Declaring Data in COBOL

Introduction

Aims

The aim of this unit is give you an understanding of the different categories of data used in a COBOL program and to demonstrate how items of each category may be created and used.

Objectives

By the end of this unit you should -

1. Know how to use and create literals, variables and Figurative Constants.
2. Understand how to declare numeric, alphabetic and alphanumeric data-items.
3. Be able to create a record structure by assigning the appropriate level numbers to its data-items.
4. Understand the difference between group and an elementary data-items.
5. Be able to assign an initial value to a variable.

Prerequisites

Introduction to COBOL.

but more information on declaring data items in COBOL can be found in the units covering

- Edited Pictures
- Sequential Files
- The USAGE clause

Further reading

More information on declaring data items in COBOL can be found in the units covering -

- Edited Pictures
- Sequential Files
- The USAGE clause

To top of page

Categories of COBOL data

Introduction

There are three categories of data item used in COBOL programs:

- Variables.
- Literals.
- Figurative Constants.

Variables

A data-name or identifier is the name used to identify the area of memory reserved for a variable. A variable is a named location in memory into which a program can put data, and from which it can retrieve data.

Every variable used in a COBOL program must be described in the DATA DIVISION.

In addition to the data-name, a variable declaration also defines the type of data to be stored in the variable. This is known as the variable's data type.

Variable Data types

Some languages like Modula-2,Pascal or Ada are described as being *strongly typed*. In these languages there are a large number of different data types and the distinction between them is rigorously enforced by the compiler. For instance, the compiler will reject a statement that attempts to assign character value to an integer data item.

In COBOL, there are really only three data types -

- numeric
- alphanumeric (text/string)
- alphabetic

The distinction between these data types is a little blurred and only weakly enforced by the compiler. For instance, it is perfectly possible to assign a non-numeric value to a data item that has been declared to be numeric.

Attempting to perform computations on numeric data items that contain non-numeric data is a frequent cause of program crashes for beginning COBOL programmers.

The problem with this lax approach to data typing is that, since COBOL programs crash (halt unexpectedly) if they attempt to do computations on items that contain non-numeric data, it is up to the programmer to make sure this never happens.

COBOL programmers must make sure that non-numeric data is never assigned to numeric items intended for use in calculations. Programmers who use strongly typed languages don't need this level of discipline because the compiler ensures that a variable of a particular types can only be assigned appropriate values.

Literals

A literal is a data-item that consists only of the data-item value itself. It cannot be referred to by a name. By definition, literals are constant data-items.

There are two types of literal -

- String/Alphanumeric Literals
- Numeric Literals

String Literals

String/Alphanumeric literals are enclosed in quotes and consist of alphanumeric characters.

For example: "Michael Ryan", "-123", "123.45"

Numeric Literals

Numeric literals may consist of numerals, the decimal point, and the plus or minus sign. Numeric literals are not enclosed in quotes.

For example: 123, 123.45, -256, +2987

Figurative Constants

Unlike most other programming languages COBOL does not provide a mechanism for creating user-defined constants but it does provide a set of special constants called Figurative Constants.

A Figurative Constant may be used wherever it is legal to use a literal but unlike literals, when a Figurative Constant is assigned to a data-item it fills the whole item overwriting everything in it.

Actually COBOL does allow you to set up single character user-defined Figurative Constants. These can be useful if you need to use the non-printable ASCII characters such as ESC or FormFeed.

User-defined Figurative Constants are declared in the SYMBOLIC CHARACTERS clause of the ENVIRONMENT DIVISION.

In an extension that previews the new COBOL specification, NetExpress does allow user-defined constants. It uses the level 78 for this purpose.

The Figurative Constants are:

SPACE or **SPACES**	Acts like one or more spaces
ZERO or **ZEROS** or **ZEROES**	Acts like one or more zeros
QUOTE or **QUOTES**	Used instead of a quotation mark
HIGH-VALUE or **HIGH-VALUES**	Uses the maximum value possible
LOW-VALUE or **LOW-VALUES**	Uses the minimum value possible
ALL literal	Allows a ordinary literal to act as Figurative Constant

Figurative Constant Notes

- When the ALL Figurative Constant is used, it must be followed by a one character literal. The designated literal then acts like the standard Figurative Constants.
- ZERO, ZEROS and ZEROES are synonyms, not separate Figurative Constants. The same applies to SPACE and SPACES, QUOTE and QUOTES, HIGH-VALUE and HIGH-VALUES, LOW-VALUES and LOW-VALUES.

Using COBOLdata ------ examples ------

The animated program fragments below demonstrate how variables, literals and Figurative Constants may be created and used.

To top of page

Declaring Data-Items in COBOL

Introduction

Because COBOL is not a typed language like Modula-2 or C it employs a different mechanism for describing the characteristics of the data-items in a program.

Rather than using types, as these languages do, COBOL uses a kind of "declaration by example" strategy. The programmer provides the system with an example, or template, or PICture of the storage required for the data-item.

In COBOL, a variable declaration consists of a line in the DATA DIVISION that contains the following items:

- A level number.
- A data-name or identifier.
- A Picture clause.

COBOL picture clauses

To create the required 'picture' the programmer uses a set of symbols. The most common symbols used in standard picture clauses are:

There are actually many more picture symbols than these. Most of these will be introduced when we cover Edited Pictures.

9	The digit nine is used to indicate the occurrence of a digit at the corresponding position in the picture.
X	The character X is used to indicate the occurrence of any character from the character set at the corresponding position in the picture.
A	The character A is used to indicate the occurrence of any alphabetic character (A to Z plus blank) at the corresponding position in the picture.
V	The character V is used to indicate the position of the decimal point in a numeric value. It is often referred to as the "assumed decimal point". It is called that because, although the actual decimal point is not stored, values are treated as if they had a decimal point in that position.
S	The character S indicates the presence of a sign and can only appear at the beginning of a picture.

Picture clause notes

Although the word PICTURE can be used when defining a picture clause it is normal to use the abbreviation PIC.

Recurring symbols may be specified by using a 'repeat' factor inside brackets. For instance:

```
PIC 9(6)       is equivalent to PICTURE 999999
PIC 9(6)V99    is equivalent to PIC 999999V99
PICTURE X(10)  is equivalent to PIC XXXXXXXXXX
PIC S9(4)V9(4) is equivalent to PIC S9999V9999
PIC 9(18)      is equivalent to PIC 999999999999999999
```

Numeric values can have a maximum of 18 (eighteen) digits.

The limit on string values is usually system dependent.

To top of page

Group and Elementary data-items

Introduction

Although we stated above that each variable declaration consists of a level number, an identifying name and a picture clause, that definition only applies to elementary data-items. Group items are defined using only a level-number and an identifying name; no picture clause is required or allowed.

Which begs the question - what is a group item and what is an elementary item?

Elementary items

An "elementary item" is the name we use in COBOL to describe a data-item that has not been further subdivided. Other languages might describe these as ordinary variables.

Elementary items **must** have a picture clause because they actually reserve the storage required for the item. The amount of storage reserved is specified by the item's picture clause.

An elementary item declaration consists of;

- a level number
- a data name
- a picture clause

A variable declaration may also have a number of other clauses such as USAGE or BLANK WHEN ZERO

A starting value may be assigned to a variable by means of an extension to the PICTURE clause called the VALUE clause.

Some examples:

```
01 GrossPay       PIC 9(5)V99 VALUE ZEROS.

01 NetPay         PIC 9(5)V99 VALUE ZEROS.

01 CustomerName   PIC X(20) VALUE SPACES.

01 CustDiscount   PIC V99 VALUE .25.
```

Group items

Sometimes when we are manipulating data it is convenient to treat a collection of elementary items as a single group. For instance, we may want to group the data-items YearofBirth, MonthofBirth, DayOfBirth under the group name - DateOfBirth. If we are recording information about students we may want to subdivide StudentName into FirstName, MiddleInitial and Surname. And we may want to use both these group items and the elementary items StudentId and CourseCode in a student record description.

We can create groups like these in COBOL using group items. A "group item" is the term used in COBOL to describe a data-item - like DateOfBirth or StudentName - that has been further subdivided. In other languages group items might be described as "structures".

A group item consists of subordinate items. The items subordinate to a group item may be elementary items or other group items. But ultimately every group item must be defined in terms of its subordinate elementary items.

In a group item, the hierarchical relationship between the various subordinate items of the group is expressed using level numbers. The higher the level number, the lower the item is in the hierarchy. Where a group item is the highest item in a data hierarchy it is referred to as a "record" and uses the level number 01.

Group items are declared using a level number and a data name only. A group item cannot have a picture clause because it does not actually reserve any storage. It is merely a name given to a collection of (ultimately) elementary items which do reserve the storage.

Therefore, the size of a group item is the sum of the sizes of its subordinate elementary items.

The type of a group item is always assumed to be PIC X because a group item may have several different data items and types subordinate to it and an X picture is the only one which could support such collections.

Level Numbers

Level numbers are used to express data hierarchy. The higher the level number, the lower the item is in the hierarchy. At the lowest level the data is completely atomic.

What is important in a structure defined with level numbers is the *relationship* of the level numbers to one another, not the actual level numbers used. For instance, the record descriptions shown below are equivalent.

Although level numbers specify the actual data hierarchy you should use indentation to provide a graphic representation of it. This will make your programs easier to read. For instance, indentation makes it obvious that DayOfBirth, MonthOfBirth and YearOfBirth are subordinate items of DateOfBirth, while CourseCode is not.

Record-A

```
01 StudentDetails.
   02 StudentId        PIC 9(7).
   02 StudentName.
      03 FirstName     PIC X(10).
      03 MiddleInitial PIC X.
      03 Surname       PIC X(15).
   02 DateOfBirth.
      03 DayOfBirth    PIC 99.
      03 MonthOfBirth  PIC 99.
      03 YearOfBirth   PIC 9(4).
   02 CourseCode       PIC X(4).
```

Record-B

```
01 StudentDetails.
   05 StudentId        PIC 9(7).
   05 StudentName.
      07 FirstName     PIC X(10).
      07 MiddleInitial PIC X.
      07 Surname       PIC X(15).
   05 DateOfBirth.
      07 DayOfBirth    PIC 99.
      07 MonthOfBirth  PIC 99.
      07 YearOfBirth   PIC 9(4).
   05 CourseCode       PIC X(4).
```

Some observations on Record-A

It is useful to examine Record-A above and to answer the following questions:

Q1. What is the size (in characters) of Record-A?

Q2. What is the size of the data-item StudentName?

Q3. What is the size of DateOfBirth?

Q4. What is the data type of DateOfBirth? Is it numeric, alphabetic or alphanumeric.

A1. The size of Record-A is the sum of the sizes of all the

elementary items subordinate to it (7+10+1+15+2+2+4+4 = 45 characters).

A2. The size of StudentName is the sum of the sizes of FirstName, MiddleInitial and Surname. So StudentName is 26 characters in size (10+1+15).

A3. DateOfBirth is 8 characters in size (2+2+4).

A4. The data type of DateOfBirth is alphanumeric (i.e. PIC X) even though all its subordinate items are numeric because the type of a group item is always alphanumeric.

Level number notes

The level numbers 01 through 49 are general level numbers but there are also special level numbers such as 66, 77 and 88.

- Level 77's can only be used to define individual elementary items. The use of 77's is banned in some shops who take the view that instead of declaring large numbers of indistinguishable 77's it is better to collect the individual items into groups.
- Level 88's are used to define Condition Names.
- Level 66's (RENAMES clause) are used to apply a new name to an identifier or group of identifiers. It is not generally used in modern COBOL programs.

Building a record structure

In the animation below we show how level numbers can be used to define a record structure as hierarchy of data-items.

To top of page

Last updated : March 1999

e-mail : CSISwebeditor@ul.ie