Searching for Variable Values with CAT Functions: An Alternative to Arrays and Loops

Mike Zdeb, University@Albany School of Public Health, Rensselaer, NY

ABSTRACT
When one wants to search through a large number of variables for one or more specific variable values, the common approach is to first use an array to store the variable values and then use a DO loop to cycle through the array elements looking for the value(s) of interest. With the advent of the various CAT (concatenation) functions in version 9, there is an alternative to the array/loop approach that reduces the amount of SAS code needed to search for values, with the added benefit of taking approximately as much elapsed time as the array/loop approach. This paper shows several scenarios where one of the CAT functions can be used in combination with version 9 FIND and COUNT functions to simplify the task of searching for variable values. The standard approach (an array plus a loop) is often shown first, followed by the CAT function solution.

INTRODUCTION
Among the many new features introduced in version 9 of SAS® were a host of new functions and call routines. Four concatenation functions (and call routine counterparts) are now available that take the place of using the concatenation operator (||) and long-standing functions such as TRIM and LEFT when creating new character strings from already existing variables. These function are: CAT, CATS, CATT, CATX.

Example 1 shows how to create a new variable using both the old and new methods. The CATX function allows you to specify a delimiter (in this case a comma followed by a single space) that is used to separate the values of all the variables listed after that delimiter. Use of the CATX function implies that both the TRIM and LEFT functions are used with each variable in the list, so in this example the function really took the place of ...

```
* example 1;
data names;
input last_name : $25. first_name : $15.  @@;
* old way;
full_name_old_way = trim(last_name) || ', ' || first_name;
* new way;
full_name_new_way = catx(', ',last_name,first_name);
datalines;
Zdeb Mike Washington George Squirrel Rocket(J)
;
run;
```

```
           DATA SET NAMES WITH CONCATENATED VARIABLES

last_name    first_name       full_name_old_way      full_name_new_way

Zdeb          Mike           Zdeb, Mike             Zdeb, Mike
Washington    George         Washington, George     Washington, George
Squirrel      Rocket(J)      Squirrel, Rocket(J)    Squirrel, Rocket(J)
```

```
full_name = trim(left(last_name)) || ', ' || trim(left(first_name));
```

Each of the four CAT functions uses a slightly different method of combining variable values and the differences can be seen in the table on the right (taken from SAS on-line documentation).

Function	Equivalent Code
CAT(OF X1-X4)	X1||X2||X3||X4
CATS(OF X1-X4)	TRIM(LEFT(X1))||TRIM(LEFT(X2))||TRIM(LEFT(X3))|| TRIM(LEFT(X4))
CATT(OF X1-X4)	TRIM(X1)||TRIM(X2)||TRIM(X3)||TRIM(X4)
CATX(SP, OF X1-X4)	TRIM(LEFT(X1))||SP||TRIM(LEFT(X2))||SP|| TRIM(LEFT(X3))||SP||TRIM(LEFT(X4))

Given the title of this paper, how can functions that are normally thought of as being used to create new character variables be used to search for variable values?

SEARCHING FOR A SINGLE CHARACTER VALUE

Assume that you have administered a questionnaire. You create a data set from your questionnaire information and the observations have an id number plus the answer to each of 10 questions coded as either Y or N. Now you want to create another data set of only those observations where at least one question was answered yes. Your task is to read each observation and check the value of 10 variables for the value 'Y'.

Example 2 shows a comparison of two approaches to finding the required observations. Each solution requires you to write code that instructs SAS to look at the value of the variables A1 through A10 and check to see if any value is a 'Y'. The array and a loop approach requires eight statements (I don't think that any can be eliminated). The array A contains all the variables A1 through A10. The loop cycles through the various values, stopping short of 10 iterations if a value of 'Y' is found. If no 'Y' is found, the loop iterates 11 times (remember, the loop stops when the value of J is greater than 10, or if a 'Y' is found). If J is less than 11, a 'Y' was found and an observation is added to the data set. The loop iterator J is not needed in the data set and it is dropped.

```
* example2;
data answers;
input id : $5. (a1-a10) (: $1.);
datalines;
A1234 Y Y Y Y Y Y Y Y N N
A2345 N N N N N N N N N N
A3456 N N N N N N N N N Y
A4567 N N N N Y N N N N N
A5678 Y N N Y N N Y N N Y
;
run;

* array and a loop;
data yes;
set answers;
array a(10);
do j=1 to 10 until (a(j) eq 'Y');
end;
if j lt 11;
drop j;
run;

* cat and find;
data yes;
set answers;
if find(cat(of a1-a10),'Y');
run;
```

That is a lot of SAS code to accomplish what the CAT and FIND approach can do in only four statements. Prior to introduction of the CAT functions, you would have to list all the values A1 through A10 and use the concatenation operator '||' to form the character string used within the FIND function, for example ...

```
if find (a1 || a2 || a3 || a4 || a5 || a6 || a7 || a8 || a9 || a10, 'Y');
```

The FIND function returns the location of 'Y' in the character string formed by concatenating the values of variables A1 through A10. If no values are found, the subsetting IF statement is FALSE (the FIND function returns a value of zero) and no observation is written to the data set. If a 'Y' is found, that statement is TRUE and a an observation is added to the data set. Notice that no new variables are created (no DROP statement needed).

SEARCHING FOR MULTIPLE CHARACTER VALUES

We can use the questionnaire data in example 2 and change the task. Now, you only want to find observations with at least two questions answered 'Y'. Rather than just search for a single 'Y', the number of occurrences must be counted. The array and loop approach in example 3 is still nine statements, but the loop is not halted until two occurrences of 'Y' are found. Within the loop, a SUM function is used to count occurrences and the SUBSETTING IF statement now specifies the value of FOUND_Y must be 2 for an observation to be added to the data set. Notice that there is no need to set FOUND_Y to zero as each observation is processed. It is set to missing during each iteration of the data step and the missing value is ignored by the SUM function within the loop.

The CAT and FIND approach is changed to a combination of CAT and COUNT functions. There are two functions that count the occurrences of user-specified text within a character string, COUNT and COUNTC. COUNT can be used to search for character string, while COUNTC searches for a single character. As in example 2, there are only four statements. No new variables are created (no DROP statement needed).

```
* example 3;
* array and a loop;
data yes;
set answers;
array a(10);
do j=1 to 10 until (found_y eq 2);
   found_y = sum(found_y,(a(j) eq 'Y'));
end;
if found_y eq 2;
drop j found_y;
run;

* cat and countc;
data yes;
set answers;
if countc(cat(of a1-a10),'Y') ge 2;
run;
```

Again, using the same data set, change the task to creating a data set that includes all the original observations with one new variable whose value is the count of the number of time 'Y' occurs in variables A1 through A10. In example 4, the array and loop solution is now seven statements. Given that all observations are output, there is no need for the SUBSETTING IF statement. Within the loop, the SUM function is used to add one to the variable FOUND_Y each time a 'Y' is found within the array variables. A 'trick' (courtesy of Paul Dorfman) is shown in the loop, using _N_ as the index variable. Since _N_ is the same name as the SAS-supplied variable (counts data step iterations), there is no need to use a DROP statement given that it is not kept. Using _N_ as a variable within the data step has no effect on its value as it counts data step iterations. The CAT function is again used with the COUNTC function rather than with FIND. The CAT function creates a string of all the questionnaire answers and the COUNTC function produces a count of responses with the value 'Y'. Only four statements are required to achieve the same results as the array and loop solution. Notice that a shortcut is used in the CAT function in example 4 with 'A:' replacing the 'A1-A10' used in examples 2 and 3 (even less SAS code).

```
* example 4;
* array and a loop;
data yes;
set answers;
array a(10);
do _n_=1 to 10;
   found_y = sum(found_y,(a(_n_) eq 'Y'));
end;
run;

* cat and countc;
data yes;
set answers;
found_y = countc(cat(of a:),'Y');
run;
```

SEARCHING FOR NUMERIC VALUES

Though the CAT functions are thought of as character functions, they can also be used with numeric variables. For example 5, again assume that you administered a questionnaire, but this time the responses were coded with values of 0 through 5 and you decided to create a data set with the responses stored in 10 numeric variables. Rather than searching the responses for the value 'Y', you want to create a new data set comprising observations with at least one question answered with the value 5. Both the array and loop and the CAT and FIND solutions look similar to example 2. A numeric array is searched for any array element with the value 5. In the CAT and FIND solution, though the variables A1 through A10 are numeric, you still must search for a character constant ('5', not a numeric 5) with the FIND function (remember, FIND is a character function). Given that CAT is a character function used in this example with numeric data, you might expect one of those messages in the SAS log about numeric-to-character conversion. However, no such messages are produced, with the SAS log looking no different from that produced when character variables are used.

```
* example5;
data answers;
input id : $5. a1-a10;
datalines;
A1234 5 5 3 3 2 1 1 1 1 1
A2345 0 0 0 0 0 0 0 5 5 5
A3456 1 1 2 2 4 0 0 0 0 0
A4567 5 5 5 5 5 5 5 5 5 5
A5678 1 0 1 0 1 0 5 5 5 3
;
run;

* array and a loop;
data five;
set answers;
array a(10);
do j=1 to 10 until (a(j) eq 5);
end;
if j jt 11;
drop j
run;

* cat and find;
data five;
set answers;
if find(cat(of a:),'5');
run;
```

```
SAS log ...

243  * cat and find;
244  data five;
245  set answers;
246  if find(cat(of a:),'5');
247  run;

NOTE: There were 5 observations read from
the data set WORK.ANSWERS.
NOTE: The data set WORK.FIVE has 4
observations and 11 variables.
```

Remember what the CAT function is doing. If you look at the box on the bottom of the first page showing both functions and equivalent code , the CAT function is said to be replacing the old concatenation operator. If we compare the results of the CAT function with the equivalent code (example 6), it is obvious that the CAT function is doing more than merely replacing the concatenation operator. Both the SAS log and new strings indicate that much more is occurring.

Using the concatenation operator produces a NOTE in the SAS log about numeric-to-character conversion.

```
* example 6;
data cat_numeric;
set answers;
new_way = cat(of a:);
old_way = a1 || a2 || a3 || a4 || a5 || a6 || a7 || a8 || a9 || a10;
run;
```

```
304   * example 6;
305   data cat_numeric;
306   set answers;
307   new_way = cat(of a:);
308   old_way = a1 || a2 || a3 || a4 || a5 || a6 || a7 || a8 || a9 || a10;
309   run;

NOTE: Numeric values have been converted to character values at the places given by:
(Line):(Column).
      308:11   308:17   308:23   308:29   308:35   308:41   308:47   308:53   308:59
308:65
NOTE: There were 5 observations read from the data set WORK.ANSWERS.
NOTE: The data set WORK.CAT_NUMERIC has 5 observations and 13 variables.
```

```
                              CONCATENATED NUMERIC VARIABLES

 new_way                                              old_way

5533211111   5      5      3     3      2      1        1      1      1        1
0000000555   0      0      0     0      0      0        0      5      5        5
1122400000   1      1      2     2      4      0        0      0      0        0
5555555555   5      5      5     5      5      5        5      5      5        5
1010105553   1      0      1     0      1      0        5      5      5        3
```

Since that conversion is equivalent to using PUT(numeric_variable,BEST12.), the concatenated string OLD_WAY contains the value of the variables A1 through A10 with each value followed by 11 blanks. The CAT function not only concatenated the numeric values, but also appears to have implied the following equivalent code ...

```
new_way = put(a1,1.) || put(a2,1.)|| ... || put(a9,1.) || put(a10,1.);
```

(QUESTION: what is the length of OLD_WAY? NEW_WAY? Answer: 120 and 200 ... do you know why?)

MORE COMPLEXITY: SEARCHING FOR DIAGNOSES (PART 1)

A common task when using medical record data is to identify patients who have been diagnosed as having one (or more) medical conditions. For example, one might want to search through a data set and extract all the observations that contain a diagnosis of diabetes, or observations with a mention of either diabetes or asthma. Since medical records normally contain multiple diagnoses, the task is similar to that posed in example 2, search through multiple character variables and identify those patients where at least one character variable contains a specific value. However, in examples 7 and 8, we are not looking for a single character ('Y'), but in the case of diabetes, a variable whose value starts with '250' or in the case of diabetes or asthma, a variable that starts with either '250' or '493'. Diagnosis codes are commonly stored as character variables with a length of five and the string '250' or '493' can be present at any position within the variable, not just at the start so the task is a bit more complicated than a situation where the diagnosis codes contain only three characters.

```
* example 7;
data diag3;
infile datalines truncover;
input (dx1-dx5) (: $3.);
datalines;
025 022
682 401 244 493
592 401 493
428 493 780 V43 250
250
414 V45 401 250
;
run;

data dia_incorrect;
set diag3;
if find(cat(of dx1-dx5),'250') ne 0;
run;

data dia_correct;
set diag3;
if find(catx('*',of dx1-dx5),'250') ne 0;
run;
```

Since there are also potential problems when the diagnosis codes are length three, example 7 shows that the CAT function will not work correctly and must be replaced by the CATX function. Just as shown in example 2, the array and loop approach can be used, but maybe by now you have become a fan of CAT and FIND. Notice in the first record of the DATALINES file, there is a diagnosis code that ends with '25' followed by another that starts with a '0'. If the CAT function is used, the FIND function searches the string '025022' for an occurrence of the string '250' and that observation is added to the data set even though no diagnosis if diabetes ('250') is present.

```
DATA SET DIA_INCORRECT

Obs    dx1    dx2    dx3    dx4    dx5

 1     025    022
 2     428    493    780    V43    250
 3     250
 4     414    V45    401    250
```

```
DATA SET DIA_CORRECT

Obs    dx1    dx2    dx3    dx4    dx5

 1     428    493    780    V43    250
 2     250
 3     414    V45    401    250
```

The CATX function allows the insertion of one or more characters separating values of the variables. When data set DIA_CORRECT is created, an asterisk is inserted between the values of the variables DX1 though DX5. Thus, in the first observation, the FIND function now searches the string '025*022' and does not find an occurrence of the string '250' in record one.

```
* example 8;
data dia_ast;
set diag3;
dia = (find(catx('*',of dx1-dx5),'250') ne 0);
ast = (find(catx('*',of dx1-dx5),'493') ne 0);
run;
```

Using the same data set, DIAG3, we can change the task and now want to add two variables to the data set, DIA and AST. The values of these variables are 1 a diagnosis is present (DIA =1 for diabetes, AST =1 for asthma) and 0 if that diagnosis is absent. Yes, the array and loop approach can be used, but so can CAT and FIND with CATX replacing CAT.

```
DATA SET DIA_AST

Obs    dx1    dx2    dx3    dx4    dx5    dia    ast

 1     025    022                          0      0
 2     682    401    244    493            0      1
 3     592    401    493                   0      1
 4     428    493    780    V43    250     1      1
 5     250                                 1      0
 6     414    V45    401    250            1      0
```

Example 8 uses the same approach as that used in example 7. This time, all observations are placed in data set DIA_AST and the variables DIA and AST indicate if a diagnosis is found. Since the CATX function is used twice, example 8 can be modified by: creating a new variable using the CATX function; searching that new variable with the FIND function for a diabetes or asthma diagnosis code; dropping the new variable. That approach is not used here mainly since I like the elegance of only five lines of SAS code replacing a much more involved array and loop approach. If you find using the CATX function twice 'objectionable', then use the modified approach and create (and remember to drop) a new variable ...

```
string = catx('*',of d1-d5);
dia = (find(string,'250') ne 0);
ast = (find(string,'493') ne 0);
drop string;
```

MORE COMPLEXITY: SEARCHING FOR DIAGNOSES (PART 2)

As mentioned in the previous section, diagnosis codes are often stored as character variables with a length of five. However, major diagnostic categories are defined by the first three characters of the diagnosis codes. For example, the values '25022' '25003' '25093' are all diabetes since they start with '250'. As shown in example 7, even when diagnosis codes have a length of three, you cannot simply search for the string '250' in a concatenated string. Combining code '025' with code '022' results in the string '025022' and that contains the diabetes code '250' even though neither diagnosis was diabetes. When diagnosis codes have a length of five, there are many situations where an individual code will contain the string '250', for example '36250' is a possible diagnosis, but it is not diabetes (fyi ...diagnoses starting with '362' are retinal disorders).

The strategy used in examples 7 and 8, using the CATX function to insert asterisks in the concatenated string, can be used when searching. Example 9 first shows the array and loop approach and then two versions of the CAT and FIND approach, one of which produces incorrect results.

Data set DIAG5 contains observations with five character diagnosis codes. Once again the task is to create two new variables, DIA and AST, that indicate the presence or absence of a diagnosis of diabetes or asthma. The data set contains a mix of observations: with diabetes; with asthma; with both diabetes and asthma; with neither diabetes nor asthma, but with diagnosis codes that contain the strings '250' and '493' not at the start of the codes, but embedded in the codes.

The array and loop approach uses EQ : rather than just EQ to find any diagnosis codes that start with either the string '250' or '493' . Without the colon modifier, the search would be for a three-character code followed by two blanks rather than a code that starts with a three-character code regardless of the subsequent characters in the code. The output below (data set DIA_AST) shows that the array and loop approach correctly classified the diagnoses in each of the observations.

Two CAT and FIND approaches produce different results. The first uses the strategy is the same as that used in examples 7 and 8, but it produces incorrect results. Look at the output on the next page (data set DIA_AST_INCORRECT) and you can see that any observation with variable DX1 starting with '250' or '493' was incorrectly classified. Why?

```
* example 9;
data diag5;
infile datalines truncover;
input (dx1-dx5) (: $5.);
datalines;
25001 V180
5680  78039 6250  49390 53081
4270  4111  4019  36250
486   7990  25082 41400
78659 25000 49320
49392 4660  2449
34839 2765  40493 4280
;
run;

data dia_ast;
set diag5;
array dx(5);
dia = 0;
ast = 0;
do j=1 to 5;
   if dx(j) eq : '250' then dia = 1;
   if dx(j) eq : '493' then ast = 1;
end;
drop j;
run;

data dia_ast_incorrect;
set diag5;
dia = (find(catx('*',of dx1-dx5),'*250') gt 0);
ast = (find(catx('*',of dx1-dx5),'*493') gt 0);
run;

data dia_ast_correct;
set diag5;
dia = (find(catx('*','*',of dx1-dx5),'*250') gt 0);
ast = (find(catx('*','*',of dx1-dx5),'*493') gt 0);
run;
```

The CATX function places an asterisk *between* each diagnosis code. No asterisk is added prior to the first variable mentioned in the list of variables that are concatenated, *separated by asterisks and not preceded by an asterisk*. The correct method produces data set DIA_AST_CORRECT and that method places an asterisk prior to variable DX1 by adding the asterisk as a character constant prior to the variable list (DX1-DX5). Data set DIA_AST_CORRECT created with the CAT and FIND approach using five lines of code is an exact match to data set DIA_AST created with the array and loop approach using eleven lines of code.

DATA SET DIA_AST

Obs	dx1	dx2	dx3	dx4	dx5	dia	ast
1	25001	V180				1	0
2	5680	78039	6250	49390	53081	0	1
3	4270	4111	4019	36250		0	0
4	486	7990	25082	41400		1	0
5	78659	25000	49320			1	1
6	49392	4660	2449			0	1
7	34839	2765	40493	4280		0	0

DATA SET DIA_AST_INCORRECT

Obs	dx1	dx2	dx3	dx4	dx5	dia	ast
1	25001	V180				0	0
2	5680	78039	6250	49390	53081	0	1
3	4270	4111	4019	36250		0	0
4	486	7990	25082	41400		1	0
5	78659	25000	49320			1	1
6	49392	4660	2449			0	0
7	34839	2765	40493	4280		0	0

DATA SET DIA_AST_CORRECT

Obs	dx1	dx2	dx3	dx4	dx5	dia	ast
1	25001	V180				1	0
2	5680	78039	6250	49390	53081	0	1
3	4270	4111	4019	36250		0	0
4	486	7990	25082	41400		1	0
5	78659	25000	49320			1	1
6	49392	4660	2449			0	1
7	34839	2765	40493	4280		0	0

ARRAY AND LOOP VERSUS CAT AND FIND: A 'REAL LIFE' EXAMPLE

A large data set with 2.5 million observations and 16 diagnosis codes was used to test the performance (elapsed and CPU times) of the two approaches. The task was to create five new variables that indicate whether an observation contained at least one diagnosis code indicating the presence of the following: diabetes ('250'); asthma ('493'); heart attack ('410'); female breast cancer ('174'); influenza ('487'). All the diagnosis codes are five characters and represent an admit diagnosis (ADX), principal diagnosis (PDX), and up to fourteen other diagnoses (ODX1 through ODX14). Two different versions of the CAT and FIND approach were used. The first *performs the concatenation once and uses the new variable* to search for diagnosis codes ...

```
string = catx('*','*',adx,pdx,of odx1-odx14:);
dia = (find(string,'*250') gt 0);
ast = (find(string,'*493') gt 0);
ami = (find(string,'*410') gt 0);
brc = (find(string,'*174') gt 0);
flu = (find(string,'*487') gt 0);
drop string;
```

The second *repeats the concatenation* for each search ...

```
dia = (find(catx('*','*',adx, pdx,of odx:),'*250') gt 0);
ast = (find(catx('*','*',adx, pdx,of odx:),'*493') gt 0);
ami = (find(catx('*','*',adx, pdx,of odx:),'*410') gt 0);
brc = (find(catx('*','*',adx, pdx,of odx:),'*174') gt 0);
flu = (find(catx('*','*',adx, pdx,of odx:),'*487') gt 0);
```

The array and loop approach is similar to that used in example 9, the only differences being: the array is larger with 16 rather than 5 diagnosis codes; there are 5 searches done within the loop, not 2; the loop is terminated as soon as a blank diagnosis code is encountered ...

```
array dx(16) adx pdx odx1-odx14;
<more>
do j=1 to 16 until (dx(j) eq ' ');
   if dx(j) eq : '250' then dia = 1;
   if dx(j) eq : '493' then ast = 1;
   if dx(j) eq : '410' then ami = 1;
   if dx(j) eq : '174' then brc = 1;
   if dx(j) eq : '487' then flu = 1;
end;
```

The following table shows the results of the three methods (using SAS version 9.1.3 SP4 on a PC with an Intel Pentium D 940 3.2GHz Dual Core Processor and 2 gigabytes of memory running Windows XP service pack 3). Prior to comparing the methods, the data set is placed in memory using ...

```
sasfile <data set name> load;
```

Each method was run 25 times and the times shown are the mean of the various jobs. The elapsed times and CPU times are longest for the array and loop approach. The CAT and FIND approach with repeated concatenation of the 16 diagnosis codes is the fastest, taking only about 80% of the elapsed and CPU times of the array and loop approach.

METHOD	ELAPSED TIME (SECONDS)	CPU TIME (SECONDS)
ARRAY AND LOOP	21.1	17.4
CAT AND FIND WITH A NEW VARIABLE	19.9	16.4
CAT AND FIND WITH REPEATED CONCATENATION	18.6	14.3

Somewhat surprising (at least to me) is that the repeated concatenation approach takes less time than the CAT and FIND approach that creates a new variables. One possible explanation is that, without a LENGTH statement, the CAT functions produce a variable with a length of 200. That produces some additional overhead when the variable is created and then set to missing during iterations of the data step. Adding a LENGTH statement to the data step when creating a new variable named STRING ...

```
length string $100;
```

reduces the CPU time for that approach from 17.4 To 13.3 seconds and the elapsed time from 19.9 to 18.4 seconds making it the fastest of all the methods attempted.

METHOD	ELAPSED TIME (SECONDS)	CPU TIME (SECONDS)
CAT AND FIND WITH A NEW VARIABLE PLUS A LENGTH STATEMENT	18.4	13.3

One last comment about the time measurements is that when all the above methods were tried in SAS Verison 9.2, the CPU times were comparable to those in V9.1.3. However, the elapsed times were consistently 5 to 15 seconds longer in V9.2 (once again, using the mean results of 25 trials of each method).

CONCLUSION

The CAT and FIND functions provide an alternative to using arrays and loops when searching for values of variables within observations. The CAT and FIND approach requires fewer data step statements than arrays and loops and the SAS code required is quite easy to understand. Though the CAT functions are character functions, they also work with numeric variables. An added benefit is that the CAT and FIND approach was found to take less elapsed time and less CPU time than using arrays and loops when applied to the same task with a very large data set.

TRADEMARK CITATION

CONTACT INFORMATION

The author can be contacted using e-mail... Mike Zdeb msz03@albany.edu

APPENDIX

The following is the full SAS code used for the 'REAL LIFE' example.

```
* put data set with diagnosis codes into memory;
sasfile temp load;


* use an array and a loop;
data array_loop;
set temp;
array dx(16) adx pdx odx1-odx14;
dia = 0;
ast = 0;
ami = 0;
brc = 0;
flu = 0;
do j=1 to 16 until (dx(j) eq ' ');
   if dx(j) eq : '250' then dia = 1;
   if dx(j) eq : '493' then ast = 1;
   if dx(j) eq : '410' then ami = 1;
   if dx(j) eq : '174' then brc = 1;
   if dx(j) eq : '487' then flu = 1;
end;
drop j;
run;


* CAT and FIND with a new variable (and a LENGTH statement);
data cat_find_new;
set temp;
length string $100;
string = catx('*','*',adx,pdx,of odx:);
dia = (find(string,'*250') gt 0);
ast = (find(string,'*493') gt 0);
ami = (find(string,'*410') gt 0);
brc = (find(string,'*174') gt 0);
flu = (find(string,'*487') gt 0);
drop string;
run;


* CAT and FIND with repeated concatenation;
data cat_find_repeat;
set temp;
dia = (find(catx('*','*',adx, pdx,of odx:),'*250') gt 0);
ast = (find(catx('*','*',adx, pdx,of odx:),'*493') gt 0);
ami = (find(catx('*','*',adx, pdx,of odx:),'*410') gt 0);
brc = (find(catx('*','*',adx, pdx,of odx:),'*174') gt 0);
flu = (find(catx('*','*',adx, pdx,of odx:),'*487') gt 0);
run;


* take data set out of memory;
sasfile temp close;
```