Making a Mountain into a Molehill in a Mainframe Environment  
Using Multi-Step Processing and other Masterly Maneuvers  

Tracee Matlock, Defense Manpower Data Center (DoD), Seaside, CA

ABSTRACT

Sharing a mainframe with tens or hundreds of other programmers can be a frustrating experience for yourself and/or others – especially if your program is calling a significant amount of resources, which could slow the processing of your co-workers projects. If all of the data you need to access for your project resides in disk packs, your job is relatively easy. On the other hand, if your organization has multiple forms of data storage media and limits on the number of tape drives you are allowed to access for a single project, then multi-step processing with temporary datasets is the solution for you. Furthermore, processing time and CPU time is very valuable in today’s information hungry world and with large projects, it is important to do all you can to get your program to run as quickly and efficiently as possible.

INTRODUCTION

This paper is designed for those who share a mainframe with other programmers – especially mainframes with tight controls on CPU time limits. If you are running SAS on a mainframe, then it is safe to say that you are working with very large files – which means efficiency is key.

All of us would love to have our program run through as efficiently as possible – the first time. That’s usually not the case though, and too often it is because of something very simple like the infamous “missing semi-colon” or a misspelled variable. Sometimes something even more frustrating occurs, like going over the CPU time limit for the job class into which you have submitted the program. And when you have to submit your job into an overnight class (meaning one chance per workday) the importance of having a successful run is crucial.

To address these issues, I have identified several methods to attain the most efficient program. To do that, I created a simulated project request. All of the files that I have utilized are flat files.

GOAL

Locate all Active Duty personnel who: 1) Had a Total Active Federal Military Service (TAFMS) of less than 24 months anytime during fiscal year 2000; 2) are currently on Active Duty; 3) are currently less than 25 years old; 4) scored more than 80% on their math portion of the Armed Services Vocational Aptitude Battery (ASVAB); and 5) the first 3 numbers of their SSN are between 014 and 110. For that population, supply their Name, Date of Birth (DOB), TAFMS, ASVAB score and State of Residence (ST).

METHODOLOGY

In order to complete this project, all twelve of the monthly Active Duty Master files during FY2000 were utilized. Each had a logical record length (LRECL) of 450 spread out over 98 variables, and averaging about 1.4 million records for a total record count of 16,864,065. Next, the population pulled from Active Duty was matched to the FY ASVAB file, which had an LRECL of 220, a variable count of 131, and a record count of 1,221,811. Finally, the monster of all monsters, the DEERS (Defense Eligibility Enrollment Reporting System) file with an LRECL of 1019, a variable count of 216 and a record count of over 18 million was used to pick up the ST and exclude everyone over 25 years old.

My organization uses an IBM CMOS processor model 9672-R24 running OS/390 version 2.8 and VM ESA 2.3. We have 1.7 Terabytes of disk storage (DASD) and ten 3490 tape drives. Because there are 80+ programmers competing for these ten drives, I am restricted by my agency to two tape drives per job. Therefore, this task could only be done with multiple steps. As each step was completed, the drives that were used in the previous step were released thereby allowing me to follow the rules of tape drive usage.

This paper has been designed as a pedagogical tool. As such, I have written two programs that will satisfy the goal. Program A, which will be referred to as A, has all of the tricks that will be found in this paper. It can be found in its entirety Appendix A. Program B, which will be referred to as B and does not have the tricks, can be found in Appendix B.

PROGRAM LOGIC

I began with a housekeeping step because I knew this program would be run more than once – even if there were no Job Control Language (JCL) errors (you will see why later). Step 1 is an HSM Delete step that deletes the final HSM data set when I run the program again.

```
//STEP1 EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*  
//SYSIN DD *  
//DELETE (DMC2TGEN31Y.TOM.WUSS.PAPR.P0106)  
/*
```

Step 2 was used to create the driver population. This is where I utilized the Active Duty Files. I read them in with the most current file first and then went back in time by month, so when I sort and drop duplicates on the SSN variable, I would keep the most current record for that person.
If I had not done it this way, I would have had to sort on two variables, the second being a file date variable, in order to keep the most current. That is inefficient because it would have made the sort take longer and use more resources. Additionally, it would have forced me to introduce an unnecessary variable (file date).

From these files I only wanted the record if the person had a TAFMS time of 24 months or less and I wanted to use the minimum amount of resource to get them. To do that, I used the trailing @ combined with an "IF/THEN/ELSE DELETE" method, also known as a SUBSETTING IF.

```
DATA POPULATION;
LENGTH SSN 5 DOB 5 TAFMS 2;
INFILE ONE;
INPUT 872 TAFMS 3.8; IF TAFMS LE 24 THEN
INPUT 81 SSN
   8222 TNAME SCHAR20.
   8242 MNAME SCHAR20.
   8116 DOB  8.1;
ELSE DELETE;
PROC SORT NODUPKEY; BY SSN;
```

By using this method, only the TAFMS variable from each record in the raw data file is moved in to the Program Data Vector (PDV) from the input buffer. Its value is tested and if it meets the condition, the rest of the record is moved in to the PDV - which reduces unnecessary time spent in the PDV. If the value of the variable TAFMS is not less than or equal to 24, then the observation is deleted, which prevents it from being output to the new SAS data set. This saves considerable CPU time than if you were to read in all of your variables and then use an IF statement at the end of the data step. This DATA Statement in A took 163.38 CPU seconds, while B took 263.06 CPU seconds to do the same way. That's a saving of 99.68 CPU seconds.

You will also notice I used a LENGTH Statement. This was done to save storage space. By default, when SAS writes a numeric variable to a dataset, it writes the number in IBM double wide floating-point format. In this format, 8 bytes are required for storing a number. Using the LENGTH statement reduced the amount of space required because SAS 8.1 has default LENGTH of 8. You can use the following table to determine the length you should specify - and always set the LENGTH according to the largest number you will process.

<table>
<thead>
<tr>
<th>LENGTH IN BYTES</th>
<th>DIGITS RETAINED</th>
<th>LARGEST INTEGER</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>256</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>65,536</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>16,777,216</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>4,294,967,296</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>1,099,511,627,776</td>
</tr>
<tr>
<td>7</td>
<td>14</td>
<td>281,474,967,10,856</td>
</tr>
<tr>
<td>8</td>
<td>16</td>
<td>72,057,594,037,927,936</td>
</tr>
</tbody>
</table>

Since I knew that the largest integer my SSN variable could possibly be was 999,999,999, and the largest integer my DOB could have been was 999,999,999, then I knew I could set my length at 5 bytes and greatly reduce the storage space required.

You can see that I have named my output data step DATA _NULL_. When the word _NULL_ appears instead of a SAS data set name, it tells SAS not to make a new SAS data set and thereby saves computer time. Furthermore, I wrote out my refined population to a temporary SAS data set that will disappear when the program is finished. This method also saves time. While B took 14.44 CPU seconds for this step to process, A, with the _NULL_ statement, took 12.94 CPU seconds. That may not seem like much, but I've had many programs cancelled by the operators in their final stages when they only needed a couple more seconds to finish.

Below my _NULL_ step is a PROC DATASETS step. This PROC has deleted the member (ONE) in the library and freed up my resources. This program would run through with the same results without the dataset deletion steps, but I chose to put them in to illustrate placement. They are actually much more valuable in a single step program with multiple datasteps and infiles.

You might be wondering if I have forgotten that I am supposed to be supplying an age to my client since I have left my DOB field alone. By waiting to pick up AGE in Step 4, I was able to keep one of my input files down to a manageable size, and avoid a time costly computation.
Next on the agenda is Step 3:

/*
//STEP3 EXEC SAS
//WORK DD SPACE=(CYL, (400,100), RLSE)
//POP DD DISP=SHR,
// DSN=&POP
//ASVAB DD DISP=SHR,
// DSN=DMDCP.GEN3FF.MEPDD.F0009
//SASOUT DD DISP=(NEW,PASS),
// INFILE POP;
//SYSIN DO
INPUT @1 SSN
//WORK DO SPACE•(CYL, (400,100),RLSE)
//POP DO DISP•SHR,
//ASVAB DO DISP=SHR,
//WORK
/*
DATA POPULATION;
INFILE POP;
INPUT &1 SSN 9.
&10 LNAME $CHAR26.
&36 FNAME $CHAR26.
&56 MNAME $CHAR26.
&76 DOB 8.;

DATA ASVAB;
LENGTH SSN 5 SCORE 2;
INFILE ASVAB;
INPUT &32 SCORE PIB1.@;
IF SCORE GT 80 THEN
INPUT &1 SSN PIB4.;
ELSE DELETE;

DATA MATCH;
MERGE POP (IN=A) ASVAB (IN=B);
BY SSN;
IF A AND B;
PROC DATASETS; DELETE POP ASVAB;
DATA _NULL_
SET MATCH;
FILE SASOUT;
PUT &1 SSN 9.
&10 LNAME $CHAR26.
&36 FNAME $CHAR26.
&56 MNAME $CHAR26.
&76 DOB 8.;
&84 SCORE 2.1;
PROC DATASETS; DELETE MATCH;

You already know the tricks I have used here. You can see the 
LENGTH Statements, the PROC DATASETS statement and the 
Temporary Data Set writeout. Something new in this step 
is the use of BINARY data. When working with binary data, 
more workspace is used because it has to be unpacked before 
anything is done with it. This entire step took 14.04 in A 
and 15.54 in B. Not much difference, but when you look at 
the workspace required, A used 6191K to read in the ASVAB file, 
while B took 8222K bytes of workspace. That’s quite a 
difference.

Now we are ready for Step 4, which required some very 
necessary programming maneuvers!

In Step 4 I needed to bring in the DEERS that I described 
earlier. Because of it’s huge size, it was very important to read 
in as little as possible.

DATA ONE;
INFILE DEERS;
LENGTH SSN 5 AGE 2;
INPUT 8780 AGE 3.
 8266 CAT $1.
 89 NNSN $9.0;
IF AGE LT 25 AND CAT = 'A' AND 
'014'<= SUBSTR (NSSN,1,3) <='110' THEN
INPUT 8798 STATE $2.
 89 SSN 9.1;
ELSE DELETE;

The big difference here is that I selected on three variables: 
AGE, CAT and a SUBSTRING of the SSN. You should notice that 
there is the need of only one occurrence of a trailing @. 
Also, I read in the SSN twice – once as Character and once as 
Numeric, with two different variable names. This was done for 
the purpose of the SUBSTR function, which is a character function. 
While it is counterproductive for my goal of saving 
memory, I chose to do it to avoid SAS Notes regarding changing 
Numeric to Character data. This choice illustrates that sometimes decisions must be made to do the least efficient thing 
to get the best output.

This step took 205.90 CPU seconds in A and 272.80 in B for a savings of 66.90 CPU seconds. Again, not a lot of savings 
there, but combined with everything else – very beneficial.

The last trick I used was a trouble shooting step utilizing OBS=0 
in the OPTIONS LINE. It is placed right below the JCL and looks like this:

OPTIONS OBS=0;

This was done so I could check all of my logic without running a multitude of records through in the process. I only wanted to 
check the syntax of my program. You could put any number in 
place of the 0 and combine that with some PROC PRINTS to do 
a check of what your observations are looking like.

SUMMARY

These coding methods allowed me to take over 36,085,876 
records and 445 variables down to 8 variables and 152 variables 
in just one program and a short amount of CPU time. To 
accomplish the same goal, A took a total of 8.28 CPU minutes, 
while B took 11.18 CPU minutes. Clearly Program A is more 
efficient.

These are just a few of the efficiency tricks that SAS provides to 
us, as we are fortunate in that we can do almost anything in SAS 
several different ways. As a Mainframe SAS user competing 
against 80+ other programmers at any given time, these methods 
have proven themselves to be extremely useful to me and I hope 
they can help you too!

118
REFERENCES

SAS is a registered trademark of SAS Institute, INC., Cary, NC, USA


SAS Online Documentation
http://wwwmdops/resources/sashtml/onldoc.htm

ABOUT THE AUTHOR

The author welcomes your comments & suggestions.

Tracee S. Matlock
Defense Manpower Data Center
Management Information and Analysis Division
(831)583-2400 x4345
DSN 878-2951
matlocks@osd.pentagon.mil

Ms. Matlock has been a Management Analyst for the Department of Defense, Defense Manpower Data Center for almost 4 years. The Defense Manpower Data Center maintains the largest archive of personnel, manpower, training and financial data in the Department of Defense. Her clients include the Office of the Secretary of Defense, the Office of the Joint Chiefs of Staff, Military Services, Congress, Unified Commands, Private Research Companies and Universities. She received her degree in Social and Behavioral Sciences from California State University in 1998.

APPENDIX A

/*
//MW#UWSS JOB (4324),TRACEE,CLASS=G
//STEP1 EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*;
//SYSIN DD *
DELETE (DMDCP.GEN31Y.TOM.UWSSPAPR.P0106)
*/

/*
//STEP2 EXEC SAS
//WORK DD SPACE=(CYL,(400,100),RLSE)
//ONE DD DISP=SHR,UNIT=AFF-ONE;
//DSN=DMDCP.GEN3FF.ACTDM.P0009
//DSN=DMDCP.GEN3FF.ACTDM.P0008
//DSN=DMDCP.GEN3FF.ACTDM.P0007
//DSN=DMDCP.GEN3FF.ACTDM.P0006
//DSN=DMDCP.GEN3FF.ACTDM.P0005
//DSN=DMDCP.GEN3FF.ACTDM.P0004
//DSN=DMDCP.GEN3FF.ACTDM.P0003
//DSN=DMDCP.GEN3FF.ACTDM.P0002
//DSN=DMDCP.GEN3FF.ACTDM.P0001
//DD DISP=SHR,UNIT=AFF-ONE
//DD DISP=SHR,UNIT=AFF-ONE;
//DSN=DMDCP.GEN3FF.ACTDM.P9912
//DD DISP=SHR,UNIT=AFF-ONE;
//DSN=DMDCP.GEN3FF.ACTDM.P9911
//DD DISP=SHR,UNIT=AFF-ONE;
//DSN=DMDCP.GEN3FF.ACTDM.P9910
//SASOUT DD DISP=(NEW,PASS);
//DCB=(LRECL=83,BLKSIZ=0,RECFM=FB);
//SPACE=(CYL,(10,5),RLSE);
//DSN=&&POP
//SYSIN DD *
OPTIONS OBS=0;

DATA POPULATION;
LENGTH SSN 5 DOB 5 TAFMS 2;
INFILE ONE;
INPUT 72 TAFMS 3.0; IF TAFMS LE 24 THEN
  INPUT @72 TAFMS 3.0; IF TAFMS LE 24 THEN
INPUT @72 TAFMS 3.0; IF TAFMS LE 24 THEN
  INPUT @72 TAFMS 3.0; IF TAFMS LE 24 THEN
INPUT @72 TAFMS 3.0; IF TAFMS LE 24 THEN
  INPUT @72 TAFMS 3.0; IF TAFMS LE 24 THEN
INPUT 81 SSN 9.8196 LNAME $CHAR26.8222 FNAME $CHAR20.8242 MNAME $CHAR20.8116 DOB 8.;
ELSE DELETE;

PROC SORT NODUPKEY; BY SSN;

DATA _NULL_; PUT @1 SSN 9.810 LNAME $CHAR26.836 FNAME $CHAR20.856 MNAME $CHAR20.876 DOB 8.;

PROC DATASETS; DELETE POPULATION;
*/

/*
//STEP3 EXEC SAS
//WORK DD SPACE=(CYL,(400,100),RLSE)
//POP DD DISP=SHR,
//DSN=&&POP
//ASVAB DD DISP=SHR,
//DSN=DMDCP.GEN3FF.MEPDD.P0009
//SASOUT DD DISP=(NEW,PASS);
//DCB=(LRECL=85,BLKSIZ=0,RECFM=FB);
//SPACE=(CYL,(50,10),RLSE);
//DSN=&&ASVAB
//SYSIN DD *

DATA POPULATION;
INFILE POP;
INPUT @1 SSN 9.810 LNAME $CHAR26.836 FNAME $CHAR20.856 MNAME $CHAR20.876 DOB 8.;

DATA ASVAB;
LENGTH SSN 5 SCORE 2;
INFILE ASVAB;
INPUT @36 SCORE PIB1.@; IF SCORE GE 80 THEN
INPUT $1 SSN PIB4.;
ELSE DELETE;
DATA MATCH;
MERGE POPULATION (IN=A) ASVAB (IN=B);
BY SSN;
IF A AND B;
PROC DATASETS;
DELETE POPULATION ASVAB;
DATA _NULL_;  
FILE SASOUT;  
PUT $1 SSN 9.  
$10 LNAME $CHAR26.  
$36 FName $CHAR20.  
$56 MNAME $CHAR20.  
$76 DOB 8.  
$84 SCORE 2.;  
PROC DATASETS;
DELETE MATCH;  
I* //STEP4 EXEC SAS  
//WORK DD SPACE=(CYL, (600,200),RLSE),  
// UNIT=(SYSDA,5)  
//POP DD DISP=SHR,  
// DSN=&&ASVABPOP  
//DEERS DD DISP=SHR,  
// DSN=DMDCP.GEN3FF.DRSDD.P0104  
//SASOUT1 DD DISP=(NEW,CATLIG,DELETE),  
// DCB=(LRECL=89,BLKSIZE=0,RECFM=FB),  
// SPACE=(CYL, (50,10),RLSE),  
// DSN=DMDCP.GEN31Y.WUSSPAPR.P0106  
//SYSIN DD *  
DATA ONE;  
INFILE DEERS;  
LENGTH SSN 5 AGE 2;  
INPUT 0780 AGE 3.  
0266 CAT $1.  
09 NSSN $9.8;  
IF AGE LT 25 AND CAT = 'A' AND  
'014'<= SUBSTR (NSSN,1,3) <='110' THEN  
INPUT 0798 STATE $2.  
09 SSN 9.;  
ELSE DELETE;  
PROC SORT DATA=ONE NODUPKEY; BY SSN;  
DATA POPULATION;  
INFILE POP;  
INPUT $1 SSN 9.  
$10 LNAME $CHAR26.  
$36 FName $CHAR20.  
$56 MNAME $CHAR20.  
$76 DOB 8.  
$84 SCORE 2.;  
DATA MATCH;  
MERGE ONE (IN=A) POPULATION (IN=B);  
BY SSN;
IF A AND B;
PROC DATASETS; DELETE ONE POPULATION;
PROC SORT DATA=MATCH NODUPKEY; BY SSN;
DATA _NULL_;  
SET MATCH;  
FILE SASOUT1;  
PUT $1 SSN 9.  
$10 LNAME $CHAR26.  
$36 FName $CHAR20.  
$56 MNAME $CHAR20.  
$76 DOB 8.  
$84 SCORE 2.  
$86 AGE 2.  
$88 STATE $2.;  
//  
APPENDIX B  
//MW$BAD JOB (4324),TRACEE,CLASS=G  
//STEP1 EXEC PGM=IDCAMS  
//SYSPRINT DD SYSOUT=*  
//SYSIN DD *  
DELETE  
(DMDCP.GEN31Y.TOM.WUSSPAPR.BAD.P0106)  
*/  
//STEP2 EXEC SAS  
//WORK DD SPACE=(CYL, (400,100),RLSE)  
//ONE DD DISP=SHR,UNIT=3490,  
// DSN=DMDCP.GEN3FF.ACTDM.P0009  
// DD DISP=SHR,UNIT=AFF-ONE,  
// DSN=DMDCP.GEN3FF.ACTDM.P0008  
// DD DISP=SHR,UNIT=AFF-ONE,  
// DSN=DMDCP.GEN3FF.ACTDM.P0007  
// DD DISP=SHR,UNIT=AFF-ONE,  
// DSN=DMDCP.GEN3FF.ACTDM.P0006  
// DD DISP=SHR,UNIT=AFF-ONE,  
// DSN=DMDCP.GEN3FF.ACTDM.P0005  
// DD DISP=SHR,UNIT=AFF-ONE,  
// DSN=DMDCP.GEN3FF.ACTDM.P0004  
// DD DISP=SHR,UNIT=AFF-ONE,  
// DSN=DMDCP.GEN3FF.ACTDM.P0003  
// DD DISP=SHR,UNIT=AFF-ONE,  
// DSN=DMDCP.GEN3FF.ACTDM.P0002  
// DD DISP=SHR,UNIT=AFF-ONE,  
// DSN=DMDCP.GEN3FF.ACTDM.P0001  
// DD DISP=SHR,UNIT=AFF-ONE,  
// DSN=DMDCP.GEN3FF.ACTDM.P9912  
// DD DISP=SHR,UNIT=AFF-ONE,  
// DSN=DMDCP.GEN3FF.ACTDM.P9911  
// DD DISP=SHR,UNIT=AFF-ONE,  
// DSN=DMDCP.GEN3FF.ACTDM.P9910  
//SASOUT DD DISP=(NEW,PASS),  
// DCB=(LRECL=89,BLKSIZE=0,RECFM=FB),  
// SPACE=(CYL, (10,5),RLSE),  
// DSN=&&POP  
//SYSIN DD *  
OPTIONS OBS=0;  
DATA POPULATION;  
INFILE ONE;
INPUT @1 SSN 9.
@196 LNAME $CHAR26.
@222 FNAME $CHAR20.
@242 MNAME $CHAR20.
@72 TAFMS 3.
@116 DOB 8.;

IF TAFMS LE 24;

PROC SORT NODUPKEY; BY SSN;

DATA FINAL;
SET POPULATION;
FILE SASOUT1;
PUT @1 SSN 9.
@10 LNAME $CHAR26.
@36 FNAME $CHAR20.
@56 MNAME $CHAR20.
@76 DOB 8.;

/*
//STEP3 EXEC SAS
//WORK DD SPACE=(CYL, (400,100), RLSE),
//POP DD DISP=SHR,
//ASN=&&POP
//ASVAB DD DISP=SHR,
//ASN=DMDCP.GEN3FF.MEPDD.F0009
//SASOUT DD DISP=(NEW,PASS),
//DCH=(LRECL=85,BLKSIZE=0,RECFM=FB),
//SPACE=(CYL,(50,10),RLSE),
//ASN=&&ASVABPOP
//@SYNIN DD *

DATA POPULATION;
INFILE POP;
INPUT @1 SSN 9.
@10 LNAME $CHAR26.
@36 FNAME $CHAR20.
@56 MNAME $CHAR20.
@76 DOB 8.;

DATA ASVAB;
INFILE ASVAB;
INPUT @1 SSN 9.
@32 PIB4.
@32 PIB1.;

IF SCORE GT 80;

DATA MATCH;
MERGE POPULATION (IN=A) ASVAB (IN=B);
BY SSN;
IF A AND B;

DATA FINAL;
SET MATCH;
FILE SASOUT1;
PUT @1 SSN 9.
@10 LNAME $CHAR26.
@36 FNAME $CHAR20.

@56 MNAME $CHAR20.
@76 DOB 8.
@84 SCORE 2.;

/*
//STEP4 EXEC SAS
//WORK DD SPACE=(CYL, (600,200), RLSE),
//UNIT=(SYSDA,5)
//POP DD DISP=SHR,
//DSN=&&ASVABPOP
//DEERS DD DISP=SHR,
//DSN=DMDCP.GEN3FF.DRSDD.P0104
//SASOUT1 DD DISP=(NEW,CATLG,DELETE),
//DCH=(LRECL=89,BLKSIZE=0,RECFM=FB),
//SPACE=(CYL,(50,10),RLSE),
//DSN=DMDCP.GEN31Y.WUSSPAPR.BAD.P0106
//@SYNIN DD *

DATA ONE;
INFILE DEERS;
INPUT @9 SSN 9.
@266 CAT $1.
@780 AGE 3.
@798 STATE $2.;

IF AGE LT 25 AND CAT = 'A' AND '014'<= SUBSTR (NSSN,1,3) <-'110';

PROC SORT DATA=ONE NODUPKEY; BY SSN;

DATA POPULATION;
INFILE POP;
INPUT @1 SSN 9.
@10 LNAME $CHAR26.
@36 FNAME $CHAR20.
@56 MNAME $CHAR20.
@76 DOB 8.
@84 SCORE 2.;

DATA MATCH;
MERGE ONE (IN=A) POPULATION (IN=B);
BY SSN;
IF A AND B;

PROC SORT DATA=MATCH NODUPKEY; BY SSN;

DATA FINAL;
SET MATCH;
FILE SASOUT1;
PUT @1 SSN 9.
@10 LNAME $CHAR26.
@36 FNAME $CHAR20.
@56 MNAME $CHAR20.
@76 DOB 8.
@84 SCORE 2.
@86 AGE 2.
@88 STATE $2.;

//