WISCONSIN COMPUTER SOCIETY

NEWSLETTER

****

Volume #2, Issue #7   July 1977   Don Stevens, Editor

****

MEETING NOTICE

Our meeting will be held at 1:00 p.m., July 9, 1977, at the Waukesha Technical Institute (Room 202 - Administration Building).

PROGRAM AGENDA

Club member Todd L. Voros will be our Main Speaker. The topic of his lecture will be SKETCHCODE - a programming technique devised by Todd. If you can, please study the article on SKETCHCODE in this Newsletter before the meeting. It could help you to better understand this new programming technique when presented at our meeting.

CLUB MEMBERSHIP DUES COLLECTION

Membership for the last 6 months of 1977 is $3.00. This will be the last Newsletter for those who have not paid membership fee.

Send your $3.00 membership fee to:

Donald Stevens
P.O. Box 159
Sheboygan Falls, Wisc. 53085

CLUB QUESTIONAIRE

I have not received many of the completed questionaires enclosed in the June Newsletter. BRING IN THESE QUESTIONAIRES TO THE JULY MEETING.

****

FOR SALE - - - - - - - - - - - - - - - - - - Contact Don Stevens

One (1) MB6A Solid State Music RAM Board - 8K Board

Two (2) MB2 Solid State Music RAM Boards - 4K Boards.

Note: These boards are made for S-100 Buss and are top quality boards.

****

Please be advised of the opening of the Madison Computer Store, 1919 Monroe Street, Madison, Wisc. 53711. Huron Smith is store Manager.

The AMIDE Corporation of New York announces availability of a PDP-8 simulator for the 8080. Priced around $20.00 and available in Paper Tape or Tarbell Cassette

****

Heath Kit announces availability of:

8080 Based System ($375 including Octal Keyboard) & compatible peripherals

LSI-11 Based System ($1295.00) & compatible peripherals

Paper Tape Reader/Punch - reads at 50 characters per second and punch operates at 10 characters per second ($395)

Photos available at JULY MEETING

****

Centronics announces Compact Microprinter - 240 characters per second and priced at $595.00

****

FREE FREE ATTENDANCE PRIZE COPIES OF JULY COMPUTER NOTES from MITS

****

GOOD THINGS TO READ

Computer Design - June 1977

Microcomputer Interfacing: Interfacing a 10-Bit DAC

A Task Scheduling Executive Program for Microcomputer Systems

Analysis of Multiple Microprocessor System Architectures

EDN - JUNE 1977

Chapters 1 thru 13 of Software Design Course (pages 67 thru 200)

Electronic Design - June 7, 1977

Getting the bugs out of your Software

Interface Circuit that teams cassette recorder with a CRT to work as a TTY/papertape unit

IEEE CIRCUITS & SYSTEMS - Feb. 1977

A simple Cassette Interface

IEEE SPECTRUM - May 1977

Everybody's Doing It ('computing' at home)

IEEE SPECTRUM - April 1977

Analog tests: The microprocessor scores

SKETCHCODE

A DOCUMENTATION TECHNIQUE FOR HOME COMPUTER HOBBYISTS AND PROGRAMMERS

By: Todd L. Voros
Systems Software Specialist
A.O. Smith Corporation
Data Systems Division

****

Problem: How can we keep from rewriting the same code over and over again for different computers?

Problem: How can we help our colleagues understand our programs quickly and efficiently?

Problem: How can we simplify and ease the debugging of our programs?

Answer: SKETCHCODE

Sketchcode? Sketchcode is a documentation technique, that if properly used can save time and effort when coding in any computer language.

What it is not: Sketchcode is not flowcharting.
Sketchcode is not a language.
Sketchcode is not hard to learn.

What it is: To answer this question, let us ask ourselves, "What are programs made of?"

Program: An implementation of one or more algorithms intended to solve a problem expressed in a machine digestible form.

What are algorithms composed of?

Processes that do not require decisions
Decisions

What is the normal method of documenting an algorithm for computer implementation? A: Flowcharting.

Does sketchcode replace Flowcharting? No, it compliments it.

So what is sketchcode?

Sketchcode is an individualistic, stylistic pseudo - language expressing the logical flow of control through a program through the use of certain elementary structures.

What are these elementary structures?

Things like: Loops

Decisions

Indentation of Logical Levels of Control

How do we express a loop in Sketchcode?

```
We write:     DO WHILE (an expression);

                PROCESSING

              END;
```

Note PROCESSING is indented two blanks to the right. All other sketchcode processing within that loop will be indented two blanks to the right.

(an expression) is the evaluation of any number of variables we desire resulting in the assignment of a TRUE or a FALSE condition.

While the condition remains true, we execute statements inside the loop.

If the condition is false, we do not execute any statements in the loop; we proceed to begin executing the statement after the END; which tells us where the loop ends. This is why the END; is not indented two columns to the right like PROCESSING.

How can we get out of a sketchcode DO loop?

> By having PROCESSING within the loop change the value of the variables tested by (an expression).

An example? Execute some processing 10 times:

```
COUNT = 0

DO WHILE (COUNT less than 10);

  PROCESS

  COUNT = COUNT + 1

END;
```

Of course, the expression that is tested for TRUE or FLASE could be much more complex:

```
DO WHILE (I =3*X OR Q=7*SQRT(35.2-E));
```

And of course, we can put a loop inside a loop:

```
DO WHILE (I less than 10)

  PROCESS

  DO WHILE (J more than 12)
    MORE PROCESSING

    MORE PROCESSING

  END;

END;
```

Notice each DO has it's own closing END statement.

Now if you think about this form of representation of the logical flow of a program for a moment or two, you may begin to see how some fairly complex situations involving loops inside of loops could be clearly and consisely expressed. Also note that the inner DO loop was indented two columns to the right and processing performed under it's control was itself indented two columns to the right.

Thus; The deeper a loop is (the more nested it is in the logical flow of control of the program), the further to the right it will appear in the program's Sketchcode representation.

Also; Code that is indented to the far right will probably be executing the most often by your program. Therefore, concentrate your optimizing efforts there first (if you make any such efforts).

However, programs are not composed entirely of loops, although they play a very important part of programming.

Decisions are also of prime importance in directing the flow of control of a program. In Sketchcode, a decision is always represented as a elementary structure of the form:

```
IF (expression)

  THEN DO;

    PROCESSING for TRUE expression

  ELSE;

    PROCESSING for FALSE expression
```

Notice, for readability that the THEN DO; and the ELSE; are indented two columns to the right and their corresponding processing is itself indented two columns further right.

Since (an expression), is always true or false in Sketchcode, either the processing under the THEN DO; will be executed and the processing under the ELSE; will be skipped, or the processing under the THEN DO; will be ignored and the processing under the ELSE; will be executed.

Sometimes in the flow of control of a program there occurs the situation that some processing is to be done only if some condition is true; otherwise nothing is to be done. This would tend to create a dangling ELSE; so Sketchcode allows the convention that a decision can also take the form:

```
IF (expression is true)

  THEN;

    PROCESSING
```

without a closing ELSE; condition. However, if the condition being evaluated by the IF is false, all processing indented to the right of the THEN; is ignored.

Sometimes loops in a program are effectively never-ending. To handle this special case, Sketchcode permits a special form of the DO loop notation:

```
DO FOREVER;

  PROCESSING

END;
```

An example of 'DO FOREVER' might be where we wish the computer to read data from the teletype forever and process it, give us an answer, and await further input. This could be done as follows:

```
DO FOREVER;

  READ INPUT
  PROCESS INPUT

END;
```

Sometimes we wish to perform once-only initialization inside of a loop in our programs. This would seem difficult to represent in Sketchcode notation but is actually not. Taking a combination of DO and IF simple structures, we are able to build a SWITCH STRUCTURE:

```
INITSW  = 'initialize'

DO WHILE (an expression);

  IF (INITSW = 'initialize)
    THEN DO;
      PERFORM INITIALIZATION PROCESSING
      SET INITSW = 'don't initialize'
    ELSE;
      PERFORM NORMAL PROCESSING

END;
```

The only tricky point to the above example is that the first time through the above example, INITSW (our initialize/don't initialize switch) will be set to 'initialize' so the THEN DO; leg of the IF will be executed on the first pass through the DO loop. Since the last thing the initialization processing does is to set the initialization switch to 'don't initialize', when performing the 2nd through last pass through the DO loop, the code under the ELSE; branch of the IF will be executed!! Thus, we have managed to provide a method of once-only initialization inside of a DO loop.

Notice we did not forget the closing END;, and of course keep in mind that both the initialization and normal processing could have deeper levels of DO's and IF's imbedded within them.

Now, what is the point we are attempting to make? The structures we have defined are completely adequate for expressing any problem capable of being implemented on a hobbyist home microprocessor system.

So what? Well, then so what is this?: WHERE WERE THE GOTO STATEMENTS?? (OR JUMPS, OR BRANCHES IF YOU PREFER)

Answer: There aren't any in Sketchcode.

Program logic always flows from top to bottom, through various levels of indentation on the way.

Program loops are always clearly documented.

Sketchcode forces you to provide a clear, concise definition of what you are trying to do, yet permit individualistic style to be used (our own examples certainly aren't part of any 'LEGAL' programming language).

When a program's logic is done in Sketchcode, it is easier to follow and debug.

If your algorithm is written in Sketchcode, it can easily be written for another, perhaps totally different microprocessor.

And last, but not least, if you really want to make your programs self-documenting, include your program's Sketchcode as part of the COMMENTS in the assembly language version of your program if you are writing it in an assembler. However, no matter what the language, Sketchcode assists in providing better documentation and insight into your efforts.

****

Brief Commentary:

A few hints from the author on the use and writing of Sketchcode follow from his experience in working with it for the last two years:

1. If you find yourself writing a lot of IF-THEN-ELSE, IF-THEN-ELSE clustered together in your Sketchcode, ask yourself the question: "Is this really a DO loop in disguise?"

2. Remember that all IF's do not necessarily require an ELSE!

3. Don't forget to indent when going to a deeper level of control!!

4. Remember that Table Searches are usually implemented by DO's.

5. Processing performed under the legs of an IF (the THEN DO and the ELSE) can be switched by negating the results of (an expression).

Thus,

```
IF (X = 0)
  THEN DO;
    A=B
  ELSE;
    A=B+B
```

is the same as

```
IF (X not equal 0)
  THEN DO;
    A=B+B
  ELSE;
    A=B
```

6. This point is tricky, but is worth consideration if your Sketchcode somehow doesn't 'seem right':

   If the ELSE condition of the IF can be gotten to by some other code prior to the IF test, then it is NOT an ELSE condition. Remove the ELSE and the indentation of code under the ELSE.

7. Before you begin to write down the very first machine or assembly language statement of your program, have the completed Sketchcode of your program in front of you and code your program from the Sketchcode!!!

8. Ask others to review your Sketchcode if you are working on a complex task.

****

The following Program was written by Tom Doyle and has been in use in my 8080 System which has a Digital Group TV Readout and Cassette Interface. The Video interface supports 512 characters (32 characters by 16 lines) on my Video Monitor.

This program for automatic Scroll resides on Page 054 in my System. The buffer area assigned in this program must occupy the top 512 bytes of an area in memory where you have no memory for at least 32 bytes above it. Program can be relocated most any place in memory as long it adheres to above conditions. My Video Output Port is 064.

SCROLL ROUTINE FOR VIDEO READOUT ---------------- by Tom Doyle
****

```
054000/ 365 PUSH  PSW
054001/ 325 PUSH  D
054002/ 345 PUSH  H
054003/ 376 CPI         015       Is it a carriage return
054005/ 312 JZ          054044
054010/ 376 CPI         012       Is it a line feed
054012/ 312 JZ          054050
054015/ 376 CPI         177       Is it a TAB    DELETE
054017/ 312 JZ          054056
054022/ 376 CPI         010       Is it a backspace
054024/ 312 JZ          054064
054027/ 376 CPI         137       Is it an "_"  Underscore
054031/ 312 JZ          054064
054034/ 376 CPI         014       Is it "FF"     CONTROL L
054036/ 312 JZ          054056
054041/ 315 CALL        054127    Call Input
054044/ 341 POP   H
054045/ 321 POP   D
054046/ 361 POP   PSW
054047/ 311 RET
054050/ 315 CALL        054207    Call Scroll  ◄———————— Line Feed
054053/ 303 JMP         054044
054056/ 315 CALL        054332    Call Clear Screen ◄————TAB
054061/ 303 JMP         054044
054064/ 076 MVI   A     040       LXI " "    ◄———————— Backspace
054066/ 052 LHLD        055376    Bottom Less 2
054071/ 167 MOV   M A
054072/ 043 INX   H
054073/ 167 MOV   M A
054074/ 042 SHLD        055376    Bottom Less 2
054077/ 315 CALL        054173    Call Print
054102/ 041 LXI   H     057375    Top Less 2
054105/ 315 CALL        054153    Call Move Cursor
054110/ 315 CALL        054317    Call Print Cursor
054113/ 303 JMP         054044    Jump Out
054116/ 076 MVI   A     377                ◄———————— Home Up
054120/ 323 OUT         064       Video Port
054122/ 076 MVI   A     000
054124/ 323 OUT         064
054126/ 311 RET
054127/ 052 LHLD        055376    Bottom Less 2 ◄———————— Input
054132/ 167 MOV   M A
054133/ 315 CALL        054173    Call Print
054136/ 315 CALL        054352    Call EOB (End of Buffer)
054141/ 332 JC          054207    JC Scroll
054144/ 053 DCX   H
054145/ 042 SHLD        055376    Bottom Less 2
054150/ 303 JMP         054312    Jump Cursor
054153/ 076 MVI   A     001                 ◄————————Move Cursor
054155/ 323 OUT         064
054157/ 076 MVI   A     000
054161/ 323 OUT         064
054163/ 315 CALL        054352    Call EOB
054166/ 330 RC
054167/ 053 DCX   H
054170/ 303 JMP         054153    Jump for Next Byte
054173/ 000 NOP                              ◄———— Print
054174/ 000 NOP
054175/ 000 NOP
```

```
054176/ 306 ADI         200
054200/ 323 OUT         064
054202/ 076 MVI    A    000
054204/ 323 OUT         064
054206/ 311 RET
054207/ 041 LXI    H    057377      Top                  <------- Scroll 1 Line
054212/ 021 LXI    D    060037      Top + 32
054215/ 176 MOV    A M
054216/ 022 STAX   D
054217/ 315 CALL        054352      Call EOB
054222/ 332 JC          054232
054225/ 053 DCX    H
054226/ 033 DCX    D
054227/ 303 JMP         054215      Jump for Next Byte
054232/ 041 LXI    H    056037      Start of Bottom Line <---Clear Bottom
054235/ 076 MVI    A    040                                  Line
054237/ 167 MOV    M A
054240/ 315 CALL        054352      Call EOB
054243/ 332 JC          054252
054246/ 053 DCX    H
054247/ 303 JMP         054235      Jump for Next Byte
054252/ 315 CALL        054116      Call Home Up
054255/ 041 LXI    H    057377      Top of Memory Area
054260/ 176 MOV    A M
054261/ 315 CALL        054173      Call Print
054264/ 315 CALL        054352      Call EOB
054267/ 332 JC          054276
054272/ 053 DCX    H
054273/ 303 JMP         054260      Jump for Next Byte
054276/ 041 LXI    H    056037      Bottom + 32          <--{Reset
054301/ 042 SHLD        055376      Bottom less 2             Buffer Pointer
054304/ 041 LXI    H    057337      Top less 32          <--{Move to
054307/ 315 CALL        054153      Call Move Cursor          Lower Left
054312/ 333 IN          377         Input Sense Switches
054314/ 007 RLC
054315/ 330 RC
054316/ 000 NOP
054317/ 076 MVI    A    137         Move ASCII "_"       <--{Print Cursor
054321/ 315 CALL        054173      Call Print                Symbol
054324/ 041 LXI    H    057376      Top less 1           <--{Advance
054327/ 303 JMP         054153      Jump to Move Cursor       Counter
054332/ 041 LXI    H    057377      Top of Memory Area <----Fill Buffer with
054335/ 076 MVI    A    040         Move ASCII " "          Spaces
054337/ 167 MOV    M A
054340/ 315 CALL        054352      Call Check for Bottom of Buffer
054343/ 332 JC          054252
054346/ 053 DCX    H
054347/ 303 JMP         054335
054352/ 325 PUSH   D
054353/ 021 LXI    D    056000      Bottom of Memory Area
054356/ 174 MOV    A H
054357/ 272 CMP    D
054360/ 302 JNZ         054374
054363/ 175 MOV    A L
054364/ 273 CMP    E
054365/ 302 JNZ         054374
054370/ 067 STC
054371/ 303 JMP         054376
054374/ 067 STC
054375/ 077 CMC
054376/ 321 POP    D
054377/ 311 RET
```

Copied
&
Stapled
4/30/04