COMPARING SAS TO TRADITIONAL PROGRAMMING LANGUAGES
Claude Jackson, ORI, Inc.

INTRODUCTION

This tutorial will discuss some of the features of SAS which differentiate SAS from traditional programming languages such as, PL/I, COBOL, and FORTRAN. By studying these unique features of SAS and seeing how the same functions are coded in traditional programming languages we should gain a better understanding of when SAS should be used instead of a traditional programming language. It should also aid us in justifying the use of SAS to others. Experience dictates that sooner or later all of us will probably have to do this. Finally it should also aid us in using SAS more effectively.

The importance to the SAS programmer of understanding what is unique about SAS should not be underestimated. There are many SAS statements that are similar if not identical to statements in PL/I, COBOL, and FORTRAN, e.g., GO TO, IF, and DO statements. It is this similarity between SAS and traditional languages that causes many programmers to underestimate SAS and not fully utilize its capabilities. I have often seen a programmer or have myself used statements in SAS similar to statements in traditional languages instead of statements with the features about to be discussed. When this is done many of the advantages of SAS are lost. Obviously programmers with experience in traditional languages are prone to this tendency but very experienced SAS Programmers overuse statements drawn from traditional languages in their efforts to use all of SAS. The point here is that often a unique SAS feature will be much better than SAS statements drawn from traditional languages. This fact should not be overlooked.

UNIQUE SAS FEATURES

- A single numeric data type
- "Smart" defaults
- Data independence
- Nonprocedural capabilities
- Powerful "programming environment"

This list of unique features is certainly not exhaustive and the features are not equal in importance. They do all share the attribute that they enable programmers to write and debug programs in SAS with much less effort than in traditional programming languages. The first two features are of lesser importance but they are two interesting examples of the subtle differences between SAS and traditional languages which I am continually discovering.

The discussions of these features which follow will try to show how a programmer with experience in traditional languages might consider these features to be disadvantages of SAS. Careful review of features suggest they are, in fact, advantages.

<table>
<thead>
<tr>
<th>PL/I</th>
<th>COBOL</th>
<th>FORTRAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIXED BINARY</td>
<td>PIC 9</td>
<td>INTEGER</td>
</tr>
<tr>
<td>FIXED DECIMAL</td>
<td>PIC 9 COMP-3</td>
<td></td>
</tr>
<tr>
<td>FLOAT BINARY</td>
<td>PIC 9 COMP-1</td>
<td>REAL</td>
</tr>
<tr>
<td>FLOAT DECIMAL</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A SINGLE NUMERIC DATA TYPE

Figure 1 lists the two or more numeric data types available in each of the major traditional languages. SAS however has only one numeric data type, double precision floating point. This would certainly seem to be a limitation of SAS. For example, it is common knowledge among good traditional language programmers that for efficient execution an integer variable should be used for subscripts of arrays. Reduction in run time efficiency appears to be the main disadvantage of having only a single numeric data. The advantage however are significant. Design, coding and testing is simplified with only a single numeric data
type and SAS takes care of all necessary conversions. In particular if the data values stored in a variable are only integers, SAS handles it appropriately and even prints it in integer format automatically. Why add features, i.e., data types, which don't aid functionality.

"SMART" DEFAULT DECLARATIONS

In PL/1 and FORTRAN default declarations will be made for a variable which is used but not explicitly declared. Figure 2 lists the rules for these defaults. These rules use the first letter of the variable to determine the type and length of the variable. This is a "dumb" rule because the first letter of a well named variable will have no relationship to the required type and length of a variable unless the programmer uses the first letter of integer to declare the type of the variable. This is usually the case when these defaults are used. Also, PL/1's defaults include only two of its many data types. In using these defaults many bugs have occurred. Therefore, many PL/1 and FORTRAN programming shops have required that all variables be declared to reduce these problems even though this increases the amount of code to be produced. Also because of these problems, language designers have even required all variables to be declared, e.g., COBOL. This is especially true for new languages. I was therefore a little concerned about default declarations in SAS of new variables. SAS however has a "smart" rule for determining the type and length of a variable. It uses the code written by the programmer to determine the type and length. Note also that SAS only has two main data types from which to choose. Programming experience and common sense should lead one to conclude as I did that SAS's default declarations are a valuable feature that reliably reduces SAS program development time.

DATA INDEPENDENCE

Data independence is the ability of a language to represent and access data independent of the physical structure in which the data is actually stored. In SAS, data is stored and accessed by name only. Data independence is an important feature of all data base management systems and is definitely not a new idea. Programs written in traditional languages are highly dependent on the physical structure of the data. A large number of statements in programs written in traditional languages are devoted to defining the physical structure of the data and in accessing these structures. A 'minor' change made in the physical structure of data, such as, adding a variable or changing the length of the variable, will require changes in all programs accessing any of the variables in the data structure.

Data independence is achieved in SAS only for data stored in SAS data sets or data stored locally during execution of DATA and PROC steps. Data accessed in non-SAS data sets via the INPUT and PUT statements is not accessed in a data independent fashion. When data in a SAS data set is retrieved by a SET, MERGE, or UPDATE statement or by a SAS PROC no definition of the format or organization of the data is required. Similarly when data is output into a SAS data set no redundant definition of variables or of the organization of the data is required. The large number of declarations eliminated by SAS's data independence is certainly one of SAS's most important features.
NONPROCEDURAL CAPABILITIES

A language has nonprocedural capabilities if it contains single statements that describe the processing of multiple records at a time or single statements that can replace a large number of statements. These nonprocedural capabilities are only of value if the functions that they can be utilized to perform are required frequently in programs.

The SAS statements DROP, KEEP, DELETE, subsetting IF, SET, OUTPUT, MERGE, and the functions automatically performed by the SAS supervisor provide the nonprocedural capabilities of SAS. For example when an OUTPUT statement is used, no statements are required to get the data to the output record as is usually required in traditional languages. In addition the SAS options/variables IN, FIRST, LAST, END, NODES are nonprocedural facilities which reduce the amount of code that must be written. The use of these capabilities not only reduces the number of statements to be written but also reduces the number of variables and buffers that would be declared in traditional languages. For example in order to perform a traditional language the function performed by a FIRST variable a variable would have to be declared to save the last value of the key being tested.

It is not only the reduction in the amount of code that is written that is important here but also the reduction in programming errors/bugs because reliable nonprocedural capabilities are used.

POWERFUL "PROGRAMMING ENVIRONMENT"

An important goal in software engineering in recent years is the development of a "programming environment" with sophisticated tools to aid programmers in their work. In other words use the computer and develop programs to perform tasks programmers now do themselves. Alternatively, existing tools (e.g. compilers) are being improved and integrated in order to optimize a programmer's use of these tools. The data dictionary capability provided by PROC CONTENTS is an example of a tool which aids the programmer in docu-
RECOMMENDATIONS

In order to take advantage of the features discussed and thus make optimal use of SAS the following recommendations are given:

- Use SAS data sets as much as possible.
- Consider designs which utilize nonprocedural facilities even if the designs require more DATA steps than a design which does not use as many nonprocedural capabilities.
- Be sure and utilize all the facilities of the SAS programming environment. PROC PRINT's and FREQ's should be used daily.
- Because many SAS features are innovatively designed e.g., "smart" default declarations, it is important to carefully evaluate them to determine their appropriate use. They may be more useful than similar features in traditional languages.