Fun!(damental) and Fun!(ctional):
An Introduction to Table-driven Programming

Ted Holt

Sr. Software Developer, Profound Logic Software
Sr. Technical Editor, Four Hundred Guru

Agenda

- **Definition**
- Demonstration
- A real-life example

Procedural Programming

- Also called imperative programming
- The programmer writes a series of commands in a computer language. The computer executes the commands to achieve a desired result.

Example

```
if Age < 6;
   AgeCategory = 'Preschool';
elseif Age < 18;
   AgeCategory = 'School age';
else;
   AgeCategory = 'Graduate';
endif;
```

Table-driven Programming

- Program logic is stored in an array (table).
- The computer interprets the instructions in the table, calling appropriate routines, to achieve the desired result.

Table-driven Logic

```
set current state = 1
do
   get Input
   get action and next state from
      the state table
   execute the action
   set current state = next state
until current state = 0
```

The engine (interpreter) is a simple fetch-execute loop.

The programmer must write a routine for each action, except for no-action.

Conditional logic (if-then-else) is handled by a state table.

A Few Advantages

- Handles logic that is not easily expressed in procedural programming.
- Logic can be changed by changing the table (permanently or at run time).
- Non-programmers can work on program logic.
- The same engine (interpreter) can be re-used.

A Few Advantages

- The table is portable to other languages.
- A program is easily enhanced by modifying the table and writing routines for new actions.

Agenda

- Definition
- **Demonstration**
- A real-life example

The Challenge

Split a DOS-like file name into:

- drive
- path
- base name
- extension.

The file name may have any or all four parts.

Do not assume the file name is valid!

Example 1

C:\Tmp\MyFile.TXT

- Drive = C
- Path = \Tmp\
- Base name = MyFile
- Extension = TXT

Example 2

\Tmp\2014\MyData

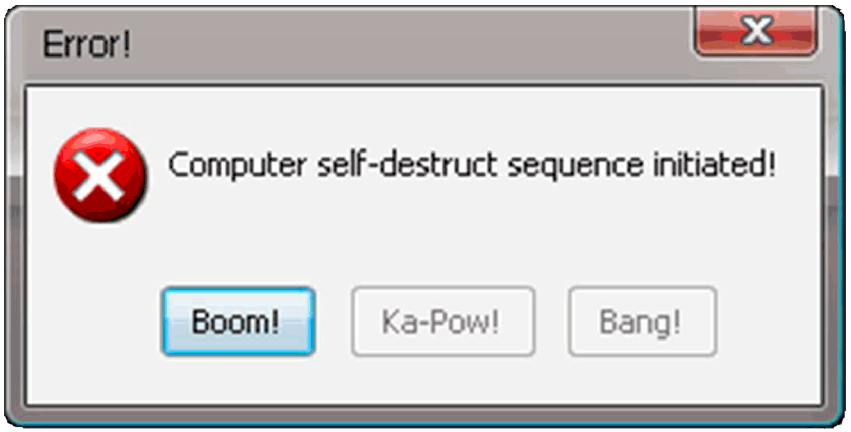
- Drive = not specified
- Path = \Tmp\2014\
- Base name = MyData
- Extension = not specified

Example 3

\Tmp/2014\piggy.pdf

Invalid!

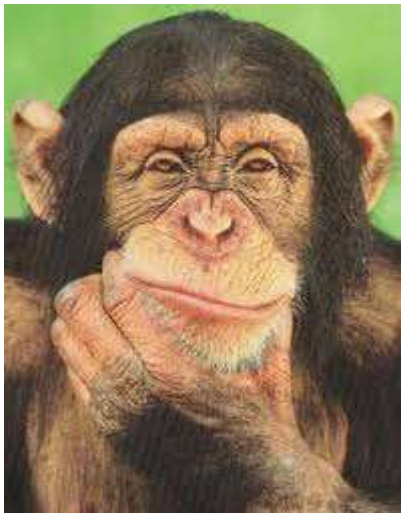
Why?



Think about It

How would you accomplish this task in your preferred programming language?

Would this be an easy task?



You would probably need a lot of IF statements. For example:

- If the second character is a colon, the first character is the drive. If, that is, the first character is a letter.

- If there is a period within the last four characters, whatever follows the period, if anything, is the extension.

And so forth.

I contend that this task would be a nightmare with procedural programming. But I could be wrong.

State Table

State	EOI	Blank	Letter	Digit	:	\	.	Other
1	NO 0	NO 1	CT 2	CT 4	ER 0	TP 4	ER 0	ER 0
2	ER 0	ER 0	CT 4	CT 4	TD 3	ER 0	ER 0	ER 0
3	TD 0	ER 0	CT 4	CT 4	ER 0	TP 4	TB 5	ER 0
4	TB 0	ER 0	CT 4	CT 4	ER 0	TP 4	TB 5	ER 0
5	TE 0	ER 0	CT 5	CT 5	ER 0	ER 0	ER 0	ER 0

The table is two-dimensional.

The rows of the table correspond to the five states of the program.

The columns correspond to the possible input values. Notice that some columns represent single characters, while others represent categories of characters.

EOI is short for "end of input". This is the imaginary character that follows the last non-blank character of the file string.

Each element (cell) of the table contains two values: an action code and next state.

States

1	**Looking for drive, path, or base name**
2	**Looking for colon, path or base name**
3	**Looking for path or base name**
4	**Building path or base name; looking for extension.**
5	**Building extension; looking for EOI.**

State Table

State	EOI	Blank	Letter	Digit	:	\	.	Other
1	NO 0	NO 1	CT 2	CT 4	ER 0	TP 4	ER 0	ER 0
2	ER 0	ER 0	CT 4	CT 4	TD 3	ER 0	ER 0	ER 0
3	TD 0	ER 0	CT 4	CT 4	ER 0	TP 4	TB 5	ER 0
4	TB 0	ER 0	CT 4	CT 4	ER 0	TP 4	TB 5	ER 0
5	TE 0	ER 0	CT 5	CT 5	ER 0	ER 0	ER 0	ER 0

Actions

CT	Append current character to token.
ER	Process an error.
NO	Take no action.
TB	Copy token to base name.
TE	Copy token to extension.
TD	Copy token to drive.
TP	Append current character to token and append token to path.

c:\tmp\br549.dat token=c

State	EOI	Blank	Letter	Digit	:	\	.	Other
1	NO 0	NO 1	CT 2	CT 4	ER 0	TP 4	ER 0	ER 0
2	ER 0	ER 0	CT 4	CT 4	TD 3	ER 0	ER 0	ER 0
3	TD 0	ER 0	CT 4	CT 4	ER 0	TP 4	TB 5	ER 0
4	TB 0	ER 0	CT 4	CT 4	ER 0	TP 4	TB 5	ER 0
5	TE 0	ER 0	CT 5	CT 5	ER 0	ER 0	ER 0	ER 0

c:\tmp\br549.dat drive=c

State	EOI	Blank	Letter	Digit	:	\	.	Other
1	NO 0	NO 1	CT 2	CT 4	ER 0	TP 4	ER 0	ER 0
2	ER 0	ER 0	CT 4	CT 4	TD 3	ER 0	ER 0	ER 0
3	TD 0	ER 0	CT 4	CT 4	ER 0	TP 4	TB 5	ER 0
4	TB 0	ER 0	CT 4	CT 4	ER 0	TP 4	TB 5	ER 0
5	TE 0	ER 0	CT 5	CT 5	ER 0	ER 0	ER 0	ER 0

c:\tmp\br549.dat path=\

State	EOI	Blank	Letter	Digit	:	\	.	Other
1	NO 0	NO 1	CT 2	CT 4	ER 0	TP 4	ER 0	ER 0
2	ER 0	ER 0	CT 4	CT 4	TD 3	ER 0	ER 0	ER 0
3	TD 0	ER 0	CT 4	CT 4	ER 0	TP 4	TB 5	ER 0
4	TB 0	ER 0	CT 4	CT 4	ER 0	TP 4	TB 5	ER 0
5	TE 0	ER 0	CT 5	CT 5	ER 0	ER 0	ER 0	ER 0

c:\tmp\br549.dat token=t

State	EOI	Blank	Letter	Digit	:	\	.	Other
1	NO 0	NO 1	CT 2	CT 4	ER 0	TP 4	ER 0	ER 0
2	ER 0	ER 0	CT 4	CT 4	TD 3	ER 0	ER 0	ER 0
3	TD 0	ER 0	CT 4	CT 4	ER 0	TP 4	TB 5	ER 0
4	TB 0	ER 0	CT 4	CT 4	ER 0	TP 4	TB 5	ER 0
5	TE 0	ER 0	CT 5	CT 5	ER 0	ER 0	ER 0	ER 0

c:\tmp\br549.dat token=tm

State	EOI	Blank	Letter	Digit	:	\	.	Other
1	NO 0	NO 1	CT 2	CT 4	ER 0	TP 4	ER 0	ER 0
2	ER 0	ER 0	CT 4	CT 4	TD 3	ER 0	ER 0	ER 0
3	TD 0	ER 0	CT 4	CT 4	ER 0	TP 4	TB 5	ER 0
4	TB 0	ER 0	CT 4	CT 4	ER 0	TP 4	TB 5	ER 0
5	TE 0	ER 0	CT 5	CT 5	ER 0	ER 0	ER 0	ER 0

c:\tmp\br549.dat token=tmp

State	EOI	Blank	Letter	Digit	:	\	.	Other
1	NO 0	NO 1	CT 2	CT 4	ER 0	TP 4	ER 0	ER 0
2	ER 0	ER 0	CT 4	CT 4	TD 3	ER 0	ER 0	ER 0
3	TD 0	ER 0	CT 4	CT 4	ER 0	TP 4	TB 5	ER 0
4	TB 0	ER 0	CT 4	CT 4	ER 0	TP 4	TB 5	ER 0
5	TE 0	ER 0	CT 5	CT 5	ER 0	ER 0	ER 0	ER 0

c:\tmp\br549.dat path=\tmp\

State	EOI	Blank	Letter	Digit	:	\	.	Other
1	NO 0	NO 1	CT 2	CT 4	ER 0	TP 4	ER 0	ER 0
2	ER 0	ER 0	CT 4	CT 4	TD 3	ER 0	ER 0	ER 0
3	TD 0	ER 0	CT 4	CT 4	ER 0	TP 4	TB 5	ER 0
4	TB 0	ER 0	CT 4	CT 4	ER 0	TP 4	TB 5	ER 0
5	TE 0	ER 0	CT 5	CT 5	ER 0	ER 0	ER 0	ER 0

c:\tmp\br549.dat token=b

State	EOI	Blank	Letter	Digit	:	\	.	Other
1	NO 0	NO 1	CT 2	CT 4	ER 0	TP 4	ER 0	ER 0
2	ER 0	ER 0	CT 4	CT 4	TD 3	ER 0	ER 0	ER 0
3	TD 0	ER 0	CT 4	CT 4	ER 0	TP 4	TB 5	ER 0
4	TB 0	ER 0	CT 4	CT 4	ER 0	TP 4	TB 5	ER 0
5	TE 0	ER 0	CT 5	CT 5	ER 0	ER 0	ER 0	ER 0

c:\tmp\br549.dat token=br

State	EOI	Blank	Letter	Digit	:	\	.	Other
1	NO 0	NO 1	CT 2	CT 4	ER 0	TP 4	ER 0	ER 0
2	ER 0	ER 0	CT 4	CT 4	TD 3	ER 0	ER 0	ER 0
3	TD 0	ER 0	CT 4	CT 4	ER 0	TP 4	TB 5	ER 0
4	TB 0	ER 0	CT 4	CT 4	ER 0	TP 4	TB 5	ER 0
5	TE 0	ER 0	CT 5	CT 5	ER 0	ER 0	ER 0	ER 0

c:\tmp\br549.dat token=br5

State	EOI	Blank	Letter	Digit	:	\	.	Other
1	NO 0	NO 1	CT 2	CT 4	ER 0	TP 4	ER 0	ER 0
2	ER 0	ER 0	CT 4	CT 4	TD 3	ER 0	ER 0	ER 0
3	TD 0	ER 0	CT 4	CT 4	ER 0	TP 4	TB 5	ER 0
4	TB 0	ER 0	CT 4	CT 4	ER 0	TP 4	TB 5	ER 0
5	TE 0	ER 0	CT 5	CT 5	ER 0	ER 0	ER 0	ER 0

c:\tmp\br549.dat token=br54

State	EOI	Blank	Letter	Digit	:	\	.	Other
1	NO 0	NO 1	CT 2	CT 4	ER 0	TP 4	ER 0	ER 0
2	ER 0	ER 0	CT 4	CT 4	TD 3	ER 0	ER 0	ER 0
3	TD 0	ER 0	CT 4	CT 4	ER 0	TP 4	TB 5	ER 0
4	TB 0	ER 0	CT 4	CT 4	ER 0	TP 4	TB 5	ER 0
5	TE 0	ER 0	CT 5	CT 5	ER 0	ER 0	ER 0	ER 0

c:\tmp\br549.dat token=br549

State	EOI	Blank	Letter	Digit	:	\	.	Other
1	NO 0	NO 1	CT 2	CT 4	ER 0	TP 4	ER 0	ER 0
2	ER 0	ER 0	CT 4	CT 4	TD 3	ER 0	ER 0	ER 0
3	TD 0	ER 0	CT 4	CT 4	ER 0	TP 4	TB 5	ER 0
4	TB 0	ER 0	CT 4	CT 4	ER 0	TP 4	TB 5	ER 0
5	TE 0	ER 0	CT 5	CT 5	ER 0	ER 0	ER 0	ER 0

c:\tmp\br549.dat base name=br549

State	EOI	Blank	Letter	Digit	:	\	.	Other
1	NO 0	NO 1	CT 2	CT 4	ER 0	TP 4	ER 0	ER 0
2	ER 0	ER 0	CT 4	CT 4	TD 3	ER 0	ER 0	ER 0
3	TD 0	ER 0	CT 4	CT 4	ER 0	TP 4	TB 5	ER 0
4	TB 0	ER 0	CT 4	CT 4	ER 0	TP 4	TB 5	ER 0
5	TE 0	ER 0	CT 5	CT 5	ER 0	ER 0	ER 0	ER 0

c:\tmp\br549.dat token=d

State	EOI	Blank	Letter	Digit	:	\	.	Other
1	NO 0	NO 1	CT 2	CT 4	ER 0	TP 4	ER 0	ER 0
2	ER 0	ER 0	CT 4	CT 4	TD 3	ER 0	ER 0	ER 0
3	TD 0	ER 0	CT 4	CT 4	ER 0	TP 4	TB 5	ER 0
4	TB 0	ER 0	CT 4	CT 4	ER 0	TP 4	TB 5	ER 0
5	TE 0	ER 0	CT 5	CT 5	ER 0	ER 0	ER 0	ER 0

c:\tmp\br549.dat token=da

State	EOI	Blank	Letter	Digit	:	\	.	Other
1	NO 0	NO 1	CT 2	CT 4	ER 0	TP 4	ER 0	ER 0
2	ER 0	ER 0	CT 4	CT 4	TD 3	ER 0	ER 0	ER 0
3	TD 0	ER 0	CT 4	CT 4	ER 0	TP 4	TB 5	ER 0
4	TB 0	ER 0	CT 4	CT 4	ER 0	TP 4	TB 5	ER 0
5	TE 0	ER 0	CT 5	CT 5	ER 0	ER 0	ER 0	ER 0

c:\tmp\br549.dat token=dat

State	EOI	Blank	Letter	Digit	:	\	.	Other
1	NO 0	NO 1	CT 2	CT 4	ER 0	TP 4	ER 0	ER 0
2	ER 0	ER 0	CT 4	CT 4	TD 3	ER 0	ER 0	ER 0
3	TD 0	ER 0	CT 4	CT 4	ER 0	TP 4	TB 5	ER 0
4	TB 0	ER 0	CT 4	CT 4	ER 0	TP 4	TB 5	ER 0
5	TE 0	ER 0	CT 5	CT 5	ER 0	ER 0	ER 0	ER 0

c:\tmp\br549.dat extension=dat

State	EOI	Blank	Letter	Digit	:	\	.	Other
1	NO 0	NO 1	CT 2	CT 4	ER 0	TP 4	ER 0	ER 0
2	ER 0	ER 0	CT 4	CT 4	TD 3	ER 0	ER 0	ER 0
3	TD 0	ER 0	CT 4	CT 4	ER 0	TP 4	TB 5	ER 0
4	TB 0	ER 0	CT 4	CT 4	ER 0	TP 4	TB 5	ER 0
5	TE 0	ER 0	CT 5	CT 5	ER 0	ER 0	ER 0	ER 0

Voilà!

- drive = c
- path = \tmp\
- base name = br549
- extension = dat

Agenda

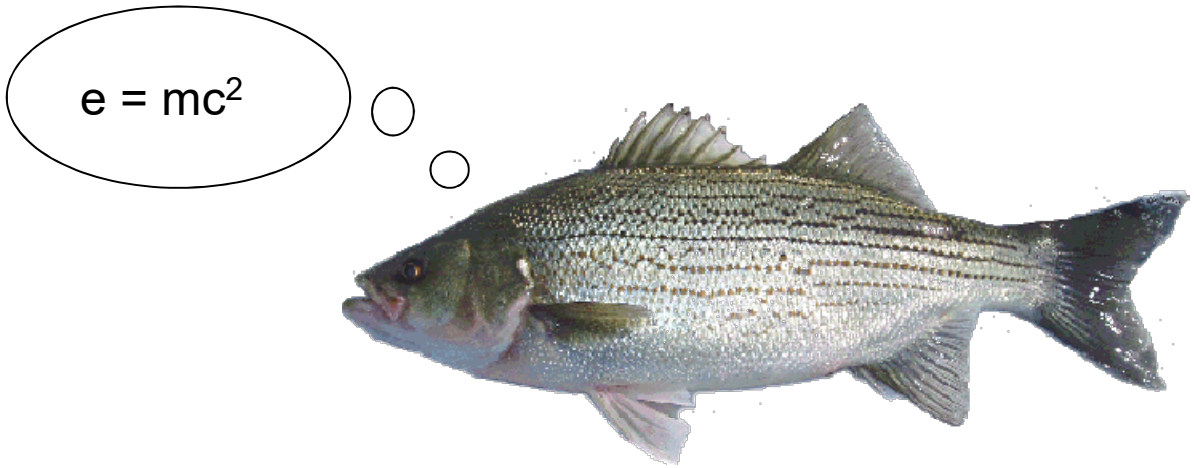
- Definition
- Demonstration
- **A real-life example**

BASS

- A scripting language
- Pronounced like the fish, not the low-voice male singer
- Short for "Build a spreadsheet"
- Reads an SCS report and uses a script to produce a CSV file
- Uses six table-driven routines
- Runs on IBM i

BASS Logic

```
loop
    read a line of the report
    if end-of-file then exit loop
    execute the script
end-loop
```



Accounts Receivable by State 11/20/14

State	Name	Account	City	Due	
MN	Abraham	583990	Isle	500.00	
MN	Alison	846283	Isle	10.00	
State total				510.00	*
NY	Lee	192837	Hector	489.50	
NY	Jones	839283	Clay	100.00	
NY	Tyron	397267	Hector	.00	
State total				589.50	*
TX	Henning	938472	Dallas	37.00	
TX	Williams	593029	Dallas	25.00	
State total				62.00	*
Grand total				1,161.50	**

** End of report **

The Script, Page 1 of 3

```
# Accounts receivable by state

   var $HeadersBuilt = NO

# Skip unneeded lines
 / $Input                    == ' '    / exit
 / $Input (  1 :   2 )  == 'Ac'   / goto 100
 / $Input (  1 :   2 )  == 'St'   / exit
 / $Input (  1 )            == '='    / exit
 / $Input (  1 :   5 )  == State / exit
 / $Input (  1 :   5 )  == Grand / stop
```

The first line is a comment. Comments begin with the pound (hash) sign and extend through the end of the line. A comment may occupy a line by itself or be placed on a line after commands.

This script contains one variable, $HeadersBuilt. Variable names begin with a dollar sign and are not case-sensitive. Variables are 256-byte variable-length string. This variable has an initial value of NO. Variables do not have to be initialized.

When BASS reads a report line, it copies the text of the line into the $Input variable .

The expressions within slashes are conditional expressions. Think of them as an IF. The first test compares $Input to a blank. This tests for blank lines on the report. If a line is blank, the exit instruction executes. The exit instruction tells BASS to quit processing this report line and proceed to the next one.

The following conditions use reference modifiers to test parts of a report line. A reference modifier consists of a beginning position and a length surrounded by parentheses and separated from one another by a colon. If the length is not given, it is assumed to mean one character. Most of these tests look for header and footer lines, which do not need to be copied into the CSV file.

The second test executes a goto command to branch to the routine that builds the heading line in the CSV file.

The Script, Page 2 of 3

```
# anything else must be a detail line

    $a = $Input (16:6)  # account number
    $b = $Input ( 7:8)  # customer name
    $c = $Input (24:8)  # city
    $d = $Input ( 1:2)  # state
    $e = $Input (32:8)  # balance due
    de-edit $e
    addrow
    exit
```

Variables $A through $Z indicate the first 26 columns of a spreadsheet. This routine copies data from the report line into the first five columns of the CSV file.

The DE-EDIT command removes dollar signs and commas, and indicates negative values with a leading minus sign.

The ADDROW command adds a row to the CSV file.

The EXIT command prevents control from falling through to the header routine.

The Script, Page 3 of 3

```
100 # header routine

    / $HeadersBuilt == YES/ exit

    $a = 'Account number'
    $b = 'Customer name'
    $c = 'City'
    $d = 'State'
    $e = 'Balance due'
    addrow
    $HeadersBuilt = YES
```

This routine builds the first row, which specifies the column headings in the CSV file .

The $HeadersBuilt variable keeps this script from writing the heading row more than once. When the heading row has been built, the value of $HeadersBuilt changes from NO to YES. The first line of the routine exits the script if the column headings have already been written to the CSV file.

The Output

```
Account number,Customer name,...*
583990,Abraham,Isle,MN,500
846283,Alison,Isle,MN,10
192837,Lee,Hector,NY,489.50
839283,Jones,Clay,NY,100
397267,Tyron,Hector,NY,0
938472,Henning,Dallas,TX,37
593029,Williams,Dallas,TX,25
```

* The first line is truncated due to lack of space on this slide.

Free-form with Ease!

The following are equivalent:

```
/$Input(1:5)==Grand/   exit
/ $Input ( 1 : 5 ) == Grand / exit
/    $Input  (  1 :5 )== Grand   /   exit
/ $Input ( 01 : 005 )== Grand / exit
/ $Input (  1 :   5 ) == Grand  /   exit

etc.
```

The BASS scripting language is free-format. Except for literals, you may use all the white space you want.

All of these expressions accomplish the same thing. If the first five characters of a report line are "Grand", BASS ceases to process that report line and moves on to the next line of the report.

Without table-driven programming, it would be difficult to make BASS a free-format language.

For More Information

- BASS: Build a Spreadsheet
  http://www.itjungle.com/fhg/fhg121212-story02.html
- Error-Checking Email Addresses, for Intelligent People
  http://www.itjungle.com/fhg/fhg050907-story02.html
- My email address:
  fourhundredguru@gmail.com

An Example Program

See program FSMX01R.

Source code is provided with this presentation.

The report shows the inner workings of the state machine.

Feel free to use the code as a basis for your own projects.

Above all, have fun!

```
CurrentState = 0;
```

(i. e. The End)