(平成 26 年 12 月 1 日版)

情報電子工学演習 II・工学基礎

バッチプログラミングによるタイプ練習ソフトの作成

1 目的

現在どのバージョンの MS–Windows でもほぼ動作するコマンドプロンプト上のバッチファイルを利用して、簡単なスクリプトプログラミングを体験する。if と goto 程度の構文を利用し、set コマンドを利用した整数計算や文字列処理などを学ぶ。

2 スケジュール

- 第 1 回 (12/03、13:30–) (計算機実習室)
  ガイダンス、エディタの使い方、バッチファイルの実行の仕方、サンプルバッチファイルの実行テスト
- 第 2 回 (12/10) (計算機実習室)
  構文とコマンド解説 no.1、テストプログラムの作成と実行テスト

  [次回までの課題 1: 作成するものの概要 (スケッチ) の作成]
- 第 3 回 (12/17) (計算機実習室)
  課題 1 の確認、構文とコマンド解説 no.2、テストプログラムの作成と実行テスト
- 第 4 回 (12/24) (計算機実習室)
  いくつかのサンプルコードの解説、最終課題のプログラミング

  [次回までの課題 2: 最終課題のフローチャートの作成]
- 第 5 回 (01/07、13:30–) (計算機実習室)
  課題 2 の確認、最終課題のプログラミング、発表準備
- 第 6 回 (01/14) (計算機実習室)
  発表会 (一人 5 分程度)

講義に関する情報は、以下の Web ページに随時追加する。

`http://takeno.iee.niit.ac.jp/~shige/math/lecture/excsiee/batch2.html`

3 バッチプログラミング

3.1 コマンドプロンプト

コマンドプロンプト は、主に文字をキーボードで入力してコンピュータを操作する形式のコマンド (命令) 実行環境[1]。

コマンドプロンプトは、この工学演習 2 では起動画面にある「計算機実習 IV」のアイコンをクリックして起動する。

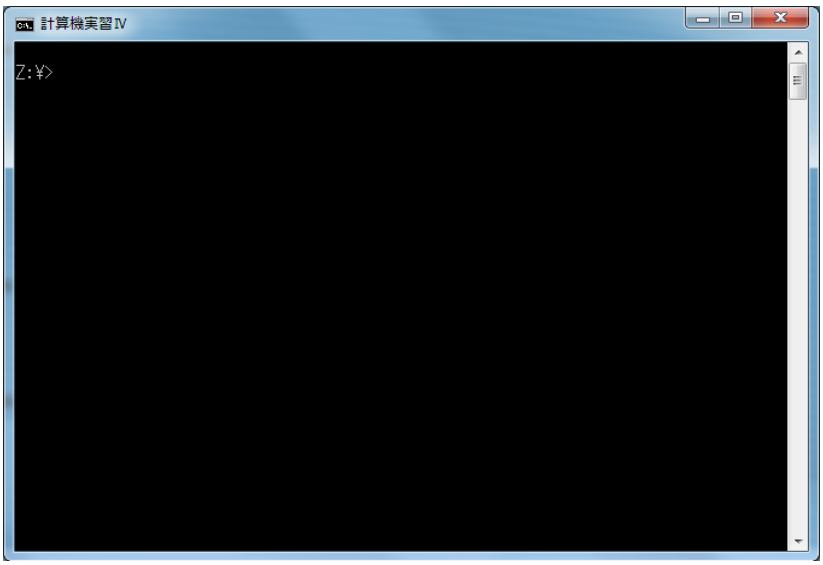


図 1: 計算機実習 IV のコマンドプロンプトの起動画面

コマンドプロンプトの「`Z:¥>`」が **プロンプト** で、この右側にコンピュータへの命令をキーボードで入力する。「`Z:¥`」は、ユーザの現在の居場所を表している。「`Z:`」は **ドライブ名**、「`¥`」は **ディレクトリ名**[2]。

- ヘルプ
  基本コマンドなどの説明 (ヘルプメッセージ) は、例えば `dir` というコマンドであれば、

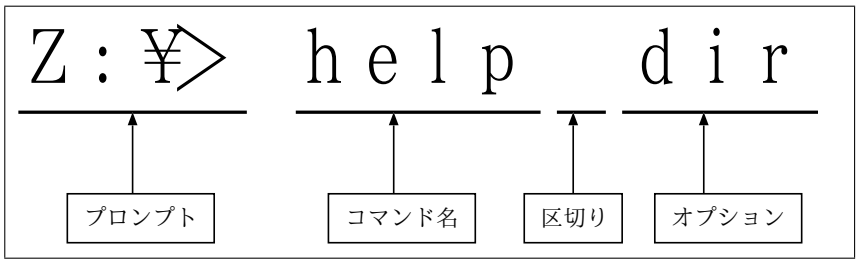
  ```
  Z:¥> help dir
  Z:¥> dir /?
  ```

  のように、`help` コマンドを使うか、`/?` オプションで表示できる。

[1] プチ解説: 「コマンド」= 命令、「プロンプト」= 入力待ち。
[2] プチ解説: 「ディレクトリ」= フォルダ。

解説

コマンドの後ろにスペースを開けてコマンド用の文字列を追加して書いたものを **オプション**、**オプションパラメータ** などと呼ぶ。オプションは、原則スペースで区切って書き並べる。



「コマンド名」= ソフトウェアの実体、「オプション」= そのソフトウェアへの指示

- 履歴
  過去に入力したコマンドは上下の矢印キーで呼び出すことができる。左右の矢印キーで入力を修正することもできる。

3.2 エディタの使い方

バッチファイルは可読文字だけからなる **テキストファイル** なので、**エディタ** と総称されるテキストファイル編集ソフトウェアで作成する。otbedit はタブ機能付きのフリーソフトのエディタ。

- OTBEdit (A.Ogawa)
  `http://www.hi-ho.ne.jp/a_ogawa/otbedit/index.htm`

各プログラム言語のキーワードの色付け/カスタマイズ機能、改行文字や全角空白の表示、フォントの設定などの機能がついていて、「メモ帳」よりはプログラミングに向いている。

otbedit のコマンド名は `otbedit`。

- 便利なショートカット:
  `Ctrl-N` (新しいファイル)、`Ctrl-O` (ファイルを開く)、`Ctrl-S` (ファイルを保存)

3.3 バッチファイルの実行

バッチファイル は、コマンドプロンプトで実行できるコマンドを一行ずつ並べて書いたファイル (拡張子 `.bat`)。

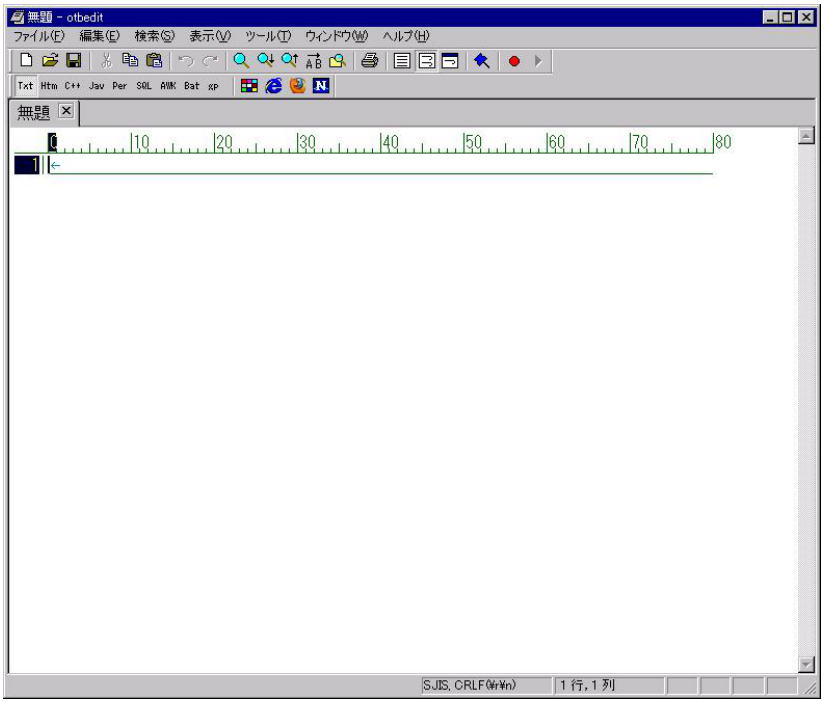


図 2: otbedit の起動画面

otbedit では、.bat の拡張子のファイルを開くか、画面の左上にある Bat というアイコンでバッチファイルモードになる。

例えば以下のバッチファイルは、ディレクトリ内のファイルの一覧を表示し (dir)、日付 (date /t) と時刻 (time /t) を表示するもの。

```
@echo off
rem バッチファイルの簡単なサンプル
dir
date /t
time /t
```

1 行目の「@echo off」は、コマンド自体の画面表示を消すための命令、2 行目の「rem」で始まる行は無視される (コメント行)。

- バッチファイルの実行方法:
  1. エクスプローラーでそのバッチファイルのアイコンをダブルクリック (マウス)
  2. コマンドプロンプト上でバッチファイル名を入力 (キーボード)

ただし、1. の方法は終了した後すぐにコマンドプロンプトウィンドウが閉じられるので、バッチファイルの最後に pause コマンド (3.5 節参照) などがないと何が表示されたかわから

なくなる。

3.4 サンプルバッチファイル

この講義の Web ページ

http://takeno.iee.niit.ac.jp/~shige/math/lecture/excsiee/batch2.html

に置いてあるサンプルバッチファイル proto1.bat のソースコード[3](詳しくは 3.11 節で解説):

```
1  @echo off
2  rem バッチファイルによるタイプ練習ソフトのプロトタイプ
3  rem shige 07/12 2013
4
5  rem ##### メニュー #####
6  :menu1
7  cls
8  color 70
9  echo.
10 echo 【タイプ練習ソフトプロトタイプ】（07/12 2013 by shige）
11 echo.
12 echo 1） 課題１：数字の入力
13 echo.
14 echo 0） 終了
15 echo.
16 :wait0
17 set /p select=   1, 0 のいずれかを入力してください。
18 if "%select%"=="0" exit /b
19 if "%select%"=="1" goto kadai1
20 goto wait0
21
22 :kadai1
23 set string1=2468097531
24 goto run1
25
26 rem ##### タイプ練習の実行部分 #####
27 :run1
28 cls
```

[3]プチ解説: 「ソースファイル」= プログラムファイル、「コード」= プログラムファイルの各命令や各行。

```
29 color 07
30 pause
31
32 echo.
33 echo 目標文字列：%string1%
34 set time1=%time%
35 set /p string2=回答文字列：
36 set time2=%time%
37
38 rem ##### 集計 #####
39 call :gettime
40 call :diffstr
41 echo.
42 echo かかった時間 = [%sec%.%csec%] 秒 (%time2% - %time1%)
43 echo 正答文字数 = [%tcount%]、誤答文字数 = [%ncount%]
44 echo.
45 pause
46
47 goto menu1
48
49 rem ########## サブルーチン ##########
50
51 rem ##### 文字列比較サブルーチン #####
52 :diffstr
53 rem 正解文字列 string1 と回答文字列 string2 の文字を比較して
54 rem 一致数、不一致数を tcount, ncount に保存して帰る
55 rem [call :diffstr] と呼び出すこと
56
57 set tmps1=X%string1%
58 set tmps2=X%string2%
59 set tcount=0
60 set ncount=0
61 :startdiff
62 rem tmps1, tmps2 から 1 文字取り出す
63 if "%tmps1%"=="X" goto enddiff
64 set tmps3=%tmps1:~1,1%
65 set tmps1=X%tmps1:~2%
66
67 set tmps4=%tmps2:~1,1%
68 set tmps2=X%tmps2:~2%
```

```
69
70 if "%tmps3%"=="%tmps4%" (
71   set /a tcount+=1
72 ) else (
73   set /a ncount+=1
74 )
75 goto startdiff
76 :enddiff
77 exit/b
78
79 rem ##### 時間差計算サブルーチン #####
80 :gettime
81 rem %time2% - %time1% の秒数を計算して（それぞれ %time% の値）
82 rem %sec%（整数部分），%csec%（センチ秒部分）に保存して帰る
83 rem [call :gettime] と呼び出すこと
84
85 set /a tmpsec1=%time1:~0,2%*3600*100
86 set /a tmpsec1=tmpsec1 + (1%time1:~3,2% - 100)*60*100
87 set /a tmpsec1=tmpsec1 + (1%time1:~6,2% - 100)*100
88 set /a tmpsec1=tmpsec1 + (1%time1:~9,2% - 100)
89
90 set /a tmpsec2=%time2:~0,2%*3600*100
91 set /a tmpsec2=tmpsec2 + (1%time2:~3,2% - 100)*60*100
92 set /a tmpsec2=tmpsec2 + (1%time2:~6,2% - 100)*100
93 set /a tmpsec2=tmpsec2 + (1%time2:~9,2% - 100)
94
95 set /a sec=(tmpsec2 - tmpsec1)/100
96 set /a csec=(tmpsec2 - tmpsec1)%%100
97 exit /b
```

3.5 基本コマンド

今回の演習で使えそうな基本コマンドは、表 1 の通り。

echo がオンのときでも、コマンドの前に @ があるとそのコマンド行自体は表示しない[4]。

color コマンドの [色] は 2 桁の 16 進数を表す 2 文字 (0x はつけない)。上の桁が背景色の色番号、下の桁が文字色の色番号。色番号と色の対応は表 2 参照。

[4] プチ解説: @ がついているとコマンドは表示しないが、コマンドの出力結果は @ があっても表示する。

コマンド	説明
cls	コマンドプロンプト画面のクリア
echo off	コマンド行自体の表示をオフにする
echo on	コマンド行自体の表示をオンにする
echo.	空行を表示
echo [文字列]	[文字列] をコマンドプロンプト画面に表示
rem	その行は実行を無視する (コメント行として使える)
color [色]	コマンドプロンプト画面の色指定
pause [文字列]	[文字列] を表示して一旦停止 (入力待ち)
set	変数操作 (3.6 節で解説)

表 1: 今回の課題用の基本コマンド

色	黒	青	緑	水色	赤	紫	黄	白	備考
色番号	0	1	2	3	4	5	6	7	暗い
	8	9	a	b	c	d	e	f	明るい

表 2: color コマンドの色番号と色の対応

例えば

```
Z:¥> color 6a
```

とすると背景が暗色の黄色 (6) で、文字色が明色の緑 (a) になる。

3.6 set コマンド

set コマンドは環境変数を表示、設定、削除する基本コマンド。**環境変数** はコマンドプロンプト、バッチファイルで利用できる変数。

コマンド	説明
set	現在の環境変数の一覧を表示
set [変数名]=[値]	[変数名] の環境変数に [値] を設定
set [変数名]=	[変数名] の環境変数を削除
set /p [変数名]=[文字列]	キーボードから環境変数値を入力
set /a [変数名]=[式]	[式] の結果の数値を環境変数に代入

表 3: set の使い方

注:

1. 環境変数はシステムが利用する重要なものもあるのでその名前を使わないようにすること (一覧表示で確認せよ)。
2. 「=」の前後にスペースを入れないこと。

設定した環境変数は「%[環境変数名]%」で利用する (例: サンプル 33 行目)。例えば、

```
Z:¥> set a=竹の
Z:¥> set b=茂治
Z:¥> set c=%a% %b%
Z:¥> echo %c%
```

とすると変数 c の値は「竹の 茂治」となり、その結果が最後の echo コマンドで表示される。

set /p は、[文字列] を表示して入力待ちになり、キーボードから入力された文字列 (Enter の前まで) を指定した環境変数に代入する (例: サンプル 17 行目)。

環境変数の値は基本的には文字列であるが、set /a では数値としても利用できる (詳しくは 3.8 節で説明)。

3.7 動的環境変数

環境変数には、表 4 などの、呼び出した際に自動的に値が設定される **動的な環境変数** がある (例: サンプル 34 行目)。

環境変数	意味
%date%	現在の日付 (例: 2010/06/26)
%time%	現在の時刻 (例: 16:48:54.26)
%random%	0 から 32767 の間の乱数

表 4: 工学演習で使いそうな動的環境変数

3.8 環境変数値の数値計算

環境変数値の整数計算は 「set /a [変数名]=[式]」で行える。この場合、= の右辺には整数計算の数式を書くことができ、その数式の計算結果の数値 (を意味する文字列) が環境変数に代入される。

右辺の数式には、整数値、環境変数名 (この場合は % で囲まなくてよい)、かっこ (丸かっこのみ) 以外に、表 5 の演算記号が使える[5] (例: サンプル 95 行目)。数値は通常は 10 進数であるが、先頭が 0x の場合は 16 進数、先頭が 0 の場合は 8 進数とみなされる。

単項演算子		
記号	意味	例など
!	!a = a が 0 以外なら 0、a が 0 なら 1	!11=0
~	~a = a のビット毎の反転	$-a-1$ と同じ
-	-a = $-a$ (符号の反転)	
四則演算		
記号	意味	例など
*	a*b = a と b の積	14*5 (=70)
/	a/b = a を b で割った商	14/5 (=2)
%	a%b = a を b で割った余り	14%5 (=4)
+	a+b = a と b の和	14+5 (=19)
-	a-b = a と b の差	14-5 (=9)
ビット演算 (引用符で囲む必要あり)		
記号	意味	例など
<<	"a<<b" = a を 2^b 倍	"14<<5" (=448)
>>	"a>>b" = a を 2^b で割った商	"14>>5" (=0)
&	"a&b" = a と b のビット毎の AND	"14&5" (=4)
|	"a|b" = a と b のビット毎の OR	"14|5" (=15)
^	"a^b" = a と b のビット毎の XOR	"14^5" (=11)
演算付き代入式 (= を置き換える)		
記号	意味	例など
*=	「a*=b」は「a=a*b」と同じ	
/=	「a/=b」は「a=a/b」と同じ	
%=	「a%=b」は「a=a%b」と同じ	
+=	「a+=b」は「a=a+b」と同じ	
-=	「a-=b」は「a=a-b」と同じ	
<<=	「a<<=b」は「a=a<<b」と同じ	
>>=	「a>>=b」は「a=a>>b」と同じ	
&=	「a&=b」は「a=a&b」と同じ	
|=	「a|=b」は「a=a|b」と同じ	
^=	「a^=b」は「a=a^b」と同じ	

表 5: set /a の右辺で使える演算記号

[5]ほぼ C 言語同様である。

「演算つき代入式」は、[変数名] の次の「=」の部分を置き換えるもので、例えば「`set /a x+=3`」は「`set /a x=x+3`」と同じ意味になる (例: サンプル 71 行目)。

例えば、

```
Z:¥> set x=10
Z:¥> set y=4
Z:¥> set /a y+=3*x%7
Z:¥> echo %y%
```

とすると、4 に 3×10 を 7 で割った余り (=2) を加えた 6 が表示され、

```
Z:¥> set /a r=%random% % 3
Z:¥> echo %r%
```

とすれば、0 か 1 か 2 がランダムに出力される。

注:

1. 余りの `%` は、バッチファイル内では `%%` と書く必要がある (例: サンプル 96 行目)。
2. `set` で変数に小数値を「文字列」として設定することはできるが、`set /a` では小数計算は行えない。

3.9 環境変数の文字列処理

環境変数から文字列の一部分を取得したり、一部分の変更ができる (表 6)。

式	意味
`%var%`	環境変数 `var` の文字列全体
`%var:~[m]%`	$[m]$ 文字目から最後までの部分文字列 (例: サンプル 65 行目)
`%var:~[m],[n]%`	$[m]$ 文字目から $[n]$ 文字分の部分文字列 (例: サンプル 64 行目)
`%var:[s1]=[s2]%`	`var` 内の [s1] をすべて [s2] に変更した文字列

表 6: 環境変数の文字列処理

$[m]$ は、文字列の先頭を「0 番目」と数えることに注意。$[m]$ に負の値を指定した場合は、文字列の先頭からではなく後ろから数える (一番後ろの文字が (-1) 番目)。また、$[n]$ として負の値を指定すると、最後の $[n]$ 文字を削除したものとなる。

例えば、環境変数 `var` が「`1234567890123`」という文字列の場合、それぞれ表 7 のようになる。

環境変数式 = 結果文字列	環境変数式 = 結果文字列
%var:~5% = 67890123	%var:~-5,2% = 90
%var:~5,2% = 67	%var:~-5,-2% = 901
%var:~5,-2% = 678901	%var:123=aBc% = aBc4567890aBc
%var:~-5% = 90123	%var:123=% = 4567890

表 7: 「%var%=1234567890123」の場合

3.10 バッチファイルの構文

3.10.1 if

if コマンドは条件分岐で使用し、以下の 4 種類の形式が使用できる。

1. 1 行で if のみ

```
if [条件] [コマンド]
```

[条件] が成立する場合のみ [コマンド] が実行され、成立しなければ何も実行しない。

2. 1 行で if と else

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

[条件] が成立すれば [コマンド 1] が実行され、成立しなければ [コマンド 2] が実行される。

実行する命令が複数ある場合は、3., 4. のように ( ) を使って書くことができる (例: サンプル 70–74 行目)。

3. 複数行で if のみ

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

4. 複数行で if と else

```
if [条件] (
    [コマンド A1]
    [コマンド A2]
    ............
) else (
    [コマンド B1]
    [コマンド B2]
    ............
)
```

注: C 言語とは違い、else の前で改行しては**いけない**。

if 文の条件部分に使用する主な形式は表 8 の通り。

条件式	意味
[値 1]==[値 2]	[値 1] = [値 2] (等しい)
[値 1] equ [値 2]	[値 1] = [値 2] (等しい)
[値 1] neq [値 2]	[値 1] ≠ [値 2] (等しくない)
[値 1] leq [値 2]	[値 1] ≤ [値 2] (以下)
[値 1] geq [値 2]	[値 1] ≥ [値 2] (以上)
[値 1] lss [値 2]	[値 1] < [値 2] (より小さい)
[値 1] gtr [値 2]	[値 1] > [値 2] (より大きい)
defined [変数名]	その環境変数が定義済み

表 8: if の条件式

表 8 の条件式の前に not をつけるといずれの場合も成立、不成立が反対の意味になる。例えば、「if not %a%==3」は、「if %a% neq 3」と同じ意味になる。

両辺の値は、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。値が数値以外の文字列ならば辞書式順序で比較され、アルファベットの大文字と小文字は区別される。辞書式順序では、数字とアルファベットの大小は以下の通り。

$$0 < 9 < A < Z < a < z$$

3.10.2 goto

goto 命令は、

```
goto [ラベル名]
```

の形式で用いて、そのバッチファイル内のラベルの場所へジャンプする (例: サンプル 19,20 行目)。ラベルは、

```
:[ラベル名]
```

の形の行で、どの場所にも置ける (例: サンプル 6,16,22 行目)。例えば、

```
@echo off
set j=1
:start1
if %j% gtr 10 goto last1
echo [%j%]
set /a j+=2
goto start1
:last1
```

というバッチファイルは、[1], [3], [5], [7], [9] を順に表示する。

3.10.3 exit

バッチファイルは最後の行の処理が終われば自動的に終了するが、途中で強制終了したい場合は、

```
exit /b
```

とする[6]。

この `exit /b` は、次の `call` で呼び出されるサブルーチンでも利用する。

3.10.4 call

バッチファイル内でいくつかのところで共通に使うものを**サブルーチン** という形で利用することができる。サブルーチンを利用する際は、

```
call :[ラベル名]
```

の形式でサブルーチンのあるラベル位置へジャンプする (例: サンプル 39,40 行目)。サブルーチン側で `exit /b` にたどりつくと (例: サンプル 77,97 行目)、`call` したところに戻り、その次の行に進む。

[6] プチ解説: この `/b` を取るとコマンドプロンプト自体も終了してしまう。

```
@echo off
set a=Dog
set b=Cat
echo [Gnu]
call :sub1
echo [Python]
exit /b

:sub1
echo [%a%]
exit /b
```

を実行すると、まず [Gnu] が表示され、:sub1 の呼び出しで [Dog] が表示されて、exit /b で元の場所に戻って [Python] が表示されて、サブルーチンではない exit /b で終了する。

3.11 サンプルプログラムの解説

3.4 節のサンプルバッチファイルを簡単に解説する。おおまかなフローチャートは、図 3 の通り。

- 5 行目から 20 行目までは起動時メニューの表示。cls で画面をクリアしてから color で色を変えて echo で表示を行っている。echo. は空行の表示。18, 19 行目は、キーで入力した文字を取得して条件判断している。

  なお、18, 19 行目の条件式の両辺を「%select%==0」のように " " で囲まないと、select の値が空文字列の場合にエラーになってしまう。

- 22 行目から 24 行目は「課題 1」の課題文字列を環境変数 string1 に保存している。

- 26 行目から 36 行目までが実際のタイプ練習の実行部分。cls と color で表示を変え、課題に進む前に pause で一旦停止している (28 行目から 30 行目)。32 行目から 36 行目までが実際の課題の実行部分で、ユーザの入力は 35 行目の set /p で string2 という環境変数に保存する。

  また、課題の実行開始時と終了時の時刻を、動的環境変数 %time% を利用して time1 と time2 に保存している。

- 38 行目から 45 行目までが入力後の表示。39 行目の gettime サブルーチンの呼び出しで time1 と time2 の時間差を計算し、40 行目の diffstr サブルーチンの呼び出しで string1 と string2 の同じ文字、違う文字の個数をそれぞれ調べている。その結果を 41 行目から 45 行目で表示し、47 行目で最初のメニューに戻っている。

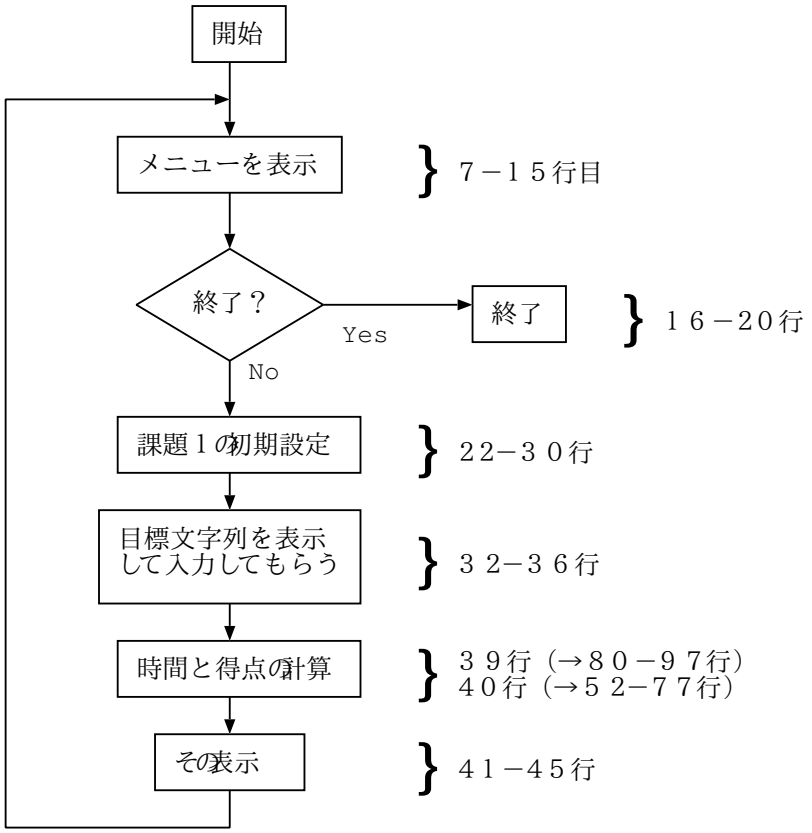


図 3: サンプルプログラムのおおまかなフローチャート

- 51 行目から 77 行目までは文字列の比較を行う `diffstr` サブルーチン。先頭から 1 文字ずつ取り出して比較し、同じならば `tcount` を 1 増やし、違っていれば `ncount` を 1 増やす、という作業を文字列の終わりまで行っている。

  なお、途中で文字列が空になってしまうとエラーになるので、実際には先頭の文字を取り出すのではなく、無理矢理先頭に `X` という文字をつけて (57, 58 行目)、2 文字目を取り出している (64 行目から 68 行目)

- 79 行目から 97 行目までは時間差を計算する `gettime` サブルーチン。`%time%` の値は「16:48:54.26」「 8:03:03.02」のような形式なので、区切り文字の : や . を飛ばしながら時刻部分 (H)、分部分 (M)、秒部分 (S)、センチ (1/100) 秒部分 (P) を取り出して、

$$X = 100(3600H + 60M + S) + P$$

  のように変換することで、2 つの時刻を今日の 00:00:00.00 からのセンチ秒数に直し (85 行目から 93 行目)、その 2 つの時刻の差の秒部分を `sec` に、センチ秒部分を `csec` に保存している。

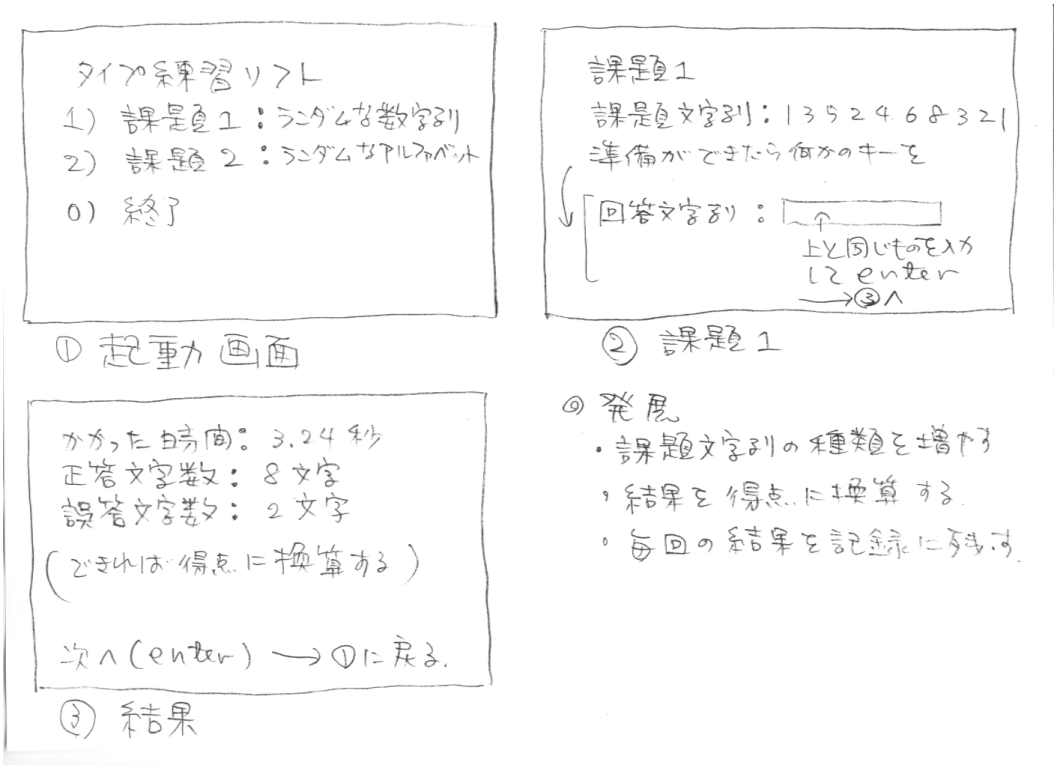
  なお、M,S,P 部分は 0 から始まる文字列になりうるが、それは数値としては 8 進数と解釈されることになっている。そのため、08, 09 のような場合はエラーとなってしまう

ため、無理矢理各文字列の先頭に 1 を追加して 108 のような 3 桁の数字にしてから 100 を引く、という作業を行って M, S, P を取り出している (86 行目から 88 行目、91 行目から 93 行目)。

4 課題について

- **課題 1:** 作成するものの概要 (スケッチ) の作成

  サンプル



  実際に作成してスケッチは手書きでよいが、面の数は、多分このサンプルよりもかなり多くなる。

- **課題 2:** 最終課題のフローチャートの作成

  まず、図 3 のようなおおまかなフローチャートを作成する。また、個々の処理部分については、簡単でないものや工夫したものについて、そのフローチャートを作成する。いずれも手書きで構わない。

参考文献

[1] 飯島弘文「Windows コマンドプロンプト スパテク 242」翔泳社 (2008)

[2] 村上俊一「ウィンドウズの仕組みがわかると『MS–DOS/コマンドプロンプト』に強くなる」メディア・テック出版 (2004)

[3] 藤田英時「コマンドまたはファイル名が違います」ナツメ社 (1992)

[4] 田中一郎「早わかり MS–DOS Ver.3.3 実用マニュアル」新星出版社 (1991)

[5] 竹野茂治「バッチファイルの概要」(2011)
`http://takeno.iee.niit.ac.jp/~shige/misc/script/script.html#bat-intro`