SAS Macro Language w pigułce

Wstęp

DEFINICJA MAKR

Język makr to język, który rozszerza możliwości standardowego 4GL. Dzięki umiejętności pisania makr, użytkownik może zautomatyzować wiele procesów, uruchamiać programy warunkowo (np. kod generujący raport tygodniowy co piątek, a raport miesięczny w ostatni dzień miesiąca), czy też dynamicznie tworzyć kod SAS-owy.

CELE UŻYTKOWANIA

- Automatyzacja procesu pisania kodu SAS-owego
- Dynamiczne generowanie kodu
- Warunkowe uruchamianie kodu
- Parametryzacja kodu

CHARAKTERYSTYKA MAKROPROGRAMÓW

- Makra rozpoczynają się znakiem procenta (*%*), po którym występuje nazwa wyrażenia
- Kończą się średnikiem (*;*)
- Wykonywane są przez procesor makr, a nie "zwykły" procesor 4GL
- Makrozmienne rozpoczynają się znakiem ampersand (*&*), po którym występuje ich nazwa

Proces kompilacji

Zanim skaner słów przekaże kod do procesora 4GL sprawdzi, czy nie zawiera on makrowyrażeń. Jeżeli takie napotka, to przekaże je do procesora makr, który je rozwinie i zwróci z powrotem do skanera. Dopiero wtedy uruchomiony zostanie procesor 4GL. W praktyce oznacza to, że makra wykonują się na samym początku procesu kompilacji.

Przykład:

```
%LET temp = 0;
%MACRO test;
   DATA test;
         DO i = 1 to 5;
               %LET temp = %eval(&temp + 1);
               temp = &temp;
               OUTPUT;
         END;
         DROP i;
   RUN;
%MEND;
%test;
```

Po przetworzeniu makra zwróci do skanera kod:

```
DATA test;
   DO i = 1 to 5;
         temp = 1;
         OUTPUT;
   END;
   DROP i;
RUN;
```

Makrozmienne

Przed utworzeniem makrozmiennej należy zaznaczyć, że kompilator traktuje wszystkie wartości jako **tekst** i nie można wymusić, aby było inaczej. Jednak nie oznacza to, że na makrozmiennych nie można wykonywać operacji arytmetycznych – odpowiada za to funkcja ***eval*** opisana w rozdziale ***Funkcje makro***.

DEFINIOWANIE

Makrozmienne można definiować na kilka różnych sposobów. Jednym z nich jest instrukcja ***%LET***. Można z niej korzystać w dowolnym miejscu programu. Aby utworzyć makrozmienną *test* o wartości *Hello, World!* należy uruchomić poniższy kod:

```
%LET test = Hello, World!;
```

Znaki cudzysłowu nie są potrzebne – wszystkie znaki (za wyjątkiem początkowych i kończących spacji) występujące do średnika są traktowane jako wartość zmiennej. Utworzona w taki sposób zmienna w dowolnym miejscu poza ciałem makra jest globalna, a tym samym dostępna w dowolnym miejscu programu.

Aby utworzyć makrozmienną w *DATA STEP* należy skorzystać z funkcji ***symput*** lub ***symputx*** opisanych w rozdziale ***Dostęp do makrozmiennych w DATA STEP***.

USUWANIE

Aby usunąć makrozmienną należy skorzystać z funkcji *%SYMDEL*.

Przykład:

```
%SYMDEL makrozmienna <makrozmienna2 makrozmienna3>;
```

ODWOŁYWANIE

Do makrozmiennych odwołuje się za pomocą znaku ampersand (*&*) i nazwy zmiennej.

Przykład: istnieje globalna makrozmienna *&temp* i chcemy wyświetlić ją w logu. Aby to zrobić należy uruchomić poniższy kod:

```
%PUT &temp;
```

Aby odwołać się do makrozmiennej w *DATA STEP* należy skorzystać z funkcji ***symget*** opisanej w rozdziale ***Dostęp do makrozmiennych w DATA STEP***.

CHARAKTERYSTYKA

- Dzielą się na **globalne** lub **lokalne**
    - Makrozmienne lokalne to takie, które są dostępne wyłącznie w makrze, w którym zostały one zdefiniowane
    - Makrozmienne globalne są dostępne w dowolnym miejscu programu, definiowane są poza ciałem makra, w ciele makra za pomocą wyrażenia *%GLOBAL nazwa_zmiennej* lub za pomocą funkcji ***symputx*** z opcją *G.*
- Mają **minimalną** długość **0** znaków (wtedy przyjmują wartość *null*)
- Mają **maksymalną** długość **65 534** znaków (64K)
- Przechowują wartości **numeryczne jako tekst**

Domyślne makrozmienne

&SYSDATE*

Data w krótkim formacie – DDMONYY, np. *26APR11*

&SYSDATE9*

Data w długim formacie – DDMONYYYY, np. *26APR2011*

&SYSDAY*

Dzień tygodnia w języku angielskim, np. *Tuesday*

&SYSTIME*

Godzina systemowa, np. *16:01*

** - data/czas rozpoczęcia sesji SAS – jeżeli SAS został uruchomiony we wtorek, a z makra odwołującego się do zmiennych zawierających datę lub czas skorzystamy w środę, to zmienne będą zwracały dane wtorkowe.*

&SYSSCP

Skrócona nazwa systemu operacyjnego, np. *WIN*

&SYSLAST

Ostatnio utworzony zbiór

&SYSPARM

Wartość argumentu *-sysparm*, który został przekazany przy uruchomieniu sesji SAS.

Przykład (uruchomienia SAS z poziomu Windows):

```
sas -sysparm work.test;
```

Kod w SAS:

```
%PUT &SYSPARM;
```

Zwróci:

```
work.test
```

AUTOMATIC**

Lista wszystkich automatycznych makrozmiennych

USER**

Lista makrozmiennych zdefiniowanych przez użytkownika

ALL**

Lista wszystkich makrozmiennych

*** - z tych wyrażeń należy korzystać z funkcją %PUT, np. %PUT _USER_ wyświetli w logu wszystkie makrozmienne zdefiniowane przez użytkownika.*

Makra

DEFINIOWANIE MAKRA

Aby móc korzystać z wyrażeń makr, najpierw należy utworzyć ciało makra. Ogólna definicja makra wygląda następująco:

```
%MACRO nazwa;
   /* kod makra */
%MEND <nazwa>;
```

MAKRA A MAKROZMIENNE

Różnica pomiędzy makrami a makrozmiennymi polega na tym, że makra można rozumieć jako małe programy, podczas gdy makrozmienne to po prostu wyrażenia, które mają przypisaną jakąś wartość tekstową i poza tym nie pełnią żadnej innej roli.

CO MAKRO MOŻE ZAWIERAĆ?

- Dowolny tekst
- Wyrażenia SAS-owe lub STEP-y
- Zmienne, funkcje, wyrażenia lub odwołania makro

MAKRA PARAMETRYZOWANE

Makra mają możliwość posiadania parametrów. Definiuje się je w nawiasach funkcyjnych za nazwą makra. Parametry są traktowane jako makrozmienne lokalne. Odwołujemy się do nich w sposób standardowy – za pomocą znaku ampersand (*&*) i ich nazwy. W przypadku, gdy chcemy, aby nasze makro posiadało kilka parametrów, to oddzielamy je przecinkami:

```
%MACRO parametryzowane(p1, p2);
   %PUT &p1 * &p2;
%MEND;
%parametryzowane(a, b);
```

Zwróci:

```
a * b
```

Przykład:

```
DATA test;
   INPUT imie $ wiek $;
   CARDS;
         Kamil 22
         Jan 33
         Ania 44
         Magda 22
   ;
RUN;

%MACRO znajdz(wiek);
   PROC PRINT DATA=work.test(WHERE=(wiek="&wiek"));
   RUN;
%MEND;

%znajdz(22);
```

Przykład utworzy zbiór danych *test* z 4 wierszami. Makro *%znajdz* będzie wymagało parametru *wiek*, który później zostanie przekazany do procedury *PRINT* za pomocą makrozmiennej *&wiek*.

Wynik działania programu (procedura *PRINT*):

```
Obs    imie      wiek
1      Kamil     22
4      Magda     22
```

Instrukcje warunkowe

Język makr ma możliwość warunkowego uruchamiania kodu. Są to standardowe instrukcje typu *if/else*, za wyjątkiem tego, że przed słowem kluczowym zawierają znak procenta. Instrukcja *%else* jest opcjonalna.

W przypadku, gdy chcemy aby po *if/else* występował ciąg instrukcji (a nie pojedyncza instrukcja), to deklarujemy go pomiędzy *%DO* a *%END*.

Przykład:

```
%MACRO czyWeekend;
   %IF &sysday = Friday OR &sysday = Saturday OR &sysday = Sunday %THEN
         %DO;
               %INCLUDE "c:\raportWeekendowy.sas";
               %PUT Uruchomiono raport weekendowy;
         %END;
   %ELSE
         %PUT Normalny dzien tygodnia;
%MEND;
%czyWeekend;
```

W piątek, sobotę oraz niedzielę makro uruchomi program z pliku C:\raportWeekendowy.sas i zwróci w logu:

```
Uruchomiono raport weekendowy
```

W pozostałe dni tygodnia zwróci:

```
Normalny dzien tygodnia
```

Pętle

%DO %WHILE

```
%DO %WHILE(wyrazenie);
   /* kod */
%END;
```

Pętla *WHILE* przed wykonaniem kodu w niej zawartego sprawdza *wyrażenie*. Kod zawarty w tej pętli wykonuje się do momentu, gdy *wyrażenie* zwróci wartość *false*.

Przykład:

```
%MACRO wyrazy(tekst, separator=*);
   %LET i = 1;
   %LET wyraz = %SCAN(&tekst, &i, &separator);
   %IF &wyraz = %THEN
         %PUT Tekst jest pusty.;
   %ELSE
         %DO %WHILE(&wyraz ne );
               %PUT wyraz &i to: &wyraz;
               %LET i = %EVAL(&i+1);
               %LET wyraz = %SCAN(&tekst, &i, &separator);
         %end;
%MEND;
%wyrazy(To*jest*przykladowy*tekst);
```

Zwróci

```
Wyraz 1 to: To
Wyraz 2 to: jest
Wyraz 3 to: przykladowy
Wyraz 4 to: tekst
```

%DO %UNTIL

```
%DO %UNTIL(wyrazenie);
   /* kod */
%END;
```

Pętla *UNTIL* najpierw wykonuje kod w niej zawarty, a później sprawdza wyrażenie. Oznacza to, że bez względu na to, czy wyrażenie jest prawdziwe, pętla wykona kod **co najmniej raz**. Dodatkową różnicą pomiędzy *WHILE* a *UNTIL* jest to, że pierwsza z nich kończy się w przypadku, gdy wyrażenie zwraca wartość *false*, a druga, gdy zwraca *true*.

Dostęp do makrozmiennych w DATA STEP

SYMPUT

```
CALL SYMPUT("makrozmienna", "wartość");
```

Ustawia wartość *makrozmienna* na *wartość*. Dodatkowo, w funkcji *SYMPUT* można zagnieżdżać inne funkcje, np:

```
CALL SYMPUT("temp", put(100, binary8.));
```

SYMPUTX

```
CALL SYMPUTX("makrozmienna", "wartość" <, L|G>);
```

Ustawia wartość *makrozmienna* na *wartość* i dodatkowo usuwa spacje na początku i na końcu łańcucha znaków. Opcjonalny trzeci argument przyjmuje wartości *L* lub *G* i tworzy odpowiednio lokalną lub globalną makrozmienną.

SYMGET

```
x = SYMGET("makrozmienna");
```

Pobiera wartość *makrozmienna* i przypisuje ją do zmiennej *x*.

Funkcje makro

LENGTH

```
%LENGTH(lancuch_znakow);
```

Zwraca długość *lancuch_znakow*.

Przykład

```
%PUT %LENGTH(test);
```

Zwróci:

```
4
```

EVAL

```
%EVAL(lancuch_znakow);
```

W języku makr wszystkie zmienne traktowane są jako łańcuch znaków, dlatego też nie działają operacje arytmetyczne. Aby wymusić na kompilatorze traktowanie znaków specjalnych (np. +, czy *) standardowo należy skorzystać z funkcji *%EVAL*.

Przykład:

```
%LET a = 2+2;
%LET b = 2;
%PUT %EVAL(&a);
%PUT %EVAL((&a-1)*&b);
```

Zwróci:

```
4
6
```

SCAN

```
%SCAN(lancuch_znakow, n <, separator>);
```

Zwraca *n*-wyraz z *lancuch_znakow*, oddzielony przez separatory (domyślnie: spacja ( ) & ! $ * ; - / , . %)

Przykład:

```
%LET temp = *a*b*c*wyraz1*wyraz2;
```

```
%PUT %SCAN(&temp, 4);
```

Zwróci:

```
wyraz1
```

SUBSTR

```
%SUBSTR(lancuch_znakow, poczatek <, n>);
```

Zwraca *n* kolejnych znaków od *poczatek* z *lancuch_znakow* lub wszystkie znaki od *poczatek* do końca łańcucha, jeżeli *n* nie zostało podane.

Przykład:

```
%LET dzisiaj = &sysdate9;
%LET temp = qwerty;

%PUT dzisiaj;
%PUT %SUBSTR(&dzisiaj, 3);
%PUT %SUBSTR(&dzisiaj, 3, 5);
%PUT %SUBSTR(&temp, 5, 1);
```

Zwróci

```
26APR2011
APR2011
APR
t
```

SYSFUNC

```
%SYSFUNC(funkcja_sasowa <, format>)
```

Uruchamia podaną funkcję SAS-ową i opcjonalnie nakłada na nią *format*.

Przykład:

```
%PUT %SYSFUNC(today(), date11.);
```

Zwróci:

```
26-APR-2011
```

UPCASE

```
%UPCASE(lancuch_znakow);
```

Przekształca *lancuch_znakow* – zamienia małe litery na wielkie

Przykład:

```
%LET temp = abc;
%PUT %UPCASE(&temp);
```

Zwróci:

```
ABC
```

Opcje makr

MCOMPILENOTE

```
OPTIONS MCOMPILENOTE=ALL|NONE;
```

Opcja *MCOMPILENOTE* odpowiada za wyświetlanie noty po skompilowaniu makra. Domyślnie opcja ma ustawioną wartość na *NONE*.

Przykład:

```
OPTIONS MCOMPILENOTE=ALL;
%MACRO test;
   %PUT Hello, World!;
%MEND test;
```

Zwróci:

```
NOTE: The macro TEST completed compilation without errors.
      3 instructions 36 bytes.
```

MLOGIC

```
OPTIONS MLOGIC|NOMLOGIC;
```

Opcja *MLOGIC* odpowiada za wyświetlanie informacji w logu na temat uruchamiania makr. Domyślnie jest ustawiona opcja *NOMLOGIC*.

Przykład:

```
OPTIONS MLOGIC;
%MACRO test;
   %PUT Hello, World!;
%MEND test;
%test;
```

Zwróci:

```
MLOGIC(TEST): Początek wykonywania.
MLOGIC(TEST): %PUT Hello, World!
Hello, World!
MLOGIC(TEST): Koniec wykonywania.
```

MPRINT

```
OPTIONS MPRINT|NOMPRINT;
```

Wyświetlanie kodu wysyłanego do kompilatora SAS (**nie** makrokompilatora) po wykonaniu makra. Domyślnie opcja ma ustawioną wartość *NOMPRINT*.

Przykład:

```
OPTIONS MPRINT;
%MACRO test;
   DATA test;
         DO i = 1 TO 3;
               PUT i;
         END;
   RUN;
%MEND test;
%test;
```

Zwróci:

```
MPRINT(TEST):   DATA test;
MPRINT(TEST):   DO i = 1 TO 3;
MPRINT(TEST):   PUT i;
MPRINT(TEST):   END;
MPRINT(TEST):   RUN;

1
2
3
```

SYMBOLGEN

```
OPTIONS SYMBOLGEN|NOSYMBOLGEN;
```

Opcja *SYMBOLGEN* odpowiada za rozwijanie wartości makrozmiennych, gdy program się do nich odwołuje. Domyślnie ustawiona jest opcja *NOSYMBOLGEN*.

Przykład:

```
OPTIONS SYMBOLGEN;
%PUT &sysdate;
```

Zwróci:

```
SYMBOLGEN: Makrozmienną SYSDATE rozwinięto do postaci 27APR11
27APR11
```

Sztuczki i kruczki

WSKAZYWANIE ZAKOŃCZENIA REFERENCJI DO MAKROZMIENNEJ

Gdy w ciągu znaków odwołujemy się do makrozmiennej i nie ma po niej spacji, to kompilator makr nie wie, gdzie kończy się jej referencja. Możemy wskazać zakończenie ręcznie, za pomocą znaku kropki.

Przykład:

```
%LET temp = 123;
%PUT ZmiennaTempMaWartosc=&temp.ATutajJestDalszyCiag;
```

Zwróci:

```
ZmiennaTempMaWartosc=123ATutajJestDalszyCiag
```

POŚREDNIE REFERENCJE DO MAKROZMIENNYCH

Gdy chcemy odwołać się do makrozmiennej za pomocą kombinacji makrozmiennych, to korzystamy z podwójnego znaku ampersand (&). W zrozumieniu przydatna jest opcja ***symbolgen***.

Przykład:

```
%LET test1 = Hello, World!;
%LET numer = 1;
%PUT &&test&numer;
```

Zwróci:

```
SYMBOLGEN: && rozwinięto do &.
SYMBOLGEN: Makrozmienną NUMER rozwinięto do postaci 1
SYMBOLGEN: Makrozmienną TEST1 rozwinięto do postaci Hello, World!
Hello, World!
```

Spis treści