FUJITSU

Fujitsu Software NetCOBOL V13.0

Syntax Samples

J2UL-2878-01ENZ0(00)
July 2023

Preface

As COBOL has grown and evolved, many new features have been added to the base language. Quite often, these are features that are in the draft version of the next COBOL standard. Other times, the features were added in response to user requests.

Often, features that have been part of the COBOL language are unused because they are not properly understood. Examples of actual COBOL statement usage are invaluable in helping the programmer figure out how to use a particular COBOL feature.

This manual, used in conjunction with the Language Reference Manual, provides syntax samples for several clauses and statements in order to clarify their usage. The syntax samples will be comparatively easy to understand because each sample has been kept to around 50 lines. This manual also introduces some coding techniques for more effective use. When introduced, these techniques are noted by text such as "This example shows an effective way of using this function".

Intended Readers

Prior to using NetCOBOL it is assumed that you have the following knowledge:

- You have some basic understanding as to how to navigate through and use the Microsoft Windows, Solaris or Linux operating system.
- You have a basic understanding of the COBOL language.

Structure of This Manual

This manual contains the following information.

COBOL Syntax Samples

Various sample programs illustrating the use of a variety of COBOL statements. Many of these are standard COBOL, and others are extensions offered by NetCOBOL.

Conventions

The conventions used in this manual are described below.

System-specific Functions

Some parts of the COBOL common syntax described in this manual depend on system functions, and differ among systems.

Such parts are indicated by the following system names:

Indication	Corresponding system	Corresponding product
HP	HP-UX	COBOL85 V20L11
Solaris	Oracle Solaris 11 Oracle Solaris 10	NetCOBOL (32bit) V10
Linux	Red Hat Enterprise Linux 5 (for x86) Red Hat Enterprise Linux 5 (for Intel64) 32bit	NetCOBOL (32bit) V7.3
LinuxIPF	Red Hat Enterprise Linux 5 (for Intel Itanium)	NetCOBOL V9.0
Linux64	Red Hat Enterprise Linux 9 (for Intel64) Red Hat Enterprise Linux 8 (for Intel64)	NetCOBOL(64bit) V13
Win32	Windows Server 2016 Windows Server 2012 R2 Windows Server 2012 Windows 10	NetCOBOL (32bit) V11 NetCOBOL (32bit) V11a
Winx64	Windows Server 2022 (*1) Windows Server 2019 (*2) Windows Server 2016	NetCOBOL (64bit) V12 NetCOBOL (64bit) V12a

Indication	Corresponding system	Corresponding product
	Windows Server 2012 R2 Windows Server 2012 Windows 11 (x64) (*1) Windows 10 (x64)	
.NET	Windows Server 2019 (*2) Windows Server 2016 Windows Server 2012 R2 Windows 10	NetCOBOL for .NET V8

*1: Windows Server 2022 and Windows 11 are supported on NetCOBOL V12a or later.

*2: Windows Server 2019 is supported on NetCOBOL V12.2 or later, and NetCOBOL for .NET V8 or later.

Abbreviations

The following abbreviations are used in this manual:

Product Name	Abbreviation
Red Hat(R) Enterprise Linux(R)	Red Hat Enterprise Linux
Microsoft(R) Windows Server(R) 2022 Datacenter Microsoft(R) Windows Server(R) 2022 Standard	Windows Server 2022
Microsoft(R) Windows Server(R) 2019 Datacenter Microsoft(R) Windows Server(R) 2019 Standard	Windows Server 2019
Microsoft(R) Windows Server(R) 2016 Datacenter Microsoft(R) Windows Server(R) 2016 Standard	Windows Server 2016
Microsoft(R) Windows Server(R) 2012 R2 Datacenter Microsoft(R) Windows Server(R) 2012 R2 Standard Microsoft(R) Windows Server(R) 2012 R2 Foundation	Windows Server 2012 R2
Microsoft(R) Windows Server(R) 2012 Datacenter Microsoft(R) Windows Server(R) 2012 Standard Microsoft(R) Windows Server(R) 2012 Foundation	Windows Server 2012
Windows(R) 11 Home Windows(R) 11 Pro Windows(R) 11 Enterprise Windows(R) 11 Education	Windows 11 or Windows 11 (x64)
Windows(R) 10 Home Windows(R) 10 Pro Windows(R) 10 Enterprise Windows(R) 10 Education	Windows 10 or Windows 10(x64)
Oracle Solaris	Solaris

Trademarks

- Microsoft, Windows, Windows Server, and .NET are trademarks of the Microsoft group of companies.

- Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.
- HP-UX is a trademark of HP Hewlett Packard Group LLC.
- Linux(R) is the registered trademark of Linus Torvalds in the U.S. and other countries.
- Red Hat and Red Hat Enterprise Linux are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.
- Intel and Itanium are trademarks of Intel Corporation or its subsidiaries.
- All other trademarks are the property of their respective owners.

Export Controls

Exportation/release of this document may require necessary procedures in accordance with the regulations of your resident country and/or US export control laws.

July 2023

Contents

Chapter 1 COBOL Syntax Samples

Note

In .NET, compiler option "MAIN" cannot be specified for @OPTIONS compiler directing statement. Specify the cobolc command option "/main".

1.1 Concatenation

The concatenation expression is used for nonnumeric literals.

The concatenation expression is useful for adding control characters because it allows nonnumeric literals, symbolic constants and hexadecimal literals to be connected.

Use caution as Windows functions are invoked in this sample and USER32.LIB stored in the COBOL installation folder (directory) must be included at linkage.

This sample applies to Win32. However, similar processing will be required for linking to C with an operating system other than Win32.

```
000010 @OPTIONS MAIN,ALPHAL(WORD)
000020*---------------------------------------------------------------------
000030* The concatenation expression can be used to connect nonnumeric
000035* literals.
000040*---------------------------------------------------------------------
000050 IDENTIFICATION   DIVISION.
000060 PROGRAM-ID.      SAMPLE.
000070 DATA             DIVISION.
000080 WORKING-STORAGE  SECTION.
000090 01 TITLE-TXT     PIC X(20).
000100 01 MSG-TXT       PIC X(20).
000110 01 RET           PIC S9(9) COMP-5 VALUE 0.
000120 PROCEDURE        DIVISION.
000130*---------------------------------------------------------------------
000140* Nonnumeric literals can be connected using &.
000150* As shown in the following example, this expression is useful for
000155* setting null terminating characters when linking to C.
000160*  -> When passing character string data to C, a null character is
000170*     required at the end of the character string.
000180*     While reference modification could be used to accomplish the same
000185*     result, the concatenation expression makes for easy and elegant
000187*     coding.
000190*---------------------------------------------------------------------
000200     MOVE "SAMPLE"        & X"00" TO TITLE-TXT.
000210     MOVE "Hello COBOL!!" & X"00" TO MSG-TXT.
000220*---------------------------------------------------------------------
000230     CALL "MessageBoxA" WITH STDCALL USING BY VALUE     0
000240                                           BY REFERENCE MSG-TXT
000250                                           BY REFERENCE TITLE-TXT
000260                                           BY VALUE     1
000270                                     RETURNING RET.
000280     EXIT PROGRAM.
000290 END PROGRAM SAMPLE.
```

1.2 Qualification

When using COBOL the programmer may use data or paragraph names that are not unique in the program, provided that the data names occur under a unique group level. For example, when information of a specific type such as a date is used for several items, the code will be easy to read if the same data name is used.

When a data name is duplicated, each name is qualified so that it can be uniquely identified. OF or IN is used to clearly specify the name of the group to which the data belongs.

```
000010 @OPTIONS MAIN
000020*-------------------------------------------------------------------
000030* Uniqueness of qualification can be used to distinguish a data item
000035* from another of the  same name.
000040*-------------------------------------------------------------------
000050 IDENTIFICATION     DIVISION.
000060 PROGRAM-ID.        SAMPLE.
000070 DATA               DIVISION.
000080 WORKING-STORAGE    SECTION.
000090*-------------------------------------------------------------------
000100* The same data name can be assigned to data that has the same meaning.
000110*-------------------------------------------------------------------
000120 01 BIRTHDAY.
000130    02 YYYY         PIC 9(4).
000140    02 MMDD         PIC 9(4).
000150 01 TODAY.
000160    02 YYYY         PIC 9(4).
000170    02 MMDD         PIC 9(4).
000180*-------------------------------------------------------------------
000190 01 AGE             PIC ZZ9.
000200 PROCEDURE          DIVISION.
000210     DISPLAY "When is your birthday?  Example:  19690123 >>" WITH NO
000215              ADVANCING.
000220     ACCEPT BIRTHDAY.
000230     MOVE FUNCTION CURRENT-DATE TO TODAY.
000240*-------------------------------------------------------------------
000250* OF is used to qualify data so that the data can be identified when it
000255* is referenced.
000260*-------------------------------------------------------------------
000270     IF MMDD OF BIRTHDAY <= MMDD OF TODAY THEN
000280       COMPUTE AGE = YYYY OF TODAY - YYYY OF BIRTHDAY
000290     ELSE
000300       COMPUTE AGE = YYYY OF TODAY - YYYY OF BIRTHDAY - 1
000310     END-IF.
000320*-------------------------------------------------------------------
000330     DISPLAY "You are" AGE " years old."
000340     EXIT PROGRAM.
000350 END PROGRAM SAMPLE.
```

1.3 Subscripting

When a data structure is to be used repeatedly, a table is defined using the OCCURS clause. A subscript is used to reference the individual table elements.

When using subscripting, be careful about setting and referencing elements outside the range defined with the OCCURS clause. Referencing an item using a subscript that is not within the bounds of the table can destroy an area of memory, or cause execution to terminate abnormally.

```
000010 @OPTIONS MAIN
000020*---------------------------------------------------------------------
000030* Data declared in an array can be referenced using a subscript.
000040*---------------------------------------------------------------------
000050 IDENTIFICATION  DIVISION.
000060 PROGRAM-ID.     SAMPLE.
000070 DATA            DIVISION.
000080 WORKING-STORAGE SECTION.
000090 01 ALP          PIC X(26) VALUE "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
000100 01             REDEFINES ALP.
000110   02 CHAR         OCCURS 26 TIMES PIC X.
000120 01 IN-DATA      PIC X.
000130 01 COUNTER      PIC 9(2).
000140 PROCEDURE       DIVISION.
000150     DISPLAY "Please input one upper-case letter. >>" WITH NO
000155             ADVANCING.
000160     ACCEPT IN-DATA.
000170     PERFORM TEST BEFORE
000180             VARYING COUNTER FROM 1 BY 1 UNTIL COUNTER > 26
000190*---------------------------------------------------------------------
000200* COUNTER is used as a subscript for CHAR to allow comparison with the
000210* input character.
000220*---------------------------------------------------------------------
000230       IF IN-DATA = CHAR(COUNTER) THEN
000240*---------------------------------------------------------------------
000250         EXIT PERFORM
000260       END-IF
000270     END-PERFORM.
000280     IF COUNTER <= 26 THEN
000290       DISPLAY IN-DATA " is character "
000295               COUNTER " in alphabetical order."
000300     ELSE
000310       DISPLAY "The input character is incorrect."
000320     END-IF.
000330 END PROGRAM SAMPLE.
```

Uniqueness of qualification and subscript reference can also be combined.

```
000010 @OPTIONS MAIN
000020*---------------------------------------------------------------------
000030* Data declared in an array is uniquely qualified.
000040*---------------------------------------------------------------------
000050 IDENTIFICATION   DIVISION.
000060 PROGRAM-ID.      SAMPLE.
000070 DATA             DIVISION.
000080 WORKING-STORAGE  SECTION.
000090 01 UPPER-CASE.
000100    02 ALP        PIC X(26) VALUE "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
000110    02           REDEFINES ALP.
000120      03 CHAR       OCCURS 26 TIMES PIC X.
000130 01 LOWER-CASE.
000140    02 ALP        PIC X(26) VALUE "abcdefghijklmnopqrstuvwxyz".
000150    02           REDEFINES ALP.
000160      03 CHAR       OCCURS 26 TIMES PIC X.
000170 01 IN-DATA       PIC X.
000180 01 COUNTER       PIC 9(2).
000190 PROCEDURE        DIVISION.
000200     DISPLAY "Please input one upper-case letter. >>" WITH NO
000205             ADVANCING.
```

```
000210     ACCEPT  IN-DATA.
000220     PERFORM TEST BEFORE
000230             VARYING COUNTER FROM 1 BY 1 UNTIL COUNTER > 26
000240*-----------------------------------------------------------------------
000250* COUNTER is a subscript used to compare the input character.
000260* The subscript is specified after the qualification.
000280*-----------------------------------------------------------------------
000290       IF IN-DATA = CHAR OF UPPER-CASE (COUNTER) THEN
000300*-----------------------------------------------------------------------
000310         EXIT PERFORM
000320       END-IF
000330     END-PERFORM.
000340     IF COUNTER <= 26 THEN
000350        DISPLAY "The lowercase letter corresponding to " IN-DATA " is "
000360               CHAR OF LOWER-CASE (COUNTER)"."
000370     ELSE
000380        DISPLAY "The input character is incorrect."
000390     END-IF.
000400 END PROGRAM SAMPLE.
```

1.4 Reference Modification

Reference modification is used to refer to a portion of a character data item.

When using reference modification, be careful about setting and referencing outside the scope, or length of the data field being referenced. Reference modification that exceeds the actual area length can destroy data or cause execution to terminate abnormally by referencing an incorrect area of memory.

```
000010 @OPTIONS MAIN
000020*-----------------------------------------------------------------------
000030* Reference modification enables part of character string data to be
000035* specified.
000040*-----------------------------------------------------------------------
000050 IDENTIFICATION  DIVISION.
000060 PROGRAM-ID.     SAMPLE.
000070 DATA            DIVISION.
000080 WORKING-STORAGE SECTION.
000090 01 ALP          PIC X(26) VALUE "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
000100 01 IN-DATA      PIC X.
000110 01 COUNTER      PIC 9(2).
000120 PROCEDURE       DIVISION.
000130     DISPLAY "Please input one upper-case letter. >>" WITH NO ADVANCING.
000140     ACCEPT IN-DATA.
000150     PERFORM TEST BEFORE
000160             VARYING COUNTER FROM 1 BY 1 UNTIL COUNTER > 26
000170*-----------------------------------------------------------------------
000180* Reference modification can specify character data of
000185* (starting location:  Length).
000190*-----------------------------------------------------------------------
000200       IF IN-DATA = ALP(COUNTER:1) THEN
000210*-----------------------------------------------------------------------
000220         EXIT PERFORM
000230       END-IF
000240     END-PERFORM.
000250     IF COUNTER <= 26 THEN
000260       DISPLAY IN-DATA " is character " COUNTER " in alphabetical order."
000270     ELSE
000280       DISPLAY "The input character is incorrect."
000290     END-IF.
000300 END PROGRAM SAMPLE.
```

Uniqueness of qualification and reference modification may also be combined.

```
000010 @OPTIONS MAIN
000020*---------------------------------------------------------------------
000030* A data name that must be uniquely qualified can also be reference
000035* modified.
000040*---------------------------------------------------------------------
000050 IDENTIFICATION    DIVISION.
000060 PROGRAM-ID.       SAMPLE.
000070 DATA              DIVISION.
000080 WORKING-STORAGE   SECTION.
000090 01 UPPER-CASE.
000100    02 ALP         PIC X(26) VALUE "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
000110 01 LOWER-CASE.
000120    02 ALP         PIC X(26) VALUE "abcdefghijklmnopqrstuvwxyz".
000130 01 IN-DATA        PIC X.
000140 01 COUNTER        PIC 9(2).
000150 PROCEDURE         DIVISION.
000160     DISPLAY "Please input one upper-case letter. >>" WITH NO ADVANCING.
000170     ACCEPT IN-DATA.
000180     PERFORM TEST BEFORE
000190             VARYING COUNTER FROM 1 BY 1 UNTIL COUNTER > 26
000200*---------------------------------------------------------------------
000210* COUNTER is used as a subscript and the input character is compared.
000220* The reference modifier is specified after the qualification to
000230* reference-modify the uniquely qualified data.
000240*---------------------------------------------------------------------
000250       IF IN-DATA = ALP OF UPPER-CASE (COUNTER:1) THEN
000260*---------------------------------------------------------------------
000270        EXIT PERFORM
000280      END-IF
000290     END-PERFORM.
000300     IF COUNTER <= 26 THEN
000310      DISPLAY "The lower-case letter corresponding to " IN-DATA " is "
000320              ALP OF LOWER-CASE (COUNTER:1)"."
000330     ELSE
000340      DISPLAY "The input character is incorrect."
000350     END-IF.
000360 END PROGRAM SAMPLE.
```

1.5 Pointer

The pointer data item is used to reference data based on a specific memory address. In this case, the data attribute and offset is defined using a based-storage section.

The pointer data item is often used in cases where an address is used as an interface such as when calling a C program.

```
000010 @OPTIONS MAIN
000020*---------------------------------------------------------------------
000030* The pointer enables data to be referenced using a memory address.
000040*---------------------------------------------------------------------
000050 IDENTIFICATION     DIVISION.
000060 PROGRAM-ID.        SAMPLE.
000070 DATA               DIVISION.
000080*---------------------------------------------------------------------
000090* An item that is to be references using a pointer reference is
```

```
000095* declared in based-storage section.
000100*-----------------------------------------------------------------
000110 BASED-STORAGE      SECTION.
000120 01 TYPE-DATE.
000130    02              PIC X(8).
000140    02 CR-HOUR      PIC 9(2).
000150    02 CR-MINUTE    PIC 9(2).
000160    02 CR-SEC       PIC 9(2).
000170*-----------------------------------------------------------------
000180 WORKING-STORAGE    SECTION.
000190 01 CR-DATE         PIC X(21).
000200 01 PTR             POINTER.
000210 PROCEDURE          DIVISION.
000220     MOVE FUNCTION CURRENT-DATE TO CR-DATE.
000230*-----------------------------------------------------------------
000240* The starting address of the actual data item is set in the pointer
000245* item and the data is pointed to and referenced using a pointer
000250* reference.
000260*-----------------------------------------------------------------
000270     MOVE FUNCTION ADDR (CR-DATE) TO PTR.
000280     DISPLAY "The current time is " PTR->CR-HOUR " hours, "
000290                                    PTR->CR-MINUTE " minutes, "
000300                                    PTR->CR-SEC " seconds."
000310*-----------------------------------------------------------------
000320 END PROGRAM SAMPLE.
```

1.6 Uniqueness of Reference of Condition Name

Condition names can also be qualified.

When a condition name has been assigned to a data item, the condition name can be uniquely referenced using qualification.

```
000010 @OPTIONS MAIN
000020*-----------------------------------------------------------------
000030* Condition names can also be qualified.
000040*-----------------------------------------------------------------
000050 IDENTIFICATION   DIVISION.
000060 PROGRAM-ID       SAMPLE.
000070 DATA             DIVISION.
000080 WORKING-STORAGE  SECTION.
000090*-----------------------------------------------------------------
000100* The correct values for month and date are both defined using the
000105* condition name "CORRECT".
000110*-----------------------------------------------------------------
000120 01 VACATION.
000130    02 V-MONTH    PIC 9(2).
000140      88 CORRECT  VALUE 1 THRU 12.
000150    02 V-DAY      PIC 9(2).
000160      88 CORRECT  VALUE 1 THRU 31.
000170*-----------------------------------------------------------------
000180 01 CR-DATE.
000190    02            PIC 9(4).
000200    02 CR-MONTH   PIC 9(2).
000210    02 CR-DAY     PIC 9(2).
000220  01 ANS          PIC 9(3).
000230 PROCEDURE        DIVISION.
000240     DISPLAY "When do you get vacation? Example:  0801 >>"
000245             WITH NO ADVANCING.
000250     ACCEPT VACATION.
000260     MOVE FUNCTION CURRENT-DATE TO CR-DATE.
```

```
000270*---------------------------------------------------------------------
000280* The condition name is be qualified using OF and uniquely referenced.
000290*---------------------------------------------------------------------
000300     IF CORRECT OF V-MONTH AND
000310        CORRECT OF V-DAY   THEN
000320*---------------------------------------------------------------------
000330       IF V-MONTH = CR-MONTH AND V-DAY >= CR-DAY THEN
000340         COMPUTE ANS = V-DAY - CR-DAY
000350       ELSE
000360         COMPUTE ANS = 30 - CR-DAY
000370         COMPUTE CR-MONTH = CR-MONTH + 1
000380         IF CR-MONTH > 12 THEN
000390           COMPUTE CR-MONTH = CR-MONTH - 12
000400         END-IF
000410         IF V-MONTH >= CR-MONTH THEN
000420           COMPUTE ANS = ANS + ((V-MONTH - CR-MONTH) * 30) + V-DAY
000430         ELSE
000440           COMPUTE ANS = ANS + ((12 - CR-MONTH) * 30) + (V-MONTH * 30)
000445                       + V-DAY
000450         END-IF
000460       END-IF
000470       DISPLAY "Approximately " ANS
000475               " days remain until you get vacation."
000480     ELSE
000490       DISPLAY "The input data is incorrect."
000500     END-IF.
000510 END PROGRAM SAMPLE.
```

1.7 Continuation of Lines

If a literal cannot be contained on one line or dividing a literal over several lines makes for easier reading, a continuation line can be used to spread a literal over several lines.

```
000010 @OPTIONS MAIN
000020*---------------------------------------------------------------------
000030* Continuation lines can be used for coding over several lines.
000040*
000050* The method for specifying a continued line depends on the Source
000055* Reference Format specified.  This example assumes that VAR (Variable)
000060* format has been specified.
000070*---------------------------------------------------------------------
000080 IDENTIFICATION   DIVISION.
000090 PROGRAM-ID       SAMPLE.
000100 DATA             DIVISION.
000110 WORKING-STORAGE  SECTION.
000120*---------------------------------------------------------------------
000130* For continuation within the definitions, the line to be continued
000140* (line 170) is coded without the closing quote mark.
000150* The continued line (line 180) is continued from the position of the
000155* leading quote mark, and specified with a "-"in the indicator area.
000160*---------------------------------------------------------------------
000170  01 COL-LINE     PIC X(60) VALUE "----+----1----+----2
000180-                               "----+----3----+----4
000190-                               "----+----5----+----6".
000200*---------------------------------------------------------------------
000210  01 STR-DATA     PIC X(60).
000220 PROCEDURE        DIVISION.
000230*---------------------------------------------------------------------
000240* Reserved words and user-defined words can also be continued.
000250*---------------------------------------------------------------------
```

```
000260     DISPLAY "Enter the character string
000270-            " to be displayed with the column line. >>" WITH NO ADV
000275-            ANCING.
000280     ACCEPT STR-D
000290-                ATA FROM CONSOLE.
000300*----------------------------------------------------------------
000310     DISPLAY " ".
000320     DISPLAY COL-LINE.
000330     DISPLAY STR-DATA.
000340 END PROGRAM SAMPLE.
```

1.8 Free Format

Free format enables coding to be done without having to worry about such things as line numbers, indicator areas, area A, or area B.

The compiler option SRF (FREE) must be specified to compile source code using free format.

This sample applies only to Win32, Winx64, Solaris, Linux, LinuxIPF, Linux64 and .NET.

```
000010 @OPTIONS MAIN,SRF(FREE)
*>-------------------------------------------------------------------------
*> This is a coding example of free format.
*>
*> Specify the compiler option SRF(FREE) to compile source using free format.
*>-------------------------------------------------------------------------
IDENTIFICATION   DIVISION.
PROGRAM-ID.      SAMPLE.
DATA             DIVISION.
WORKING-STORAGE  SECTION.
77 MSG           PIC X(60).
PROCEDURE        DIVISION.
*>-------------------------------------------------------------------------
*> The comment ("*>") is used for comments.
*> Comments can be written from any column location.
*>-------------------------------------------------------------------------
DISPLAY " ".    *> The A or B areas need not be considered.
*>-------------------------------------------------------------------------
*> Blank lines can also be written.
*>-------------------------------------------------------------------------
*>-------------------------------------------------------------------------
*> Nonnumeric literals can also be continued.
*> -  Unlike variable-length format, the line is closed once using quote
*>    marks.
*> -  A "-" is coded immediately after the closing quote mark.
*> -  Blank characters to be coded after a "-".
*> -  The continued line starts from the next quote mark.
*>-------------------------------------------------------------------------
  MOVE "Hello, I'm "-
       "a NetCOBOL Program. Ha"-
       "ve a good time!!"    TO MSG.
  DISPLAY MSG.
END PROGRAM SAMPLE.
```

1.9 PROGRAM COLLATING SEQUENCE Clause

The PROGRAM COLLATING SEQUENCE clause is used to change the sequence of characters within a program. (Normally, the sequence conforms to the character sequence of ASCII.)

```
000010 @OPTIONS MAIN
000020*---------------------------------------------------------------------
000030* The PROGRAM COLLATING SEQUENCE clause in this example, compares
000040* characters using the collating sequence of the EBCDIC character set.
000050*---------------------------------------------------------------------
000060 IDENTIFICATION    DIVISION.
000070 PROGRAM-ID        SAMPLE.
000080 ENVIRONMENT       DIVISION.
000090 CONFIGURATION     SECTION.
000100*---------------------------------------------------------------------
000110* The ALPHABET clause declares the alphabet name (EBCDIC-CODE) for
000115* EBCDIC.
000120* And the PROGRAM COLLATING SEQUENCE clause specifies the alphabet
000125* name.
000130*---------------------------------------------------------------------
000140  OBJECT-COMPUTER. FM-V
000150         PROGRAM COLLATING SEQUENCE IS EBCDIC-CODE.
000160  SPECIAL-NAMES.
000170         ALPHABET EBCDIC-CODE IS EBCDIC.
000180*---------------------------------------------------------------------
000190 DATA              DIVISION.
000200 WORKING-STORAGE   SECTION.
000210 01 DATA-1         PIC X(3).
000220 01 DATA-2         PIC X(3).
000230**
000240 PROCEDURE         DIVISION.
000250       DISPLAY "PLEASE INPUT 3 CHARACTERS (DATA-1) >>" WITH NO
002555               ADVANCING.
000260       ACCEPT DATA-1.
000270       DISPLAY "PLEASE INPUT 3 CHARACTERS (DATA-2) >>" WITH NO
000275                ADVANCING.
000280       ACCEPT DATA-2.
000290**
000300       DISPLAY " ".
000310       DISPLAY "*** RESULT OF RELATION INPUT-DATA ***".
000320       EVALUATE TRUE
000330         WHEN DATA-1 = DATA-2
000340           DISPLAY DATA-1 " (DATA-1) = " DATA-2 " (DATA-2)"
000350         WHEN DATA-1 < DATA-2
000360           DISPLAY DATA-1 " (DATA-1) < " DATA-2 " (DATA-2)"
000370         WHEN DATA-1 > DATA-2
000380           DISPLAY DATA-1 " (DATA-1) > " DATA-2 " (DATA-2)"
000390       END-EVALUATE.
000400 END PROGRAM SAMPLE.
```

1.10 Function Name-2 Clause

The function name-2 clause allows the use of external switches.

An external switch controls information that can be switched at execution. The external switch is specified by the runtime option when an application is started.

A runtime option is specified by one of the following methods:

- As a command line parameter following "-CBL" argument
- As an environment variable: "GOPT" for Solaris, Linux, LinuxIPF, Linux64 and LinuxIPF or "@GOPT" for Win32, Winx64 and .NET.

Refer to "NetCOBOL User's Guide" for details on runtime options.

In the sample below, the message is output with upper-case letters if s10000000 is specified for the runtime option. If s00000000 is specified or nothing is specified for the runtime option, the message is output with lower-case letters.

```
000010 @OPTIONS MAIN
000020*----------------------------------------------------------------
000030* In this sample, the message is switched using an external switch.
000040*----------------------------------------------------------------
000050 IDENTIFICATION   DIVISION.
000060 PROGRAM-ID       SAMPLE.
000070 ENVIRONMENT      DIVISION.
000080 CONFIGURATION    SECTION.
000090*----------------------------------------------------------------
000100* The external switch is defined.
000110* Only one external switch is defined here.  However, up to eight
000115* external switches can be defined.
000120*----------------------------------------------------------------
000130  SPECIAL-NAMES.
000140          SWITCH-0 IS SW0 ON  STATUS IS U-CASE
000150                          OFF STATUS IS L-CASE.
000160*----------------------------------------------------------------
000170 PROCEDURE       DIVISION.
000180     IF U-CASE THEN
000190       DISPLAY "HELLO, I AM A NETCOBOL PROGRAM."
000200     ELSE
000210       DISPLAY "Hello, I am a NetCOBOL program."
000220     END-IF
000230 END PROGRAM SAMPLE.
```

1.11 Function Name-3 Clause

The function name-3 clause can be used to manipulate command line arguments and environment variables.

Command line arguments are arguments that are passed when an application is started. The ACCEPT statements can use to retrieve the number of arguments or their values.

In addition, the ACCEPT and DISPLAY statements can be combined to retrieve and set the values of environment variables in the same way.

```
000010 @OPTIONS MAIN
000020*----------------------------------------------------------------
000030* The function name-3 clause is used to accept arguments when an
000040* application is started, sort the arguments, and display them.
000050*
000060* Sample activation example:  SAMPLE.EXE USA England Japan Italy China
000070*----------------------------------------------------------------
000080 IDENTIFICATION  DIVISION.
000090 PROGRAM-ID.     SAMPLE.
000100 ENVIRONMENT     DIVISION.
000110 CONFIGURATION   SECTION.
000120*----------------------------------------------------------------
000130* Mnemonic names are assigned to the argument number (ARGUMENT-NUMBER)
000140* and argument value (ARGUMENT-VALUE).
000150*----------------------------------------------------------------
000160 SPECIAL-NAMES.
000170          ARGUMENT-NUMBER IS ARG-NUM
000180          ARGUMENT-VALUE  IS ARG-VAL.
000190*----------------------------------------------------------------
000200 INPUT-OUTPUT    SECTION.
```

```
000210 FILE-CONTROL.
000220     SELECT SORT-FILE ASSIGN TO SORT-WORK.
000230 DATA            DIVISION.
000240 FILE            SECTION.
000250 SD SORT-FILE.
000260 01 SORT-REC.
000270    02 SORT-DATA PIC X(10).
000280 WORKING-STORAGE SECTION.
000290 01 CNT          PIC 9(4) BINARY.
000300 01 ELM-NUM      PIC 9(4) BINARY.
000310 PROCEDURE       DIVISION.
000320*---------------------------------------------------------------------
000330* The ACCEPT statement can be used to obtain the number of arguments.
000340*---------------------------------------------------------------------
000350      ACCEPT ELM-NUM FROM ARG-NUM.
000360*---------------------------------------------------------------------
000370      DISPLAY "** The arguments are sorted. **".
000380      SORT SORT-FILE ON ASCENDING KEY SORT-DATA
000390                      INPUT PROCEDURE IS IN-PROC
000400                      OUTPUT PROCEDURE IS OUT-PROC.
000400
000410  IN-PROC.
000420       PERFORM TEST BEFORE
000430               VARYING CNT FROM 1 BY 1 UNTIL CNT > ELM-NUM
000440*---------------------------------------------------------------------
000450* For an argument value, the DISPLAY statement first specifies what
000460* number the argument is.  The ACCEPT statement can then be specified
000465* to retrieve the argument.
000470*---------------------------------------------------------------------
000480         DISPLAY CNT UPON ARG-NUM
000490         ACCEPT SORT-DATA FROM ARG-VAL
000500*---------------------------------------------------------------------
000510         RELEASE SORT-REC
000520       END-PERFORM.
000530 OUT-PROC.
000540      RETURN SORT-FILE AT END GO TO P-EXIT.
000550      DISPLAY SORT-DATA.
000560      GO TO OUT-PROC.
000570 P-EXIT.
000580      EXIT PROGRAM.
000590 END PROGRAM SAMPLE.
```

1.12 ALPHABET Clause

The ALPHABET clause assigns names (alphabet names) in a collating sequence.

The alphabet names defined here are enabled when specified in the PROGRAM COLLATING SEQUENCE clause or SORT/MERGE statement.

```
000010 @OPTIONS MAIN
000020*---------------------------------------------------------------------
000030* In this sample, the ALPHABET clause is used to sort a character
000035* string in an arbitrary collating sequence.
000040*---------------------------------------------------------------------
000050 IDENTIFICATION   DIVISION.
000060 PROGRAM-ID.      SAMPLE.
000070 ENVIRONMENT      DIVISION.
000080 CONFIGURATION    SECTION.
000090*---------------------------------------------------------------------
000100* Normally, the collating sequence of ASCII is
```

```
000110*   numeric characters < upper-case letters < lower-case letters.
000112* However,
000115*   lower-case letters < upper-case letters < numeric
000117* is declared as the original alphabet name.
000120*-----------------------------------------------------------------
000130 SPECIAL-NAMES.
000140         ALPHABET ORG-SEQ IS "a" THRU "z"
000150                             "A" THRU "Z"
000160                             "0" THRU "9".
000170*-----------------------------------------------------------------
000180 INPUT-OUTPUT     SECTION.
000190 FILE-CONTROL.
000200     SELECT SORT-FILE ASSIGN TO SORT-WORK.
000210 DATA             DIVISION.
000220 FILE             SECTION.
000230 SD SORT-FILE.
000240 01 SORT-REC.
000250    02 SORT-DATA    PIC X(10).
000260 PROCEDURE        DIVISION.
000270      DISPLAY "** The input character string is sorted. **".
000280*-----------------------------------------------------------------
000290* The character string is sorted using the original collating sequence.
000300*-----------------------------------------------------------------
000310      SORT SORT-FILE ON ASCENDING KEY SORT-DATA
000320                      COLLATING SEQUENCE IS ORG-SEQ
000330                      INPUT PROCEDURE IS IN-PROC
000340                      OUTPUT PROCEDURE IS OUT-PROC.
000350*-----------------------------------------------------------------
000360 IN-PROC.
000370      DISPLAY "Please input the data (Enter space to end) >> " WITH NO
000375              ADVANCING.
000380      ACCEPT SORT-DATA FROM CONSOLE.
000390      IF SORT-DATA NOT = SPACE THEN RELEASE SORT-REC
000400                                    GO TO IN-PROC.
000410      DISPLAY "----------".
000420 OUT-PROC.
000430      RETURN SORT-FILE AT END GO TO P-EXIT.
000440      DISPLAY SORT-DATA.
000450      GO TO OUT-PROC.
000460 P-EXIT.
000470      EXIT PROGRAM.
000480 END PROGRAM SAMPLE.
```

1.13 CLASS Clause

The CLASS clause assigns names (class names) to arbitrary character sets.

The defined class names can be used in conditional expressions.

```
000010 @OPTIONS MAIN
000020*-----------------------------------------------------------------
000030* In this sample, the CLASS clause is used to determine the validity
000035* of the input data.
000040*-----------------------------------------------------------------
000050 IDENTIFICATION   DIVISION.
000060 PROGRAM-ID.      SAMPLE.
000070 ENVIRONMENT      DIVISION.
000080 CONFIGURATION    SECTION.
000090*-----------------------------------------------------------------
000100* The class names of hexadecimal character scope are defined.
```

```
000110*-----------------------------------------------------------------
000120  SPECIAL-NAMES.
000130          CLASS HEX IS "0" THRU "9"
000140                       "A" THRU "F"
000145                       "a" THRU "f".
000150*-----------------------------------------------------------------
000160 DATA             DIVISION.
000170 WORKING-STORAGE  SECTION.
000180 01 IN-DATA       PIC X(4).
000190 PROCEDURE        DIVISION.
000200     DISPLAY "Please input hexadecimal data (4 digits). >>" WITH NO
000205             ADVANCING.
000210      ACCEPT IN-DATA FROM CONSOLE.
000220*-----------------------------------------------------------------
000230* The defined class-names can be used in conditional expressions.
000240*-----------------------------------------------------------------
000250      IF IN-DATA IS HEX THEN
000260        DISPLAY "The hexadecimal value is correct."
000270      ELSE
000280        DISPLAY "The input data is incorrect."
000290      END-IF
000300*-----------------------------------------------------------------
000310 END PROGRAM SAMPLE.
```

1.14 CURRENCY SIGN Clause

For the PICTURE clause, the currency sign is $. The CURRENCY SIGN clause can be used to change the currency sign to an arbitrary character. However, there are some characters that cannot be specified as a currency sign. See the Language Reference for Details.

```
000010 @OPTIONS MAIN
000020*-----------------------------------------------------------------
000030* In this sample, the CURRENCY SIGN clause is used to change
000035* to Japanese Yen from dollars for display.
000040*-----------------------------------------------------------------
000050 IDENTIFICATION   DIVISION.
000060 PROGRAM-ID.      SAMPLE.
000070 ENVIRONMENT      DIVISION.
000080  CONFIGURATION   SECTION.
000090*-----------------------------------------------------------------
000100* The currency sign is changed to ¥.  The default value is $.
000110*-----------------------------------------------------------------
000120   SPECIAL-NAMES.
000130           CURRENCY SIGN IS "¥".
000140*-----------------------------------------------------------------
000150 DATA             DIVISION.
000160  WORKING-STORAGE SECTION.
000170  01 DOLLAR       PIC 9(7).
000180*-----------------------------------------------------------------
000190* The ¥ is specified in the PICTURE clause.
000200*-----------------------------------------------------------------
000210  01 YEN          PIC ¥,¥¥¥,¥¥9.
000220*-----------------------------------------------------------------
000230 PROCEDURE              DIVISION.
000240     DISPLAY "Dollars is changed to Yen."
000250     DISPLAY "Please input Dollars (up to 5 digits). >>"
000255             WITH NO ADVANCING.
000260     ACCEPT DOLLAR.
000270     COMPUTE YEN = DOLLAR * 110.
```

```
000280     DISPLAY "**                                  = " YEN " ($1 = ¥110)".
000290 END PROGRAM SAMPLE.
```

1.15 DECIMAL-POINT IS COMMA Clause

Specifying the DECIMAL-POINT IS COMMA clause switches the meanings of the comma (,) and decimal point (.) when used in data descriptions.

```
000010 @OPTIONS MAIN
000020*-----------------------------------------------------------------------
000030* The DECIMAL-POINT IS COMMA clause is used to switch the meanings of
000040* the comma and decimal point.
000050*-----------------------------------------------------------------------
000060 IDENTIFICATION         DIVISION.
000070 PROGRAM-ID.            SAMPLE.
000080 ENVIRONMENT            DIVISION.
000090 CONFIGURATION          SECTION.
000100*-----------------------------------------------------------------------
000110* The DECIMAL-POINT IS COMMA clause is specified.
000120*-----------------------------------------------------------------------
000130  SPECIAL-NAMES.
000140           DECIMAL-POINT IS COMMA.
000150*-----------------------------------------------------------------------
000160 DATA                   DIVISION.
000170 WORKING-STORAGE        SECTION.
000180 01 Distance-traveled   PIC 9(4).
000190 01 Fuel-consumed       PIC 9(3).
000200*-----------------------------------------------------------------------
000210* The decimal point is represented using a comma.
000220*-----------------------------------------------------------------------
000230 01 Fuel-cost           PIC 999,99.
000240*-----------------------------------------------------------------------
000250 PROCEDURE              DIVISION.
000260     DISPLAY "The fuel cost is calculated."
000270     DISPLAY "Please input the distance traveled.  (unit:  Km) >>"
000275              WITH NO ADVANCING.
000280     ACCEPT  Distance-traveled.
000290     DISPLAY "Please input fuel consumed.  (unit:  L) >>"
000295              WITH NO ADVANCING.
000300     ACCEPT Fuel-consumed.
000310     COMPUTE Fuel-cost = Distance-traveled / Fuel-consumed
000320     DISPLAY "The fuel cost is " fuel-cost " Km/L."
000330 END PROGRAM SAMPLE.
```

1.16 POSITIONING UNIT Clause

The POSITIONING UNIT clause is specified when defining unit names for determining printing positions. The defined positioning unit names are specified in the PRINTING POSITION clause of the data description entry.

```
000010 @OPTIONS MAIN
000020*-----------------------------------------------------------------------
000030* The POSITIONING UNIT clause is used to control the printing
000035* positions.
000040*-----------------------------------------------------------------------
000050 IDENTIFICATION   DIVISION.
000060 PROGRAM-ID.      SAMPLE.
```

```
000070 ENVIRONMENT      DIVISION.
000080 CONFIGURATION    SECTION.
000090*-----------------------------------------------------------------------
000100* A positioning unit is defined.
000110* "5 CPI" represents a spacing where 5 characters can be printed per
000115* inch.
000120*-----------------------------------------------------------------------
000130  SPECIAL-NAMES.
000140          POSITIONING UNIT CPI-5 IS 5 CPI.
000150*-----------------------------------------------------------------------
000160 INPUT-OUTPUT     SECTION.
000170 FILE-CONTROL.
000180        SELECT PRT-FILE ASSIGN TO PRINTER.
000190 DATA             DIVISION.
000200 FILE             SECTION.
000210 FD PRT-FILE.
000220 01 PRT-REC       PIC X(80).
000230 WORKING-STORAGE  SECTION.
000240*-----------------------------------------------------------------------
000250* The column positions of the items are specified.
000260* -  ADDR is printed from column 11 (= 3 inches) when 1 column uses a
000270*    spacing of 5 characters per inch.
000280* -  TEL is printed from column 26 (= 6 inches) when 1 column uses a
000290*    spacing of 5 characters per inch.
000300*-----------------------------------------------------------------------
000310 01 PROF-DATA.
000320    02 NAME       PIC X(16).
000330    02 ADDR       PIC X(32) PRINTING POSITION IS 11 BY CPI-5.
000340    02 TEL        PIC X(15) PRINTING POSITION IS 26 BY CPI-5.
000350*-----------------------------------------------------------------------
000360 PROCEDURE        DIVISION.
000370     DISPLAY "Please input your name    >> " WITH NO ADVANCING.
000380     ACCEPT NAME FROM CONSOLE.
000390     DISPLAY "                  Address >> " WITH NO ADVANCING.
000400     ACCEPT ADDR FROM CONSOLE.
000410     DISPLAY "                  Tel     >> " WITH NO ADVANCING.
000420     ACCEPT TEL  FROM CONSOLE.
000430     DISPLAY "Now printing .....".
000440**
000450     OPEN OUTPUT PRT-FILE.
000460     WRITE PRT-REC FROM PROF-DATA AFTER ADVANCING PAGE.
000470     CLOSE PRT-FILE.
000480 END PROGRAM SAMPLE.
```

1.17 PRINTING MODE Clause

The PRINTING MODE clause is specified when assigning names to printing attributes when printing. The defined printing mode names are specified in the CHARACTER TYPE clause of the data description entry.

```
000010 @OPTIONS MAIN
000020*-----------------------------------------------------------------------
000030* The PRINTING MODE clause is used to define the printing mode name.
000040*-----------------------------------------------------------------------
000050 IDENTIFICATION   DIVISION.
000060 PROGRAM-ID.      SAMPLE.
000070 ENVIRONMENT      DIVISION.
000080 CONFIGURATION    SECTION.
000090*-----------------------------------------------------------------------
000100* The printing attributes are defined.
000110* SIZE 12 POINT: The character size is 12 point.
```

```
000120* PITCH 2 CPI:   The character pitch is 2 CPI.
000130* FONT GOTHIC:   Gothic font is used.
000140* ANGLE 90:      Characters are rotated 90 degrees
000145*                (that is, vertical printing).
000150* FORM F0201:    Wide body is used.
000160*----------------------------------------------------------------
000170  SPECIAL-NAMES.
000180          PRINTING MODE PRT-ATR FOR ALL
000190                                IN SIZE 12 POINT
000200                                AT PITCH 2 CPI
000210                                WITH FONT GOTHIC
000220                                AT ANGLE 90 DEGREES
000230                                BY FORM F0201.
000240*----------------------------------------------------------------
000250  INPUT-OUTPUT    SECTION.
000260  FILE-CONTROL.
000270        SELECT PRT-FILE ASSIGN TO PRINTER.
000280 DATA             DIVISION.
000290 FILE             SECTION.
000300 FD PRT-FILE.
000310 01 PRT-REC       PIC X(80).
000320 WORKING-STORAGE  SECTION.
000330*----------------------------------------------------------------
000340* The defined printing attributes are specified in the CHARACTER TYPE
000345* clause.
000350*----------------------------------------------------------------
000360 01 PRT-DATA      PIC X(20) CHARACTER TYPE IS PRT-ATR.
000370*----------------------------------------------------------------
000380 PROCEDURE        DIVISION.
000390     DISPLAY "Now printing .....".
000400     OPEN OUTPUT PRT-FILE
000410       MOVE "NetCOBOL" TO PRT-DATA
000420       WRITE PRT-REC FROM PRT-DATA AFTER ADVANCING PAGE.
000430     CLOSE PRT-FILE.
000440 END PROGRAM SAMPLE.
```

1.18 SYMBOLIC CHARACTERS Clause

The SYMBOLIC CHARACTERS clause is specified when assigning names to character codes. The SYMBOLIC CHARACTERS clause can be used for assigning names to control characters.

```
000010 @OPTIONS MAIN
000020*----------------------------------------------------------------
000030* In this sample, the SYMBOLIC CHARACTERS clause is used to control the
000035* TAB character code.
000040* Executing this sample creates a sequential file named SAMPLE.TXT in
000050* the current folder (directory).
000060*----------------------------------------------------------------
000070 IDENTIFICATION   DIVISION.
000080 PROGRAM-ID.      SAMPLE.
000090 ENVIRONMENT      DIVISION.
000100 CONFIGURATION    SECTION.
000110*----------------------------------------------------------------
000120* The integer specified in the SYMBOLIC CHARACTERS clause is the
000130* sequential position of a character in the character set.  That is,
000135* the integer indicates the order of a character in the
000140* list where X"00" in the ASCII code table is 1. Because the TAB code
000145* is X"09", 10 is specified for the integer.
000150*----------------------------------------------------------------
000160 SPECIAL-NAMES.
```

```
000170         SYMBOLIC CHARACTERS TAB IS 10.
000180*-----------------------------------------------------------------
000190 INPUT-OUTPUT     SECTION.
000200 FILE-CONTROL.
000210      SELECT TXT-FILE ASSIGN TO "SAMPLE.TXT"
000220                      ORGANIZATION IS LINE SEQUENTIAL.
000230 DATA             DIVISION.
000240 FILE             SECTION.
000250 FD TXT-FILE.
000260 01 TXT-REC       PIC X(80).
000270 PROCEDURE        DIVISION.
000280     OPEN OUTPUT TXT-FILE.
000290*-----------------------------------------------------------------
000300* The defined symbolic character can be used in the same way as a
000305* nonnumeric literal.
000310* For this example, the symbolic character is used in a concatenation
000315* expression.
000320*-----------------------------------------------------------------
000330     MOVE "Fujitsu Ltd." & TAB & "Sales department" & TAB &
000335          "Fujitsu Taro" TO TXT-REC.
000340*-----------------------------------------------------------------
000350     WRITE TXT-REC.
000360     CLOSE TXT-FILE.
000370 END PROGRAM SAMPLE.
```

1.19 SYMBOLIC CONSTANT Clause

The SYMBOLIC CONSTANT clause is specified when assigning names to literals. Because the literals are located in a single place in the program, maintainability and extendibility are enhanced. The literals need only be modified at one place in the program.

```
000010 @OPTIONS MAIN
000020*-----------------------------------------------------------------
000030* The SYMBOLIC CONSTANT clause assigns a name to a literal.
000040*-----------------------------------------------------------------
000050 IDENTIFICATION       DIVISION.
000060 PROGRAM-ID.          SAMPLE.
000070 ENVIRONMENT          DIVISION.
000080 CONFIGURATION        SECTION.
000090*-----------------------------------------------------------------
000100* The consumption tax rate has been defined.
000110* By making the consumption tax rate a symbolic constant,
000120* future changes in the rate can be easily handled.
000130*-----------------------------------------------------------------
000140 SPECIAL-NAMES.
000150          SYMBOLIC CONSTANT
000160          CONSUMPTION-TAX-RATE IS 0.05.
000170*-----------------------------------------------------------------
000180 DATA                 DIVISION.
000190 WORKING-STORAGE      SECTION.
000200 01 PRICE             PIC 9(8).
000210 01 CONSUMPTION-TAX   PIC ZZZZZ9.
000220 PROCEDURE            DIVISION.
000230     DISPLAY "Calculate the consumption tax."
000240     DISPLAY "How much does the item you want cost? >>" WITH NO
000245             ADVANCING
000250     ACCEPT PRICE.
000260*-----------------------------------------------------------------
000270* Symbolic constants can be used in the same way as literals.
000280*-----------------------------------------------------------------
000290     COMPUTE CONSUMPTION-TAX = PRICE * CONSUMPTION-TAX-RATE.
```

```
000300     DISPLAY "The consumption tax rate is " CONSUMPTION-TAX-RATE "."
000305             "  The tax amount is " CONSUMPTION-TAX " dollars."
000310*----------------------------------------------------------------
000320     EXIT PROGRAM.
000330 END PROGRAM SAMPLE.
```

1.20 RENAMES Clause

The RENAMES clause can be used to assign a different name to a list of contiguous data items.

Specify the new name and which data items make up this new name in the description entry of level-number 66.

```
000010 @OPTIONS MAIN
000020*----------------------------------------------------------------
000030* The RENAMES clause can be used to rename an area.
000040*----------------------------------------------------------------
000050 IDENTIFICATION   DIVISION.
000060 PROGRAM-ID.      SAMPLE.
000070 DATA             DIVISION.
000080 WORKING-STORAGE  SECTION.
000090 01 EMPLOYEE-DATA.
000100    02 EMPLOYEE-NUMBER   PIC 9(8)   VALUE 19990120.
000110    02                   PIC X(2)   VALUE SPACE.
000120    02 WHERE-EMPLOYED    PIC X(20)  VALUE "Sales department".
000130    02 EMPLOYEE-NAME     PIC X(30)  VALUE "Fujitsu taro".
000140    02 STREET-ADDRESS    PIC X(50)  VALUE "123 Main Street".
000150    02 TELEPHONE-NUMBER  PIC X(15)  VALUE "(123) 456-7890".
000160*----------------------------------------------------------------
000170* The RENAMES clause can be used to declare a different memory
000175* organization.
000180* Employee information: Information items from the employee number to
000185*                       the employee name are renamed.
000190* Personal information: Information items from the employee name to the
000195*                       telephone number are renamed.
000200*----------------------------------------------------------------
000210 66 EMPLOYEE-INFORMATION  RENAMES EMPLOYEE-NUMBER THRU EMPLOYEE-NAME.
000220 66 PERSONAL-INFORMATION  RENAMES EMPLOYEE-NAME  THRU TELEPHONE-NUMBER.
000230*----------------------------------------------------------------
000024 77 INFORMATION-TYPE     PIC 9.
000250 PROCEDURE               DIVISION.
000260     DISPLAY "Please input the information type."
000270     DISPLAY "Employee information (1), Personal information (2), All
000275-            "information (3) >>" WITH NO ADVANCING.
000280     ACCEPT INFORMATION-TYPE.
000290*----------------------------------------------------------------
000300* The renamed data name can be used in the same way as a group item.
000310*----------------------------------------------------------------
000320     EVALUATE INFORMATION-TYPE
000330     WHEN 1
000340       DISPLAY EMPLOYEE-INFORMATION
000350     WHEN 2
000360       DISPLAY PERSONAL-INFORMATION
000370     WHEN 3
000380       DISPLAY EMPLOYEE-DATA
000390     WHEN OTHER
000400       DISPLAY "The input data is incorrect."
000410     END-EVALUATE.
000420*----------------------------------------------------------------
000430 END PROGRAM SAMPLE.
```

1.21 VALUE Clause for Conditional Names

An 88 level can be used to assign a conditional name representing a particular value or values for a data item. The defined conditional name can be used in conditional expressions.

```
000010 @OPTIONS MAIN
000020*---------------------------------------------------------------------
000030* Level-number 88 items can be used to define conditional names when
000040* data is declared.
000050*---------------------------------------------------------------------
000060 IDENTIFICATION     DIVISION.
000070 PROGRAM-ID.        SAMPLE.
000080 DATA               DIVISION.
000090 WORKING-STORAGE SECTION.
000100 01 TODAY.
000110    02             PIC 9(2).
000120*---------------------------------------------------------------------
000130* The conditional names are defined based on data item values.
000140*---------------------------------------------------------------------
000150    02 MONTH        PIC 9(2).
000160      88 SPRING-M   VALUE  3 THRU 5.
000170      88 SUMMER-M   VALUE  6 THRU 8.
000180      88 AUTUMN-M   VALUE  9 THRU 11.
000190      88 WINTER-M   VALUE 12, 1, 2.
000200*---------------------------------------------------------------------
000210    02             PIC 9(2).
000220 PROCEDURE          DIVISION.
000230      ACCEPT TODAY FROM DATE.
000240*---------------------------------------------------------------------
000250* The conditional names can be used in conditional expressions.
000260*---------------------------------------------------------------------
000270     EVALUATE TRUE
000280      WHEN SPRING-M
000290       DISPLAY "It is Spring.  Let's go hiking!!"
000300      WHEN SUMMER-M
000310       DISPLAY "It is Summer.  Let's go swimming!!"
000320      WHEN AUTUMN-M
000330       DISPLAY "It is Autumn.  Let's play tennis!!"
000340      WHEN WINTER-M
000350       DISPLAY "It is Winter.  Let's go skiing!!"
000360     END-EVALUATE.
000370*---------------------------------------------------------------------
000380 END PROGRAM SAMPLE.
```

1.22 BLANK WHEN ZERO Clause

The BLANK WHEN ZERO clause replaces zeros with blanks.

The BLANK WHEN ZERO clause is useful for displaying spaces when a zero is used to represent an invalid value in a data item.

```
000010 @OPTIONS MAIN
000020*---------------------------------------------------------------------
000030* The BLANK WHEN ZERO clause displays a blank when a value is zero.
000040*---------------------------------------------------------------------
000050 IDENTIFICATION          DIVISION.
000060 PROGRAM-ID.             SAMPLE.
000070 DATA                    DIVISION.
000080 WORKING-STORAGE         SECTION.
```

```
000090*---------------------------------------------------------------------
000100* The BLANK WHEN ZERO clause is specified for selected subjects
000105* (Japanese-history and world-history).
000110*---------------------------------------------------------------------
000120  01 TEST-RESULTS.
000130    02 ENGLISH           PIC 9(3) BLANK WHEN ZERO.
000140    02                   PIC X(6) VALUE SPACE.
000150    02 MATHEMATICS       PIC 9(3) BLANK WHEN ZERO.
000160    02                   PIC X(10) VALUE SPACE.
000170    02 US-HISTORY        PIC 9(3) BLANK WHEN ZERO.
000180    02                   PIC X(10) VALUE SPACE.
000190    02 WORLD-HISTORY     PIC 9(3) BLANK WHEN ZERO.
000210 77 TEMP                 PIC 9(3).
000220 PROCEDURE               DIVISION.
000230     DISPLAY "Please input the test results."
000240     DISPLAY "Please input a zero for courses that have not been
000245-            "selected."
000250     DISPLAY "English          >> " WITH NO ADVANCING
000260     ACCEPT TEMP.
000261     MOVE TEMP TO ENGLISH.
000270     DISPLAY "Mathematics      >> " WITH NO ADVANCING
000280     ACCEPT TEMP.
000281     MOVE TEMP TO MATHEMATICS.
000290     DISPLAY "US history       >> " WITH NO ADVANCING
000300     ACCEPT TEMP.
000310     MOVE TEMP TO US-HISTORY.
000320     DISPLAY "World history    >> " WITH NO ADVANCING
000330     ACCEPT TEMP.
000340     MOVE TEMP TO WORLD-HISTORY.
000350     DISPLAY " ".
000360     DISPLAY "          English  Mathematics  US history   World
000365-            " History".
000370     DISPLAY "Results:  " TEST-RESULTS.
000380 END PROGRAM SAMPLE.
```

1.23 GLOBAL Clause

The GLOBAL clause is used to define data that is to be shared among different nested programs. Data items that are defined as GLOBAL can be referenced from child programs.

```
000010 @OPTIONS MAIN
000020*---------------------------------------------------------------------
000030* The GLOBAL clause can define variables that have a global attribute.
000040*---------------------------------------------------------------------
000050 IDENTIFICATION   DIVISION.
000060 PROGRAM-ID.      SAMPLE.
000070 DATA             DIVISION.
000080 WORKING-STORAGE  SECTION.
000090*---------------------------------------------------------------------
000100* The GLOBAL clause is specified to declare a global attribute.
000110* Variables that have a global attribute can also be referenced from
000115* child programs.
000120*---------------------------------------------------------------------
000130  01 IN-DATA      IS GLOBAL PIC X(80).
000140  01 ED-DATA      IS GLOBAL.
000150     02 ELM       OCCURS 8 TIMES PIC X(16).
000160*---------------------------------------------------------------------
000170 PROCEDURE        DIVISION.
000180     CALL "IN-PROC".
000190     CALL "ED-PROC".
```

```
000200     CALL "OUT-PROC".
000210     EXIT PROGRAM.
000220**
000230 IDENTIFICATION   DIVISION.
000240 PROGRAM-ID.      IN-PROC.
000250 PROCEDURE        DIVISION.
000260     DISPLAY "The input character string is delimited by a space."
000270     DISPLAY "Please input the character string. >>" WITH NO ADVANCING.
000280*-----------------------------------------------------------------------
000290* Values are set for the item IN-DATA that has the global attribute.
000300*-----------------------------------------------------------------------
000310     ACCEPT IN-DATA FROM CONSOLE.
000320*-----------------------------------------------------------------------
000330     EXIT PROGRAM.
000340 END PROGRAM IN-PROC.
000350**
000360 IDENTIFICATION   DIVISION.
000370 PROGRAM-ID.      ED-PROC.
000380 PROCEDURE        DIVISION.
000390*-----------------------------------------------------------------------
000400* The item IN-DATA that has the global attribute is edited.
000410*-----------------------------------------------------------------------
000420     UNSTRING IN-DATA DELIMITED BY SPACE
000430                      INTO ELM(1) ELM(2) ELM(3) ELM(4)
000440                           ELM(5) ELM(6) ELM(7) ELM(8).
000450*-----------------------------------------------------------------------
000460     EXIT PROGRAM.
000470 END PROGRAM ED-PROC.
000480**
000490 IDENTIFICATION   DIVISION.
000500 PROGRAM-ID.      OUT-PROC.
000510 PROCEDURE        DIVISION.
000520     DISPLAY " ".
000530*-----------------------------------------------------------------------
000540* The item ED-DATA that has the global attribute is referenced.
000550*-----------------------------------------------------------------------
000560     DISPLAY "Unstring data:" ED-DATA.
000570*-----------------------------------------------------------------------
000580     EXIT PROGRAM.
000590 END PROGRAM OUT-PROC.
000600 END PROGRAM SAMPLE.
```

1.24 JUSTIFIED Clause

Normally, character data is left-justified. However, items that specify the JUSTIFIED clause are right-justified. The JUSTIFIED clause is useful for displaying data right-justified.

```
000010 @OPTIONS MAIN
000020*-----------------------------------------------------------------------
000030* The JUSTIFIED RIGHT clause specifies that the character string
000035* is right-justified.
000040*-----------------------------------------------------------------------
000050 IDENTIFICATION   DIVISION.
000060 PROGRAM-ID.      SAMPLE.
000070 DATA             DIVISION.
000080 WORKING-STORAGE SECTION.
000090*-----------------------------------------------------------------------
000100* The JUSTIFIED RIGHT clause is specified for items that are to be
000110* stored or displayed right-justified.
000120*-----------------------------------------------------------------------
```

```
000130 01 CARD.
000140    02 TEAM-1     PIC X(10).
000150    02            PIC X(02) VALUE "VS".
000160    02 TEAM-2     PIC X(10) JUSTIFIED RIGHT.
000170    02            PIC X(02) VALUE SPACE.
000180    02 PLACE      PIC X(25).
000190*-----------------------------------------------------------------
000200 PROCEDURE      DIVISION.
000210     DISPLAY "** Today's match card **".
000220     DISPLAY " ".
000230     MOVE "Japan"  TO TEAM-1.
000240*-----------------------------------------------------------------
000250* The character string is stored right-justified.
000260*-----------------------------------------------------------------
000270     MOVE "Brazil" TO TEAM-2.
000280*-----------------------------------------------------------------
000290     MOVE "Sydney" TO PLACE.
000300     DISPLAY CARD.
000310**
000320     MOVE "Italia"      TO TEAM-1.
000330     MOVE "USA"   TO TEAM-2.
000340     MOVE "Canberra" TO PLACE.
000350     DISPLAY CARD.
000360 END PROGRAM SAMPLE.
```

1.25 OCCURS Clause

The OCCURS clause is used to define table (also called array) data.

The OCCURS clause can be used to easily define a formalized table structure. A variable number of occurrences may be defined. Tables may be nested up to seven levels.

1.25.1 OCCURS Clause (format 1) Specific Occurrence Count

```
000010 @OPTIONS MAIN
000020*-----------------------------------------------------------------
000030* OCCURS clause (format 1)
000040* Array data is defined.
000050*-----------------------------------------------------------------
000060 IDENTIFICATION   DIVISION.
000070 PROGRAM-ID.      SAMPLE.
000080 DATA             DIVISION.
000090 WORKING-STORAGE  SECTION.
000100 01 RESIDENTS.
000110    02              PIC X(25) VALUE "Room 101:  Suzuki".
000120    02              PIC X(25) VALUE "Room 102:  Nakamura".
000130    02              PIC X(25) VALUE "Room 103:  Saito".
000140    02              PIC X(25) VALUE "Room 201:  Yamamoto".
000150    02              PIC X(25) VALUE "Room 202:  Kimura".
000160    02              PIC X(25) VALUE "Room 203:  Tanaka".
000170*-----------------------------------------------------------------
000180* The resident data is redefined as a table.
000190* Apartment Fujitsu is a two-story building with three rooms on each
000195* floor (two-dimensional array).
000200*-----------------------------------------------------------------
000210 01             REDEFINES RESIDENTS.
000220    02 FLOOR        OCCURS 2 TIMES.
000230      03 OCCUPANT     OCCURS 3 TIMES PIC X(25).
000240*-----------------------------------------------------------------
```

```
000250 77 FLOOR-NUMBER   PIC 9(1).
000260 77 ROOM-NUMBER    PIC 9(1).
000270 PROCEDURE          DIVISION.
000280     DISPLAY "This is the Apartment Fujitsu residents guide."
000290     DISPLAY "The room is on which floor? (1 or 2) >>"
000295             WITH NO ADVANCING.
000300     ACCEPT FLOOR-NUMBER.
000310     DISPLAY "Which room number? (1 to 3) >>"
000315             WITH NO ADVANCING.
000320     ACCEPT ROOM-NUMBER.
000330     DISPLAY " ".
000340*-----------------------------------------------------------------
000350* The arrayed data can be referenced by subscripting.
000360* Because the data is a two-dimensional array, two subscripts are used
000365* to specify the data.
000370*-----------------------------------------------------------------
000380     DISPLAY OCCUPANT (FLOOR-NUMBER, ROOM-NUMBER) " is the resident."
000390*-----------------------------------------------------------------
000400 END PROGRAM SAMPLE.
```

1.25.2 OCCURS clause (format 2) Variable Number of Occurrences

The OCCURS clause (format 2) is used when there is a variable number of occurrences for the data item that specifies the OCCURS clause.

The size of the data item that specifies the OCCURS clause is not variable -- the number of occurrences is variable.

```
000010 @OPTIONS MAIN
000020*-----------------------------------------------------------------
000030* OCCURS clause (format 2)
000040* The number of occurrences can change dynamically.
000050*-----------------------------------------------------------------
000060 IDENTIFICATION   DIVISION.
000070 PROGRAM-ID.      SAMPLE.
000080 DATA             DIVISION.
000090 WORKING-STORAGE SECTION.
000100*-----------------------------------------------------------------
000110* If the data name is specified in the DEPENDING ON clause, the number
000120* of occurrences can be controlled dynamically when the program is
000125* executed.
000130*-----------------------------------------------------------------
000140 01 DSP-DATA.
000150    02            OCCURS 80 TIMES DEPENDING ON LEN.
000160      03            PIC X VALUE "*".
000170 77 LEN        PIC 9(2).
000180*-----------------------------------------------------------------
000190 PROCEDURE      DIVISION.
000200     DISPLAY "Display how many bytes? (1 to 80) >>" WITH NO ADVANCING.
000210     ACCEPT LEN.
000220*-----------------------------------------------------------------
000230* The value of the data (LEN) specified in the DEPENDING ON clause
000235* becomes the number of occurrences.
000240*-----------------------------------------------------------------
000250     DISPLAY DSP-DATA.
000260*-----------------------------------------------------------------
000270 END PROGRAM SAMPLE.
```

1.26 Numeric Edited Data PICTURE Clause

Numeric edited data items can be used to edit numeric data into the desired format for presentation.

```
000010 @OPTIONS MAIN
000020*---------------------------------------------------------------------
000030* The PICTURE clause defines the numeric edited data item characters.
000040*---------------------------------------------------------------------
000050 IDENTIFICATION   DIVISION.
000060 PROGRAM-ID.      SAMPLE.
000070 DATA             DIVISION.
000080 WORKING-STORAGE  SECTION.
000090*---------------------------------------------------------------------
000100* This is an example of defining numeric edited data items.
000110* Execute the program to see the results.
000130*---------------------------------------------------------------------
000140 01 NUMEDIT-1    PIC   ----,--9.
000150 01 NUMEDIT-2    PIC   -ZZZ,ZZ9.
000160 01 NUMEDIT-3    PIC    $$$,$$9.
000170 01 NUMEDIT-4    PIC 9999/99/99.
000180 01 NUMEDIT-5    PIC   +++9.999.
000190 01 NUMEDIT-6    PIC *********9.
000200*---------------------------------------------------------------------
000210 01 WDATE.
000220    02 TODAY        PIC 9(8).
000230 PROCEDURE      DIVISION.
000240     DISPLAY "INPUT -> PICTURE        = OUTPUT"
000250     DISPLAY "-----------------------------------"
000260**
000270     MOVE -3000  TO  NUMEDIT-1
000280     DISPLAY "-3000 -> PIC   ----,--9 =   "  NUMEDIT-1
000290**
000300     MOVE   980  TO  NUMEDIT-2
000310     DISPLAY "  980 -> PIC   -ZZZ,ZZ9 =   "  NUMEDIT-2
000320**
000330     MOVE  3210  TO  NUMEDIT-3
000340     DISPLAY " 3210 -> PIC    $$$,$$9 =    " NUMEDIT-3
000350**
000360     MOVE FUNCTION CURRENT-DATE  TO  WDATE
000370     MOVE TODAY  TO  NUMEDIT-4
000380     DISPLAY "TODAY -> PIC 9999/99/99 = "    NUMEDIT-4
000390**
000400     MOVE 12.34  TO  NUMEDIT-5
000410     DISPLAY "12.34 -> PIC   +++9.999 =   "  NUMEDIT-5
000420**
000430     MOVE 67890  TO  NUMEDIT-6
000440     DISPLAY "67890 -> PIC *********9 = "    NUMEDIT-6
000450 END PROGRAM SAMPLE.
```

1.27 REDEFINES Clause

The REDEFINES clause is used to redefine data items.

The REDEFINES clause is used to reference a specific item using a different item definition or to assign a different role to one data area.

```
000010 @OPTIONS MAIN
000020*---------------------------------------------------------------------
000030* The REDEFINES clause is used to redefine items.
000040*---------------------------------------------------------------------
000050 IDENTIFICATION   DIVISION.
000060 PROGRAM-ID.      SAMPLE.
000070 DATA             DIVISION.
```

```
000080 WORKING-STORAGE  SECTION.
000090*-----------------------------------------------------------------
000100* The name of the item to be redefined is specified in the REDEFINES
000110* clause.
000120* The attribute of the item to be redefined can be different from the
000122* attribute of the item that is redefined.  However, take the internal
000124* format into consideration when using the REDEFINES clause so that the
000126* new attributes make sense.
000130*-----------------------------------------------------------------
000140 01 IN-CHAR       PIC X(1).
000150 01 BOOL          REDEFINES IN-CHAR PIC 1(8) BIT.
000160*-----------------------------------------------------------------
000170 PROCEDURE        DIVISION.
000180     DISPLAY "The input character is displayed in bit expression."
000190     DISPLAY "Please input a character (one character). >>"
000195             WITH NO ADVANCING.
000200     ACCEPT IN-CHAR.
000210     DISPLAY "The bit expression of character """ IN-CHAR
000215             """ is " BOOL ".".
000220 END PROGRAM SAMPLE.
```

1.28 SIGN Clause

The letter S indicates that a numeric field is signed. To specify that a numeric field is to have a sign, the letter S is added to the character string in the PICTURE clause.

The actual internal representation of the sign is controlled by the SIGN clause.

```
000010 @OPTIONS MAIN
000020*-----------------------------------------------------------------
000030* The SIGN clause specifies the internal sign format.
000040*-----------------------------------------------------------------
000050 IDENTIFICATION   DIVISION.
000060 PROGRAM-ID.      SAMPLE.
000070 DATA             DIVISION.
000080 WORKING-STORAGE  SECTION.
000090 77 UNSIGNED      PIC  9(4).
000100 77 SIGNED        PIC S9(4).
000110 77 TRAIL-SIGNED  PIC S9(4) SIGN TRAILING.
000120 77 LEAD-SIGNED   PIC S9(4) SIGN LEADING.
000130 77 TRAIL-SEP     PIC S9(4) SIGN TRAILING SEPARATE.
000140 77 LEAD-SEP      PIC S9(4) SIGN LEADING SEPARATE.
000150 PROCEDURE        DIVISION.
000160*-----------------------------------------------------------------
000170* For UNSIGNED data items, the value is stored as an absolute value.
000180* For the following, the contents of item UNSIGNED become X"31323334".
000190*-----------------------------------------------------------------
000200      MOVE +1234 TO UNSIGNED.
000210      DISPLAY "+1234 ->  9(4)                       =  " UNSIGNED.
000220*-----------------------------------------------------------------
000230* For SIGNED, the sign is managed using the four high-order bits of the
000240* least significant digit in the same way as TRAILING SIGN format.
000250* (Positive:  X"4x", Negative:  X"5x")
000255* For the following, the contents of item SIGNED become X"31323344".
000260*-----------------------------------------------------------------
000270      MOVE +1234 TO SIGNED.
000280      DISPLAY "+1234 -> S9(4)                       =  " SIGNED.
000290*-----------------------------------------------------------------
000300* For the TRAILING specification, the sign is managed using the four
000310* high-order bits of the least significant digit.  (Positive:  X"4x",
```

```
000320* Negative:  X"5x")
000325* For the following, the contents of item TRAIL-SIGNED become
000327* X"31323344".
000330*-----------------------------------------------------------------
000340      MOVE +1234 TO TRAIL-SIGNED.
000350      DISPLAY "+1234 -> S9(4) SIGN TRAILING          =  " TRAIL-SIGNED.
000360*-----------------------------------------------------------------
000370* For the LEADING specification, the sign is managed using the four
000380* high-order bits of the most significant digit.  (Positive:  X"4x",
000390* Negative:  X"5x")
000395* For the following, the contents of item LEAD-SIGNED become
000397* X"41323334".
000400*-----------------------------------------------------------------
000410      MOVE +1234 TO LEAD-SIGNED.
000420      DISPLAY "+1234 -> S9(4) SIGN LEADING           =  " LEAD-SIGNED.
000430*-----------------------------------------------------------------
000440* For the TRAILING SEPARATE specification, the sign is added to the
000450* right of the least significant digit using the character information.
000460* (Positive:  X"2B", Negative:  X"2D")
000465* For the following, the contents of item TRAIL-SEP become
000467* X"313233342B".
000470*-----------------------------------------------------------------
000480      MOVE +1234 TO TRAIL-SEP.
000490      DISPLAY "+1234 -> S9(4) SIGN TRAILING SEPARATE =  " TRAIL-SEP.
000500*-----------------------------------------------------------------
000510* For the LEADING SEPARATE specification, the sign is added to the left
000520* of the most significant digit using the character information.
000530* (Positive: X"2B", Negative:  X"2D")
000535* For the following, the contents of item LEAD-SEP become
000537* X"2B31323334".
000540*-----------------------------------------------------------------
000550      MOVE +1234 TO LEAD-SEP.
000560      DISPLAY "+1234 -> S9(4) SIGN LEADING SEPARATE  =  " LEAD-SEP.
000570 END PROGRAM SAMPLE.
```

1.29 TYPEDEF and TYPE Clauses

The TYPEDEF clause can define an arbitrary data type.

The defined data type is referenced by the TYPE clause.

The TYPEDEF and TYPE clauses are unique to Win32, Winx64, Solaris, Linux, LinuxIPF, Linux64 and .NET.

```
000010 @OPTIONS MAIN
000020*-----------------------------------------------------------------
000030* The TYPEDEF clause can define an arbitrary data type.
000040* The defined data type can be referenced by the TYPE clause.
000050*
000060* This sample uses a date type to measure the execution performance of
000065* the ADD statement.
000070*-----------------------------------------------------------------
000080 IDENTIFICATION   DIVISION.
000090 PROGRAM-ID.      SAMPLE.
000100 DATA             DIVISION.
000110 WORKING-STORAGE  SECTION.
000120*-----------------------------------------------------------------
000130* The TYPEDEF clause is used to define the date type (DATE-EDITED).
000140*-----------------------------------------------------------------
000150 01 DATE-EDITED  TYPEDEF.
000160    02 YEARS        PIC 9(4).
000170    02             PIC X(1) VALUE "/".
```

```
000180    02 MONTHS       PIC 9(2).
000190    02              PIC X(1) VALUE "/".
000200    02 DAYS         PIC 9(2).
000210    02              PIC X(1) VALUE " ".
000220    02 HOURS        PIC 9(2).
000230    02              PIC X(1) VALUE ":".
000240    02 MINUTES      PIC 9(2).
000250    02              PIC X(1) VALUE ":".
000260    02 SECONDS      PIC 9(2).
000270    02              PIC X(1) VALUE ".".
000280    02 M-SECS       PIC 9(2).
000290*-------------------------------------------------------------------
000300* The date type defined above can be used by specifying the TYPE
000305* clause.
000310*-------------------------------------------------------------------
000320 01 STARTED     TYPE DATE-EDITED.
000330 01 ENDED       TYPE DATE-EDITED.
000340*-------------------------------------------------------------------
000350 01 WK-DATE.
000360    02  YEARS       PIC  9(4).
000370    02  MONTHS      PIC  9(2).
000380    02  DAYS        PIC  9(2).
000390    02  HOURS       PIC  9(2).
000400    02  MINUTES     PIC  9(2).
000410    02  SECONDS     PIC  9(2).
000420    02  M-SECS      PIC  9(2).
000430 01 COUNTER         PIC S9(8) VALUE 0.
000440 PROCEDURE          DIVISION.
000450      MOVE FUNCTION CURRENT-DATE TO WK-DATE.
000460      MOVE CORR WK-DATE          TO STARTED.
000470      DISPLAY "STARTED-TIME IS " STARTED.
000480**
000490      PERFORM 1000000 TIMES
000500        ADD 1 TO  COUNTER
000510      END-PERFORM.
000520**
000530      MOVE FUNCTION CURRENT-DATE TO  WK-DATE.
000540      MOVE CORR WK-DATE          TO ENDED.
000550      DISPLAY "ENDED-TIME   IS " ENDED.
000560      EXIT PROGRAM.
000570 END PROGRAM SAMPLE.
```

1.30 BASED ON Clause

Normally, data items defined using the Based-Storage Section make use of pointers in order to be referenced. However, using the BASED ON clause to specify specific pointer data items enables data to be referenced without the pointer' qualification.

```
000010 @OPTIONS MAIN
000020*-------------------------------------------------------------------
000030* The BASED ON clause enables referencing with an implicit pointer.
000040*-------------------------------------------------------------------
000050 IDENTIFICATION   DIVISION.
000060 PROGRAM-ID.      SAMPLE.
000070 DATA             DIVISION.
000080*-------------------------------------------------------------------
000090* Specifying the BASED ON clause enables defined items to be referenced
000100* with implicit pointing.
000110*-------------------------------------------------------------------
000120 BASED-STORAGE   SECTION.
000130 01              BASED ON MENU-PTR.
```

```
000140    02 MENU         OCCURS 3 TIMES.
000150       03 M-NAME       PIC X(20).
000160       03 M-DETAIL     PIC X(30).
000170*-----------------------------------------------------------------------
000180 WORKING-STORAGE SECTION.
000190 01  MENU-NO     PIC 9(1).
000200 01  MENU-PTR    POINTER.
000210 CONSTANT        SECTION.
000220 01 SAMPLE-DATA.
000230    02  MENU-1.
000240        03        PIC X(20)  VALUE "A-Lunch".
000250        03        PIC X(30)  VALUE "Curry rice, Salad, Fruit".
000260    02  MENU-2.
000270        03        PIC X(20)  VALUE "B-Lunch".
000280        03        PIC X(30)  VALUE "Sandwich, Salad, Coffee".
000290    02  MENU-3.
000300        03        PIC X(20)  VALUE "C-Lunch".
000310        03        PIC X(30)  VALUE "Spaghetti, Salad, Ice Cream".
000320 PROCEDURE        DIVISION.
000330     DISPLAY "*** Today's Lunch Menu ***".
000340     MOVE FUNCTION ADDR (SAMPLE-DATA) TO MENU-PTR.
000350     PERFORM TEST BEFORE
000360             VARYING MENU-NO FROM 1 BY 1 UNTIL  MENU-NO > 3
000370*-----------------------------------------------------------------------
000380* Reference with implicit pointing is enabled.
000390* For the following, coding MENU-PTR->M-NAME is the same as MENU-NAME
000400* and MEMU-PTR->M-DETAIL is the same as M-DETAIL.
000410*-----------------------------------------------------------------------
000420       DISPLAY " "
000430       DISPLAY "Name   : " M-NAME(MENU-NO)
000440       DISPLAY "Details: " M-DETAIL(MENU-NO)
000450*-----------------------------------------------------------------------
000460     END-PERFORM.
000470     EXIT PROGRAM.
000480 END PROGRAM SAMPLE.
```

1.31 Boolean Expression

When coding a program, there will be several situations where you might want to perform logical operations using Boolean items. COBOL enables this using Boolean expressions.

```
000010 @OPTIONS MAIN
000020*-----------------------------------------------------------------------
000030* Boolean expressions are used for operations using Boolean data items.
000040*-----------------------------------------------------------------------
000050 IDENTIFICATION   DIVISION.
000060 PROGRAM-ID.      SAMPLE.
000070 DATA             DIVISION.
000080 WORKING-STORAGE  SECTION.
000090 01 IN-DATA       PIC S9(4) COMP-5.
000100 01 CNT           PIC S9(4) COMP-5.
000110 01 .
000120    02 RESULT     PIC 1(12) BIT VALUE ALL B"0".
000130    02 RES-BIT    REDEFINES RESULT OCCURS 12 PIC 1(1) BIT.
000140 CONSTANT         SECTION.
000150 01 ELM-TBL.
000160    02             PIC X(10) VALUE "USA".
000170    02             PIC X(10) VALUE "Korea".
000180    02             PIC X(10) VALUE "Germany".
000190    02             PIC X(10) VALUE "Russia".
```

```
000200    02              PIC X(10) VALUE "England".
000210    02              PIC X(10) VALUE "Japan".
000220    02              PIC X(10) VALUE "Spain".
000230    02              PIC X(10) VALUE "France".
000240    02              PIC X(10) VALUE "Kenya".
000250    02              PIC X(10) VALUE "China".
000260    02              PIC X(10) VALUE "Brazil".
000270    02              PIC X(10) VALUE "Italy".
000280 01             REDEFINES ELM-TBL.
000290    02 ELM-NAME     PIC X(10)  OCCURS 12.
000300*-----------------------------------------------------------------
000310* The category information of each country is defined using Boolean
000315* expressions.
000320* UNSC:  The resident country of the United Nations is set to 1.
000330* NATO:  The member nations of NATO are set to 1.
000340*-----------------------------------------------------------------
000350  01 SUBSET-TBL.
000360    02 UNSC         PIC 1(12) BIT VALUE B"100110010100".
000370    02 NATO         PIC 1(12) BIT VALUE B"101010110001".
000380*-----------------------------------------------------------------
000390 PROCEDURE        DIVISION.
000400     DISPLAY "The following countries are categorized.".
000410     PERFORM TEST BEFORE
000420             VARYING CNT FROM 1 BY 1 UNTIL CNT > 12
000430       IF CNT = 6 OR 12 THEN
000440         DISPLAY ELM-NAME(CNT)
000450       ELSE
000460         DISPLAY ELM-NAME(CNT) WITH NO ADVANCING
000470       END-IF
000480     END-PERFORM.
000490     DISPLAY " ".
000500     DISPLAY "<Category>".
000510     DISPLAY "Resident country of the United Nations :1".
000520     DISPLAY "Member nations of NATO                 :2".
```

```
000530     DISPLAY "Countries of 1 and 2                   :3".
000540     DISPLAY "Countries of neither                   :4".
000550     DISPLAY " ".
000560     DISPLAY "Please select the category. >>" WITH NO ADVANCING.
000570     ACCEPT  IN-DATA.
000580*-----------------------------------------------------------------
000590* Boolean operators such as AND and OR can be used to obtain the
000600* countries that match the conditions.  Expressions that use these
000605* Boolean operators are referred to as Boolean expressions.
000610*-----------------------------------------------------------------
000620     EVALUATE IN-DATA
000630      WHEN 1  COMPUTE RESULT = UNSC
000640      WHEN 2  COMPUTE RESULT = NATO
000650      WHEN 3  COMPUTE RESULT = UNSC AND NATO
000660      WHEN 4  COMPUTE RESULT = NOT (UNSC OR NATO)
000670     END-EVALUATE.
000680*-----------------------------------------------------------------
000690     PERFORM TEST BEFORE
000700             VARYING CNT FROM 1 BY 1 UNTIL CNT > 12
000710       IF RES-BIT(CNT) = B"1" THEN
000720         DISPLAY ELM-NAME(CNT)
000730       END-IF
000740     END-PERFORM.
000750 END PROGRAM SAMPLE.
```

1.32 Class Condition

The class condition is used to check the contents of data items. For example, it can be used to check to see if the data item consists of only numeric data.

This sample uses classes that have already been defined (alphabetic character check and numeric data check). However, defining a class using the CLASS clause of the SPECIAL-NAMES paragraph enables checking using any arbitrarily defined class.

```
000010 @OPTIONS MAIN
000020*---------------------------------------------------------------------
000030* The type of character data can be checked using a class condition.
000040*---------------------------------------------------------------------
000050 IDENTIFICATION   DIVISION.
000060 PROGRAM-ID.      SAMPLE.
000070 DATA             DIVISION.
000080 WORKING-STORAGE  SECTION.
000090 01 NAME          PIC X(20).
000100 01 BIRTHDAY      PIC X(8).
000110 PROCEDURE        DIVISION.
000120     DISPLAY "Please input your name in alphabetic characters. >>"
000125             WITH NO ADVANCING.
000130     ACCEPT NAME FROM CONSOLE.
000140*---------------------------------------------------------------------
000150* The ALPHABETIC condition is used for the alphabetic character check.
000160*---------------------------------------------------------------------
000170     IF NAME IS NOT ALPHABETIC THEN
000180       DISPLAY "The input data is incorrect."
000190       EXIT PROGRAM
000200     END-IF.
000210*---------------------------------------------------------------------
000220     DISPLAY "Please input your birthday.  Example:  19690123  >>"
000225             WITH NO ADVANCING.
000230     ACCEPT BIRTHDAY FROM CONSOLE.
000240*---------------------------------------------------------------------
000250* The NUMERIC condition is used for the numeric data check.
000260*---------------------------------------------------------------------
000270     IF BIRTHDAY IS NOT NUMERIC THEN
000280       DISPLAY "The input data is incorrect."
000290       EXIT PROGRAM
000300     END-IF.
000310*---------------------------------------------------------------------
000320     DISPLAY " ".
000330     DISPLAY "Name    : "  NAME.
000340     DISPLAY "Birthday: "  BIRTHDAY.
000350 END PROGRAM SAMPLE.
```

1.33 Abbreviating Combined Relation Conditions

Combined relation conditions (multiple condition expressions) can be abbreviated if the left part or right part is the same. The source will be easier to read if combined relation conditions are abbreviated.

```
000010 @OPTIONS MAIN
000020*---------------------------------------------------------------------
000030* Combined relation conditions can be abbreviated depending on the
000035* conditions.
000040*---------------------------------------------------------------------
000050 IDENTIFICATION   DIVISION.
```

```
000060 PROGRAM-ID.      SAMPLE.
000070 DATA             DIVISION.
000080 WORKING-STORAGE  SECTION.
000090 01 IN-DATA       PIC 9(1).
000100 01 COUNTER       PIC 9(1).
000110 CONSTANT         SECTION.
000120 01 SPORTS-DATA.
000130    02              PIC X(9) VALUE "Marathon".
000140    02              PIC X(9) VALUE "Baseball".
000150    02              PIC X(9) VALUE "Tennis".
000160    02              PIC X(9) VALUE "Skiing".
000170    02              PIC X(9) VALUE "Judo".
000180    02              PIC X(9) VALUE "Soccer".
000190 01              REDEFINES SPORTS-DATA.
000200    02 SPORTS       OCCURS 6 PIC X(9).
000210 PROCEDURE        DIVISION.
000220     PERFORM TEST BEFORE
000230             VARYING COUNTER FROM 1 BY 1 UNTIL COUNTER > 6
000240       DISPLAY COUNTER ". " SPORTS(COUNTER) WITH NO ADVANCING
000250     END-PERFORM.
000260     DISPLAY " ".
000270     DISPLAY " ".
000280     DISPLAY "Balls are used in which sport? >>" WITH NO ADVANCING.
000290     ACCEPT  IN-DATA.
000300     DISPLAY " ".
000310*-----------------------------------------------------------------------
000320* Multiple condition expressions where the left part is the same but
000330* the right part changes can be abbreviated as shown below:
000340* If abbreviation is not used, the condition expressions must be
000350* written as shown below:
000360* IF IN-DATA = 2 OR
000370*    IN-DATA = 3 OR
000380*    IN-DATA = 6 THEN
000390*-----------------------------------------------------------------------
000400     IF IN-DATA = 2 OR 3 OR 6 THEN
000410       DISPLAY "This is correct."
000420     ELSE
000430       DISPLAY "This is incorrect."
000440     END-IF.
000450*-----------------------------------------------------------------------
000460 END PROGRAM SAMPLE.
```

1.34 COMPUTE Statement

1.34.1 ROUNDED Phrase

Normally for COBOL arithmetic expressions, truncation is performed based on the number of digits of the data item where the operation results are stored. However, the ROUNDED phrase can be used to round the operation results instead of truncating them.

```
000010 @OPTIONS MAIN
000020*-----------------------------------------------------------------------
000030* The ROUNDED phrase can be used to round the operation results.
000040*-----------------------------------------------------------------------
000050 IDENTIFICATION    DIVISION.
000060 PROGRAM-ID.       SAMPLE.
000070 DATA              DIVISION.
000080 WORKING-STORAGE   SECTION.
000090 01 INPUT-NUM      PIC S9(4) VALUE ZERO.
000100 01 THE-SUM        PIC S9(8) VALUE ZERO.
```

```
000110 01 AVERAGE-VALUE  PIC S9(4).
000120 01 COUNTER        PIC  9(4) BINARY.
000130 PROCEDURE         DIVISION.
000140     DISPLAY "Round off the average value of the input data (up to 4
000145-            " digits)."
000150     PERFORM TEST AFTER
000160             VARYING COUNTER FROM 1 BY 1 UNTIL INPUT-NUM = 0
000170       DISPLAY "Please input a value (end in 0). >>" WITH NO ADVANCING
000180       ACCEPT INPUT-NUM
000190       COMPUTE THE-SUM = THE-SUM + INPUT-NUM
000200     END-PERFORM.
000210     IF COUNTER > 1 THEN
000220*---------------------------------------------------------------------
000230* The ROUNDED phrase rounds the operation results.
000240* Truncation is performed if the ROUNDED phrase is omitted.
000250*---------------------------------------------------------------------
000260       COMPUTE AVERAGE-VALUE ROUNDED = THE-SUM / (COUNTER - 1)
000270*---------------------------------------------------------------------
000280       DISPLAY " "
000290       DISPLAY "The average value is " AVERAGE-VALUE "."
000300     END-IF.
000310 END PROGRAM SAMPLE.
```

1.34.2 ON SIZE ERROR Phrase

The ON SIZE ERROR phrase can be used to define special processing that is to take place when a size error occurs for a data item used to store the results of a mathematical operation. Because COBOL continues execution even if a size error occurs, a size error can cause a loop or processing result error at execution. To prevent such problems, the ON SIZE ERROR phrase can be used to clearly define the operation to be executed when a size error occurs.

```
000010 @OPTIONS MAIN
000020*---------------------------------------------------------------------
000030* The ON SIZE ERROR phrase can define the processing operation when a
000035* size error occurs.
000040*---------------------------------------------------------------------
000050 IDENTIFICATION    DIVISION.
000060 PROGRAM-ID.       SAMPLE.
000070 DATA              DIVISION.
000080 WORKING-STORAGE   SECTION.
000090 01 INPUT-NUM      PIC S9(4) VALUE ZERO.
000100 01 THE-SUM        PIC S9(4) VALUE ZERO.
000110 01 AVERAGE-VALUE  PIC S9(4).
000120 01 COUNTER        PIC  9(4) BINARY.
000130 PROCEDURE         DIVISION.
000140     DISPLAY "Obtain the average value of the input data (up to 4
000145-            " digits)."
000150     PERFORM TEST AFTER
000160             VARYING COUNTER FROM 1 BY 1 UNTIL INPUT-NUM = 0
000170       DISPLAY "Please input a value (end in 0). >>"  WITH NO ADVANCING
000180       ACCEPT INPUT-NUM
000190*---------------------------------------------------------------------
000200* The logic is coded so that the PERFORM statement is exited when a
000210* size error occurs for a data item used to store the sum of the
000215* numeric data.
000220*---------------------------------------------------------------------
000230       COMPUTE THE-SUM = THE-SUM + INPUT-NUM
000240         ON SIZE ERROR DISPLAY "The intermediate data is out of range."
000250                       MOVE ZERO TO COUNTER
000260                       EXIT PERFORM
000270       END-COMPUTE
```

```
000280*---------------------------------------------------------------------
000290     END-PERFORM.
000300     IF COUNTER > 1 THEN
000310       COMPUTE AVERAGE-VALUE = THE-SUM / (COUNTER - 1)
000320       DISPLAY " "
000330       DISPLAY "The average value is " AVERAGE-VALUE "."
000340     END-IF.
000350 END PROGRAM SAMPLE.
```

1.35 INITIALIZE Statement

The INITIALIZE statement is used to initialize data items. The INITIALIZE statement is particularly useful for initializing all items that belong to a group level item.

```
000010 @OPTIONS MAIN
000020*---------------------------------------------------------------------
000030* The INITIALIZE statement is used to initialize data items.
000040*---------------------------------------------------------------------
000050 IDENTIFICATION   DIVISION.
000060 PROGRAM-ID.      SAMPLE.
000070 DATA             DIVISION.
000080 WORKING-STORAGE  SECTION.
000090 01 EMPLOYEE-DATA.
000100    02 PERSONAL-INFORMATION  OCCURS 5 TIMES.
000110       03 EMPLOYEE-NUMBER       PIC 9(8).
000120       03                      PIC X(1) VALUE ":".
000130       03 NAME                  PIC X(20).
000140       03                      PIC X(1) VALUE ":".
000150       03 WHERE-EMPLOYED        PIC X(20).
000160 01 COUNTER       PIC S9(4) BINARY.
000170 CONSTANT         SECTION.
000180 01 HEADER        PIC X(40) VALUE "<Number>  <Name>
000185-                               "              <Address>".
000190 PROCEDURE        DIVISION.
000200*---------------------------------------------------------------------
000210* Specifying a group item in the INITIALIZE statement sets appropriate
000220* initial values based on the attributes of the items that belong to
000225* the group item.  Filler items, such as the colons above are not
000227* initialized.
000230*---------------------------------------------------------------------
000240     INITIALIZE EMPLOYEE-DATA.
000250*---------------------------------------------------------------------
000260     PERFORM DISPLAY-PROCESSING.
000270*---------------------------------------------------------------------
000280* In addition, arbitrary initial values can be set.
000290*---------------------------------------------------------------------
000300     INITIALIZE EMPLOYEE-DATA REPLACING NUMERIC  DATA BY 99999999
000310                                        ALPHANUMERIC DATA BY ALL "-".
000320     PERFORM DISPLAY-PROCESSING.
000330     EXIT PROGRAM.
000340**
000350 DISPLAY-PROCESSING    SECTION.
000360     DISPLAY " "
000370     DISPLAY HEADER.
000380     PERFORM TEST BEFORE
000390             VARYING COUNTER FROM 1 BY 1 UNTIL COUNTER > 5
000400       DISPLAY PERSONAL-INFORMATION (COUNTER)
000410     END-PERFORM.
000420 END PROGRAM SAMPLE.
```

1.36 INSPECT Statement

The INSPECT statement (format 1) is used to count the occurrence of characters or character strings in a data item.

```
000010 @OPTIONS MAIN
000020*---------------------------------------------------------------------
000030* The INSPECT statement (format 1) is used to count the occurrence of
000035* characters or character strings.
000040*---------------------------------------------------------------------
000050 IDENTIFICATION        DIVISION.
000060 PROGRAM-ID.           SAMPLE.
000070 DATA                  DIVISION.
000080 WORKING-STORAGE SECTION.
000090 01 NUMBER-OF-MALES    PIC 9(4) VALUE ZERO.
000100 01 NUMBER-OF-FEMALES  PIC 9(4) VALUE ZERO.
000110 CONSTANT              SECTION.
000120 01 PARTICIPANT-NAMES  PIC X(50)
000130     VALUE "Mr.Brown Mrs.Brown Mr.Jones Mrs.Margaret Mr.Smith".
000140 PROCEDURE             DIVISION.
000150     DISPLAY "Name of travelers: " PARTICIPANT-NAMES.
000160     DISPLAY " ".
000170*---------------------------------------------------------------------
000180* The INSPECT statement is used to find character strings.
000190*---------------------------------------------------------------------
000200     INSPECT PARTICIPANT-NAMES TALLYING NUMBER-OF-MALES FOR ALL "Mr.".
000210     DISPLAY "For males, there are " NUMBER-OF-MALES " participants.".
000220     INSPECT PARTICIPANT-NAMES TALLYING NUMBER-OF-FEMALES FOR ALL
000335             "Mrs.".
000230     DISPLAY "For females, there are " NUMBER-OF-FEMALES "
000235-            " participants.".
000240*---------------------------------------------------------------------
000250 END PROGRAM SAMPLE.
```

In addition, the inspected character or character string can be replaced with another character or character string (format 2).

```
000010 @OPTIONS MAIN
000020*---------------------------------------------------------------------
000030* The INSPECT statement (format 2) is used to replace the character
000035* string.
000040*---------------------------------------------------------------------
000050 IDENTIFICATION   DIVISION.
000060 PROGRAM-ID.      SAMPLE.
000070 DATA             DIVISION.
000080 WORKING-STORAGE SECTION.
000090 01 CR-DATE.
000100    02 CR-YEAR      PIC 9(4).
000110    02 CR-MON       PIC 9(2).
000120    02 CR-DAY       PIC 9(2).
000130 01 COPYRIGHT    PIC X(60)
000140     VALUE "Copyright yyyy Fujitsu Limited".
000150 PROCEDURE        DIVISION.
000160     MOVE FUNCTION CURRENT-DATE TO CR-DATE.
000170*---------------------------------------------------------------------
000180* The desired word is found in the character string, and replaced with
000185* the specified word.
000190*---------------------------------------------------------------------
000200     INSPECT COPYRIGHT REPLACING ALL "yyyy" BY CR-YEAR.
```

```
000210*---------------------------------------------------------------------
000220     DISPLAY COPYRIGHT.
000230 END PROGRAM SAMPLE.
```

1.37 MOVE Statement with CORRESPONDING

When moving a group item, the CORRESPONDING phrase can be used to code a move that operates only on the elementary data items that share the same name in both groups.

```
000010 @OPTIONS MAIN
000020*---------------------------------------------------------------------
000030* The CORRESPONDING phrase can be used to execute statements that
000040* process only corresponding data items.
000050*---------------------------------------------------------------------
000060 IDENTIFICATION     DIVISION.
000070 PROGRAM-ID.        SAMPLE.
000080 DATA               DIVISION.
000090 WORKING-STORAGE    SECTION.
000100 01 EDITING-DISPLAY.
000110    02             PIC X(6) VALUE " Year ".
000120    02 YEAR         PIC 9(4).
000130    02             PIC X(7) VALUE " Month ".
000140    02 MONTH        PIC 9(2).
000150    02             PIC X(5) VALUE " Day ".
000160    02 MONTH-DAY    PIC 9(2).
000170    02             PIC X(6) VALUE " Hour ".
000180    02 HOUR         PIC 9(2).
000200    02             PIC X(8) VALUE " Minute ".
000210    02 MINUTE       PIC 9(2).
000220    02             PIC X(8) VALUE " Second ".
000230    02 SECOND       PIC 9(2).
000240 01 CURRENT-DATE-AND-TIME.
000250    02 YEAR         PIC 9(4).
000260    02 MONTH        PIC 9(2).
000270    02 MONTH-DAY    PIC 9(2).
000280    02 HOUR         PIC 9(2).
000290    02 MINUTE       PIC 9(2).
000300    02 SECOND       PIC 9(2).
000310 PROCEDURE          DIVISION.
000320      MOVE FUNCTION CURRENT-DATE TO CURRENT-DATE-AND-TIME.
000330*---------------------------------------------------------------------
000340* The CORRESPONDING phrase moves to the corresponding data item (having
000345* the same name).
000350*---------------------------------------------------------------------
000360      MOVE CORRESPONDING CURRENT-DATE-AND-TIME TO EDITING-DISPLAY.
000370*---------------------------------------------------------------------
000380      DISPLAY "The current date and time are ".
000385      DISPLAY EDITING-DISPLAY ".".
000390 END PROGRAM SAMPLE.
```

1.38 SEARCH Statement

The SEARCH statement is used to search tables (also called arrays, declared using the OCCURS clause) for specific elements.

The SEARCH statement has two formats: Sequential search (format 1) and non-sequential search (format 2).

1.38.1 SEARCH Statement (format 1)

```
000010 @OPTIONS MAIN
000020*---------------------------------------------------------------------
000030* The SEARCH statement (format 1) retrieves specific elements from
000035* tables.
000040*---------------------------------------------------------------------
000050 IDENTIFICATION   DIVISION.
000060 PROGRAM-ID.      SAMPLE.
000070 DATA             DIVISION.
000080 WORKING-STORAGE  SECTION.
000090 01 GET-GOODS     PIC X(15).
000100 01 GET-NUM       PIC 9(4) BINARY.
000110 01 TOTAL         PIC 9(4) BINARY VALUE ZERO.
000120 01 COUNTER       PIC 9(1).
000130 01 PRICE-ED      PIC $$$$$9.
000140 01 TOTAL-ED      PIC ZZZZZ9.
000150 CONSTANT         SECTION.
000160 01 GOODS-DATA.
000170    02            PIC X(15) VALUE "PRINTER".
000180    02            PIC 9(4)  VALUE 400.
000190    02            PIC X(15) VALUE "MODEM".
000200    02            PIC 9(4)  VALUE 80.
000210    02            PIC X(15) VALUE "HARD DISK".
000220    02            PIC 9(4)  VALUE 280.
000230    02            PIC X(15) VALUE "DESKTOP TYPE".
000240    02            PIC 9(4)  VALUE 1500.
000250    02            PIC X(15) VALUE "NOTEBOOK TYPE".
000260    02            PIC 9(4)  VALUE 2200.
000270*---------------------------------------------------------------------
000280* An index name is specified for any table referenced by a SEARCH
000285* statement.
000290*---------------------------------------------------------------------
000300 01               REDEFINES GOODS-DATA.
000310   02 GOODS       OCCURS 5 TIMES INDEXED BY IX.
000320      03 NAME         PIC X(15).
000330      03 PRICE        PIC 9(4).
000340*---------------------------------------------------------------------
000350 PROCEDURE        DIVISION.
000360     PERFORM TEST BEFORE
000370             VARYING COUNTER FROM 1 BY 1 UNTIL COUNTER > 5
000380       MOVE PRICE(COUNTER) TO PRICE-ED
000390       DISPLAY COUNTER "." NAME(COUNTER) PRICE-ED
000400     END-PERFORM.
000410     DISPLAY " ".
000420     DISPLAY "What would you like to buy?  Goods name >>" WITH NO
000425             ADVANCING.
000430     ACCEPT GET-GOODS FROM CONSOLE.
000440     DISPLAY "How many would you like?               >>" WITH NO
000445             ADVANCING.
000450     ACCEPT GET-NUM.
000460*---------------------------------------------------------------------
000470* The operation to be executed upon a successful search is coded in the
000480* SEARCH statement.
000490* Because search is executed sequentially from the index, a value is set
000500* for the index to indicate the search starting position.  In addition,
000510* search is stopped when the search object is found.  The value of the
000515* index is then set at that point.  As a result, a subsequent SEARCH
000517* statement can be coded to continue searching from the last position
000518* where an item was found.
000520*---------------------------------------------------------------------
000530     SET IX TO 1.
000540     SEARCH GOODS
```

```
000550       WHEN NAME(IX) = GET-GOODS
000560         MOVE PRICE(IX) TO TOTAL
000570     END-SEARCH.
000580*-----------------------------------------------------------------------
000590     DISPLAY " ".
000600     IF TOTAL NOT = ZERO THEN
000610       COMPUTE TOTAL = TOTAL * GET-NUM
000620       MOVE TOTAL TO TOTAL-ED
000630       DISPLAY "The total amount is " TOTAL-ED " Dollars."
000640     ELSE
000650       DISPLAY "The input data is incorrect."
000660     END-IF.
000670 END PROGRAM SAMPLE.
```

1.38.2 SEARCH Statement (format 2)

```
000010 @OPTIONS MAIN
000020*-----------------------------------------------------------------------
000030* The SEARCH statement (format 2) can also perform a random search.
000040*-----------------------------------------------------------------------
000050 IDENTIFICATION   DIVISION.
000060 PROGRAM-ID.      SAMPLE.
000070 DATA             DIVISION.
000080 WORKING-STORAGE  SECTION.
000090 01 GET-GOODS     PIC X(15).
000100 01 GET-NUM       PIC 9(4) BINARY.
000110 01 TOTAL         PIC 9(4) BINARY VALUE ZERO.
000120 01 COUNTER       PIC 9(1).
000130 01 PRICE-ED      PIC $$$$$9.
000140 01 TOTAL-ED      PIC ZZZZZ9.
000150 CONSTANT         SECTION.
000160 01 GOODS-DATA.
000170    02           PIC X(15) VALUE "DESKTOP TYPE".
000180    02           PIC 9(4)  VALUE 1500.
000190    02           PIC X(15) VALUE "HARD DISK".
000200    02           PIC 9(4)  VALUE 280.
000210    02           PIC X(15) VALUE "MODEM".
000220    02           PIC 9(4)  VALUE 80.
000230    02           PIC X(15) VALUE "NOTEBOOK TYPE".
000240    02           PIC 9(4)  VALUE 2200.
000250    02           PIC X(15) VALUE "PRINTER".
000260    02           PIC 9(4)  VALUE 400.
000270*-----------------------------------------------------------------------
000280* An index key and index name are specified for the table used by the
000290* SEARCH statement (format 2).
000300* The data used as the search key must be in ascending (ASCENDING) or
000310* descending (DESCENDING) order.
000320*-----------------------------------------------------------------------
000330  01              REDEFINES GOODS-DATA.
000340    02 GOODS      OCCURS 5 TIMES
000350                  ASCENDING KEY IS NAME INDEXED BY IX.
000360      03 NAME     PIC X(15).
000370      03 PRICE    PIC 9(4).
000380*-----------------------------------------------------------------------
000390 PROCEDURE        DIVISION.
000400     PERFORM TEST BEFORE
000410             VARYING COUNTER FROM 1 BY 1 UNTIL COUNTER > 5
000420       MOVE PRICE(COUNTER) TO PRICE-ED
000430       DISPLAY COUNTER "." NAME(COUNTER) PRICE-ED
000440     END-PERFORM.
```

```
000450     DISPLAY " ".
000460     DISPLAY "What would you like to buy?  Goods name >> "
000470             WITH NO ADVANCING.
000480     ACCEPT GET-GOODS FROM CONSOLE.
000490     DISPLAY "How many would you like?                >> "
000500             WITH NO ADVANCING.
000510     ACCEPT GET-NUM.
000520*-----------------------------------------------------------------
000530* For the SEARCH statement (format 2), the initial value of the index
000540* is ignored.  The value of the index used for searching is changed
000550* based on the search method determined by the compiler.
000560* Use formats 1 or 2 based on the required situation.
000570*-----------------------------------------------------------------
000580     SEARCH ALL GOODS
000590       WHEN NAME(IX) = GET-GOODS
000600         MOVE PRICE(IX) TO TOTAL
000610     END-SEARCH.
000620*-----------------------------------------------------------------
000630     DISPLAY " ".
000640     IF TOTAL NOT = ZERO THEN
000650       COMPUTE TOTAL = TOTAL * GET-NUM
000660       MOVE TOTAL TO TOTAL-ED
000670       DISPLAY "The total amount is " TOTAL-ED " dollars."
000680     ELSE
000690       DISPLAY "The input data is incorrect."
000700     END-IF.
000710 END PROGRAM SAMPLE.
```

1.39 STRING Statement

The STRING statement is used to connect several character data items.

Character data items can be connected up to an arbitrary delimiter and an arbitrary length.

```
000010 @OPTIONS MAIN
000020*-----------------------------------------------------------------
000030* The STRING statement is used to connect character data items.
000040*-----------------------------------------------------------------
000050 IDENTIFICATION   DIVISION.
000060 PROGRAM-ID.      SAMPLE.
000070 DATA             DIVISION.
000080 WORKING-STORAGE  SECTION.
000090 01 LISTING       PIC X(30) VALUE SPACE.
000100 01 COUNTER       PIC 9(4)  BINARY.
000110 CONSTANT         SECTION.
000120 01 SHOPPING-LIST.
000130    02             PIC X(15) VALUE "Apple,10,$10".
000140    02             PIC X(15) VALUE "Bread,5,$20".
000150    02             PIC X(15) VALUE "Eggs,10,$2".
000160    02             PIC X(15) VALUE "Tomato,5,$5".
000170    02             PIC X(15) VALUE "Milk,5,$8".
000180 01              REDEFINES SHOPPING-LIST.
000190    02 ELM         OCCURS 5 TIMES PIC X(15).
000200 PROCEDURE        DIVISION.
000210     PERFORM TEST BEFORE
000220             VARYING COUNTER FROM 1 BY 1 UNTIL COUNTER > 5
000230*-----------------------------------------------------------------
000240* The data items to be connected are specified in the STRING statement.
000250* In this example, only the names of products are fetched from a
000255* shopping list and listed.
```

```
000260*---------------------------------------------------------------------
000270       STRING LISTING      DELIMITED BY SPACE
000280              ELM(COUNTER) DELIMITED BY ","
000290              ","          DELIMITED BY SIZE  INTO LISTING
000300*---------------------------------------------------------------------
000310     END-PERFORM.
000320     DISPLAY "Shopping list: " LISTING.
000330 END PROGRAM SAMPLE.
```

1.40 UNSTRING Statement

The UNSTRING statement is used to split character string data using an arbitrary delimiter and store the split character string data in individual data items.

```
000010 @OPTIONS MAIN
000020*---------------------------------------------------------------------
000030* The UNSTRING statement is used to split a character string using an
000035* arbitrary delimiter.
000040*---------------------------------------------------------------------
000050 IDENTIFICATION   DIVISION.
000060 PROGRAM-ID.      SAMPLE.
000070 DATA             DIVISION.
000080 WORKING-STORAGE  SECTION.
000090 01 .
000100    02 WORD       OCCURS 10 TIMES PIC X(10).
000110 01 COUNTER       PIC 9(4) BINARY.
000120 01 SRCH-POS      PIC 9(4) BINARY VALUE 1.
000130 01 WORD-NUM      PIC 9(4) BINARY VALUE 0.
000140 CONSTANT         SECTION.
000150 77 STR-DATA      PIC X(25) VALUE "Are you hungry? Yes,I am.".
000160 PROCEDURE        DIVISION.
000170      DISPLAY "String: " STR-DATA.
000180      DISPLAY " ".
000190      PERFORM TEST BEFORE
000200              VARYING COUNTER FROM 1 BY 1 UNTIL COUNTER > 10
000210*---------------------------------------------------------------------
000220* The UNSTRING statement can specify the following items:
000230* -  Delimiter (DELIMITED BY)
000240* -  Storage area (INTO)
000250* -  Search starting position (WITH POINTER)
000260* -  Storage area of split count (TALLYING IN)
000270*
000280*---------------------------------------------------------------------
000290       UNSTRING STR-DATA
000300                DELIMITED BY ALL SPACE OR "," OR "."
000310                INTO WORD(COUNTER)
000320                WITH POINTER SRCH-POS
000330                TALLYING IN  WORD-NUM
000340       END-UNSTRING
000350*---------------------------------------------------------------------
000360     END-PERFORM.
000370     PERFORM TEST BEFORE
000380             VARYING COUNTER FROM 1 BY 1 UNTIL COUNTER > WORD-NUM
000390       DISPLAY COUNTER "." WORD(COUNTER)
000400     END-PERFORM.
000410 END PROGRAM SAMPLE.
```

1.41 USE Statement

The USE statement is used to code the procedure to be executed when an exception occurs. (For this example an input-output error is used.)

```
000010 @OPTIONS MAIN
000020*---------------------------------------------------------------------
000030* In this sample, the USE statement is coded to capture input-output
000035* errors.
000040*---------------------------------------------------------------------
000050 IDENTIFICATION   DIVISION.
000060 PROGRAM-ID.      SAMPLE.
000070 ENVIRONMENT      DIVISION.
000080 INPUT-OUTPUT     SECTION.
000090 FILE-CONTROL.
000100      SELECT TMP-FILE ASSIGN TO "C:\NONFILE"
000110                      FILE STATUS IS F-STAT.
000120 DATA             DIVISION.
000130 FILE             SECTION.
000140 FD TMP-FILE.
000150 01 TMP-REC       PIC X(80).
000160 WORKING-STORAGE  SECTION.
000170 01 F-STAT.
000180    02 STAT-1     PIC X(1).
000190    02 STAT-2     PIC X(1).
000200 PROCEDURE        DIVISION.
000210*---------------------------------------------------------------------
000220* The input-output error procedure (USE statement) is coded in the
000225* declarative section.
000230*---------------------------------------------------------------------
000240 DECLARATIVES.
000250*---------------------------------------------------------------------
000260* Here, the input-output error procedure is defined for TMP-FILE.
000270* If an input-output error occurs for TMP-FILE when this procedure is
000280* present, control will be passed to the start of this procedure
000290* without issuing a runtime system error.
000300*---------------------------------------------------------------------
000310 IO-ERR           SECTION.
000320     USE AFTER EXCEPTION PROCEDURE TMP-FILE.
000330     EVALUATE F-STAT
000340     WHEN "35"
000350       DISPLAY "The file cannot be found."
000360     WHEN "47"
000370       DISPLAY "The READ statement was executed for an unopened file."
000380     WHEN "42"
```

```
000390       DISPLAY "The CLOSE statement was executed for an unopened file."
000400     WHEN OTHER
000410       DISPLAY "Another input-output error occurred."
000420     END-EVALUATE.
000430     DISPLAY ">>>> Processing continues. ".
000440 END DECLARATIVES.
000450*---------------------------------------------------------------------
000460     OPEN INPUT TMP-FILE.
000460     READ TMP-FILE AT END GO TO P-EXIT.
000480     CLOSE TMP-FILE.
000490 P-EXIT.
000500     EXIT PROGRAM.
000510 END PROGRAM SAMPLE.
```

1.42 SIN, COS and TAN Function

The SIN, COS and TAN functions return approximate values of sine, cosine and tangent for angles specified as arguments to the respective functions.

This sample applies only to Win32.

```
000010 @OPTIONS MAIN
000020*---------------------------------------------------------------------
000030* Sine (SIN), cosine (COS), and tangent (TAN) graphs are plotted.
000040*---------------------------------------------------------------------
000050 IDENTIFICATION   DIVISION.
000060 PROGRAM-ID.      SAMPLE.
000070 DATA             DIVISION.
000080  WORKING-STORAGE SECTION.
000090  01 PI           PIC S9(3)V9(15) VALUE 3.141592653589793.
000100  01 VAL          PIC S9(3)V9(15).
000110  01 LINE-POS     PIC  9(2).
000120  01 COL-POS      PIC  9(2).
000130  01 GRAPH-CODE   PIC  X(1).
000140  01 COUNTER      PIC  9(4) BINARY.
000150  01 S-COUNTER    PIC S9(4) BINARY.
000160 PROCEDURE        DIVISION.
000170     DISPLAY "Which graph do you want to plot? (SIN:  S, COS:  C,
000180-            " TAN:  T) >> " WITH NO ADVANCING.
000190     ACCEPT GRAPH-CODE.
000200     PERFORM TEST BEFORE
000210             VARYING COUNTER FROM 1 BY 1 UNTIL COUNTER > 80
000220       DISPLAY "-" AT LINE 13 COLUMN COUNTER
000230     END-PERFORM.
000240     PERFORM TEST BEFORE
000250             VARYING COUNTER FROM 1 BY 1 UNTIL COUNTER = 26
000260       DISPLAY "|" AT LINE COUNTER COLUMN 40
000270     END-PERFORM.
000280     DISPLAY "+" AT LINE 13 COLUMN 40.
000290*---------------------------------------------------------------------
000300* A sine (SIN) graph is plotted.
000310*---------------------------------------------------------------------
000320     EVALUATE GRAPH-CODE
000330     WHEN "S"
000340       PERFORM TEST BEFORE
000350               VARYING S-COUNTER FROM -39 BY 1 UNTIL S-COUNTER = 40
000360         COMPUTE VAL = 12 * (FUNCTION SIN (PI / 39 * S-COUNTER))
000370         COMPUTE LINE-POS ROUNDED = 13 - VAL
000380         COMPUTE COL-POS = 40 + S-COUNTER
000390         DISPLAY "*" AT LINE LINE-POS COLUMN COL-POS
000400       END-PERFORM
000410*---------------------------------------------------------------------
000420* A cosine (COS) graph is plotted.
000430*---------------------------------------------------------------------
000440     WHEN "C"
000450       PERFORM TEST BEFORE
000460               VARYING S-COUNTER FROM -39 BY 1 UNTIL S-COUNTER = 40
000470         COMPUTE VAL = 12 * (FUNCTION COS (PI / 39 * S-COUNTER))
000480         COMPUTE LINE-POS ROUNDED = 13 - VAL
000490         COMPUTE COL-POS = 40 + S-COUNTER
000500         DISPLAY "*" AT LINE LINE-POS COLUMN COL-POS
000510       END-PERFORM
000520*---------------------------------------------------------------------
000530* A tangent (TAN) graph is plotted.
000540*---------------------------------------------------------------------
```

```
000550     WHEN "T"
000560       PERFORM TEST BEFORE
000570               VARYING S-COUNTER FROM -38 BY 1 UNTIL S-COUNTER = 39
000580         COMPUTE VAL = 0.5 * (FUNCTION TAN (PI / 2 / 39 * S-COUNTER))
000590         COMPUTE LINE-POS ROUNDED = 13 - VAL
000600         COMPUTE COL-POS = 40 + S-COUNTER
000610         DISPLAY "*" AT LINE LINE-POS COLUMN COL-POS
000620       END-PERFORM
000630     END-EVALUATE.
000640 END PROGRAM SAMPLE.
```

1.43 ADDR and LENG Functions

The ADDR function is used to obtain memory addresses of data items.

The obtained address can be used for pointer qualification, non-COBOL linkage of an address interface, and so on.

The LENG function is used to obtain the actual length of data items in bytes.

The LENGTH function is similar to the LENG function. The LENGTH function, however, returns the number of character positions. That is, the LENGTH function returns the number of characters. For example, using a DBCS (Double Byte Character Set) the LENGTH function might return 2, where the LENG function will return 4 - the actual number of bytes.

```
000010 @OPTIONS MAIN
000020*---------------------------------------------------------------------
000030* The pointer obtained using the ADDR function and the two forms
000040* defined in the based-storage section are used to store data in the
000050* work area. In addition, the LENG function is used to obtain the data
000055* item length.
000060*---------------------------------------------------------------------
000070 IDENTIFICATION   DIVISION.
000080 PROGRAM-ID.      SAMPLE.
000090 DATA             DIVISION.
000100 BASED-STORAGE    SECTION.
000110 01 LONG-FORM.
000120    02 FIRST-NAME-L   PIC X(8).
000130    02 LAST-NAME-L    PIC X(8).
000140    02 AGE-L          PIC 9(2).
000150    02 SEPARATER-L    PIC X(1).
000160    02 NEXT-TOP-L     PIC X(1).
000170 01 SHORT-FORM.
000180    02 FIRST-NAME-S   PIC X(8).
000190    02 AGE-S          PIC 9(2).
000200    02 SEPARATER-S    PIC X(1).
000210    02 NEXT-TOP-S     PIC X(1).
000220 WORKING-STORAGE  SECTION.
000230 01 PTR            USAGE POINTER.
000240 01 WORK           PIC X(255).
000250 01 FORMS          PIC 9(1).
000260 01 COUNT-L        PIC 9(2) VALUE 0.
000270 01 COUNT-S        PIC 9(2) VALUE 0.
000280 01 TOTAL-SIZE     PIC 9(3).
000290 PROCEDURE        DIVISION.
000300*---------------------------------------------------------------------
000310* The ADDR function obtains the starting address of the work area
000315* (data storage area).
000320*---------------------------------------------------------------------
000330     MOVE FUNCTION ADDR (WORK) TO PTR.
000340*---------------------------------------------------------------------
000350     PERFORM TEST AFTER
000360             UNTIL FORMS = 0
```

```
000370       DISPLAY " "
000380       DISPLAY "Which forms do you want to select?"
000390       DISPLAY "1:  Long forms (First-name  Last-name  Age)"
000400       DISPLAY "2:  Short forms (First-Name  Age)"
000410       DISPLAY "0:  End processing                  >>" WITH NO
000415               ADVANCING
000420       ACCEPT FORMS
000430       EVALUATE FORMS
000440       WHEN "1"
000450         DISPLAY "First-name >> " WITH NO ADVANCING
000460         ACCEPT PTR->FIRST-NAME-L FROM CONSOLE
000470         DISPLAY "Last-name  >> " WITH NO ADVANCING
000480         ACCEPT PTR->LAST-NAME-L FROM CONSOLE
000490         DISPLAY "Age        >> " WITH NO ADVANCING
000500         ACCEPT PTR->AGE-L FROM CONSOLE
000510         MOVE   "/" TO PTR->SEPARATER-L
000520         COMPUTE COUNT-L = COUNT-L + 1
000530         MOVE FUNCTION ADDR (PTR->NEXT-TOP-L) TO PTR
000540       WHEN "2"
000550         DISPLAY "First-Name >> " WITH NO ADVANCING
000560         ACCEPT PTR->FIRST-NAME-S FROM CONSOLE
000570         DISPLAY "Age        >> " WITH NO ADVANCING
000580         ACCEPT PTR->AGE-S FROM CONSOLE
000590         MOVE   "/" TO PTR->SEPARATER-S
000600         COMPUTE COUNT-S = COUNT-S + 1
000610         MOVE FUNCTION ADDR (PTR->NEXT-TOP-S) TO PTR
000620       END-EVALUATE
000630     END-PERFORM.
000640*-----------------------------------------------------------------------
000650* The LENG function is used to obtain the data item length.
000660* Using the LENG function enables coding that will not be affected when
000670* items are added (the group item length is changed).
000680*-----------------------------------------------------------------------
000690     COMPUTE TOTAL-SIZE =
000695         (FUNCTION LENG (PTR->LONG-FORM) - 1 ) * COUNT-L
000700         + ( FUNCTION LENG (PTR->SHORT-FORM) - 1 ) * COUNT-S.
000710*-----------------------------------------------------------------------
000720     DISPLAY "DATA           :" WORK.
000730     DISPLAY "TOTAL DATA SIZE:" TOTAL-SIZE.
000740 END PROGRAM SAMPLE.
```

1.44 CURRENT-DATE Function

The CURRENT-DATE function is used to obtain the current date and time.

The FROM phrase of the ACCEPT statement can also be used to obtain the current date and time. However, the ACCEPT statement returns only the two low-order digits of the year. To obtain all four digits of the year, use the CURRENT-DATE function.

```
000010 @OPTIONS MAIN
000020*-----------------------------------------------------------------------
000030* The date, time, day of the week, and time difference with Greenwich
000040* mean time obtained using the CURRENT-DATE function are displayed.
000050*-----------------------------------------------------------------------
000060 IDENTIFICATION   DIVISION.
000070 PROGRAM-ID.      SAMPLE.
000080 DATA             DIVISION.
000090 WORKING-STORAGE  SECTION.
000100 01 TODAY.
000110    02 CR-YEAR      PIC X(4).
000120    02 CR-MON       PIC X(2).
```

```
000130    02 CR-DAY       PIC X(2).
000140    02 CR-HOUR      PIC X(2).
000150    02 CR-MIN       PIC X(2).
000160    02 CR-SEC       PIC X(2).
000170    02 CR-MSEC      PIC X(2).
000180    02 LOCAL-TIME.
000190      03 TIME-DIF     PIC X(1).
000200      03 TIME-DIF-H   PIC X(2).
000210      03 TIME-DIF-M   PIC X(2).
000220 01 CR-DOW        PIC 9(1).
000230 CONSTANT         SECTION.
000240 01 DOW-TABLE.
000250    02               PIC X(9) VALUE "Monday".
000260    02               PIC X(9) VALUE "Tuesday".
000270    02               PIC X(9) VALUE "Wednesday".
000280    02               PIC X(9) VALUE "Thursday".
000290    02               PIC X(9) VALUE "Friday".
000300    02               PIC X(9) VALUE "Saturday".
000310    02               PIC X(9) VALUE "Sunday".
000320 01               REDEFINES DOW-TABLE.
000330    02 DOW           OCCURS 7 TIMES PIC X(9).
000340 PROCEDURE         DIVISION.
000350*-----------------------------------------------------------------------
000360* The CURRENT-DATE function obtains the current date and time.
000370* Because the CURRENT-DATE function cannot obtain the day of the week,
000380* use the DAY-OF-WEEK phrase of the ACCEPT statement.
000390*-----------------------------------------------------------------------
000400     MOVE FUNCTION CURRENT-DATE TO TODAY.
000410     ACCEPT CR-DOW FROM DAY-OF-WEEK.
000420*-----------------------------------------------------------------------
000430     DISPLAY "Date: Year " CR-YEAR  " Month " CR-MON
000435             " Day " CR-DAY "(" DOW(CR-DOW) ")".
000440     DISPLAY "Time: Hour " CR-HOUR  " Minute " CR-MIN
000445             " Second " CR-SEC "." CR-MSEC.
000450     IF LOCAL-TIME NOT = 0 THEN
000460       DISPLAY "Time difference with Greenwich mean time for this time
000465-              "zone: "
000470              TIME-DIF TIME-DIF-H " Hours " TIME-DIF-M " Minutes"
000480     END-IF.
000490 END PROGRAM SAMPLE.
```

1.45 SUM Function

The SUM function returns the sum of the values specified in the arguments.

```
000010 @OPTIONS MAIN
000020*-----------------------------------------------------------------------
000030* The SUM function returns the sum of the values specified in the
000035* arguments.
000040* In this sample, the following functions are used at the same time:
000050* - MEAN function:   Returns the arithmetic average value of the
000055*                    arguments.
000060* - MEDIAN function: Returns the median value of the arguments.
000070*
000080*-----------------------------------------------------------------------
000090 IDENTIFICATION   DIVISION.
000100 PROGRAM-ID.      SAMPLE.
000110 DATA             DIVISION.
000120 WORKING-STORAGE  SECTION.
000130 01 .
```

```
000140   02 VAL         PIC S9(4) OCCURS 5 TIMES.
000150 01 TOTAL         PIC S9(8) VALUE 0.
000160 01 MEAN          PIC S9(8) VALUE 0.
000170 01 MEDIAN        PIC S9(8) VALUE 0.
000180 01 MIDRANGE      PIC S9(8) VALUE 0.
000190 01 SELECT-SW     PIC 9(1).
000200    88 SW-ALL     VALUE 1.
000210    88 SW-PART    VALUE 2.
000220 01 COUNTER       PIC 9(1).
000230 PROCEDURE        DIVISION.
000240     DISPLAY "Please input 5 values one at a time, not exceeding four
000245-            "digits.".
000250     PERFORM TEST BEFORE
000260             VARYING COUNTER FROM 1 BY 1 UNTIL COUNTER > 5
000270       DISPLAY "Value(" COUNTER ")" " >> " WITH NO ADVANCING
000280       ACCEPT VAL(COUNTER) FROM CONSOLE
000290     END-PERFORM.
000300     DISPLAY "(" WITH NO ADVANCING.
000310     PERFORM TEST BEFORE
000320             VARYING COUNTER FROM 1 BY 1 UNTIL COUNTER > 5
000330       DISPLAY VAL(COUNTER) " " WITH NO ADVANCING
000340     END-PERFORM.
000350     DISPLAY ")".
000360     DISPLAY " ".
000370     DISPLAY "Please select the processing method.".
000380     DISPLAY "1:  Process all.".
000390     DISPLAY "2:  Ignore the first and last. >>" WITH NO ADVANCING.
000400     ACCEPT SELECT-SW.
000410*-----------------------------------------------------------------------
000420* The sum, average value, and median value are obtained.  If all
000430* elements of a table are specified, the subscript ALL phrase can be
000435* substituted.
000440*-----------------------------------------------------------------------
000450     EVALUATE TRUE
000460     WHEN SW-ALL
000470       COMPUTE TOTAL  = FUNCTION SUM (VAL(ALL))
000480       COMPUTE MEAN   = FUNCTION MEAN(VAL(ALL))
000490       COMPUTE MEDIAN = FUNCTION MEDIAN(VAL(ALL))
000500*-----------------------------------------------------------------------
000510* The sum, average value, and median value are obtained.
000520*-----------------------------------------------------------------------
000530     WHEN SW-PART
000540       COMPUTE TOTAL  = FUNCTION SUM (VAL(2), VAL(3), VAL(4))
000550       COMPUTE MEAN   = FUNCTION MEAN(VAL(2), VAL(3), VAL(4))
000560       COMPUTE MEDIAN = FUNCTION MEDIAN(VAL(2), VAL(3), VAL(4))
000570*-----------------------------------------------------------------------
000580     WHEN OTHER
000590       DISPLAY "The selection is incorrect."
000600       EXIT PROGRAM
000610     END-EVALUATE.
000620     DISPLAY " ".
000630     DISPLAY "The total value is " TOTAL.
000640     DISPLAY "The average value is " MEAN.
000650     DISPLAY "The median value is " MEDIAN.
000660 END PROGRAM SAMPLE.
```

1.46 REM Function

The REM function returns the remainder that results when argument 1 is divided by argument 2.

The REM function is used when the remainder of a division operation is required.

```
000010 @OPTIONS MAIN
000020*---------------------------------------------------------------------
000030* The REM function is used to obtain the remainder of a division
000035* operation.
000040*---------------------------------------------------------------------
000050 IDENTIFICATION   DIVISION.
000060 PROGRAM-ID.      SAMPLE.
000070 DATA             DIVISION.
000080 WORKING-STORAGE  SECTION.
000090 01 BIRTHDAY      PIC 9(8).
000100 01 ELAPSED-DAYS  PIC 9(8) BINARY.
000110 01 REMAINDER-N   PIC 9(8) BINARY.
000120 CONSTANT         SECTION.
000130 01 DAYS-OF-THE-WEEK-TABLE.
000140    02            PIC X(9) VALUE "Sunday".
000150    02            PIC X(9) VALUE "Monday".
000160    02            PIC X(9) VALUE "Tuesday".
000170    02            PIC X(9) VALUE "Wednesday".
000180    02            PIC X(9) VALUE "Thursday".
000190    02            PIC X(9) VALUE "Friday".
000200    02            PIC X(9) VALUE "Saturday".
000210 01                REDEFINES DAYS-OF-THE-WEEK-TABLE.
000220    02 DAY-OF-THE-WEEK OCCURS 7 TIMES PIC X(9).
000230 PROCEDURE         DIVISION.
000240     DISPLAY "When is your birthday?  Example:  19690123 >>"
000245             WITH NO ADVANCING.
000250     ACCEPT BIRTHDAY.
000260*---------------------------------------------------------------------
000270* The INTEGER-OF-DATE obtains the elapsed days from January 1, 1601
000280* (Monday), divides the days by 7, and uses the remainder to obtain the
000281* day of the week.
000290*---------------------------------------------------------------------
000300     COMPUTE ELAPSED-DAYS = FUNCTION INTEGER-OF-DATE (BIRTHDAY).
000310     COMPUTE REMAINDER-N  = FUNCTION REM (ELAPSED-DAYS  7).
000320*---------------------------------------------------------------------
000330     DISPLAY "You were born on " DAY-OF-THE-WEEK (REMAINDER-N + 1) ".".
000340 END PROGRAM SAMPLE.
```

1.47 INTEGER-OF-DATE and DATE-OF-INTEGER Functions

The INTEGER-OF-DATE function accepts a date in the format YYYYMMDD and returns the number of days since January 1, 1601. The converse function, DATE-OF-INTEGER accepts an integer argument and converts it to a date in the format YYYYMMDD, where the integer argument is the number of days since January 1, 1601.

```
000010 @OPTIONS MAIN
000020*---------------------------------------------------------------------
000030* In this sample, the INTEGER-OF-DATE and DATE-OF-INTEGER functions are
000040* used to obtain the date after the specified number of days has
000045* elapsed.
000050*---------------------------------------------------------------------
000060 IDENTIFICATION   DIVISION.
000070 PROGRAM-ID.      SAMPLE.
000080 DATA             DIVISION.
000090 WORKING-STORAGE  SECTION.
000100 01 TODAY.
000110    02 YYYYMMDD   PIC 9(8).
000120 01 OTHER-DAY1    PIC S9(9) BINARY.
000130 01 OTHER-DAY2    PIC 9(8).
```

```
000140 01 DAYS          PIC S9(4) BINARY.
000150 PROCEDURE        DIVISION.
000160     MOVE FUNCTION CURRENT-DATE TO TODAY.
000170     DISPLAY "Today is " TODAY ".".
000180     DISPLAY "Obtain the date after how many days? >> " WITH NO
000185             ADVANCING.
000190     ACCEPT DAYS.
000200*-----------------------------------------------------------------
000210* The date after the specified number of days has elapsed is obtained
000215* by calculating the number of days from January 1, 1601 and adding the
000216* days entered by the user.
000220*-----------------------------------------------------------------
000230     COMPUTE OTHER-DAY1 = FUNCTION INTEGER-OF-DATE (YYYYMMDD) + DAYS.
000240*-----------------------------------------------------------------
000250* To display the date, the newly computed number of days is converted
000255* to standard format (YYYYMMDD).
000260*-----------------------------------------------------------------
000270     COMPUTE OTHER-DAY2 = FUNCTION DATE-OF-INTEGER (OTHER-DAY1).
000280*-----------------------------------------------------------------
000290     DISPLAY " ".
000300     DISPLAY "The date after " DAYS " days from " TODAY " is "
000305             OTHER-DAY2 ".".
000310 END PROGRAM SAMPLE.
```

1.48 LOWER-CASE and UPPER-CASE Functions

The LOWER-CASE function is used to convert upper-case letters in a character string specified as an argument to lower-case letters.

The LOWER-CASE function is useful for uniformly managing alphabetic data using lower-case letters.

The UPPER-CASE function is used to convert lower-case letters in a character string specified in an argument to upper-case letters.

The UPPER-CASE function is useful for uniformly managing alphabetic data using upper-case letters.

```
000010 @OPTIONS MAIN
000020*-----------------------------------------------------------------
000030* The LOWER-CASE function is used to uniformly manage alphabetic data
000040* using lower-case letters.
000050* The UPPER-CASE function is used to uniformly manage alphabetic data
000055* using upper-case letters.
000060*-----------------------------------------------------------------
000070 IDENTIFICATION  DIVISION.
000080 PROGRAM-ID.     SAMPLE.
000090*
000100 DATA            DIVISION.
000110 WORKING-STORAGE SECTION.
000120 01 IN-STR       PIC X(40).
000130 01 LOWER-STR    PIC X(40).
000140 01 UPPER-STR    PIC X(40).
000150*
000160 PROCEDURE       DIVISION.
000170     DISPLAY "Please input a name using alphabetic characters. >>"
000175             WITH NO ADVANCING.
000180     ACCEPT IN-STR FROM CONSOLE.
000190*-----------------------------------------------------------------
000200* The upper-case letters are converted to lower-case letters.
000210*-----------------------------------------------------------------
000220     MOVE FUNCTION LOWER-CASE (IN-STR) TO LOWER-STR.
000223*-----------------------------------------------------------------
000225* The lower-case letters are converted to upper-case letters.
000227*-----------------------------------------------------------------
```

```
000230     MOVE FUNCTION UPPER-CASE (IN-STR) TO UPPER-STR.
000240*----------------------------------------------------------------
000250     DISPLAY " ".
000260     DISPLAY "Lower-case letter notation: " LOWER-STR.
000270     DISPLAY "Upper-case letter notation: " UPPER-STR.
000280 END PROGRAM SAMPLE.
```

1.49 MAX and MIN Function

The MAX function returns the maximum value of an argument consisting of several different values. Likewise, the MIN function returns the minimum value from the list of values that is the argument to the function. A table of values may be passed as an argument to these functions, as illustrated below.

```
000010 @OPTIONS MAIN
000020*----------------------------------------------------------------
000030* In this sample, the MAX and MIN functions are used to obtain the
000040* maximum and minimum values of three input values.
000050*----------------------------------------------------------------
000060 IDENTIFICATION   DIVISION.
000070 PROGRAM-ID.      SAMPLE.
000080 DATA             DIVISION.
000090 WORKING-STORAGE  SECTION.
000100 01 .
000110    02 VAL        OCCURS 3 TIMES PIC 9(4).
000120 01 MAX-VAL       PIC 9(5).
000130 01 MIN-VAL       PIC 9(5).
000140 01 COUNTER       PIC 9(1).
000150 PROCEDURE        DIVISION.
000160     DISPLAY "Please input three values.".
000170     PERFORM TEST BEFORE
000180             VARYING COUNTER FROM 1 BY 1 UNTIL COUNTER > 3
000190       DISPLAY "Value (up to four digits) >> " WITH NO ADVANCING
000200       ACCEPT VAL(COUNTER)
000210     END-PERFORM.
000220     DISPLAY " ".
000230     DISPLAY "The maximum and minimum values of the input data are
000235-            "determined."
000240*----------------------------------------------------------------
000250* The maximum and minimum values are determined.
000260*----------------------------------------------------------------
000270     COMPUTE MAX-VAL = FUNCTION MAX (VAL(ALL)).
000280     COMPUTE MIN-VAL = FUNCTION MIN (VAL(ALL)).
000290*----------------------------------------------------------------
000300     DISPLAY "The maximum value is " MAX-VAL ".".
000310     DISPLAY "The minimum value is " MIN-VAL ".".
000320 END PROGRAM SAMPLE.
```

1.50 REVERSE Function

The REVERSE function reverses the order of characters in the string passed as an argument to the function.

```
000010 @OPTIONS MAIN
000020*----------------------------------------------------------------
000030* The REVERSE function reverses the order of character strings in
000035* arguments.
000040*----------------------------------------------------------------
```

```
000050 IDENTIFICATION   DIVISION.
000060 PROGRAM-ID.      SAMPLE.
000070 DATA             DIVISION.
000080 WORKING-STORAGE  SECTION.
000090 01 IN-STR        PIC X(10).
000100 01 REV-STR       PIC X(10).
000110 PROCEDURE        DIVISION.
000120     DISPLAY "Please input 10 alphabetic characters. >> " WITH NO
000125              ADVANCING.
000130     ACCEPT IN-STR FROM CONSOLE.
000140*-------------------------------------------------------------------------
000150* The order of the characters in the input string is reversed.
000160*-------------------------------------------------------------------------
000170     MOVE FUNCTION REVERSE (IN-STR) TO REV-STR.
000180*-------------------------------------------------------------------------
000190     DISPLAY " ".
000200     DISPLAY "Input character string:    " IN-STR.
000210     DISPLAY "Reversed character string: " REV-STR.
000220 END PROGRAM SAMPLE.
```

1.51 STORED-CHAR-LENGTH Function

The STORED-CHAR-LENGTH function returns the length of valid data excluding any trailing spaces included in an argument. Because the STORED-CHAR-LENGTH function returns the number of character positions, this function is useful for checking for numeric truncation or for determining significant data in an alphanumeric field for use with reference modification.

The STORED-CHAR-LENGTH function is unique to Win32, Winx64, Solaris, Linux, LinuxIPF, Linux64 and .NET.

```
000010 @OPTIONS MAIN
000020*-------------------------------------------------------------------------
000030* The STORED-CHAR-LENGTH function returns the valid data length in a
000035* data item.
000040*-------------------------------------------------------------------------
000050 IDENTIFICATION   DIVISION.
000060 PROGRAM-ID.      SAMPLE.
000070 DATA             DIVISION.
000080 WORKING-STORAGE  SECTION.
000090 01 IN-DATA       PIC X(80).
000100 01 DATA-LEN      PIC 9(4) BINARY.
000110 01 BUFFER        PIC X(40) VALUE ALL "*".
000120**
000130 PROCEDURE        DIVISION.
000140     DISPLAY "Please input a character string. >> " WITH NO ADVANCING.
000150     ACCEPT IN-DATA FROM CONSOLE.
000160*-------------------------------------------------------------------------
000170* If the argument is a national data item, the returned value of the
000180* STORED-CHAR-LENGTH function is number of characters.  If the argument
000190* is an alphanumeric data item, the value returned is the number of
000200* bytes.
000210*-------------------------------------------------------------------------
000220     COMPUTE DATA-LEN = FUNCTION STORED-CHAR-LENGTH (IN-DATA).
000230*-------------------------------------------------------------------------
000240     IF DATA-LEN > FUNCTION LENGTH(BUFFER) THEN
000250       DISPLAY "The input data length has exceeded the buffer length."
000260     ELSE
000270       MOVE IN-DATA(1:DATA-LEN) TO  BUFFER(1:DATA-LEN)
000280       DISPLAY "BUFFER = " BUFFER
000290     END-IF.
000300 END PROGRAM SAMPLE.
```

1.52 WHEN-COMPILED Function

The WHEN-COMPILED function returns the date, time, and time difference with Greenwich mean time when a program was compiled.

```
000010 @OPTIONS MAIN
000020*---------------------------------------------------------------------
000030* The WHEN-COMPILED function is used to obtain the program compilation
000035* date.
000040*---------------------------------------------------------------------
000050 IDENTIFICATION   DIVISION.
000060 PROGRAM-ID.      SAMPLE.
000070 DATA             DIVISION.
000080 WORKING-STORAGE  SECTION.
000090 01 COMPILATION-DATE.
000100    02 YEAR         PIC 9(4).
000110    02 MONTH        PIC 9(2).
000120    02 MONTH-DAY    PIC 9(2).
000130    02 HOUR         PIC 9(2).
000140    02 MINUTE       PIC 9(2).
000150    02 SECOND       PIC 9(2).
000160 01 EDITING-DATA.
000170    02 YEAR         PIC 9(4).
000180    02             PIC X(1) VALUE "/".
000190    02 MONTH        PIC 9(2).
000200    02             PIC X(1) VALUE "/".
000210    02 MONTH-DAY    PIC 9(2).
000220    02             PIC X(1) VALUE SPACE.
000230    02 HOUR         PIC 9(2).
000240    02             PIC X(1) VALUE ":".
000250    02 MINUTE       PIC 9(2).
000260    02             PIC X(1) VALUE ":".
000270    02 SECOND       PIC 9(2).
000280 PROCEDURE        DIVISION.
000290*---------------------------------------------------------------------
000300* The program compilation date and compile time are retrieved.
000310*---------------------------------------------------------------------
000320     MOVE FUNCTION WHEN-COMPILED TO COMPILATION-DATE.
000330*---------------------------------------------------------------------
000340     MOVE CORRESPONDING COMPILATION-DATE TO EDITING-DATA.
000350     DISPLAY "This program was compiled " EDITING-DATA ".".
000360 END PROGRAM SAMPLE.
```

1.53 COPY Statement

When multiple programs use common data declarations or procedures, the COPY statement is used to create and retrieve copy libraries as required.

At this time, words in a library can be retrieved and replaced (format 1) or partial character strings in words can be retrieved and replaced (format 2).

1.53.1 COPY Statement (format 1)

<Library "CP-SMPL1.CBL">

```
000010*---------------------------------------------------------------------
000020* Date data is defined in this library.
000030*---------------------------------------------------------------------
000040  01 FMT-DATE.
000050    02 YYYY       PIC 9(4).
```

```
000060    02 MMDD.
000070      03 MM         PIC 9(2).
000080      03 DD         PIC 9(2).
```

<Source program>

```
000010 @OPTIONS MAIN
000020*---------------------------------------------------------------------
000030* The COPY statement is specified to read the library.
000040*---------------------------------------------------------------------
000050 IDENTIFICATION   DIVISION.
000060 PROGRAM-ID.      SAMPLE.
000070 DATA             DIVISION.
000080 WORKING-STORAGE SECTION.
000090*---------------------------------------------------------------------
000100* Specifying REPLACING in the COPY statement enables words (text words)
000110* in a library to be replaced and expanded.
000120* In this example, one group item name is replaced with "TODAY" and
000130* another group item name is replaced with "BIRTHDAY".
000140*---------------------------------------------------------------------
000150 COPY CP-SMPL1 REPLACING FMT-DATE BY TODAY.
000160 COPY CP-SMPL1 REPLACING FMT-DATE BY BIRTHDAY.
000170*---------------------------------------------------------------------
000180 01 AGE          PIC 9(3).
000190 PROCEDURE        DIVISION.
000200     DISPLAY "When is your birthday?  Example:  19690123 >>"
000205             WITH NO ADVANCING.
000210     ACCEPT BIRTHDAY.
000220     MOVE FUNCTION CURRENT-DATE TO TODAY.
000230     COMPUTE AGE = YYYY OF TODAY - YYYY OF BIRTHDAY.
000240     IF MMDD OF TODAY < MMDD OF BIRTHDAY THEN
000250       COMPUTE AGE = AGE - 1
000260     END-IF.
000270     DISPLAY " ".
000280     DISPLAY "You are " AGE " years old.".
000290 END PROGRAM SAMPLE.
```

1.53.2 COPY Statement (format 2)

<Library "CP-SMPL2.CBL">

```
000010*---------------------------------------------------------------------
000020* Date data is defined in this library.
000030*---------------------------------------------------------------------
000040  01 XXX-DATE.
000050    02 XXX-YYYY   PIC 9(4).
000060    02 XXX-MMDD.
000070      03 MM        PIC 9(2).
000080      03 DD        PIC 9(2).
```

<Source program>

```
000010 @OPTIONS MAIN
000020*---------------------------------------------------------------------
000030* When expanding a copy library, partial character strings can also be
000035* replaced.
000040*---------------------------------------------------------------------
000050 IDENTIFICATION   DIVISION.
000060 PROGRAM-ID.      SAMPLE.
000070 DATA             DIVISION.
000080 WORKING-STORAGE  SECTION.
```

```
000090*---------------------------------------------------------------
000100* The DISJOINING and JOINING phrases can be used to replace and expand
000110* partial character strings (words linked using a hyphen) of text words
000115* coded in a library.
000120*---------------------------------------------------------------
000130  COPY CP-SMPL2 DISJOINING XXX JOINING TODAY    AS PREFIX.
000140  COPY CP-SMPL2 DISJOINING XXX JOINING BIRTHDAY AS PREFIX.
000150*---------------------------------------------------------------
000160 01 AGE           PIC 9(3).
000170 PROCEDURE        DIVISION.
000180     DISPLAY "When is your birthday?  Example:  19690123 >>"
000185             WITH NO ADVANCING.
000190     ACCEPT BIRTHDAY-DATE.
000200     MOVE FUNCTION CURRENT-DATE TO TODAY-DATE.
000210     COMPUTE AGE = TODAY-YYYY - BIRTHDAY-YYYY.
000220     IF TODAY-MMDD < BIRTHDAY-MMDD THEN
000230       COMPUTE AGE = AGE - 1
000240     END-IF.
000250     DISPLAY " ".
000260     DISPLAY "You are " AGE " years old.".
000270 END PROGRAM SAMPLE.
```

Index