2016 年 05 月 20 日

計算機実習 III (2016 年度)
第 5 回: コマンドプロンプトとバッチファイル その 5

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

目 次

1 かっこによるグループ化

C 言語ではいくつかの命令を { } で囲むことでグループ化できるが、バッチファイルでも複数のコマンドを ( ) で囲むことでグループ化できる。これは、後で説明する if や for とともに良く用いられる。例:

```
@echo off
(
    echo == 日時の表示 ==
    echo 今日の日付は %date%
    echo 現在の時刻は %time%
)
```

「(」の後ろ (右) に最初のコマンドを書いてもいいし、この例のように「(」の後ろで改行してから最初のコマンドを書いてもよい。また、最後のコマンドの後ろに「)」を書いてもいいし、この例のように最後のコマンドの後ろで改行してから行頭に「)」を書いてもよい。

「( )」の中にさらに「( )」によるコマンドグループを入れることも可能。

グループ化の最後の「)」の後ろに「> [ファイル]」や「>> [ファイル]」をつけると、グループ内のコマンドすべての出力がそのファイルに書き出される。例えば、

```
( echo 3
    echo 5 ) > file
```

は、ほぼ (いくつかの空白の違いを除けば)

```
echo 3 > file
echo 5 >> file
```

とすることと同じになる。

グループ化に関する注意:

- ( ) 内では echo コマンドで「(」や「)」を文字として表示させることはできない。
- 「( )」の中にある %[変数名]% は、「( )」内のコマンドが**実行される前**に変数値への置き換えがすべて行われてからコマンドが実行される。例えば、

  ```
  @echo off
  set x=0
  set y=0
  ( set x=1
      set /a y=%x% + 3
      echo %y% )
  ```

  というバッチファイルの場合、3 行目が実行された後、4 行目以降の ( ) の部分が実行される前に ( ) 内の %[変数名]% の部分が

  ```
  ( set x=1
      set /a y=0 + 3
      echo 0 )
  ```

  と、その時点 (実行前の時点) での値に置き換えられてから実行される。よって結果は 4 ではなく 0 が表示され、y の値も 4 ではなく 3 になる。

  しかし、( ) 内では %[変数名]% を使わず、echo 文も ( ) の外に出して

  ```
  @echo off
  set x=0
  set y=0
  ( set x=1
      set /a y=x + 3 )
  echo %y%
  ```

のようにすると 4 が表示され、y の値も 4 になる。なお、これらは先頭の「@echo off」を rem でコメントアウトして実行してみるとよくわかるだろう。

2 if 文の基本構造

コマンドプロンプトやバッチファイルでも if 文 (正確には if コマンド) を用いることで条件分岐を行うことができる。if は、基本的に以下の 2 種類の形式が使用できる。

1. `if [条件] [コマンド群]`
2. `if [条件] [コマンド群 1] else [コマンド群 2]`

この [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。例:

```
@echo off
if %x%==0 ( echo x は偽です。) else echo x は真です。
```

if, else を複数つなげたい場合、C 言語同様

```
if [条件 1] ( [コマンド 1]
) else if [条件 2] ( [コマンド 2]
) else [コマンド 3]
```

のように書くこともできるし、

```
if [条件 1] ( [コマンド 1]
) else (
    if [条件 2] ( [コマンド 2]
    ) else [コマンド 3]
)
```

としてもよい。

注意:

- 上の最初の例のように、2. の形式の場合は [コマンド群 1] がコマンド 1 つであってもそこにスペース区切りのオプションが含まれるときは、それを ( ) で囲んで

  `if [条件] ( [コマンド 1] ) else [コマンド 2]`

  とする必要がある。そうしないと、その後の「else」以降も [コマンド 1] のオプションの続きとみなされ、1. の形式として実行されてしまう。

- 2. の形式では、else を行頭に書くと新たなコマンド行とみなされエラーになる。よって、else は [コマンド群 1] の直後、またはそのグループ化の閉じかっこ「)」の後ろに改行せずに続けて書かなければいけない。例えば、

```
if [条件] ( [コマンド 1]
) else [コマンド 2]
```

はよいが、

```
if [条件] ( [コマンド 1] )
else [コマンド 2]
```

はエラーになる。

- if の [条件式] の後ろや else の後ろに「(」を続けて書くときは、その間に空白を入れないとエラーになりうる。

3 if 文の条件式

if 文の条件部分には、表 1 のいずれかの形式を使う。

形式	意味
[値 1]==[値 2]	等しければ真
[値 1] [比較演算子] [値 2]	文字列、数値の比較
exist [ファイル名]	ファイルやディレクトリが存在すれば真
errorlevel [番号]	直前コマンド終了コードが番号以上なら真
defined [変数名]	その環境変数が定義されていれば真

表 1: if 文の条件で使える形式

表 1 の 2 番目の形式の比較演算子は、表 2 の 3 文字の演算子である。

演算子	意味	演算子	意味
equ	等しい (=)	neq	等しくない ($\neq$)
leq	以下 ($\leq$)	lss	より小さい (<)
geq	以上 ($\geq$)	gtr	より大きい (>)

表 2: 比較演算子

表 1 の条件式の前に、例えば「not %x%==0」のように not をつけるといずれの場合も真偽が反転される。

以下のバッチファイルは、ほぼ 4 割の確率で「ヒット」、6 割の確率で「内野ゴロ」とランダムに表示する:

```
@echo off
set /a x=%random% %% 10
if %x% lss 4 ( echo ヒット ) else echo 内野ゴロ
```

表 1 の 1 つ目、2 つ目の形式では、両辺の値がどちらも整数値を表す文字列であれば整数値として比較が行われる。 値が文字列ならば辞書式順序で比較され、デフォルトではアルファベットの大文字と小文字は区別される。文字の大小関係は、数字、アルファベット大文字、小文字は以下の通り (ASCII コード順ではない)。

$$0 < 9 < \mathrm{a} < \mathrm{A} < \mathrm{z} < \mathrm{Z}$$

また、表 1 の 1 つ目、2 つ目の形式では、if と条件の間に 「/i」を置くとアルファベットの大文字、小文字を区別しない。例えば「`if A==a`」は偽で「`if /i A==a`」は真となる。

if の条件式の注意:

- C 言語とは違い、条件式の値の部分に「`%x%+1 gtr 3`」のように数式を書くことはできない。
- C 言語とは違い、複数の条件を AND (`&&`) や OR (`||`) で結ぶことはできないので、複数の条件での分岐は `if` の入れ子や、4 節の `goto` によるジャンプなどを使う必要がある。例えば、C 言語の「`if (x==1 || y==2)`」と同等の処理は

  ```
  if %x%==1 ( echo 真
  ) else if %y%==2 ( echo 真
  ) else echo 偽
  ```

  のようにすれば可能で、「`if (x==1 && y==2)`」と同等の処理は

  ```
  if %x%==1 (
      if %y%==2 ( echo 真
      ) else echo 偽
  ) else echo 偽
  ```

  のようにして実現できる。
- 文字列の比較は、アルファベットの大文字、小文字が混じった文字列の場合は厳密には辞書式順序ではなく、「$\mathrm{a} < \mathrm{A} < \mathrm{aa} < \mathrm{aA}$」のようになるので、十分テストを行う必要がある。
- 表 1 の 1 つ目、2 つ目の形式では、両辺の一方の値が空文字になるとエラーとなる。よって、両辺の一方が空文字になる可能性がある場合 (特に環境変数やオプ

ション変数などを使う場合) は、両辺をそれぞれ " " で囲んで「if "%1"=="1"」のようにするとよい (" " つきの文字列として比較される)。

- C 言語とは違い、条件部分を「if (%3==1)」のようにかっこつきで書くとエラーになる。

バッチファイルのオプション %1, %2, . . . は、指定されてないもの以降は空文字になるので、オプションがいくつあるかを if 文で判定することができる。例:

```
@echo off
if "%1"=="" ( echo オプションが一つもない
) else if "%2"=="" ( echo オプションは 1 つだけ [%1]
) else echo オプションは 2 つ以上ある [%1][%2]
```

コマンド「exit /b」を使うとバッチファイルを途中で強制終了できる。これを使えば、上のバッチファイルは else なしにもできる (同様のことは次節の goto でも可能):

```
@echo off
if "%1"=="" ( echo オプションが一つもない
    exit /b )
if "%2"=="" ( echo オプションは 1 つだけ [%1]
    exit /b )
echo オプションは 2 つ以上ある [%1][%2]
```

4 goto 命令

goto 命令は、実行の流れをバッチファイル内の任意の箇所へジャンプさせる (表 3)。ラベル行は、バッチファイル内の任意の位置に置くことができる。

命令形式	意味
goto [ラベル名]	「:[ラベル名]」の行へ処理をジャンプ

表 3: goto 命令

goto は C 言語ではほとんど使われないが、ある処理を飛ばすために使ったり、いくつかの処理の中から一つを選んで処理させたり、ループを作るのに使うことができる。

例えば、3 節の注意として書いた C 言語の「if (x==1 || y==2)」に相当するものは、goto で以下のように書くこともできる。

```
if %x%==1 goto runTrue
if %y%==2 goto runTrue
echo 偽
goto nextf
:runTrue
echo 真
:nextf
```

goto 命令と if 文を組み合わせることで簡単なループ (C 言語の for や while) も作れる。例えば

```
set j=1
:start1
if %j% gtr 9 goto last1
set /a x+=j
set /a y+=j*j
set /a j+=2
goto start1
:last1
```

は、「set /a x+=j」と「set /a y+=j*j」までの部分を 1,3,5,7,9 の j の値に対して行うループになっていて、C 言語での「for(j=1; j<=9; j+=2)」のループに相当し、x には $1+3+5+7+9$, y には $1^2+3^2+5^2+7^2+9^2$ が追加される。同様のことは後で紹介する for 文でも行える。

なお、この goto 命令を使うと簡単に無限ループが作れるが、無限ループとなったバッチファイルの実行は、コマンドプロンプトでは Ctrl-C で終了できる。

5 if 文のグループ化に関する注意

1 節で述べたように、if での ( ) 内の変数値は実行前にすべて展開されてから実行されるので、( ) 内で set /a で計算したり、表示させたりする場合は注意が必要。

例えば、変数 n が偶数ならその旨と $n/2$ の値を表示させ、n が奇数なら何も表示しない、という処理は、n が偶数かどうかの判別は n を 2 で割った余りが 0 かどうかでわかるので、

```
set /a m=n %% 2
if %m%==0 ( set /a x=n / 2
    echo n は偶数、%x% )
```

でできるように見えるかもしれないが、echo の部分の %x% は、計算する前の値に置き換えられてしまってうまくいかない。よって、%x% を ( ) の外に出すために if の実行文を 2 つに分けて

```
set /a m=n %% 2
if %m%==0 set /a x=n / 2
if %m%==0 echo n は偶数、%x%
```

のようにする (最初の if 文で x の値を確定し、次の if 文でその値を利用する) か、または処理の部分はかっこを使わず別の場所に書き、その処理をするかどうかを if と goto で分岐するようにして、

```
set /a m=n %% 2
if not %m%==0 goto last1
set /a x=n / 2
echo n は偶数、%x%
:last1
```

のようにする必要がある。なお、%x% をかっこの外に出すためにかっこ内では変数の設定だけをするようにして

```
set /a m=n %% 2
if %m%==0 ( set /a x=n / 2
    set str=n は偶数、)
echo %str%%x%
```

とすると、n が奇数の場合にうまくいかず、結局その echo にも if をつけないといけなくなり、上の 2 つ目のものとほぼ同じになってしまう。

( ) 内の処理を避けたい場合、特に AND や OR のように if の入れ子が必要で表示を伴う場合や、変数値を整数として使うのではなく文字列として使う場合 (/a なしの set の右辺では変数は % % で囲む必要がある) には、この goto の用法は有用となる。ただし、goto を多用すると処理の流れがわかりにくくなるので注意が必要。

なお、( ) 内で「%[変数名]%」の展開を事前にやらずに ( ) 内の処理と平行して行う「遅延展開」という方法もある。これについては後で説明する。