2016 年 05 月 27 日

計算機実習 III (2016 年度)
第 6 回: コマンドプロンプトとバッチファイル その 6

(http://takeno.iee.niit.ac.jp/%7Eshige/
math/lecture/comp4/comp4.html)

目 次

1 増分形式の for 文

バッチファイルで繰り返しの作業を行うのに、C 言語と同様 for 文 (正確には for コマンド) が使える。for 文の使い方は 2 通りあるが[1]、まずは C 言語に近い増分形式の for 文を紹介する。

増分形式の for 文の使用書式は以下の通り。

```
for /l %[変数名] in ([初期値],[増分],[終了値]) do [コマンド群]
```

この増分形式では、for の後ろに「/l」(イチではなくエル) のオプションを指定する。

[コマンド群] の部分には、コマンド 1 つ、あるいは ( ) でグループ化した複数コマンドを置く。

[初期値], [増分], [終了値] はいずれも整数で、for 文はループ変数である %[変数名] にまず [初期値] を代入して [コマンド群] を 1 回実行し、次に %[変数名] に [増分] を加えて [コマンド群] を 1 回実行する。これを %[変数名] の値が [終了値] 以下の間繰り返す。例えば、

```
Z:¥> for /l %a in (2,3,20) do echo %a
```

[1] 本当は、for コマンドの形式は 3 通りあるが、この実習ではそのうち 2 つを紹介する。

は、ループ変数 %a に、2, 5, 8, 11, 14, 17, 20 が順に代入され、それぞれに対して「echo %a」が実行されるので、これら 7 つの数字が 1 行ずつ表示されることになる。

for 文は終了値を越えたときに終了するので、上の例は終了値を 21, 22 に変えても結果は変わらない。

for 文に関する注意:

- C 言語では「for (j=1; j<=100; j+=2)」のように変数を使った条件式や代入式を書くが、こちらの for 文は数字だけをかっこ内に並べる。
- ループ変数の名前はアルファベット 1 文字。大文字と小文字は区別される。
- ループ変数の値を参照するときは、(set /a の右辺でも) %[変数名] とする。
- コマンドプロンプトでは %[変数名] と書くが、バッチファイル内では % を重ねて %%[変数名] とする必要がある。

例 (100 以下の奇数の和を表示):

```
@echo off
set x=0
for /l %%a in (1,2,100) do set /a x=x+%%a
echo %x%
```

なお、do の前で改行して do を次の行頭に置いたり、do とその後ろの「(」の間で改行するとエラーになる。例:

```
@echo off
set x1=1
set x2=1
for /l %%a in (3,1,40) do (
    set /a x=x1 + x2
    set /a x1=x2
    set /a x2=x
)
echo %x2%
```

このバッチファイルは、フィボナッチ数列:

$$x_1 = 1,\ x_2 = 1,\ x_n = x_{n-1} + x_{n-2}\ (n \geq 3)$$

の x_{40} の値を表示する。 なお、for 文の ( ) 内で x1, x2 への単なる代入で「set」でなく「set /a」を使っているのは、右辺を %x2%, %x% と書くのを避けるため。これを %x2% 等と書いた場合の問題点については、3 節、4 節で説明する。

増分や初期値、終了値には負の値も使用でき、例えば、

```
Z:¥> for /l %a in (9,-2,-3) do echo %a
```

とすると、%a には順に 9,7,5,3,1,−1,−3 が代入され、これらが 1 行ずつ表示される。増分が負の場合は、ループ変数が終了値「以上」の間実行を繰り返す。

2 リスト形式の for 文

もう一つの形式である、リスト形式の for 文の使用書式は以下の通り。

```
for %[変数名] in ([リスト]) do [コマンド群]
```

[リスト] には複数の文字列を、スペースかカンマ (,) 区切りで書き並べる。すると、それぞれの文字列がループ変数にひとつずつ代入されて、その分だけ [コマンド群] のコマンドが実行される。よってこの形式では、ループ変数の値は整数ではなく、文字列となる。なお、この形式は for の後ろにオプションをつけない[2]。例えば、

```
Z:¥> for %a in (dog cat mouse) do echo 名前は%a
```

は、%a に順番に dog, cat, mouse という文字列を代入し、そのそれぞれに対してコマンド「echo 名前は%a」を実行するので、結果として「名前は dog」「名前は cat」「名前は mouse」が 1 行ずつ表示される。

バッチファイル中で、バッチファイルへのオプションすべての文字列を表す「%*」を for 文のリスト指定に使用すれば、そのバッチファイルに指定したすべてのオプションに対する処理を行うことができる。例:

```
@echo off
set x=0
for %%a in (%*) do set /a x=x+%%a
echo %x%
```

このバッチファイルは、指定したオプションの整数値としての総和を表示する。

ワイルドカードを for 文のリストに使用すると、そのワイルドカードにマッチするファイル名、ディレクトリ名をすべてリストに指定したことと同じになる。例:

```
Z:¥> for %a in (AKB*.JPEG) do move %a picture¥%~na.jpg
```

[2]つまり、このリスト形式の方がバッチファイルの for 文ではデフォルト形式。

これは、カレントディレクトリにある該当ファイルの拡張子を、.JPEG から .jpg に変更しながら picture というディレクトリに移動する。

上の例の for 文のコマンド部分の「%~na」は、ループ変数の「%」と変数名「a」の間に「~n」を入れたものだが、これは for 文のループ変数がファイル名やディレクトリ名である場合に表 1 のような特別な変換処理を行うもので、「%~na」は「%a」が指すファイル名の拡張子を除いた部分に変換される。例えば「%a」が「AKB048.JPEG」の場合は 「%~na」は「AKB048」となり、よって上のコマンドにより「move AKB048.JPEG picture¥AKB048.jpg」が実行されることになる。

変数置換形式	意味	例
%~f[変数名]	フルパス名に変換	Z:¥dir1¥dir2¥test1.bat
%~d[変数名]	ドライブ名に変換	Z:
%~p[変数名]	ファイルの居場所のパスに変換	¥dir1¥dir2¥
%~n[変数名]	ファイル名の拡張子以外に変換	test1
%~x[変数名]	拡張子部分に変換	.bat
%~a[変数名]	ファイルの属性に変換	--a------
%~t[変数名]	ファイルの日付/時刻に変換	2014/05/20 10:40
%~z[変数名]	ファイルのサイズに変換	238

表 1: for 文ループ変数のファイル名変換処理

例えば、以下のバッチファイルは、オプションで指定したファイルのうち、拡張子が .xls か .doc のファイルを Data というディレクトリに移動する。

```
@echo off
for %%a in (%*) do (
    if "%%~xa"==".xls" move %%a Data
    if "%%~xa"==".doc" move %%a Data )
```

3 for 文の if–goto ループへの書き換え

通常、1 行のコマンド、またはグループ化したコマンド群の中で 「%[変数名]%」が使われていると、第 5 回で述べたように、実行される**前**にそれらが値に置き換わるので、for 文のコマンド部分でループ変数以外の環境変数を使う場合は注意が必要。

例えば、$1, 1+3, 1+3+5, \ldots, 1+3+\cdots+99$ の値を表示させるために以下のようにすると、「echo [%x%]」の変数展開は、for 文の実行前にその時点の x の値である 0

に展開されてしまい、結局 50 回「echo 0」が実行されることになる。

```
@echo off
set x=0
for /l %%a in (1,2,99) do (
    set /a x=x+%%a
    echo [%x%]
)
```

なお、「set /a x=x+%%a」の文の方は、%[変数名]% は使われていないし、ループ変数 %%a は毎回値が更新されるので、変数 x の値は正しく計算される。しかし、これを「set /a x=%x%+%%a」としてしまうと「set /a x=0+%%a」が 50 回行われてしまい x の値も正しく計算されなくなる。

この問題を解消するには、

- for を使わず、if と goto によるループ (第 5 回) に書き換える
- 環境変数の値を逐次展開させる「遅延展開機能」(次節) を有効にする

などの方法がある。ここでは if, goto によるループへの書き換えを紹介する。

「for /l %a in (x,y,z) do [コマンド]」のフローチャートは (y が正の場合) 図 1 (左) のようになるから、例えば上の for 文の例を if と goto のループで書くと、図 1 (右) のようになる。こうすれば変数は展開されながら 1 行ずつ順に実行されるため、変数が逐次展開され期待通りの結果が得られる。

なお、リスト形式の for 文は if, goto でのループ化はできないので、その場合は次節の遅延展開を使う必要がある。

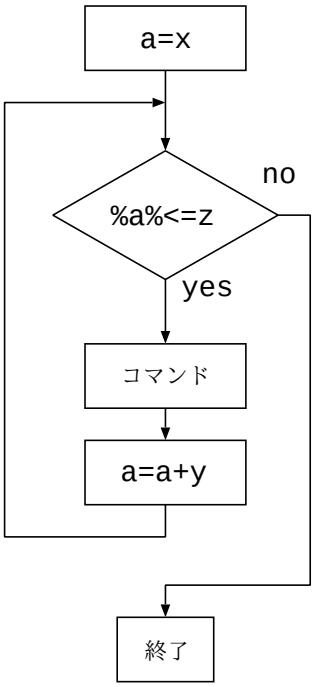
4 環境変数の遅延展開

この節では、変数展開の問題に対するもう一つの方法である、環境変数の遅延展開について説明する。**遅延展開** は、変数の展開を実行前でなく実行時に行う機能である。遅延展開を有効にするには、

```
setlocal enabledelayedexpansion
```

というコマンドを事前に実行し[3]、そして遅延展開させる環境変数を「% %」でなく、

[3]長いオプション部分は、enable = 有効、delayed = 遅れた、expansion = 展開、の 3 つの単語を連続させたもの。setlocal は局所変数の設定コマンド。




```
@echo off
set x=0
set a=1
:loopstart
if not %a% leq 99 goto loopend
set /a x=x+a
echo [%x%]
set /a a+=2
goto loopstart
:loopend
```

図 1: 「for 文のフローチャートと前の例の if, goto での書き換え

「! !」で囲む。例えば、「timeout /t 3」は 3 秒待つだけのコマンドであるが、

```
@echo off
( echo %time%
    timeout /t 3
    echo %time% ) > time.dat
```

とすると、かっこ部分の実行**前**に 2 箇所の %time% が同時に展開されるので、ファイル time.dat には同じ時刻が 2 つ記録される。これを、遅延展開を使って、

```
@echo off
setlocal enabledelayedexpansion
( echo !time!
    timeout /t 3
    echo !time!  )  > time.dat
```

とすると、かっこ内の各行の実行**時**に !time! の動的変数の値が参照されるので、異なる実行時刻 (3 秒ほどずれたもの) が記録されるようになる。

リスト形式の for 文

```
@echo off
set s=abc
for %%a in (dog cat mouse) do set s=%s% %%a
echo %s%
```

では、for 文のコマンド部分の環境変数 %s% は、for 文の実行**前**にその時点での値に展開されるので、「set s=abc dog」「set s=abc cat」「set s=abc mouse」を実行したのと同じことになり、結果として「abc mouse」が表示されるが、これを遅延展開を用いて、

```
@echo off
setlocal enabledelayedexpansion
set s=abc
for %%a in (dog cat mouse) do set s=!s! %%a
echo %s%
```

とすると、!s! の部分は set の実行**時**に s のその時点での値に展開されるので、「set s=abc dog」「set s=abc dog cat」「set s=abc dog cat mouse」が順に実行されることになり、「abc dog cat mouse」が表示されるようになる。

150 から 200 までの乱数を、ファイル rand1.dat に 100 行書き出すバッチファイル:

```
@echo off
setlocal enabledelayedexpansion
copy nul rand1.dat
for /l %%a in (1,1,100) do (
    set /a x=!random! %% 51 + 150
    echo %%a !x! >> rand1.dat
)
```

でも、for 文の内部の「!random!」を「%random%」にしてしまうと、for 文実行**前**にそれが一つの乱数に置き換えられ、100 行その同じ値が表示されてしまうことになる。

遅延展開を使って展開のタイミングをずらすことにより、変数名に変数値を使って「疑似配列変数」を作ることもできるようになる[4]。例えば、a1 から a100 という変数を使う場合、添字を j として例えば a3 を使う場合、

```
set j=3
echo %a%j%%
```

[4]元々コマンドプロンプト、バッチファイルには正式な配列変数はない。

では当然適切には展開されない (a3 の値にはならない) が、

```
setlocal enabledelayedexpansion
set j=3
echo !a%j%!
```

とすると、3 行目の実行**前**に先に %j% が 3 に置き換わり、次に !a3! の値が参照されて置き換わる。もちろん、C 言語風に a[3] のような使い方もできる。例:

```
@echo off
setlocal enabledelayedexpansion
set N=0
for %%k in (%*) do (
    set /a N+=1
    set a[!N!]=%%k
)
for /l %%j in (%N%,-1,1) do echo !a[%%j]!
```

このバッチファイルは、バッチファイルへのオプション (%*) を、最初の for 文で一旦 a[1], a[2], ... という名前の環境変数[5]に保存し、次の for 文で、それらを指定したのとは逆順に表示する。

なお、この後ろの for 文では疑似配列の添字はループ変数なので、添字は逐次更新され、本来遅延展開は必要ないが、それに対する疑似配列の値を遅らせて展開させるために遅延展開が必要となる。これを %a[%%j]% としてしまうと適切には展開されない。

5 ページの例も、遅延展開を有効にして「echo [%x%]」を「echo [!x!]」に変えれば意図した結果となる。

[5]例えば a[1] は、あくまでそういう 4 文字の名前の環境変数というだけであり、配列変数ではなく、[ ] にも特別な意味はない。