

Detailed Notes on Working with Numeric Variables

Numeric variables are relatively straight-forward to work with in SAS. The main difficulty that can occur results from attempts to save space in SAS data sets. First, note that since all computers ultimately use BASE 2 to perform all calculations, there are not exact BASE 2 expressions for numbers like 1/3, or 1/6 or other decimal numbers. Furthermore, there are limits in precision for even integer numbers. These limits can cause problems when operating on numeric variables. The program LEC7P27.SAS illustrates some of these limits.

Partial Listing of LEC7P27.SAS

```
OPTIONS LS=128 PS=60 NODATE NONUMBER NOCENTER;
*p*****;
*p*lec7p27.SAS 10/16/95 c691f #1 Ed Stanek *;
*p* DESCRIPTION: Issues in storing Numeric variables;
*p* LENGTH statement *;
*p*****;
LIBNAME old4 V604 'C:\SPH\OLD'; * use SAS default engine for data files;
LIBNAME old10 V610 'C:\SPH\OLD'; * use SAS default engine for data files;
LIBNAME new10 V610 'C:\SPH\NEW'; * new SAS data sets;

DATA new1;
  LENGTH c $8.;
  a=1/3;
  b=a*3;
  if b=1 THEN c="MATCH";
  ELSE c="No Match";
  d=9007199254740992;
  OUTPUT;
  a=1/3;
  b=(a*1000-333)*3;
  if b=1 THEN c="MATCH";
  ELSE c="No Match";
  d=9007199254740993;
  OUTPUT;
PROC PRINT;
  var a b c d;
  FORMAT a 20.19 b 20.19 d 21.;
RUN;
```

Partial OUTPUT from LEC7P27.SAS

Names and Follow-up Status for Subjects

OBS	A	B	C	D
1	.33333333333333330000	1.000000000000000000	MATCH	9007199254740992
2	.33333333333333330000	.999999999999999940000	No Match	9007199254740992

Note that while the first statements produce a “MATCH” for variables A and B, the value of A is not equal to the value of 1/3 specified in the program. Furthermore, note that the value of D in the program differs, while the value of D in the OUTPUT is the same. Both examples are due to limits of precision on the computer for numeric value representations.

SAS uses 8 bytes to represent a numeric value (such as the value of D in LEC7P27.SAS). With 8 bytes, SAS can represent an integer as large as $2^{53}=9,007,199,254,740,992$ exactly. Many integer values can be represented exactly with fewer bytes. Specifying a length less than 8 bytes means that variables will occupy less space in SAS data sets. The Table below (from SAS LANGUAGE GUIDE, Version 6.03, p198) indicates the maximum integer that can be expressed exactly by the number of Bytes.

Table 7.8. Number of Bytes and Maximum Integer that Can Be Represented Exactly

# Bytes	Power of 2	Maximum Integer
3	13	8,192
4	21	2,087,152
5	29	536,870,912
6	37	137,438,953,472
7	45	35,184,372,088,832

In practice, there are many settings where variables take on discrete values (such as the values 1-5). SAS will automatically assign a LENGTH of 8 Bytes to numeric variables. This length should not be changed if the variables contain decimal points. However, if the variables correspond to integers (less than the Maximum values given above) the variables can be stored using fewer Bytes without loss of precision. Such storage is accomplished using a LENGTH statement in the same manner as a LENGTH statement is used with Character variables.

1 Storage of Numeric Values using DOS on PC's

Problems in comparisons of numbers stored on the computer are not unique to the SAS system. These problems relate to the basic functioning of computers. A good introduction to storage of numeric values is given in the SAS Language Reference Manual (Version 6 1st edition, pp86-92). We summarize the material presented in the manual in this section, and place it in the context of using a length statement for numeric variables with SAS using DOS on a personal computer.

1.1. Storing Numbers with Floating Point Notation

Floating point notation is used to store numbers on a computer. Floating point notation is like scientific notation. A number is represented as a value (called the mantissa) multiplied by some other number (the base) raised to a power (called the exponent). The mantissa is always set to range from 0 to 1. For example, the numbers 98, 100, and 102 can be expressed as

	Mantissa	Base	Exponent
$98 = 0.98 \cdot 10^2$	0.98	10	2
$100 = 1.00 \cdot 10^2$	1.00	10	2
$102 = 0.102 \cdot 10^3$	0.102	10	3

All integers can be expressed in this form. Rational numbers may contain repeating patterns, so that an exact representation using a finite set of numbers may not be possible. For example, the numbers $1/7$ and $1/3$ are represented as follows:

$$1/7 = 0.142857142857\underline{142857}\dots 10^0$$

$$1/3 = 0.333333\underline{3}\dots 10^0$$

where the repeating portion of the mantissa is underlined.

Note that while numbers can be expressed in scientific notation, there is a certain degree of arbitrariness as to how this notation is defined. Consider for example the number 100 (in base 10). This number could be represented in the following manner:

	Mantissa	Base	Exponent
$100 = 10. 10^1$	10.	10	1
$100 = 1.00 10^2$	1.00	10	2
$100 = 0.1 10^3$	0.10	10	3
$100 = 0.01 10^4$	0.01	10	4

Some convention needs to be established to define how the mantissa and exponent are defined. We require the mantissa to be strictly less than one, and take exponent to be the exponent that places the first significant digit in the first place to the right of the decimal point. We make an exception to this rule when the number can be represented by a perfect power of the exponent. In such a case, we use 0.0000 to represent the mantissa. Using this convention, the mantissa and exponent for several numbers are given as follows:

	Mantissa	Base	Exponent
$0.085 = 0.85 10^{-1}$	0.85	10	-1
$98 = 0.98 10^2$	0.98	10	2
$100 = 1.00 10^2$	0.00	10	2
$102 = 0.102 10^3$	0.102	10	3

Note that with this convention, we can not think of the number as being simply the product of the mantissa times the base raised to the exponent. This convention avoids the inefficiency of having many ways of representing the number 0. This notation can be used to uniquely all rational numbers, with the exception of the number 0.

1.2 Storing Characters and Numbers on the Computer

Computers store numbers using on/off switches. Since there are only two possible values for the on/off switch (corresponding to 1/0), numbers are stored in the computer using base 2. A byte (corresponding to 8 bits) is the basic unit of storage on the computer. The 8 bits correspond to 8 on/off switches. The 8 bits in a byte can form $2^8=256$ possible ON/OFF patterns. These patterns are the ASCII codes. Particular patterns correspond to letters, numbers and other special characters. For this reason, a “character” is stored on the computer using 1 byte.

Generally, a number is stored on a computer using 8 bytes. Use of 8 bytes for storing a number is standard for many computer types (including IBM mainframes, Data General, DEC, etc., PC’s and MACs). Use of 8 bytes is often called “double precision”. It is referred to as “double” in the C language, and “REAL*8” in Fortran, and “FLOAT BINARY(53)” in the language PL/I. The 8 bytes contain 64 bits, or 2^{64} patterns of ON/OFF switches. When storing the number, floating point notation is used.

Although many common computer hardware systems use 8 bytes to store a floating point number, the manner in which they use the 8 bytes differs. Basically, the 64 bits contained in the 8 bytes are divided up into bits that store the sign (stored in 1 bit with “plus”=0 and “minus”=1), the exponent, and the mantissa. The base used for storage may differ by computer hardware, and by operating systems. We describe the way floating point numbers are stored on MACs and PCs using the DOS, OS/2 and UNIX operating systems. This storage system is called the Institute of Electrical and Electronic Engineers (IEEE) standard. This basic storage system we describe is used by MAC’s, SUN-3-UNIX, and HP UNIX systems. With DOS and OS/2, the storage system is identical, but the order of the bytes is reversed.

1.3. Standard IEEE Floating Point Storage

Floating point numbers are stored on a PC using DOS, OS/2, Unix, and MAC’s using base 2. Positions in base 2 represent powers of 2, where the power corresponds to the position relative to the position immediately left of the radix (a general term for the “.”). Thus, the base 2 number 111.11 can be interpreted as follows:

Base 2 Representation	1	1	1	.	1	1
Power of 2	2	1	0		-1	-2
Base 10 number	4	2	1		½	1/4

and is equivalent to 7.75 in Base 10. As a second example, the base two number 1000.011 corresponds to 8.375 in Base 10. Similarly, the base 2 number 0.0011 is equivalent to the base 10 number 0.1875, which is represented as

Base 2 Representation	0	.	0	0	1	1
Power of 2	0		-1	-2	-3	-4
Base 10 number	1		½	1/4	1/8	1/16

Understanding how numbers are stored on the computer requires converting numbers to a mantissa and exponent in base 2. The same rules for converting numbers in base 2 are used as were outlined for storing numbers in base 10 in Section 1.1. The mantissa is strictly less than 1, and when a value equals exactly a power of 2, the mantissa is set to 0.0000. Examples of some numbers in base 10, their representation in base 2, and the mantissa and exponent (expressed in base 10) are given below.

Base 10	Base 2	Mantissa (base 2)	Exponent (base 10)
3/16	0.0011	0.11	-2
1/4	0.01	0.0000	-2
3/8	0.011	0.11	-1
7/16	0.0111	0.111	-1
1/2	0.1	0.0000	-1
3/4	0.11	0.11	0
1	1.00	0.0000	0
1 1/2	1.1	0.11	1
2	10	0.0000	1
2 1/2	10.1	0.101	2
3	11	0.11	2
4	100	0.0000	2
5	101	0.101	3
6	110	0.110	3
7	111	0.111	3
8	1000	0.0000	3

Numbers that are to be stored may be positive or negative. The sign of the number is stored as the value of the first bit. When representing numbers using floating point notation, the sign, mantissa, and exponent need to be indicated. Exponents can also be positive or negative. In order to avoid having to store the sign of the exponent, a constant is added to the exponent to make the value always positive. This constant is called the bias. When the constant is added to the exponent, the resulting quantity is called the characteristic.

The 64 bits used to represent a floating point number for IEEE standard notation has 1 bit for the sign, 11 bits for the exponent (with a bias of 1023) and 52 bits for the mantissa. The bits are arranged as follows using IEEE standard representation for SUN-3-Unix and HP Unix operating systems:

General Representation of Floating Point Number for IEEE Standard (Unix/MAC)

byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7	byte 8
SEEE	EEEE	MMMM	MMMM	MMMM	MMMM	MMMM	MMMM

where “S” indicates the bit used to store the sign (+ or -), “E” represents the bits used to store the exponent (with 11 bits reserved for the exponent, where the exponent is expressed in as a base 2 number), and “M” represents the bits used to store the mantissa (also represented as a base 2 number). Using PC-DOS or OS/2 operating systems, the only difference in representation is the order of the bytes, with the bytes displayed in reverse order.

Note that 11 bits are used to store the exponent. The largest base 10 number that can be represented with 11 bits is the base 2 number 11111111111 = 2047. If the exponent was always positive, this implies that the largest power than could be represented is 2^{2047} . However, exponents can be positive (for large numbers) or negative (for small numbers). In order to represent both positive and negative exponents as positive numbers (called the characteristic) a bias of 1023 is added to the power of 2. Thus, numbers in base 2 are represented as powers as follows:

Number (Base 10)	Exponent (of Base 2)	Characteristic (exponent + 1023)	Characteristic in Base 2
$\frac{1}{2}^{-1023}$	-1023	0	000 0000 0000
1/64	-6	1017	011 1111 0111
1/32	-5	1018	011 1111 1000
1/16	-4	1019	011 1111 1001
1/8	-3	1020	011 1111 1010
1/4	-2	1021	011 1111 1011
$\frac{1}{2}$	-1	1022	011 1111 1110
1	0	1023	011 1111 1111
2	1	1024	100 0000 0000
4	2	1025	100 0000 0001
8	3	1026	100 0000 0010
16	4	1027	100 0000 0011
64	6	1029	100 0000 0101
2^{1023}	1023	2046	111 1111 1110
2^{1024}	1024	2047	111 1111 1111

1.4. Examples of IEEE Floating Point Storage

We illustrate several example of storage of floating point numbers using the more intuitive order used by MAC, SUN-3-Unix or HP Unix systems. We begin by considering how the number "1" is represented using standard IEEE floating point notation. We need to represent three quantities to see how the number is stored: the sign, the characteristic, and the mantissa. First, the sign of the number is positive, so the first bit is set at "0". Next, using the convention for representing the mantissa in Section 7.8.1.3, the mantissa in base 2 is given as 0.00000, while the exponent is given as "0". Adding the bias of 1023, the characteristic (in base 2) is given as 011 1111 1111. Thus, using the first bit for the sign, and the next 11 bits for the characteristic in base 2, the first 12 bits are given as follows:

0011 1111 1111 (base 2)
 3F F (base 16)

Rather than display the detailed pattern of on/off switches in base 2, base 16 is often used to represent the set of 8 bytes. Each byte consists of two base 16 numbers. The corresponding representation of numbers in base 2, base 10 and base 16 numbers are show on the next page.

Base 2	Base 10	Base 16
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F
10000	16	10

Since the values of the mantissa for number “1” (base 10) are all zero, the number is stored in floating point notation as follows:

```

Representation of Floating Point Number 1 (base 10) using IEEE Standard (Unix/MAC)
byte 1   byte 2   byte 3   byte 4   byte 5   byte 6   byte 7   byte 8
0011 1111 1111 0000 0000 0000 0000 0000 0000 0000 0000 0000 (base 2)
 3F      F0      00      00      00      00      00      00 (base 16)

```

As a second example, we consider how the number 3 (in base 10) is stored. From section 7.8.1.3, the mantissa (in base 2) is given by 0.11, while the characteristic is given as 1025 (base 10) or equivalently as 100 0000 0001 in base 2. Using these values, we can represent the number 3 (base 10) as a floating point number using 8 bytes as follows:

```

Representation of Floating Point Number 3 (base 10) using IEEE Standard (Unix/MAC)
byte 1   byte 2   byte 3   byte 4   byte 5   byte 6   byte 7   byte 8
0100 0000 0001 1100 0000 0000 0000 0000 0000 0000 0000 0000 (base 2)
 40      1C      00      00      00      00      00      00 (base 16)

```

Using PC DOS or OS/2, the bytes are represented in reverse order, where

Representation of Floating Point Number 3 (base 10) using IEEE Standard (DOS/OS/2)

byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7	byte 8	
0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	0001 1100	0100 0000	(base 2)
00	00	00	00	00	00	1C	40	(base 16)

We consider as a third example the number -8.375 in base 10, which is represented in base 2 as 1000.011. Next, we represent this number as a number between 0 and 1, such that in base 2,

$$1000.011 = 0.100011 \cdot 2^4.$$

Now, adding the bias (of 1023) to the exponent (of 4) gives the characteristic of 1027. In base 2, we express 1027 as 100 0000 0011. Finally, using these values, the floating point representation of -8.375 is

Representation of Floating Point Number -8.375 (base 10) using IEEE Standard (Unix)

byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7	byte 8	
1100 0000	0011 1000	1100 0000	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	(base 2)
C0	38	C0	00	00	00	00	00	(base 16)

Values in base 10 have a direct correspondence to sums of powers of 2 for base 2. The table below gives the power of two, and the base 10 corresponding number.

Correspondence of Powers of 2 and Base 10 Numbers and Cumulative Numbers
 Power of 2 Base 10 Cumulative

0	$2^0 =$	1	1
1	$2^1 =$	2	3
2	$2^2 =$	4	7
3	$2^3 =$	8	15
4	$2^4 =$	16	31
5	$2^5 =$	32	63
6	$2^6 =$	64	127
7	$2^7 =$	128	255
8	$2^8 =$	256	511
9	$2^9 =$	512	1,023
10	$2^{10} =$	1,024	2,047
11	$2^{11} =$	2,048	4,095
12	$2^{12} =$	4,096	8,191
13	$2^{13} =$	8,192	16,383
14	$2^{14} =$	16,384	32,767
15	$2^{15} =$	32,768	65,535
16	$2^{16} =$	65,536	131,071
17	$2^{17} =$	131,072	262,143
18	$2^{18} =$	262,144	524,287

Using this table, we can determine the floating point notation for the following integers:

Base 10 Number	Base 2 Representation	Exponent for base 2 number (base 10)	Characteristic (base 2 (bias=1023))
8189	1111 1111 1101	12	100 0000 1011
8190	1111 1111 1110	12	100 0000 1011
8191	1111 1111 1111	12	100 0000 1011
8192	1000 0000 0000 0	12	100 0000 1011
8193	1000 0000 0000 1	13	100 0000 1100
8194	1000 0000 0001 0	13	100 0000 1100
8195	1000 0000 0001 1	13	100 0000 1100

These numbers are represented in floating point notation as follows:

Representation of Floating Point Numbers 8189-8196 (base 10) using IEEE Standard (Unix)

byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7	byte 8	BASE 10
0100 0000	1011 1111	1111 1101	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	8189
0100 0000	1011 1111	1111 1110	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	8190
0100 0000	1011 1111	1111 1111	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	8191
0100 0000	1011 1000	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	8192
0100 0000	1100 1000	0000 0000	1000 0000	0000 0000	0000 0000	0000 0000	0000 0000	8193
0100 0000	1100 1000	0000 0001	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	8194
0100 0000	1100 1000	0000 0001	1000 0000	0000 0000	0000 0000	0000 0000	0000 0000	8195
0100 0000	1100 1000	0000 0010	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	8196

Finally, consider the representation of the number 227,303 (base 10) as a floating point number. The first step in converting the number to floating point is to express it in base 2. Since $2^{17}=131,072$, while $2^{18}=262,144$, the highest digit in base 2 is for 2^{17} . Working backwards,

	Base 2 #	Remainder from 227,303
$2^{17} = 131,072$	1	96,231
$2^{16} = 65,536$	1	30,695
$2^{15} = 32,768$	0	30,695
$2^{14} = 16,384$	1	14,311
$2^{13} = 8,192$	1	6,119
$2^{12} = 4,096$	1	2,023
$2^{11} = 2,048$	0	2,023
$2^{10} = 1,024$	1	999
$2^9 = 512$	1	487
$2^8 = 256$	1	231
$2^7 = 128$	1	103
$2^6 = 64$	1	39
$2^5 = 32$	1	7
$2^4 = 16$	0	7
$2^3 = 8$	0	7
$2^2 = 4$	1	3
$2^1 = 2$	1	1
$2^0 = 1$	1	0

so that in base 2, we represent 227,303 as

110111011111100111 .

Re-expressing this after moving the radix 18 places, 227,303 equals

$$0.1101\ 1101\ 1111\ 1001\ 11\ 2^{18}$$

where the base 2 exponent is 18. Adding the bias of 1023, the characteristic is given as 1041. This characteristic expressed in base 2 is given by 100 0001 0001. Finally, using these values, we can represent the number 227,303 using floating point notation as

Representation of Floating Point Number 227,303 (base 10) using IEEE Standard (Unix)															
byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7	byte 8								
0100	0001	0001	1101	1101	1111	1001	1100	0000	0000	0000	0000	0000	0000	0000	(base 2)
41	1D	DF	9C	00	00	00	00								

1.5. The effect of LENGTH on Floating Point Numbers

It is possible to use less bytes to store a floating point number with the length statement. The length statement reduces the number of bytes that are available for storage of the mantissa. Thus, with a length of 8, full double precision is retained. With a length of 7, only 7 bytes are available, resulting in 44 bits that can be used to represent the mantissa. With a length of 3, only 12 bits are available to represent the mantissa.

We can illustrate the problems that can occur when floating point numbers are represented with fewer bytes than are needed to express the number exactly in base 2. Consider, for example, the integers from 8189 to 8195. The floating point representation of these numbers is given in the previous section. Supposing we set the length of these variables to 3, implying that only the first three bytes will be used to store the number. The result is given below.

Representation of Floating Point Number 8189-8196 (base 10) using IEEE Standard (Unix) with Length= 3									
byte 1	byte 2	byte 3	Base 10 Number						
0100	0000	1011	1111	1111	1101	8189			
0100	0000	1011	1111	1111	1110	8190			
0100	0000	1011	1111	1111	1111	8191			
0100	0000	1011	1000	0000	0000	8192			
0100	0000	1100	1000	0000	0000	8193			
0100	0000	1100	1000	0000	0001	8194			
0100	0000	1100	1000	0000	0001	8195			
0100	0000	1100	1000	0000	0010	8196			

Note that when the length is specified too short, the values of the variables will be truncated. As can be seen via the floating point representation above, the values of the numbers 8192 and 8193 have the same representation, as do the values of 8194 and 8195. Notice that to evaluate the number produced by SAS when the length is specified too short, we need to express the number in base 10 using the exponent and mantissa that are expressed in base 2. For example, the number stored by SAS corresponding to 8193 (in base 10 with length 3) has a characteristic given by (1024 + 8+4) (base 10) resulting in an exponent of 13, resulting in a value of 2^{13} =8192.

We illustrated this with the program LEC7P34.SAS.

Partial Listing of LEC7P34.SAS

```
DATA d;
  INPUT id;
  id3=id;
  id4=id;
  id5=id;
  id8=id;
  length id3 3
         id4 4
         id5 5
         id8 8;
CARDS;
8190
8191
8192
8193
8194
8195
8196
;
PROC PRINT;
RUN;
```

Listing of Results from LEC7P34.SAS

OBS	ID	ID3	ID4	ID5	ID8
1	8190	8190	8190	8190	8190
2	8191	8191	8191	8191	8191
3	8192	8192	8192	8192	8192
4	8193	8192	8193	8193	8193
5	8194	8194	8194	8194	8194
6	8195	8194	8195	8195	8195
7	8196	8196	8196	8196	8196

We consider one more example to illustrate how truncation due to improper length specification can alter the stored values of the data. Previously, we developed the 8 byte representation for the number 227,303 as follows:

```
Representation of Floating Point Number 227,303 (base 10) using IEEE Standard (Unix)
byte 1   byte 2   byte 3   byte 4   byte 5   byte 6   byte 7   byte 8
0100 0001 0001 1101 1101 1111 1001 1100 0000 0000 0000 0000 0000 0000 (base 2)
  41      1D      DF      9C      00      00      00      00
```

As we reduce the length of the variable, the number of bytes used for storage is reduced. When the length is set to 4, only four bytes are used, and the number is represented as

```
Representation of Floating Point Number 227,303 (base 10) using IEEE Standard (Unix)
with a LENGTH of 4

byte 1   byte 2   byte 3   byte 4
0100 0001 0001 1101 1101 1111 1001 1100
  41      1D      DF      9C
```

Note that with a length of 4, no information is lost about the mantissa in base 2. However, if we set the length of the variable to 3, some information is lost, and the number that should represent 227,303 is truncated as follows:

Representation of Floating Point Number 227,303 (base 10) using IEEE Standard (Unix)
with a LENGTH of 3

```

byte 1   byte 2   byte 3
0100 0001 0001 1101 1101 1111
   41         1D         DF

```

In base 2, this number is given by $0.110111011111 \times 2^{18}$, where the exponent 18 is given by the number $18+1023$ in base 2 = 10000010001. Thus, in base 2, the number with length 3 is given by 1101 1101 1111 0000 00, and is equal to

$131,072 + 65,536 + 16,384 + 8,192 + 4,096 + 1,024 + 512 + 256 + 128 + 64 = 227,264$.

The program LEC7P35.SAS attempts to illustrate this truncation in SAS. Unfortunately, the result returned is larger than 227,264 by 32. I am not sure how SAS added the additional 32 units to the value.

Partial Listing of LEC7P35.SAS

```

DATA d;
  INPUT id;
  id3=id;
  id4=id;
  id5=id;
  id8=id;
  length id3 3
         id4 4
         id5 5
         id8 8;
CARDS;
227303
227302
227301
227300
227299
227298
227297
227296
227295
227294
;
PROC PRINT;
RUN;

```

Listing of Results from LEC7P35.SAS

OBS	ID	ID3	ID4	ID5	ID8
1	227303	227296	227303	227303	227303
2	227302	227296	227302	227302	227302
3	227301	227296	227301	227301	227301
4	227300	227296	227300	227300	227300
5	227299	227296	227299	227299	227299

6	227298	227296	227298	227298	227298
7	227297	227296	227297	227297	227297
8	227296	227296	227296	227296	227296
9	227295	227264	227295	227295	227295
10	227294	227264	227294	227294	227294

Note the pattern in truncation of the values.

2. Dangers in Setting Lengths for Numeric Variables

It is desirable to limit the length for integer variables so as to minimize data set size. For example, in the program LEC7P28.SAS, 1000 records with 100 variables in each record are created, where in one data set, all variables have length 8, while in the other data set, all variables have length 3. The data set with the smaller length is stored as a smaller data set on the C:\SPH\NEW directory.

Selective Listing of LEC7P28.SAS

```
OPTIONS LS=128 PS=60 NODATE NONUMBER NOCENTER;
*p*****;
*p*lec7p28.SAS 10/17/95 c691f #1 Ed Stanek *;
*p* DESCRIPTION: Comparison of Length statements on*;
*p* data set size for numeric variables *;
*p*****;
LIBNAME old4 V604 'C:\SPH\OLD'; * use SAS default engine for data files;
LIBNAME old10 V610 'C:\SPH\OLD'; * use SAS default engine for data files;
LIBNAME new10 V610 'C:\SPH\NEW'; * new SAS data sets;

*****;
*** Create data set with integers with length 8 for;
*** each variable
*****;
DATA new.d1;
  ARRAY V(100);
  DO i=1 to 1000;
    DO j=1=1 to 100;
      V{j}=j;
    END;
  OUTPUT;
END;

*****;
*** Create data set with integers with length 3 for;
*** each variable
*****;
DATA new.d2;
  ARRAY V{100};
  LENGTH V1-V100 3;
  DO i=1 to 1000;
    DO j=1=1 to 100;
      V{j}=j;
    END;
  OUTPUT;
END;
RUN;
```

The file stored as D1.SD2 contains 835,840 Bytes, while the file D2.SD2 contains 340,736 bytes. Thus the savings in storage space is roughly proportional to the percent reduction in the length of the variables.

Problems occur when variables are stored with a given length (to save storage space) that is inadequate for all of the values of the variables. An example is given in LEC7P29.SAS, where a variable, age, is reported in years for subjects, with a value of 99.9 corresponding to the missing value. The length has been set for age to 3, since the integer values of age are less than 3 digits long. However, since the missing value code is a decimal value, defining age of length 3 will mis-represent the decimal code.

Selective Listing of LEC7P29.SAS

```

*****;
*** Create data set with integers where length *;
*** should be longer *;
*****;
DATA d1;
  INPUT @1 id 2.
        @4 age
        @4 age8 ;
  LENGTH id age 3 ;
CARDS;
1 23
2 34
3 42
4 99.9
5 25
;
PROC CONTENTS ;
PROC PRINT NOOBS;
  VAR id age age8 ;

*****;
*** Convert missing age to SAS missing values ***;
*****;
DATA d2;
  SET d1;
  agev1=age;
  IF agev1=99.9 then agev1=.;
  age8v1=age8;
  IF age8v1=99.9 then age8v1=.;
PROC PRINT NOOBS DATA=d2;
  VAR id age agev1 age8 age8v1;
  TITLE1 "Listing of Age data with Missing values set to SAS Missing Values";
RUN;

```

Selective Listing of OUTPUT from LEC7P29.SAS

CONTENTS PROCEDURE					
Data Set Name: WORK.D1					Observations: 5
#	Variable	Type	Len	Pos	

2	AGE	Num	3	3	
3	AGE8	Num	8	6	
1	ID	Num	3	0	

Listing of Age data for D1

ID	AGE	AGE8
1	23.0000	23.0
2	34.0000	34.0
3	42.0000	42.0
4	99.8906	99.9
5	25.0000	25.0

Listing of Age data with Missing values set to SAS Missing Values for D2

ID	AGE	AGEV1	AGE8	AGE8V1
1	23.0000	23.0000	23.0	23
2	34.0000	34.0000	34.0	34
3	42.0000	42.0000	42.0	42
4	99.8906	99.8906	99.9	.
5	25.0000	25.0000	25.0	25