A GENERAL TEST DATA GENERATOR FOR COBOL

Lt. Richard L. Sauder
Automation Techniques Branch
Headquarters, Air Force Logistics Command
Wright-Patterson Air Force Base, Ohio

This article discusses the effort being made by the Air Force Logistics Command in developing a method of generating effective program test data. This "Test Data Generator" is designed to operate in conjunction with the COBOL compiler implemented by AFLC. As such, the system not only builds data conforming to descriptions given in the Data Division of a COBOL program but also places in these items necessary data relationships to test the logic of the COBOL program. Both the utilization and the method of operation of the system are discussed in this paper.

Introduction

One of the major underdeveloped areas that still exists in the development of programming techniques is that of insuring adequate checkout of programs before release for use. Often the logical paths within a program are highly complex. The effort to insure that each of these is functionally correct can be prohibitive - so much so that only the most obvious and most frequently employed segments of the program are tested thoroughly.

Compilers that have been employed in the past decade have aided in reducing this problem. By eliminating many careless coding errors, they have not only insured a greater degree of operation in a shorter time but have also released more time for more elaborate testing of the compiled program. The fact still remains, however, that unless adequate test data is available the debugging effort is hampered. Recognizing this deficiency, the Air Force Logistics Command last year began developing a method of producing this data. The immediate result is a Test Data Generator designed to operate in conjunction with the COBOL compiler implemented by AFLC. As such it performs two main functions:

1. Builds data of the format and description specified in the COBOL source program and
2. Inserts in these elements data specified by the user to meet inter (and intra) element requirements and relationships.

These functions basically fulfill the needs for producing effective test data. Obviously it is necessary to produce elements conforming to their given formats. But since there are few business oriented runs which do not require dependency among various elements, a more pertinent criterion is that of controlling the content of these elements.

The method of operation and the information necessary to fulfill these functions will now be described in detail.

Utilization of the COBOL Data Division

Determining the format of data fields results from a thorough interpretation of the Data Division of the COBOL source program being tested. Within this Division, options exist to describe in detail both the structure of every element within a record and the relationships of these elements to one another. Those options which affect the generation of data are defined as follows:

1. Name
2. Level number to establish the relationship of one unit of data to others.
3. Size in number of characters.
4. Class indicating the type of data, i.e., alphabetic, numeric, alphanumeric.
5. Usage to establish the number system in which the element is represented.
6. Occurs to define repeated occurrences of the same element.
7. Range to establish limits for the value of the element.
8. Sign specifying the operational sign of the element.
9. Synchronize positioning the element within or across computer words.
10. Redefines to allow the same area to have more than one description.
11. Picture - a graphical representation of the element.
12. Value to define a stated value for the element.

In addition to these element descriptions, options are present which describe the files to which the records belong, e.g., tape and file labels, the number of records within a file and the number of elements within these records.

Construction of Formatted Data

Since in the generation of data we are concerned only with those records in a program designated for input, the interpretation of the source program Data Division produces only sets of parameters dealing with elements of input records.
In all cases this set includes the size of the element of data and its relative location within the record. Other information produced depends on the type of element. Type in this context is defined as one of four classifications:

1. **Literal** indicating that a single value has been defined for the element.
2. **Conditional** indicating that a restricted set of variables only may appear in the field. (Conditional values in COBOL data descriptions are designated by a level number of 88.)
3. **Random** indicating that no values were assigned to the field so that the content of the element is determined only by the options: class, range, and base.
4. **Sequential** indicating that the original value of the element is to be incremented by a fixed or a random amount.

All information concerning these types can be obtained from the data descriptions with the exception of those items to be sequenced, and the amount, if known, by which they are to be incremented. Therefore, disregarding these items, it is now possible to generate "garbage" test records with only the COBOL source tape for the program serving as input. The method of generation if this were desired would be as follows:

1. **Interpretation of the program's Data Division**; construction of sets of parameters and grouping of these parameters by record designation.
2. **Isolation of all parameters for a particular record and employment of each set of parameters in turn, to build a single test record with the following criteria:**
   a. If the element is literal or conditional, insert the value or chosen value into its relative position in the record.
   b. If the item is random, generate random characters (determined by the class of the item) filling the element.
3. **Repetition of step two until the desired number of test records has been generated. Repetition of the entire process for the next record description, and so forth.**

In testing applications, it is often true that some information which would be generated in this manner is insignificant or unnecessary. For instance, a thirty character random field reserved for a manufacturer's name could possibly be deleted or replaced by a two character code representing the name if this element is not involved in any logical decision in the program. This action would not only reduce the time needed for generation, but also would considerably improve the appearance of the data. In addition, elements may not be described in the Data Division to a sufficient degree for testing purposes. For instance, a stock-number may be described as alphanumeric when in reality the first half of the number is alphabetic and the remainder numeric.

For these reasons the Data Generator includes the option to modify data descriptions by the deletion or replacement of these descriptions or by the insertion of new descriptions. The method by which these modifications are expressed will be discussed later.

**Specification of Data Relationships**

The data generated in the manner described falls far short of being adequate for production testing. Though some control on the content of the records is obtained by stating literal and conditional values for elements, the great majority of the output is unrelated "garbage" adequate only for testing error logic. For this reason the ability to specify data relationships and requirements is included with the generator. The type of relationships desired may be of the following nature:

If field-1 of record-A equals field-1 of record-B then field-2 of record-A must be in the range 7-15. If field-2 of record-A is less than 10 and field-3 of record-A is not less than 24 then field-7 of record-B must equal "A"; but if field-2 of record-A equals 10, then field-7 of record-B equals "B". If field-7 of record-B equals "C" then field-8 of record-B equals 0 or 2 . . . . . . .

Via this option the formatted records produced can be edited to contain combinations of these relationships, which in turn can test the logic of the program concerned. To allow a flexibility of this magnitude, a procedural type system similar in objective to the COBOL Procedure Division is provided. Just as the Procedure Division contains a series of statements arranged to specify the logic of a computer program, this system (named the Relation Section) is a series of statements arranged to specify the logic of data relationships.

The Relation Section is formed by a series of conditional and declarative statements consisting of a restricted set of operations and the operands involved. The operands are those element names which appear in the Data Division of the program or those defined by the method of modification explained before, and any quantity appearing as a literal value. The same conventions apply to expressing these operands as in the COBOL Procedure Division.

**Example:**

\[
\text{If } \text{FIELD-1} \text{ equals SUBFIELD-X of FIELD-A of RECORD-1 . . .} \\
\text{If } \text{CLASS-CODE-1} \text{ equals "ABC" then IN-CODE of FIELD-C equals 377 . . .} \\
\]

The set of operations allowed consists of three types: relational, arithmetic and control.
A GENERAL TEST DATA GENERATOR FOR COBOL

Relational Operations

The relational elements are the same as those defined in the COBOL language for expressing conditional relationships. In the Relation Section, however, these elements serve a dual purpose. They first express conditional relationships as in COBOL. In addition, however, they are employed in imperative statements to prescribe an action between given operands. Used this way, the convention is established that given

\[ \text{opnd-m} \ (\text{relation}) \ \text{opnd-n}, \]

\[ \text{opnd-m} \] is adjusted, if necessary, to satisfy the given relation. For instance, given: \( \text{FLD-A} \) equals \( \text{FLD-B} \). If inequality exists, the content of \( \text{FLD-A} \) is made equal to that of \( \text{FLD-B} \).

The relations (and their associated mnemonics) are:

- \( \text{EQ} \) equals
- \( \text{NEQ} \) is not equal to
- \( \text{GR} \) is greater than
- \( \text{NGR} \) is not greater than
- \( \text{LS} \) is less than
- \( \text{NLS} \) is not less than
- \( \text{ZER} \) is zero
- \( \text{NZR} \) is not zero
- \( \text{POS} \) is positive
- \( \text{NEG} \) is negative
- \( \text{NEG} \) is not negative
- \( \text{RNG} \) range of \( \text{element-j} \) is \( (\text{value-a}) \) - \( (\text{value-b}) \), \( (\text{value-c}) \), \( (\text{value-d}) \)...(value-m).

Examples:

1. If \( \text{FLD-B} \) \( \text{EQ} \) \( \text{FLD-C} \), \( \text{ITEM-NUM-L} \) \( \text{MUL} \) \( 200017 \).
2. \( \text{FLD-B} \) of \( \text{RECORD-1} \) \( \text{RNG} \); \( \text{FLD-2} \) of \( \text{FLD-B} \) of \( \text{RECORD-2} \).
3. If \( \text{ELEMENT-C} \) \( \text{GR} \) 7 or if \( \text{ELEMENT-P} \) \( \text{EQ} \) \( "\text{AB}" \), \( \text{RNG} \) \( \text{FLD-3A} \) is 7-19.
4. \( \text{RNG} \) \( \text{ITEM-PQ} \) is \( "x", "d", "b", "x", 9, 13 \).

Arithmetic Operations

- \( + \text{element-j} \) \( \text{PLUS} \) \( \text{element-k} \)
- \( - \text{element-1} \) \( \text{MINUS} \) \( \text{element-m} \)
- \( * \text{element-n} \) \( \text{TIMES} \) \( \text{element-p} \)
- \( / \text{element-q} \) \( \text{DIVIDED BY} \) \( \text{element-r} \)

Examples:

1. If \( \text{FLD-B} \) \( \text{EQ} \) \( \text{FLD-C} \) \( + \) \( \text{FLD-D} \), then \( \text{ITEM-1 EQ} \) \( \text{ELEMENT-1} \) \( \times \) 3.
2. If \( \text{ELEMENT-B} - 35 \text{GR} \) 7, then \( \text{RNG} \) \( \text{FLD-F} \) is 0, 1.

Control Operations

At present three control operations are permitted.

1. \( \text{END} \) denotes completion of data specifications.
2. \( \text{GO} \) to \path{1}.

Example: If \( \text{FLD-R} \) \( \text{EQ} \) 7 and \( \text{FLD-P} \) \( \text{MUL} \) \( 100 \), \( \text{GO} \) to \path{1}; otherwise \( \text{GO} \) to \path{2}.

3. \( \text{BILD} \) \( \text{BUILD} \) record-name-j.
   This operation signals the construction of a particular record described in the Data Division.
   If references are made to elements within a record for which the BILD option has not been stated, these items are ignored.

Example: 1. \( \text{BILD RECORD-1}, \text{RECORD-2} \).
2. If \( \text{ITEM-CODE} \) is \( \text{POS} \), \( \text{BILD} \) \( \text{RECORD-3} \).

Additional Relation Section Options

Several storage areas of varying sizes are set aside for use with the Relation Section. Each of these areas has a fixed name and may be referenced as an operand by any appropriate operation stated. The major reason for providing these areas is to allow trailer type items to be specified. For instance, if a stock-number of an item must appear in fifteen successive trailers, this stock-number can be generated and placed in a storage area and then be moved later to the appropriate element when the trailer items are built.

These areas must be used when an element of a table is referenced by subscripting.

Examples.

1. If \( \text{TRAILER-CODE-1 EQ} \) 1, \( \text{BILD-19 EQ} \) \( \text{STOCK-NO-1} \).
2. \( \text{BILD-6 EQ} \) 7. If \( \text{TABLE-1} (\text{BILD-6}) \) \( \text{EQ} \) \( \text{TABLE-2} (\text{BILD-6}) \).

NOTE: \( \text{BILD-6} \) and \( \text{BILD-19} \) are names of assigned storage areas.

Preparation of Relation Section for Input

The examples that have been given for the use of the Relation Section have indicated a
moderately free format. On the contrary, a tabular form of input has been chosen because the task of specifying the operands and the desired operation (or operations) is generally simplified by employing tables. Two tables - one containing operands, the other containing operations - are required to specify a set of relations and actions. Figure 1 describes the formats of these tables. In this illustration also is the tabular representation of the following series of statements.

A1. If IN-1 EQ FLD-2, then STOCK of RECORD-A EQ STOCK of RECORD-2, BILD-6 EQ 5, TBL-1 (BILD-6) OR TBL-2 (BILD-6).
A2. If START-ODDS NLS 4, then RNG FLD-18 is "A", "B", "X", go to A3; otherwise go to A4.
A4. BLD RECORD-C. Go to A5.
A5. BLD RECORD-A, RECORD-B. Go to A3.

In this tabular form, series of disjunctive statements are expressed horizontally across successive columns; conjunctive statements are expressed vertically in one column. For example, given the series of expressions:

If (A EQ B and C LS D) or J NLS K or (E NQ "X" and F FOS), THEN BID Q.

This series is expressed tabularly as follows:

<table>
<thead>
<tr>
<th>Condition-Group</th>
<th>Operation-Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>P2</td>
</tr>
<tr>
<td>10 A</td>
<td>20 B</td>
</tr>
<tr>
<td>50 E</td>
<td>60 F</td>
</tr>
</tbody>
</table>

Additional Necessary Information

Two additional sections are provided in the input format for expressing those options needed in constructing formatted data.

Control Section. Within this section information is given as to the number of test items to be constructed and those elements to be sequenced. The COUNT option specifies this total number for each record within one file. The format is as follows:

COUNT file-name
record-1 amount
record-2 amount
... ...
record-n amount

Only those input files for which test records are to be constructed must be listed.

The SEQUENCE option is specified in the following manner:

SEQUENCE (Name) (Orig. value) (Increment).

Examples:

CONTROL SECTION.
COUNT FILE-1
RECD-A 5000
RECD-B 1500
COUNT FILE-2
RECD-C 7000
SEQUENCE FLD-2C 100000 100
OF RECD-B
SEQUENCE ELIN-PL ABODE

Overlay Section. This section contains those elements of the Data Division which are to be modified. Any of three options may be employed depending upon whether an element is to be deleted, replaced, or inserted. (The "line-number" referred to in the following explanations is that found in the COBOL Data Division.)

1. Deletion:
line-no-1 DELET (thru line-no-2).

Example: 100024 DELET THRU 100027.
100105 DELET.

2. Insertion:
line-no-1 INSERT (n) lines.
The lines to be inserted must directly follow the INSERT statement.

Example: 100046 INSERT 4 LINES.
02 FIELD-A size is 6. Value is "ABC123".
100083 02 FIELD-B size is 9. Class is numeric synchronized.

3. Replacement.
line-no-1 (contents of new line).

Example: 100049 03 IN-CODE size is 30.
100063 -WE IS "X-00".

The Control, Overlay, and Relation Sections provide sufficient information to generate test items fulfilling both functions of the Test Data Generator. The combined system is titled the "Requirements Division." A complete example of this division is given in figure 2.

Construction of Edited Data

At present work is proceeding on a network analysis routine which will serve the following purpose. After the parameters describing record formats have been constructed, these parameters
**NAME-TABLE-J**

<table>
<thead>
<tr>
<th>Path</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
<th>A5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Condition-Group</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>IN-1</td>
<td>FLD-2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>START-CODE</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(REF #)</td>
<td>(OPERANDS)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**OPERATION-TABLE-J**

<table>
<thead>
<tr>
<th>Path</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
<th>A5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Condition-Group</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>EQ</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>NLS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(REF #)</td>
<td>(OPERATIONS)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Result-Group**

<table>
<thead>
<tr>
<th>Path</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
<th>A5</th>
</tr>
</thead>
<tbody>
<tr>
<td>174</td>
<td>RECORD-C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>175</td>
<td>RECORD-A</td>
<td>RECORD-B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>184</td>
<td>STOCK OF RECORD-A</td>
<td>STOCK OF RECORD-2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>190</td>
<td>BILD-6</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>195</td>
<td>TBL-1 (BILD-6)</td>
<td>TBL-2 (BILD-6)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>201</td>
<td>FLD-18</td>
<td>&quot;A&quot;, &quot;B&quot;, &quot;X&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(REF #)</td>
<td>(OPERANDS)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| (REF #) | (OPERATIONS) | | | | |

**GoTo**

<table>
<thead>
<tr>
<th>Path</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
<th>A5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Else</td>
<td></td>
<td>A2</td>
<td>A5</td>
<td>A5</td>
<td>A3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Path</th>
<th>A1</th>
<th>A2</th>
<th>A4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>A2</td>
<td></td>
</tr>
</tbody>
</table>

Figure 1. Relation Section Input Format
Figure 2. Sample Problem Employing the Requirements Division
will be modified before each test record is generated so that relationships specified may appear in the record. In this manner, different branches in the networks of relations given can be traced until the maximum number of records requested has been generated. 

This scheme introduces a significant problem. Two branches must emanate from every branch point in a logic network; therefore, for \( n \) branch points, there exist \( 2^n \) paths. In terms of the problem at hand, if only fifteen conditional statements (stemming from a common origin) were expressed, there would be over 30,000 possible methods of combining the data relations stated. With a limited number of test items to be generated, then, it is likely that several important combinations would be omitted.

To overcome this problem, it is possible to designate priority branches in the network. All path names in the Relation Section which are prefixed by the letter 'P' will be considered priority paths and will be investigated first.

Generation of a Test COBOL Source Program

A temporary method of generating edited test items has been devised to operate until an acceptable network analysis routine is completed. Via this method, the information given in the Data Requirements Division and portions of the Data Division of the program to be tested are edited into a COBOL Source Program. This program when compiled accepts as input the unedited data files generated as a result of the interpretation of the Data Division. Each of the records defined in a file is assigned an input area in Working Storage of the edited source program. When a \( 
\) operation is encountered, an edited item, if present, is released from this area and a new 'garbage' item is read in. The source coding generated to test relational operations merely involves a test comparing the operands involved. If the test fails, the first operand is adjusted to satisfy the relation.

To test error logic of the program, the test COBOL program is designed to generate improper relations at random. An error indicator is inserted in these items if space is available. This method of editing data relationships is admittedly less efficient than an analysis type system. However, the extensive use of this approach will help in indicating necessary revisions to the analysis routine being developed.

Conclusion

The use of the Test Data Generator requires work on the part of the programmer or systems analyst. This has a definite advantage, however, in that it forces him to review the logic of the data structure of a program. If this can be accomplished without reference to the program, the chances that the data generated will locate logical discrepancies in the program are increased. For this same reason, the concept of determining necessary data relationships by analyzing the Procedure Division of the program has been discarded. Although this method would require little work for the programmer or analyst, it would likewise eliminate all possibilities of locating missing branches in the program. Some thought has been given, however, to an analyzer of this type which would cross-check the statements of the Requirements Division and thereby possibly indicate logical discrepancies before any test data is generated. This method would also help to locate logical errors in the Requirements Division, should they exist.

Any implications that the Data Generator described is the final solution to program debugging are hereby denied! The system in its present state is merely an intermediate stage of development. Extensive applications of the generator will indicate the advisability of whether to continue with this line of thought or to revise the approach to any expedient degree. It is felt that this development will help to further understanding of how best to express data in a manner which may be of value to COBOL and problem-oriented languages in general.