2012 年 05 月 11 日

計算機実習 IV (2012 年度)
第 4 回: コマンドプロンプトとバッチファイル その 4

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

目 次

1 リダイレクション

コマンドプロンプトで起動する非ウィンドウプログラムからの画面出力、またはキーボードからプログラムへの入力は、それぞれ

- **標準出力** (stdout: 通常はコマンドプロンプト画面につながっている)
- **標準入力** (stdin: 通常はキーボード入力につながっている)

を介して行われている。C 言語で言えば、printf() や putchar() 等の関数は標準出力への出力を行い、scanf() や getchar() 等は標準入力から入力をもらう関数である。

このつながりは、**リダイレクション** 機能でそれぞれファイルにつなぎかえることができる。

- > [ファイル]
  プログラムから標準出力 (通常画面) に出力されるものを、[ファイル] に出力する。[ファイル] がなければ新規に作成され、[ファイル] があれば上書きされる (出力リダイレクション)。

- < [ファイル]
  標準入力 (通常キーボード) からプログラムへ入力するものを、[ファイル] から入力する (入力リダイレクション)。

- >> [ファイル]
  プログラムから標準出力 (通常画面) に出力されるものを、[ファイル] に出力する。[ファイル] がなければ新規に作成され、[ファイル] があればその末尾に追加出力される (追加出力リダイレクション)。

これは、「< [ファイル 1] > [ファイル 2]」のように入出力の両方をファイルにつなぎかえることもできる。

これらのリダイレクションは、いずれもコマンド行の最後に書き加える。例えば、

```
Z:> time /t > time1.dat
```

とすれば現在の時刻が time1.dat に保存される。ただ、実際は time /t の出力よりも、動的環境変数 %time% の方が細かい時刻を持っているので、

```
Z:> echo %time% > time1.dat
```

の方が時刻の保存にはよいだろう。

```
Z:> echo %time% >> time1.dat
```

のように追加出力にすれば、ログイン時刻の記録を残すことなどにも使える。

また、バッチファイル中で

```
echo abc > tmpf
set /p x= < tmpf
```

とすると、これは「abc」(と改行) のみの 1 行のファイル tmpf を作成し、それを変数 x に代入して先に進む。このようにすれば、echo でなくても任意のコマンドの画面出力を一時ファイルを経由することで環境変数の値に代入できる。

特殊ファイル nul は、出力リダイレクションに与えてもファイルを残さないので、プログラムの画面出力を消すのに使うことができる。

```
@echo off
echo 適当なキーを打て
pause > nul
```

これは実質的には pause と同じであるが、そのメッセージを変更していることになる。

課題 4-1. ログイン日とログイン時刻を保存するバッチファイル kadai4-1.bat を作成せよ。

課題 4-2. 1 行目に 1、2 行目に 2、3 行目に 3 とだけ書いたファイル file1 を作り、それを「set /p x= <」に与えることで変数 x に何が代入されるか確認せよ。

課題 4-3. for /l とリダイレクションを用いて、1 行目に 1、2 行目に 2、. . .、100 行目に 100 と書かれた 100 行のファイル file.dat を作るバッチファイル kadai4-3.bat を作成せよ。

課題 4-4. for /l とリダイレクションを用いて、1 が 1 行だけ書かれたファイル file1.dat、2 が 1 行だけ書かれたファイル file2.dat、. . .、10 が 1 行だけ書かれたファイル file10.dat の 10 個のファイルを作成するバッチファイル kadai4-4.bat を作成せよ。

課題 4-5. for /l と set /p, set /a を用いて、file1.dat, file2.dat, . . ., file10.dat に 1 行ずつ書かれた数字を全部加えた値を出力するバッチファイル kadai4-5.bat を作成せよ。

2 パイプ

あるコマンド (comm1 とする) の出力を、別のコマンド (comm2 とする) の入力に切り変えたい場合、入出力リダイレクションを用いて

```
Z:> comm1 > tmpf
Z:> comm2 < tmpf
```

とすればできるが、これを 1 行で、かつ中間ファイル (上の tmpf) を作らずに行う仕組みが **パイプ** である。パイプは、記号 | を用いて、以下のように行う。

```
Z:> comm1 | comm2
```

これで、comm1, comm2 が実行されて、comm1 の標準出力への出力データが、そのまま comm2 の標準入力へと渡される。

なお、「set /p」には「echo abc | set /p x=」のようにしてパイプで値を渡すことはできないようである[1]。

パイプは多段階につなぐことも可能で、例えば

```
Z:> comm1 | comm2 | comm3
```

のようにすれば、comm2 の出力がさらに comm3 の入力として渡されることになる。もちろん、これとリダイレクションを組み合せることも可能であり、例えば、

```
Z:> comm1 < f1.dat | comm2 | comm3 > f2.dat
```

とすると、f1.dat に対する comm1 の処理の結果が comm2 に、その処理の結果が comm3 に渡され、その結果が f2.dat として保存されることになる。

標準入力からデータをもらって、標準出力にその処理結果を書き出すソフトを **フィルタ** と呼ぶが、パイプはフィルタを組み合わせてデータを加工するのに向いている。一つ一つのフィルタが単一の機能しかなくても、複数のフィルタを組み合わせることで、流れ作業的にデータの複雑な加工処理が行える。

パイプに関連して良く用いられるコマンドやフィルタには、以下のようなものがある。

- type : 引数として指定したテキストファイル (複数も可) の中身を順に標準出力に流す
- more : 引数として指定したテキストファイル (省略した場合は標準入力) の中身を 1 画面ごとに停止しながら表示
- sort : 引数として指定したテキストファイル (省略した場合は標準入力) の中身を辞書順にソートして標準出力に流す。/r オプションで逆順のソートとなる。
- find "文字列" : 引数として指定したテキストファイル (複数も可、省略した場合は標準入力) から、指定した文字列の含まれる行を抜き出して標準出力に流す。/v オプションで含まない行の方を出力、/c でマッチする行数のみを出力、/n で行番号も出力、/i で大文字小文字の区別をせずに検索する[2]。

[1] おそらく、set がコマンドプロンプトの内部コマンドであるためではないかと思う。
[2] これらのオプションは、find と "文字列" の間に書く。

- `fc` : 引数として指定した 2 つのテキストファイルの違いを標準出力に流す

例えば、長いヘルプメッセージの場合、`help if` のように 1 画面ずつ表示してくれるものもあるが、`help sort` のように一気に画面に流してしまって前の方が読めない場合がある。そのような場合に、

```
Z:> help sort | more
```

のようにすると `more` が 1 画面ずつ表示してくれる。

課題 4-6. `sort /?` のヘルプを `more` とパイプを用いて 1 画面ずつ読め。

課題 4-7. `for /l` と追加出力リダイレクションを用いて、1 行ずつ各行に「2012 01/01」から「2012 12/31」までが書かれた (366 行の) ファイル `data1.dat` を作成するバッチファイル kadai4-7.bat を作成し、実際にその `data1.dat` を作成せよ。

課題 4-8. `data1.dat` から日付に 3 がつく行のみを `find` で取り出し、それをパイプと `more` を用いて 1 画面ずつ読め。

課題 4-9. `data1.dat` から日付に 3 がつく行のみを `find` で取り出し、それをパイプと `sort /r` を用いて逆順に並べ、それを `data3.dat` にリダイレクトせよ。

課題 4-10. `dir c:¥windows¥fonts` の出力から、`find /i` とパイプを用いて、“times” という名前が含まれるフォントファイルの出力のみを抜き出せ。

3 call とサブルーチン

goto 命令は、コマンドの実行をある場所へ移動するものだったが、一旦別なところにジャンプして、また元の場所に戻りたい場合がある。その場合、飛ぶ先を普通は **サブルーチン** と呼ぶが、`call` コマンドを使うとジャンプ位置へサブルーチンコールできる[3]。

[3]すなわち、サブルーチンへジャンプし、そちらの処理が終わったら元のところに戻る。

call コマンドにはオプションを与えることもでき、その場合、サブルーチン側ではそのオプションを %1, %2 のような名前で利用できるが、これはバッチファイル自体のオプションとは別物になる。

- call :[ラベル名]
- call :[ラベル名] [オプション] ...

サブルーチン側の処理を終了するには exit /b を使う。つまり、サブルーチン部分は独立した別のバッチファイルのようなものになる。なお、goto とは違い、call のサブルーチンコールではラベル名の前に ':' をつけなければいけない[4]。

例えば

```
@echo off
set x=6
set y=8
call :sum %x% %y%
echo %x% + %y% = %z%
exit/b

:sum
set /a z=%1+%2
exit /b
```

のようなバッチファイルを実行すると、「$6 + 8 = 14$」が表示される。4 行目の call で :sum の場所へジャンプして、exit /b によって 5 行目に戻り、6 行目の exit /b でバッチファイルが終了する。

繰り返し使う部分などをサブルーチンにすると便利だが、C 言語の関数とは違い、値を直接返すことはできないことに注意する[5]。

課題 4-11. 上記にならい、引数 3 つを加えた値を total という名前の変数に代入して元に戻るようなサブルーチンを作成せよ。

課題 4-12. 指定された引数すべての値を加えた値を total という名前の変数に代入して元に戻るようなサブルーチンを作成せよ。ただし、与える引数の個数は、6 個以下 (不定) とする。

[4] 後述の call による子バッチの呼び出しと区別するため。

[5] exit /b のオプションを使って errorlevel 変数に返すこともできなくはないが、上の環境変数への代入とそう違わない。

課題 4-13. 1 以上 12 以下の 1 つの引数を取り、その月の日数を days という変数に代入して元に戻るようなサブルーチンを作成せよ。

元々、call コマンドはバッチファイル内から別のバッチファイル (子バッチ) を呼びだすのに使うもので、「call [バッチファイル名]」として呼びだすと、そのバッチファイルを実行して、それが終わったらこの call の次の行に戻る。

4 shift

バッチファイルの実行時に渡したオプションは、バッチファイル内部で %1, %2, ... のように参照できるが、一度に参照できるのは %9 までであり、%10 という変数を直接使うことはできない[6]。それより後の引数を参照したり、すべてのオプションに渡る処理を順にしたい場合には、「for %%a in (%*)」を用いる方法と「shift」を用いる方法がある。

- shift : %[n] をそれぞれ %[n-1] にシフトする
- shift /[番号] : 指定番号以降のみシフトする

%0 も %1 だったものに変わるので、%0 を変更したくなければ shift /1 とすればよい。なお、引数全体の文字列 %* は shift をしても変わらない。

これと if, goto によるループを組み合わせることで、すべてのオプションに渡る処理を順に行うことができる。例えば、以下の 2 つのバッチファイルはほぼ同等で、いずれもバッチファイルに指定したすべてのオプションを 1 行ずつ表示する。

```
@echo off
for %%a in ( %* ) do (
    echo [%%a]
)
```

```
@echo off
:startlabel
if "%1"=="" exit /b
echo [%1]
shift
goto startlabel
```

[6] 試してみればわかるが、%1 に文字 0 がついているとみなされる。

ただし、for のリスト処理にはワイルドカードの展開処理が含まれるので、このバッチの引数としてワイルドカードを含む文字列が指定された場合、例えば * を指定すると、前者はカレントディレクトリのファイルすべてに対して 1 回ずつ for 文が実行されることになるが、後者は単に一つの文字列 [*] のみが表示される。

前者の方が簡単そうであるが、ワイルドカード文字をファイル名に展開したくない場合や、for 文の実行ブロックでは実行できない処理[7]をしたい場合などでは shift と if, goto の組み合わせの方がいいだろう。

なお、例えば shift /1 を行うと元の %1 を復帰させることはできないが、%* は shift では変わらないので、サブルーチンを利用すれば再び %1 を使うことができる。例えば、

```
@echo off
echo [main:  %1]
shift
echo [main:  %1]
call :sub1 %*
exit /b

:sub1
echo [sub:  %1]
shift
echo [sub:  %1]
exit /b
```

というバッチファイルに「1 2」というオプションをつけて実行すると、

```
[main:  1]
[main:  2]
[sub:  1]
[sub:  2]
```

と表示される。逆に、shift を使う処理はサブルーチンの方で行い、元の方では shift を使わないようにすることで %1 などが変わらないようにするという方法もある。

課題 4-14. 与えられたすべての引数の和の値を表示するバッチファイル

[7]例えば for 文の ( ) ブロック内では文字 ')' を echo できないし、for 文のところで説明したように環境変数の更新に制限がある。

kadai4-14.bat を `shift` を用いて作成せよ。ただし、引数がない場合は使い方 (ヘルプ) を表示するようにせよ。

課題 4-15. 与えられた引数がすべてファイル名であるとして、それら一つ一つが存在するか確認して、存在すれば trash というディレクトリに移動し、存在しなければそのファイルは存在しない、というメッセージを表示するバッチファイル kadai4-15.bat を作成せよ。

課題 4-16. `shift` と `if`, `goto` を組み合わせて、バッチファイルに与えた引数の個数を表示するバッチファイル kadai4-16.bat を作成せよ。

課題 4-17. `shift` と `if`, `goto` を組み合わせて、バッチファイルに与えた引数の個数を `args` という環境変数に代入するサブルーチンを作成せよ。

コラム: テキストファイルとバイナリファイル

「メモ帳」で編集できるような、可読文字がただ並んでいるファイルを**テキストファイル** と呼び、それ以外のものを **バイナリファイル** と呼ぶ。C のコンパイル前のプログラムファイル (ソースファイルという) はテキストファイルであるが、ワープロファイル (.doc) や表計算データ (.xls) はバイナリファイルである。

テキストファイルもバイナリファイルも、バイト (0x00–0xFF) のデータが並んでいるだけ、ということでは同じだが、ASCII コードの可読文字である 0x20–0x7E の英数半角記号、および改行記号 (0x0A) 等の印字用制御文字だけで構成されるものを狭義にはテキストファイルと呼ぶ。しかしこれだと日本語文字が入らないので、日本語コード (通常 2 バイトコード) などのある特定の文字コードが入ったものも広義にはテキストファイルと読んでいる。

“実”		“習”		(空白)	4	‘¥r’	‘¥n’
0x8e	0xc0	0x8f	0x4b	0x20	0x34	0x0d	0x0a

テキストデータのサンプル

一方、バイナリファイルは文字を表さないバイトが並ぶので、普通のエディタで表示させることはできないか、表示させても意味のないデタラメなものが並んで見えることになる。画像データ (.gif, .png, .jpg)、音声データ (.wav,

.mp3)、圧縮ファイル (.zip)、ワープロ文書データ (.doc, .rtf, .ppt)、表計算データ (.xls) などはバイナリファイルである。

0xff	0xd8	0xff	0xe0	0x00	0x10	0x4a	0x46

バイナリデータのサンプル (JPEG ファイルの先頭)

プログラムソースファイルはテキストファイルであるが、コンパイルして実行できるようになったプログラム実行ファイル (.exe) はバイナリファイルである。しかし、インタプリタ言語ではコンパイルをせずにソースファイルをそのままインタプリタに実行させるので、バイナリファイルは作らない。

ワープロ文書のような整形文書ファイル形式でも、LaTeX ソースファイルや HTML ファイルはマークアップ形式のテキストファイルであり、 表計算データも CSV 形式やタブ区切り形式などの形でテキストファイルとして保存できる。

バイナリ形式の利点は、

- 文字ではないデータを保存する場合はバイナリの方が小さくて済む (0x1234 という数値は、バイナリなら 0x12 と 0x34 に分けて表現すれば 2 バイトで済むがテキストだと '0', 'x', '1', '2', '3', '4' の 6 バイト必要)
- 専用のソフトでしか参照できないので簡単に見られることがないし、他のソフトで使われることがない

テキスト形式の利点は、

- データを容易に参照、検索ができる
- データを処理するのに専用のソフトは必要ないので、多くのソフトで共有したり色々な処理や使い回しが可能

等であろうと思う。

バイナリデータだと処理をするのに MS–Excel などの特定のソフトの上で、そしてそれに従った操作でないと処理はできないが、テキストデータならば MS–Excel などを使わなくても AWK 等のスクリプト言語で複雑な処理が可能だし、しかもその方が作業は普通は早く、MS–Excel が起動するのを待つ間に仕事が終わってしまうこともあるだろう。簡単なプログラミングができる人には、多分テキストデータを簡単なスクリプトで処理する方がずっと楽なことが多いだろうと思う。