                       Delka          3     +1
o                       Cislo          Q     +2
```

Ediční kód N použitý ve funkci %EDITC upravuje číslo tak, že oddělí desetinná místa, jsou-li jaká, potlačí vedoucí nuly, vloží záporné znaménko dopředu, oddělí trojice čísliců; nulové číslo vytiskne. Ediční kód Q je použit v popisu výstupu (formulář O). Liší se od kódu N tak, že neodděluje trojice číslic a nulové číslo netiskne.

Operace s časovými údaji

V jazyku RPG jsou k dispozici prostředky k práci s časovými údaji. Časové údaje jsou *datum*, *čas* a *časové razítko* (datum a čas společně s přesností na mikrosekundy).

Tyto typy dat se objevují také v externích popisech DDS, kde byly zavedeny již dříve, ale i v SQL. Kódy pro jejich označení v RPG jsou tyto:

DATE	Date, datum (v DDS se označuje kódem L)
TIME	Time, čas (v DDS se označuje kódem T)
TIMESTAMP	Timestamp, časové razítko (v DDS se označuje kódem Z)

U jejich definic se neuvádí délka ani počet desetinných míst, protože délka je dána jejich formátem (a desetinná místa nemají význam).

Příslušné konstanty (literály) se v RPG označují písmeny D, T, Z.

Formáty s ukázkami konstant jsou uvedeny v následujících odstavcích.

Datum (DATE)

Kód	*Význam*	*Stand. formát*	*Oddělovač*	*Délka*	*Konstanta*
*ISO	Mezinárodní norma	rrrr-mm-dd	-	10	D'2005-12-31'
*CYMD	Století, rok, měsíc, den	srr/mm/dd	/ - . , &	9	D'105/12/31'
*YMD	Rok, měsíc, den	rr/mm/dd	/ - . , &	8	D'05/12/31'
*DMY	Den, měsíc, rok	dd/mm/rr	/ - . , &	8	D'31/12/05'
*MDY	Měsíc, den, rok	mm/dd/rr	/ - . , &	8	D'12/31/05'
*JUL	Rok, den v roce	rr/ddd	/ - . , &	6	D'05/365'
*USA	IBM USA	mm/dd/rrrr	/	10	D'12/31/2005'
*EUR	IBM Evropa	dd.mm.rrrr	.	10	D'31.12.2005
*JIS	Japonská norma	rrrr-mm-dd	-	10	D'2005-12-31'

První z oddělovačů je standardní, někde je možný jen jediný. Znak & znamená mezerový oddělovač. Konkrétní oddělovač lze zadat ve funkci DATFMT, např. DATFMT(*YMD-).

V operaci TEST lze použít ještě další kódy, které slouží k testování jiných typů dat (zejména číselných a znakových) na datumový typ.

Kód	*Význam*	*Tvar*	*Oddělovač*	*Délka*	*Hodnota*
*CYMD	Století, rok, měsíc, den	srr/mm/dd	/ - . , &	9	105/12/31
*CMDY	Století, měsíc, den, rok	smm/dd/rr	/ - . , &	9	112/31/05
*CDMY	Století, den, měsíc, rok	sdd/mm/rr	/ - . , &	9	131/12/05
*JOBRUN	Podle údajů DATFMT a DATSEP úlohy				
*LONGJUL	Dlouhé Juliánské datum	rrrr/ddd	/ - . , &	8	2005/365

Datumové kódy lze rozšířit o znak oddělovače resp. 0. Nula znamená, že datum nemá *žádný oddělovač*. Tyto kódy lze zapsat v příkazu TEST, kde dáváme najevo, že *znaková* proměnná, kterou testujeme, má určitý oddělovač, resp. nemá oddělovač vůbec.

Kód	*Význam*	*Tvar*	*Oddělovač*	*Délka*	*Hodnota*
*ISO0	Mezinárodní norma	'rrrrmmdd'	žádný	8	'20051231'
*CYMD0	Století, rok, měsíc, den	'srrmmdd'	žádný	7	'1051231'
*YMD0	Rok, měsíc, den	'rrmmdd'	žádný	6	'051231'

atd...

Čas (TIME)

Kód	*Význam*	*Tvar*	*Oddělovač*	*Délka*	*Konstanta*
*ISO	Mezinárodní norma	hh.mm.ss	.	8	T'23.59.59'
*HMS	Hodiny, minuty, sekundy	hh:mm:ss	: . , &	8	T'23:59:59'
*USA	IBM USA	hh:mm AM/PM	:	8	T'11.59 PM'
*EUR	IBM Evropa	hh.mm.ss	.	8	T'23.59.59'
*JIS	Japonská norma	hh:mm:ss	:	8	T'23:59:59'

První z oddělovačů je standardní. Znak & znamená mezeru. Konkrétní oddělovač lze zadat ve funkci TIMFMT, např. TIMFMT(*HMS&).

Podobně jako u datumových kódů, lze i u časových kódů dopsat oddělovač nebo 0, která vyjadřuje nepřítomnost oddělovače. Kódy *ISO0, *HMS0, atd. lze u příkazu TEST, a to jen pracujeme-li se *znakovou* proměnnou.

Časové razítko (TIMESTAMP)

Časové razítko nemá žádný kód formátu, protože jeho tvar je jen jediný, a to:

Tvar	*Oddělovač*	*Délka*	*Konstanta*
rrrr-mm-dd-hh.mm.ss.xxxxxx	- .	26	Z'2005-12-31-23.59.59.999999'

Časové razítko se skládá z údajů rok (rrrr), měsíc (mm), den (dd), hodiny (hh), minuty (mm), sekundy (ss), mikrosekundy (xxxxxx). Údaje jsou odděleny pomlčkou (v datumové části) a tečkou (v časové části). Časová část je od datumové oddělena pomlčkou. Když zapisujeme konstantu v programu, můžeme mikrosekundy od konce vynechávat. Kompilátor si vynechaná místa doplní nulami. Například:

Z'2005-12-31-23.59.59.99' bude převedeno na 2005-12-31-23.59.59.990000,

Z'2005-12-31-23.59.59' bude převedeno na 2005-12-31-23.59.59.000000.

Definice časových údajů

Časové údaje mohou být definovány

- v RPG programu v popisu dat,
- v DDS (u souborů popsaných externě),
- v SQL (příkazem CREATE TABLE v definici sloupců jako typy DATE, TIME, TIMESTAMP).

```
CTL-OPT DATFMT(*YMD.) TIMFMT(*ISO);
...
// Proměnné
DCL-S Vcera DATE INZ(D'05.08.04') DATFMT(*ISO);
DCL-S Dnes  DATE INZ(D'05.08.05') DATFMT(*ISO);
DCL-S Zitra DATE INZ(D'05.08.06') DATFMT(*ISO);
DCL-S Poledne TIME INZ(T'12.00.00') TIMFMT(*ISO);

// Konstanty (nelze u nich zapsat DATFMT, TIMFMT)
DCL-C Vcera      CONST(D'05.08.04');
DCL-C Dnes       CONST(D'05.08.05');
DCL-C Zitra      CONST(D'05.08.06');
DCL-C Pulnoc     T'00.00.00';
DCL-C Silvestr   Z'2005-12-31-23.59.59.999999';
DCL-C NovyRok    Z'2006-01-01-00.00.00';
```

Konstanty ve parametrech INZ a CONST musí být zapsány ve formátu platném pro celý program, tj. v tom, který je zadán v řídicím příkazu CTL-OPT symboly DATFMT a TIMFMT (chybí-li tam, pak ve formátu *ISO). To platí i tehdy, je-li u konstanty zadán jiný formát (funkcí DATFMT či TIMFMT). Kompilátor si ovšem konstanty převede do formátů určených funkcemi DATFMT a

TIMFMT v popisu dat a nadále pracuje s převedenými hodnotami. Tak např. konstanta *Pulnoc* je zadána ve formátu *ISO (protože ten je předepsán v řídicím příkazu CTL-OPT), ale hodnota v programu bude mít formát *HMS (protože ten je předepsán funkcí TIMFMT).

Formát externě popsaných časových údajů musí být popsán v popisu příslušného souboru, a to funkcemi DATFMT a TIMFMT tak, aby odpovídal formátu definovanému v externím popisu (např. DDS). Nezadáme-li tyto funkce v popisu souboru, hlásí kompilátor chybu.

Pořadí při aplikaci formátu časových údajů je tedy následující:

RPG proměnné

DATFMT/TIMFMT - popis dat
DATFMT/TIMFMT - řídicí příkaz
*ISO

Externí proměnné

kód formátu (*YMD,...) - formulář I
DATFMT/TIMFMT - formulář O
DATFMT/TIMFMT - popis souboru
chyba při překladu

Přesuny a konverze časových údajů

Pomocí příkazů MOVE, MOVEL a funkcí %DATE, %TIME, %TIMESTAMP lze časové údaje přesunovat mezi následujícími typy dat:

Kód formátu	***Zdroj***	***Cíl***	***Vysvětlivka***
	datum	datum	s případnou konverzí formátu
	čas	čas	s případnou konverzí formátu
	časové razítko	časové razítko	(jen jediný formát)
	datum	časové razítko	čas se v razítku nezmění
	čas	časové razítko	datum se v razítku nezmění
	časové razítko	datum	
	časové razitko	čas	
*YMD, ...	datum	číselná proměnná	oddělovač bude odstraněn
*HMS, ...	čas	číselná proměnná	oddělovač bude odstraněn
*YMD, ...	datum	znaková proměnná	
*HMS, ...	čas	znaková proměnná	
*YMD, ...	číselná proměnná	datum	
*YMD, ...	znaková proměnná	datum	
*HMS, ...	číselná proměnná	čas	
*HMS, ...	znaková proměnná	čas	
	číselná proměnná	časové razítko	
	znaková proměnná	časové razítko	
	*DATE	datum	
	UDATE	datum	

Je-li zdroj *DATE nebo UDATE, kód formátu se nezadává.

Kód formátu může být také *JOBRUN, *LONGJUL, *CYMD, *CMDY nebo *CDMY. Kód *JOBRUN znamená formát data nebo také času odpovídající atributům úlohy (viz CL příkaz DSPJOB, parametry DATFMT, DATSEP). K času se vztahuje atribut DATSEP (nikoliv TIMSEP).

Doba trvání

Doba trvání (duration) je časový interval vyjádřený v různých časových jednotkách, které se zadávají následujícími obraznými konstantami (kódy):

Plný tvar	*Zkratka*	*Význam*
*YEARS	*Y	roky
*MONTHS	*M	měsíce
*DAYS	*D	dny
*HOURS	*H	hodiny
*MINUTES	*MN	minuty
*SECONDS	*S	sekundy
*MSECONDS	*MS	mikrosekundy

Program DATADD – přičítání doby trvání k datu a času

```
**FREE
dcl-s CasoveRazitko   timestamp;
dcl-s DatumFakturace  date Inz(*LoVal);
dcl-s DatumPlat       date;
dcl-s DatDodani       date Inz(*LoVal);
dcl-s ZarucDoba       int(10) Inz(2);     //dva roky
dcl-s DatKonZaruky    date;
dcl-s PrijPalety      timestamp;
dcl-s ZasklPalety     timestamp;
dcl-s OchrDoba        int(10) Inz(3);     // tri hodiny

// Systemovy cas pouzijeme jako pocatecni casovy udaj
CasoveRazitko =  %TimeStamp();
DatumFakturace = %date(CasoveRazitko);
DatDodani =      %date(CasoveRazitko);
PrijPalety =     CasoveRazitko;

PrijPalety += %Hours(12);

// K datu vystaveni faktury pricteme 14 dni
DatumPlat = DatumFakturace + %Days(14);

// Datum dodani plus zarucni doba dava datum konce zaruky
DatKonZaruky = DatDodani + %Years(ZarucDoba);

// Cas prijeti palety + ochranna doba v hodinach dava cas pro zaskladneni
ZasklPalety = PrijPalety + %Hours(OchrDoba);

dump(a) 'DATADD';
*inlr = *on;
```

Symbol *LOVAL představuje nejnižší hodnotu, v tomto případě datum 0001-01-01. Pro záruční dobu je použit binární celočíselný typ INT(10). Desítka znamená počet dekadických číslic.
V programu se používá příkaz TIME, který získá ze systémový čas a umístí jej do časového razítka. Operátor += je zkratka pro přičítání pravé strany k levé, abychom nemuseli na pravé straně opakovat proměnnou z levé strany příkazu. Přičítání doby trvání provádí operátor +.

Program DATPODM – použití data v podmínkách

```
**FREE
dcl-s DatumZac    date inz(d'2005-01-01')
dcl-s DatumKon    date inz(d'2005-12-31')
dcl-s DatumPrac   date
dcl-s PocetDni    int(10)

// zjistim pocet dnu v unoru 2005
DatumPrac = DatumZac;
dow DatumPrac <= DatumKon;
    if %subdt(DatumPrac: *months) = 2;
        pocetDni += 1;
    endif;
    DatumPrac += %days(1);
enddo;
*inlr = *on;
```

Příkaz DOW provádí cyklicky tzv. tělo cyklu (což jsou příkazy mezi DOW a ENDDO), dokud platí podmínka uvedená vpravo, tedy dokud *DatumPrac* je menší nebo rovno *DatumKon*. Funkce %SUBDT vyjme z data počet měsíců a vrátí jej jako svou hodnotu.

Program DATSUB – odečítání časových údajů a volání programu

```
**free
dcl-s DatumFakt   date inz(d'2005-07-17');
dcl-s DatumPlat   date inz(d'2005-08-17');
dcl-s Cena        packed(7: 2) inz(100,00);
dcl-s Penale      packed(7: 2);
dcl-s DobaOdFakt  packed(5);

dcl-s Odchod      timestamp inz(z'2005-07-17-15.00.00');
dcl-s Prichod     timestamp inz(z'2005-07-17-07.15.00');
dcl-s MinutZaDen  zoned(6);
dcl-s Hodin       zoned(2: 1);

dcl-pr penaleProgram extpgm('PENALE');  // prototyp volání programu
    DobaOdFaktur    packed(5);
    CenaDodavky     packed(7: 2);
    CastkaPenale    packed(7: 2);
end-pr;

DobaOdFakt =  %diff(DatumPlat: DatumFakt: *days);
if DobaOdFakt > 14;
    //callp penaleProgram (DobaOdFakt: Cena: Penale);  // volání programu
    penaleProgram (DobaOdFakt: Cena: Penale);  // volání programu bez CALLP
endif;

MinutZaDen = %diff(Odchod: Prichod: *mn);
eval(h)   Hodin = MinutZaDen / 60;
dump(a) 'DATSUB';
return;
```

Odečtení doby trvání od data provádíme operátorem – (minus). K odečtení dvou datumů však musíme použít funkci %DIFF (difference). V programu DATSUB se vyvolává jiný RPG program nazvaný PENALE.

Než se program vyvolá, musí se mu zadat tři parametry pomocí *prototypu volání* v příkazu DCL-PR se zvoleným jménem programu (penaleProgram) a parametrem EXTPGM, kde je skutečné jméno volaného programu PENALE zapsané *velkými písmeny* a uvedené *mezi apostrofy*. Prototyp končí příkazem END-PR.

Příkaz CALLP (call with prototype) volá program zvoleného jména *penaleProgram*. První dva parametry předáváme volanému programu, třetí parametr je výsledek, který dostaneme po provedení programu PENALE, kdy se výpočet vrátí zpět. Dělení se provádí operátorem /. Modifikátor H v

závorce u příkazu EVAL znamená, že výsledek dělení se *zaokrouhlí* na 1 desetinné místo, protože tak je definovaná proměnná *Hodin*.

Program PENALE – výpočet penále pro program DATSUB

```
**free
// Prijima dva parametry a vraci jeden (částku penále)
dcl-pi *n extpgm('PENALE');            // program interface
    DobaOdFaktur    packed(5);
    CenaDodavky     packed(7: 2);
    CastkaPenale    packed(7: 2);
end-pi;

dcl-c Sazba         1;                 // jedno procento/den

CastkaPenale = DobaOdFaktur * CenaDodavky * Sazba/100;
return;
```

Program PENALE je volán s parametry, které jsou zadány v příkazu DCL-PI (program interface, rozhraní programu) a které přesně odpovídají parametrům v prototypu volání (kromě jmen, která se nemusí shodovat). Popis rozhraní končí příkazem END-PI.

Program DATTST – testy správnosti časových údajů s konverzemi

```
**FREE
// ************************************************************
// *   Test spravnosti data a casu s konverzemi
// ************************************************************
dcl-s CisloDat        zoned(6) inz(050930);
dcl-s ZnakyDat        char(6)  inz('050930');
dcl-s ZnakyCas        char(8)  inz('23:59:59');
dcl-s ZnakyRazitko    char(20) inz('20051231235959000000');

dcl-s DatCislo        date(*dmy.);
dcl-s DatZnaky        date(*iso);
dcl-s CasZnaky        time(*hms&);
dcl-s RazitkoZnaky    timestamp;

dcl-s DnesCislo       date;
dcl-s ZnakyDnes       varchar(20);

// Cislo představuje platné datum
//  (pripona 0 nema smysl, protoze cislo nemuze obsahovat oddelovace):
test(de) *ymd  CisloDat;
if not %error;  // když není chyba
    DatCislo = %date(CisloDat: *ymd);
endif;

// Znaky predstavuji platne datum
//  (pripona 0 znamena, že testovana promenna neobsahuje oddelovace):
test(de) *ymd0 ZnakyDat;
if not %error;
    DatZnaky = %date(ZnakyDat: *ymd0);
endif;

// ZnakyCas predstavuje platny cas
test(te) *hms: ZnakyCas;
if        not %error;
    CasZnaky = %time(ZnakyCas: *hms);
endif;

// RazitkoZnaky predstavuje platne casove razitko
//  (pripona 0 znamena, ze testovana promenna neobsahuje oddelovace):
test(ze) *iso0 ZnakyRazitko;
if not %error;
    RazitkoZnaky = %timestamp(ZnakyRazitko: *iso0);
endif;

// Ziskam znakovou podobu systemoveho data ve tvaru *jobrun
```

```
ZnakyDnes = %char(DnesCislo: *jobrun);

dump(a) 'DATTEST';
*inlr = *on;
```

Příkaz TEST s modifikátory v závorkách zkoumá datum, čas nebo časové razítko zadané jako číselná nebo znaková proměnná. Jaký typ zkoumá, je předepsáno modifikárorem D, T nebo Z. Modifikátor E umožňuje použít funkci %ERROR, která vrací hodnotu *ON (chyba) nebo *OFF. Vyjde-li test správně, převedeme takovou proměnnou na příslušný časový typ (datum, čas nebo časové razítko). V programu DATTST všechny testy vyjdou správně a proměnné se převedou.

Funkce %CHAR převádí do znakové podoby nejen číslo, ale i časové údaje, zde datum. Podobně i funkce %DEC převádí do číselné podoby nejen znaky, ale i časové údaje.

Cvičení

Zkuste změnit některé vstupní hodnoty na chybné, abyste ve výpisu paměti (dump) zjistili, zda se převod dat provedl nebo ne.

Datové struktury a vektory

Vektor

Vektor (array) je skupina datových položek, které mají stejnou délku a stejný typ a jsou umístěny v operační paměti. Vektorem je např. tabulka částek za 12 měsíců roku. Pojem *array* se někdy také překládá jako *pole*, spíše však v jiných programovacích jazycích. Vektor se definuje klíčovým slovem DIM určujícím počet jeho položek.

```
DCL-S  Vektor  char(1) DIM(10);
```

Datová struktura

Datová struktura (data structure) je skupina datových položek, které mají obecně různou délku a typ, ale významově spolu souvisí. Datová struktura připomíná záznam (record) v databázi. Složky datové struktury se nazývají *podpole (subfields)*. Popis datové struktury začíná příkazem DCL-DS. Následuje jeden nebo více příkazů podpolí. Podpole čili složky datové struktury se definují stejným způsobem jako samostatné proměnné, ale *bez* klíčového slova DCL-S. Shoduje-li se však jméno podpoe s rezervovaným slovem RPG, musí mu předcházet klíčové slovo DCL-SUBF.

```
DCL-DS DatovaStruktura;
    zbozi char(50);
    DCL-SUBF packed packed(4: 0);
END-DS;
```

Datové struktury bez překryvů

```
// Datová struktura beze jména
dcl-ds *N;
    dcl-subf pole1  char(5);
    dcl-subf pole2  packed(12: 2);
end-ds;

// Datová struktura se jménem a inicializací
dcl-ds Strukt_1 INZ;
    dcl-subf Num1      packed(2: 0);
    dcl-subf Num2      packed(4: 2);
    dcl-subf Jmeno     char(35);
end-ds;
```

První datová struktura *Strukt_1* obsahuje dvě číselné proměnné (podpole datové struktury) a jednu znakovou proměnnou. Protože v řádku s DCL-DS je zapsáno klíčové slovo INZ, budou číselné

proměnné nulové a znaková proměnná bude prázdná. Druhá datová struktura *nemá jméno*. Místo jména je zápis *N. Podpole jsou *nepovinně* označena klíčovým slovem DCL-SUBF.

Datové struktury s překryvy

```
// překryv - OVERLAY - nesmí použít jméno struktury, jen jméno podpole
DCL-DS struktura;
      znak  char(1);
      cislo uns(3) OVERLAY(znak); // binární číslo bez znaménka v 1 bajtu (3 dekadické číslice)
END-DS;

// překryvy - OVERLAY *NEXT - následující pozice v překrývaném podpoli
DCL-DS datum;
      RRRRMMDD zoned(8);
      RRRR     zoned(4) OVERLAY(*NEXT);
      MM       zoned(2) OVERLAY(RRRR: *NEXT);
      DD       zoned(2) OVERLAY(MM: *NEXT);
END-DS;

// pozice ve struktuře - POS
DCL-DS Qwc_JOBI0100_t QUALIFIED; // Retrieve Job Information API QUSRJOBI
       Job_Type     char(1)  POS(61);
       Job_Subtype  char(1)  POS(62);
       Run_Priority char(10) POS(73);
END-DS;
```

Podpole můžeme zapsat jednotlivě nebo je můžeme určit hromadně pomocí externích definic DDS. *Externí definice* samostatné datové struktury je určena klíčovým slovem EXT nebo EXTNAME, anebo LIKEREC. K externí definici lze přidávat další podpole (i s překryvem). Podpolem může být i jiná datová struktura určená slovem LIKEDS. Slovo QUALIFIED znamená, že podpole při použití musíme kvalifikovat jménem datové struktury, například *zakaznik.jmeno*.

```
// Vnořená datová struktura - pomocí LIKEDS
DCL-DS zakaznik QUALIFIED;
    cislo           int(10);
    jmeno           varchar(50);
    mesto           varchar(50);
    objednavky      LIKEDS(DatovaStruktura) DIM(100);
    pocetObjednavek int(10);
END-DS zakaznik;

// Vnořená datová struktura - přímo
DCL-DS zakaznik QUALIFIED;
    cislo           int(10);
    jmeno           varchar(50);
    mesto           varchar(50);
    DCL-DS DatovaStruktura DIM(10);
        zbozi char(50);
        DCL-SUBF packed packed(4: 0);
    END-DS;
    pocetObjednavek int(10);
END-DS zakaznik;

// jméno struktury je shodné s externím jménem
DCL-DS STAVYP EXT END-DS;  // v programu nesmí být DCL-F CENIKP

// jméno struktury není shodné s externím jménem
DCL-DS zaznam EXTNAME('STAVYP') END-DS;

// případné EXT, musí být zapsáno před EXTNAME
// *EXTDFT znamená předvolené hodnoty
DCL-DS zaznam EXT inz(*EXTDFT) EXTNAME('STAVYP') END-DS;
```

Vektor datových struktur

Datová struktura může mít *více výskytů* a představuje tak vlastně vektor datových struktur. Datová struktura s více výskyty je definována klíčovým slovem DIM. Navíc musí být označena slovem QUALIFIED. Datovou strukturu lze také definovat na základě jiné datové struktury pomocí slova LIKEDS.

Počet položek vektoru nebo počet výskytů datové struktury můžeme zjistit funkcí %ELEM.

Délku položky vektoru nebo podpole datové struktury můžeme určit funkcí %LEN nebo %SIZE. Rozdíl mezi těmito funkcemi je ten, že %LEN představuje délku číselné proměnné v počtu dekadických číslic, kdežto %SIZE délku v počtu bajtů. U znakových proměnných dávají obě funkce stejnou hodnotu, a to počet znaků. Na vnořené datové struktury nelze tyto funkce aplikovat.

```
// Datová struktura s pěti výskyty
DCL-DS VektorStruktur DIM(5);
    pole1 char(1);
    pole2 int(5);
END-DS;
```

Program VEKDS01 – vektor datových struktur

```
**FREE
// Databazovy soubor stavu
// -----------------------
dcl-f STAVYP keyed;

// Datova struktura opakovana "pocet" krat, definovana externe
// stejne jako format souboru stavu. Je oznacena jako kvalifikovana,
// abychom ji mohli pouzit jako vektor, cili pole.

dcl-DS Stavy dim(pocet) extname('STAVYP': 'STAVYPF0': *input) qualified;
    DruhMater char(2) overlay(MATER);
end-DS;

// Konstanta urcujici pocet opakovani struktury Stavy
// (nemusi byt definovana predem)
dcl-c pocet      10;

// Index do pole Stavy
dcl-s idx        int(10);

// Stradac mnozstvi o 1 cislici vetsi nez MNOZ
dcl-s MnozCelkem like(MNOZ: +1);

// Hlavni program
// --------------
MnozCelkem = 0;

// Prectu prvni zaznam souboru stavu do prvni polozky pole struktur
idx = 1;
read STAVYP Stavy(idx);

// Cyklus zpracovani - nactu vsechny zaznamy souboru do struktur,
// dokud se tam vejdou a pritom scitam mnozstvi
dow  not %eof and idx < %elem(Stavy);
    if Stavy(idx).DruhMater = '44' or Stavy(idx).DruhMater = '55';
        MnozCelkem += Stavy(idx).MNOZ;
    endif;
    idx += 1;
    read STAVYP Stavy(idx);
enddo;

dump(a) 'VEKDS01';
*InLR = *ON;
```

Datová struktura *Stavy* má 10 výstkytů - DIM(pocet), jejichž podpole jsou určena externím popisem pomocí klíčového slova EXTNAME. Podpole se čerpají ze souboru STAVYP a jeho formátu

STAVYPF0. Hodnota *INPUT znamená, že struktura bude obsahovat podpole shodná s poli formátu STAVYPF0 schopnými vstupu. U databázových souborů jsou ovšem všechna pole schopná vstupu (i výstupu). Právě tak by mohlo být použita hodnota *OUTPUT nebo *ALL. Kromě toho bychom mohli vynechat jméno formátu, protože fyzický databázový soubor může mít jen jeden formát. Hodnoty *INPUT a *OUTPUT jsou určena spíše pro obrazovkové soubory. Kromě těchto hodnot existuje ještě *KEY, což bychom zadali, kdybychom chtěli definovat jen klíčová pole databázového souboru. Podpole *DruhMater* překrývá podpole MATER přes první dva bajty použitím slova OVERLAY.

Výraz LIKE(MNOZ) znamená délku pole MNOZ a zápis **+1** značí, že proměnná *MnozCelkem* bude mít délku o 1 číslici větší (aby se do ní vešel součet hodnot množství) .

Příkaz *read* má v operandu zapsán výraz *Stavy(idx)*, což znamená, že přečtený záznam souboru STAVYP se umístí do výskytu datové struktury *Stavy* s pořadovým číslem *idx*.

Výraz *%elem(Stavy)* označuje počet výskytů struktury *Stavy*.

Výraz *Stavy(idx).MNOZ* znamená obsah proměnné MNOZ vzatý z idx-tého výskytu struktury *Stavy*.

Program VEKDS02 – vektor v datové struktuře, cyklus FOR

```
      // ************************************************************
      // *   Vektor v datove strukture, cyklus FOR
      // ************************************************************

      dcl-ds Struktura;
          Vektor      packed(5: 2) dim(5) ctdata perrcd(1) extfmt(s);
          Soucet      packed(6: 2) inz(0);
      end-ds;
      dcl-s idx       packed(4: 0);

      //  Hlavni program
      // ---------------
      for idx = 1 to %elem(Vektor);
          Soucet += Vektor(idx);
      endfor;
      dump(a) 'VEKDS02';
      *inlr = *on;

1...+.*. 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
**CTDATA Vektor - 5 mist, z toho dve desetinna
00100
00200
00300
00400
00500
```

Vektor je určen jako 5 číselných položek dlouhých 5 číslic s 2 desetinnými místy. Slovo CTDATA (compile time data) znamená, že hodnoty položek budou známy již při kompilaci programu. Musí být umístěny za koncem programu od 1. sloupce a uvedeny příkazem **CTDATA s mezerou, za níž je zapsáno jméno vektoru a případný komentář. Slovo PERRCD(1) (per record) znamená, že na každém řádku za řádkem **CTDATA je zapsána jedna položka. Slovo EXTFMT(S) znamená, že tvar čísel je zónový (znakový), bez vyznačené desetinné čárky (s případným záporným znaménkem skrytým v posledním znaku).

Příkaz *for – endfor* provádí tělo cyklu tolikrát, kolik má *Vektor* položek.

Data area

Datová oblast je objekt typu *DTAARA a vytváří se CL příkazem CRTDTAARA. V RPG se označuje klíčovým slovem **DTAARA**.

Klíčové slovo svazuje *existující datovou oblast* s jednoduchou proměnnou, datovou strukturou nebo podpolem datové struktury. Nelze je použít v podproceduře (o procedurách se jedná v jiném školicím materiálu).

DTAARA { { *AUTO } { *USRCTL } { (name) } }

- *AUTO - datová struktura je datovou strukturou datové oblasti (data area) bez řízení operacemi IN, OUT, UNLOCK. Pro použití těchto operací je nutné zadat ještě parametr *USRCTL. Není-li parametr *AUTO zadán, dosadí se *USRCTL.
- *USRCTL - datová struktura je datovou strukturou datové oblasti (data area) řízenou operacemi IN, OUT, UNLOCK.
- Parametr *name* může být
  - *znaková konstanta* tvaru (s velkými písmeny)
    `'KNIHOVNA/AREA'`
    `'*LIBL/AREA'`
    `'AREA'`
  - *znaková proměnná* obsahující znakovou konstantu podle výše uvedeného vzoru. Proměnná musí být nastavena před otevřením souboru, např. jako vstupní parametr programu nebo jako počáteční hodnota v definici dat klíčovým slovem INZ.

  Je-li parametr *name* zadán, musí být z parametrů poslední. Není-li zadán, je použito jméno objektu datové struktury (zde AREA).

DTAARA { (name) }

Tento tvar se použije u popisu jednoduché proměnné nebo podpole datové struktury. Nepovinný parametr *name* určuje *externí* jméno datové oblasti (viz výše). Není-li zadán, dosadí se jméno objektu.

```
// jednoduchá proměnná DAREA představuje jméno objektu *LIBL/AREA
DCL-S darea CHAR(15) DTAARA;

// jméno DSAREA představuje datovou strukturu datové oblasti *LIBL/AREA
DCL-DS dsarea DTAARA('AREA');
    podpole1 CHAR(10);
    podpole2 CHAR(5);
END-DS;

// Local data area (*LDA) s bezejmennou datovou strukturou
dcl-DS *N DTAARA;
   podpole   char(10);
   podpole2  char(5);
end-DS;
```

Příklad operací s datovou oblastí

```
dcl-ds dsarea DTAARA('AREA');
   podpole1 CHAR(10);
   podpole2 CHAR(5);
end-ds;

in *lock dsarea;         // zamknu
podpole1 = 'aaaaaaaaaa'; // naplním podpole
podpole2 = 'BBBBB';
out dsarea;              // přepíšu
in dsarea;               // přečtu
unlock dsarea;           // odemknu

dsply dsarea;
return;
```

Přesměrování souborů

Klíčová slova popisu souborů

EXTFILE (filename | *EXTDESC)

Určuje jméno existujícího souboru (objektu) pro dobu výpočtu (otevření a zpracování).

- Parametr *filename* může být *znaková konstanta* tvaru (s velkými písmeny a v apostrofech)
  `'KNIHOVNA/SOUBOR'`
  `'*LIBL/SOUBOR'`
  `'SOUBOR'`
  nebo *znaková proměnná* obsahující znakovou konstantu podle výše uvedeného vzoru. Proměnná musí být nastavena před otevřením souboru, např. jako vstupní parametr programu nebo jako počáteční hodnota v definici dat klíčovým slovem INZ.
- Parametr **EXTDESC* přebírá údaj z klíčového slova EXTDESC, které musí být také zadáno (viz příklad u klíčového slova EXTDESC).

EXTMBR (membername)

Určuje jméno členu (member) existujícího souboru pro dobu výpočtu (otevření a zpracování).

- Parametr *membername* může být
  znaková konstanta tvaru
  `'*ALL'`
  `'*FIRST'`
  `'MEMBER01'` (jméno členu v apostrofech)
  nebo znaková proměnná obsahující znakovou konstantu podle výše uvedeného vzoru. Proměnná musí být nastavena před otevřením souboru, např. jako vstupní parametr programu nebo jako počáteční hodnota v definici dat klíčovým slovem INZ.

EXTDESC (external-filename)

Určuje jméno existujícího souboru, ale jen pro kompilaci.

Parametr *external-filename* může být jen
znaková konstanta tvaru (s velkými písmeny a v apostrofech)
`'KNIHOVNA/SOUBOR'`
`'*LIBL/SOUBOR'`
`'SOUBOR'`

Program EXTFILE

```
*****************************************************************
*  Definice externího jména souboru pro výpočet
*  -------------------------------------------------------------
*  Program se volá CL příkazem
*  CALL PGM(EXTFILE) PARM('VZRPG_FREE/CENIKP')
*
*****************************************************************
 // rozhraní programu (program interface)
 DCL-PI *N extpgm('EXTFILE'); // program nemá interní jméno, jen externí
     filename char(21);       // vstupní parametr s plným jménem objektu
 END-PI;

 // Popis vstupního souboru
 DCL-F ceny EXTFILE(filename);

 read ceny;

 dsply NAZEV; // proměnná ze souboru CENIKP
 *inlr = *on;
```

Program EXTDESC

```
*****************************************************************
*  Definice externího jména souboru pro kompilaci i pro výpočet
*  -------------------------------------------------------------
*  Program se volá CL příkazem
*  CALL PGM(EXTDESC)
*
*****************************************************************
  dcl-f ceny EXTDESC('VZRPG_FREE/CENIKP') // určí objekt pro kompilaci
             EXTFILE(*EXTDESC)            // převezme objekt i pro výpočet
             EXTMBR('*FIRST');            // určí datový člen

  read ceny;

  dsply NAZEV;
  *inlr = *on;
```

Chybové stavy

Metody ošetřování mimořádných stavů v RPG

Mimořádné stavy výpočtu jsou takové stavy, které vzniknou v důsledku

- nežádoucího stavu (chyby) vstupu a výstupu (např. pokus o čtení z databázového souboru, kterému chybí člen - member),
- neobvyklého chování uživatele (např. násilné zrušení úlohy),
- chyby programu (např. neošetřené přeplnění podsouboru),
- vnějších okolností (např. porucha hardware).

Zde probereme mimořádné stavy vstupu a výstupu (I/O - Input/Output) a chyby programu. Vlivy vnějších okolností se řeší kombinací systémových a programových prostředků, např. ukládáním a obnovou objektů (save, restore), žurnálováním databází a řízením transakcí (commitment control).

Následující seznam ukazuje způsoby zachycování a ošetřování chyb v pořadí priorit. Není-li použit jeden, uplatní se další v pořadí.

- *Zápis modifikátoru* E k příkazu pro vstup a výstup, např. OPEN(E), CHAIN(E), READ(E), WRITE(E) apod. nebo i příkazu pro manipulaci s textem např. CHECK(E), CHECKR(E), SCAN(E), SUBST(E), XLATE(E), ale i některých jiných, např. CALL(E). Způsobí to, že po provedení takové operace můžeme testovat funkci **%ERROR**, která je buď vypnutá (*OFF – bez chyby) nebo zapnutá (*ON – chyba). Navíc lze testovat funkci **%STATUS** poskytující kód chyby.
- *Monitorování úseků programu* příkazem **MONITOR** se sdruženými příkazy **ON-ERROR** a **ENDMON**. U příkazů obsahujících *vestavěné funkce* pro manipulaci s textem (např. %SUBST, %DEC) nelze použít modifikátor E. Monitorováním takových příkazů však můžeme chyby zachytit a zpracovat. Monitor lze použít i pro příkazy vstupu/výstupu a ostatní příkazy, které mohou způsobit mimořádné stavy. Je tudíž obecnější než modifikátor E a testování funkce %ERROR. Je ovšem k dispozici teprve od verze 5.1 operačního systému. V příkazu ON-ERROR lze určit, které mimořádné stavy chceme zachytit. Můžeme zadat konkrétní číslo chyby, např. 0100 pro chybu délky nebo pozice při manipulaci s textem, nebo souhrnný kód, např. *ALL (pro všechny druhy chyb).
- *Definování speciálních podprogramů*, do nichž přejde výpočet, když dojde k mimořádnému stavu. K zachycení chyb vstupu /výstupu slouží podprogramy pro jednotlivé soubory, které se nazývají *File Exception/Error Subroutine* a označují se kódem **INFSR** u popisu příslušného souboru v popisu souboru. K zachycení programových chyb lze definovat speciální podprogram, který se nazývá *Program Exception/Error Subroutine* a označuje se kódem ***PSSR** v příkazu BEGSR. Tento podprogram lze zadat také v popisu souboru jako jméno u INFSR.

Ve všech výše uvedených způsobech zachycování chyb lze využít *speciální datové struktury*, které jazyk RPG poskytuje. Tam jsou obsaženy i jiné informace než o chybě, takže je lze využívat i pro jiný účel než je ošetření chyby.

Pro každý datový soubor (databázový, obrazovkový, tiskový, atd.) může být definována samostatná datová oblast členěná podle struktury zvané *File Information Data Structure*, zkráceně **INFDS**. Informace můžeme využívat i k normální práci v programu (např. *zjišťovat hexadecimální kód funkční klávesy* apod.).

Pro ošetřování *chyb programu* lze využít speciální datovou strukturu zvanou *Program Status Data Structure*, zkráceně **PSDS**. Rovněž v této struktuře jsou kromě chyby obsaženy i jiné informace o stavu programu (např jméno právě probíhajícího programu).

Využití funkce %ERROR

U řady příkazů můžeme zadat modifikátor E, který umožňuje použít následnou funkci %ERROR. Tato funkce má hodnotu *OFF, nedojde-li v příkazu k chybě a hodnotu *ON, dojde-li k chybě. Jde většinou o příkazy vstupu/výstupu a příkazy pro manipulaci s textem. Jejich seznam lze nalézt v referenční příručce *ILE/RPG Reference* v kapitole *20. Operations* společně s operacemi používajícími jiné modifikátory.

Informační datová struktura souboru (INFDS)

Informační datová struktura souboru se definuje takto (příklad):

```
// file information data structure

dcl-f STAVYP disk(*ext) INFDS(infoStavy);

DCL-DS infoStavy;
    file     *FILE;              // File name
    open_ind ind      pos(9);    // File open '1'
    eof_ind  ind      pos(10);   // File at eof '1'
    status   *STATUS;            // Status code
    opcode   *OPCODE;            // Last opcode
    record   *RECORD;            // Record name
    msgid    char(7)  pos(46);   // Error MSGID
    fileName char(10) pos(83);   // File name
    library  char(10) pos(93);   // Library name
    rec_fmt  char(10) pos(261);  // Record format
END-DS;
```

Podpole datové struktury mají své pevně dané pozice, které lze v některých případech nahradit klíčovým slovem, např. *FILE (odpovídá pozicím 1 až 8) nebo *STATUS (odpovídá pozicím 11 až 15). Jména podpolí použijeme jako holá. Zapíšeme-li u struktury slovo QUALIFIED,

```
DCL-DS infoStavy Qualified;
   ...
```

musíme odkaz na podpole kvalifikovat jménem struktury, např. `infoStavy.open_ind`.

Informace, které operační systém dosazuje do datové struktury se dělí do čtyř úseků:

- **File Feedback** (pozice **1** až **80)** - obsahuje údaje o souboru, které jsou specifické pro RPG.
- **Open Feedback** (pozice **81** až **240)** - obsahuje údaje o otevřeném souboru; některé údaje jsou závislé na druhu souboru.
- **I/O Feedback** (pozice **241** až **286)** - obsahuje údaje o poslední I/O operaci; některé údaje jsou platné jen pro určitý druh souboru.
- **Device Specific Feedback** (**367** a dále) - obsahuje informace o poslední I/O operaci, závislé na druhu souboru (disk, obrazovka, tiskárna, atd.).

Podrobný popis najdeme v příručce *ILE/RPG Reference SC09-2508, část File and Program Exception/Errors*.

Podprogram pro obsluhu stavu souboru (INFSR)

Jde o obyčejný podprogram, který si pojmenujeme (SUBR1) a zapíšeme mezi příkazy BEGSR a ENDSR, a jeho jméno zapíšeme do popisu souboru ke klíčovému slovu INFSR. Soubor CENY použije podprogram *subrCeny*, tj. když dojde k chybě v souboru CENIKP, výpočet přejde do podprogramu *subrCeny*.

V podprogramu můžeme zapsat libovolné příkazy, dokonce i ty, které používají soubor CENIKP. V posledním případě může dojít k opětnému vyvolání podprogramu *subrCeny*, aniž by se z něj před tím výpočet vrátil. Pak bychom měli v podprogramu zjišťovat, pokolikáté byl program *subrCeny* vyvolán, aniž by byl předtím opuštěn, a podle toho se zařídit. Můžeme např. použít proměnnou, do

níž budeme přičítat jedničku, když výpočet vstoupí do podprogramu, a odečítat jedničku, když jej opouští.

```
dcl-f CENIKP keyed usage(*update) INFDS(infoCeny)
                                  INFSR(subrCeny);
chain CENIKP; // příkaz CHAIN nemá modifikátor E
 ...
begsr subrCEny;
 ...
endsr;
```

*Informační struktura stavu programu (PSDS) a podprogram *PSSR*

Datová struktura je označena slovem PSDS, které ji jednoznačně určuje. Nemusí být pojmenována, v tom případě místo jména zapíšeme symbol *N. Údaje jsou obsaženy ve stanovené délce v pozicích 1 až 429. U některých údajů lze místo pozic použít klíčové slovo, např. *STATUS (pozice 11 až 15, stejně jako v INFDS) nebo *PROC (pozice 1 až 10), která obsahuje jméno procedury nebo programu.

```
dcl-ds progStruct PSDS;
    proc_name   *PROC;      // Procedure name
    pgm_status  *STATUS;    // Status code
    parms       *PARMS;     // Number of parameters passed
    prog_MSGID  char(7) pos(40);   // Message ID
end-ds;
```

Podpole této datové struktury můžeme v programu analyzovat kdykoliv, ale význam to má tehdy, když použijeme speciální podprogram ***PSSR**, do nějž přejde výpočet v okamžiku, kdy nastane programová chyba, např. dělení nulou (status 00102) nebo chybný index vektoru (status 00121) apod.

```
begsr *PSSR;
 ...
endsr;
```

Podrobný popis datové struktury najdeme v příručce *ILE/RPG Reference SC09-2508, část File and Program Exception/Errors*.

Program TEXTCH

V programu jsou dva monitorované úseky. První monitor zachytí chybu 100 ve funkci %SUBST, protože funkce %SCAN nenajde vzorek *'IJK'* v textu *TextFix* a dosadí pozici 0. Pozice 0 však není platná a způsobí chybu 100 ve funkci %SUBST. Kódy chyb lze nalézt v referenční příručce *ILE RPG Reference, SC09-2508,* v kapitole *File and Program Exception/Errors, odst. Program Status Codes*.

Druhý monitor zachytí chybu v druhém příkazu – ON-ERROR *ALL, protože funkce %SCAN nenajde vzorek nulové délky v textu TextVar a dosadí pozici 0 pro funkci %SUBST, která způsobí opět chybu 100. Protože v prvním příkazu ON-ERROR je číslo chyby 101, neuplatní se.

```
        dcl-f qprint printer(100);

        dcl-s TextFix  char(20) inz('ABCDEFGH');
        dcl-s TextFix2 char(30) inz('abcdefgh');

        dcl-s TextVar  varchar(20) inz('ABCDEFGH');
        dcl-s TextVar2 varchar(30) inz('abcdefgh');

        dcl-s Vzorek   varchar(10);
        dcl-s Delka    packed(3: 0);

        dcl-ds progStruct PSDS;
            proc_name   *PROC;      // Procedure name
            pgm_status  *STATUS;    // Status code
            parms       *PARMS;     // Number of parameters passed
            prog_MSGID  char(7) pos(40);   // Message ID
        end-ds;

        // Textova promenna s pevnou delkou

        Vzorek = 'IJK';   // hledam vzorek, ktery v retezci neni
        MONITOR;
           TextFix2 = %subst(TextFix:
                             %scan(Vzorek: TextFix: 1):
                             %len(Vzorek));
           Delka = %len(TextFix2);
           except    fix;
        ON-ERROR  100;
           dump(a)   'PoziceFix';   // leva zavorka musi byt tesne za znakem prikazu
        ENDMON;

        // Textova promenna s promennou delkou

        Vzorek = '';    // hledam vzorek nulove delky
        MONITOR;
           TextVar2 = %subst(TextVar:
                             %scan(Vzorek: TextVar: 1):
                             %len(Vzorek));
           Delka = %len(TextVar2);
           except    var;
        ON-ERROR  101;      // tady jsem se spletl a testuji nespravny kod
           dump(a)   'DelkaVar1';   // leva zavorka musi byt tesne za znakem prikazu
        ON-ERROR  *ALL;     // zde se vsak chyba zachyti
           dump(a)   'DelkaVar2';
        ENDMON;

        *inlr = *on;

      *   Popis vystupu - testovaci vypis

12345O*ilename++DF..N01N02N03Excnam++++B++A++Sb+Sa+.........................Comments++++++++++++
     oqprint    e            fix
     O*............N01N02N03Field+++++++++YB.End++PConstant/editword/DTformat++Comments++++++++++++
     o                                           'TextFix2 '
```

```
     o                       Delka         3     +1
     o                       TextFix2            +2

     oqprint    e            var
     o                                             'TextVar2 '
     o                       Delka         3     +1
     o                       TextVar2            +2
```

Některé další příkazy a funkce

Příkazy

EVALR (Evaluate expression, right adjust)
EVAL-CORR (Assign corresponding subfields)
OPEN (Open File for Processing)
ON-EXIT (On Exit)
SORTA (Sort an Array)

Vestavěné funkce

%ABS (Absolute Value of Expression)
%ADDR(Get Address of Variable)
%ALLOC (Allocate Storage)
%BITAND (Bitwise AND Operation)
%BITNOT (Invert Bits)
%BITOR (Bitwise OR Operation)
%BITXOR (Bitwise Exclusive-OR Operation)
%DIV (Return Integer Portion of Quotient)
%EDITFLT (Convert to Float External Representation)
%HOURS (Number of Hours)
%KDS (Search Arguments in Data Structure)
%LOOKUPxx (Look Up an Array Element)
%MAX (Maximum Value)
%MIN (Minimum Value)
%MINUTES (Number of Minutes)
%MONTHS (Number of Months)
%MSECONDS (Number of Microseconds)
%OCCUR (Set/Get Occurrence of a Data Structure)
%OPEN (Return File Open Condition)
%PADDR (Get Procedure Address)
%PARMS (Return Number of Parameters)
%PARMNUM (Return Parameter Number)
%REALLOC (Reallocate Storage)
%SCAN (Scan for Characters)
%SCANR (Scan Reverse for Characters)
%SCANRPL (Scan and Replace Characters)
%SECONDS (Number of Seconds)
%SHTDN (Shut Down)
%SQRT (Square Root of Expression)
%STATUS (Return File or Program Status)
%STR (Get or Store Null-Terminated String)
%SUBARR (Set/Get Portion of an Array)
%SUBDT (Extract a Portion of a Date, Time, or Timestamp)
%TIME (Convert to Time)
%TIMESTAMP (Convert to Timestamp)
%TLOOKUPxx (Look Up a Table Element)
%UNS (Convert to Unsigned Format)
%XFOOT (Sum Array Expression Elements)
%YEARS (Number of Years)

Ladění programů v RPG IV

Kromě běžnných ladicích prostředků jako je příkaz DSPJOBLOG nebo výpis paměti (DUMP) je nejdůležitějším ladicím prostředkem v prostředí RPG IV ladicí program volaný příkazem STRDBG.

Postup ladění

1. 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. Předvolená hodnota je *STMT, která nepracuje s obrazem zdrojového textu nebo protokolu, ale jen s pořadovými čísly příkazů, a to podle protokolu o kompilaci.
2. Vydáme příkaz **STRDBG** s příslušnými parametry, čímž zahájíme ladicí režim (debug mode).
3. Zobrazí se zdrojový text programu s příkazovým řádkem a přehledem funkčních kláves.

```
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
```

4. Zadáme alespoň jeden bod zastavení (breakpoint) nastavením kurzoru na některý řádek zobrazeného zdrojového textu a stiskem klávesy F6.
5. Pomocí klávesy F21 vyvoláme příkazový řádek CL a *vyvoláme program* příkazem CALL.
6. Používáme funkce ladicího programu (vydáváme příkazy na příkazovém řádku, nebo nastavujeme kurzor a tiskneme klávesové funkce).
7. **Ukončíme** ladicí režim příkazem **ENDDBG**.

Stručný přehled ladicích příkazů

Při ladění použijeme klávesové funkce, ale můžeme používat příkazy ladicího programu, které zadáváme na jeho příkazovém řádku (označeném slovem Debug). Přehled příkazů s návodem a příklady získáme stiskem klávesy F1 nebo Help, když je kurzor na příkazovém řádku. Přehled dosud vydaných příkazů a jejich výsledků získáme klávesou Enter. Klávesou Enter se z přehledu vracíme zpět.

Podtržené začátky jmen příkazů představují jejich rovnocenné zkratky.

Příkaz	*Význam příkazu*
ATTR jméno-proměnné	zjištění atributů proměnné
BREAK číslo-příkazu {WHEN logický-výraz}	nastavení bodu zastavení
CLEAR číslo-příkazu {PGM | WATCH {ALL | watch-číslo} }	vyčištění bodů zastavení či pozorovacích míst
DISPLAY { MODULE jméno-modulu | EQUATE }	zobrazení modulu nebo EQUATE jmen
EQUATE jméno výraz	definice EQUATE jména (zkratky pro výraz)
EVAL výraz {C | X {délka}}	vyhodnocení výrazu s příp. změnou proměnné
FIND {číslo-příkazu | řetězec | PREVIOUS | NEXT }	vyhledání příkazu nebo řetězce
QUAL čislo-příkazu | jméno-procedury/číslo-příkazu	rozsah platnosti pro EVAL nebo WATCH
SET { volba hodnota }	zobrazení nebo nastavení voleb pro ladění
STEP {1 | číslo-příkazu } { OVER | INTO }	krokování programu
WATCH výraz délka	nastavení pozorované adresy (proměnné)
UP počet-řádků	posun obrazu nahoru
DOWN počet-řádků	posun obrazu dolů
LEFT počet-sloupců	posun obrazu vlevo
RIGHT počet-sloupců	posun obrazu vpravo

TOP	posun obrazu na první řádek
BOTTOM	posun obrazu na poslední řádek
NEXT	posun obrazu na další bod zastavení
PREVIOUS	posun obrazu na předchozí bod zastavení
HELP	zobrazení návodu k použití ladicích příkladů

Příklady ladicích příkazů

Příkaz ATTR s odpovědí:

```
 Debug . . .   ATTR mnoz

 F3=End program   F6=Add/Clear breakpoint   F10=Step   F11=Display variable
 F12=Resume       F17=Watch variable   F18=Work with watch   F24=More keys
 TYPE = PACKED(10,2), LENGTH = 6 BYTES
```

Příkaz BREAK s odpovědí:

```
    BREAK 47
Breakpoint added to line 47.
```

Podmíněný příkaz BREAK s odpovědí:

```
    br 56 when *in12 <> '1'
Breakpoint replaced at line 56.
```

Program se pak zastaví na příkazu č. 56, jestliže indikátor 12 bude vypnutý (’0’). Jestliže v tom okamžiku zobrazíme proměnnou *IN12, nemusíme nutně vidět hodnotu ’0’. Ta se do proměnné může dostat až v dalším kroku F10, kdy se příkaz 56 dokončí.

Příkaz WATCH s odpovědí. Proměnná MNOZ bude sledována, zda se změní. po změně se ladicí program zastaví za příkazem, který způsobil jeho změnu.

```
    watch mnoz
Watch number 1 added.
```

V dalším výpočtu se po změně proměnné ladicí program zastaví za příkazem, který způsobil jeho změnu a vypíše

```
Watch number 1 at line 54, variable: MNOZ
```

V tomto okamžiku můžeme vypsat hodnotu této proměnné obvyklým způsobem.

Příkaz CLEAR WATCH ALL s odpovědí:

```
    clear watch all
All debug watches cleared in this job.
```

Příkaz CLEAR PGM s odpovědí:

```
    clear pgm
All breakpoints removed from program STAPOR.
```

Příkazy EVAL s odpovědí:

```
    eval mater : x
         00000     F0F0F0F0 F1...... ........ ........   - 00001...........

    eval %addr(mater)
%ADDR(MATER) = SPP:C07542EA63077D48

    eval mater = '00002'
MATER = '00002' = '00002'
```

V ladicím režimu lze použít i další CL příkazy.

ADDBKP - add breakpoint - přidat bod přerušení,
RMVBKP - remove breakpoint - odstranit bod přerušení,
ADDTRC - add trace - přidat stopu (rozsah čísel sledovaných příkazů),
RMVTRC - remove trace - odstranit stopu,
DSPPGMVAR - display program variable - zobrazit programovou proměnnou,
CHGPGMVAR - change program variable - změnit programovou proměnnou,
DSPTRCDTA - display trace data - zobrazit údaje o stopě výpočtu od posledního výmazu,
CLRTRCDTA - cleat trace data - smazat údaje o stopě výpočtu,
DSPTRC - display trace - zobrazit všechny rozsahy stop výpočtu.