by Steven Yates

Because of the nature of the Action! system, typing checkers like D:CHECK (issue 16) cannot be implemented. A lack of line numbers leaves no way to communicate to the user where typos are. Moreover, the flexibility in the source program's form makes finding errors difficult—without requiring the reader to type the program exactly as it appears.

D:CHECK in Action! gives you a program (D:CHECK, ACT) which works with the Action! system to provide interactive checking and correcting of typing errors. Instead of printing a list of numbers which you compare to a similar list in the magazine, this Action! version finds the errors and puts you back into the editor at their approximate locations in your source.

This extra power takes some more typing on your part. If you look at Listing 1, you'll notice the first few lines contain a set of numbers headed "CHECKSUM DATA." All other Action! programs printed in ANALOG Computing from now on will have similar lines at the start of their listings.

These lines give D:CHECK in Action! its power. The program generates numbers from what you've typed and compares them to the numbers here. If there are any discrepancies, it will then help you to find the problems, so they can be corrected.

When typing in the listing, these lines must be typed in exactly as printed.

There are square brackets at the beginning and end of the checksum list, and one space between each value. All values are two-digit numbers and must be typed as such. After these lines are typed, the remainder of the program can be entered in any way you wish.

You may type it as listed, or you may decide not to include the blank lines inserted for easier reading. You may add more blank lines, if you like. You may combine short lines to form one line with a space or more between the originals; or you could break long lines into two or more lines, so everything fits on the screen.

The program ignores spaces, so you don't need to indent lines, and you can add spaces if it's more readable for you. D:CHECK in Action! also ignores comments, so you can leave them out and save some typing, or add some of your own.

This flexibility of program form is basic to the Action! system which D:CHECK in Action! preserves. Remember, any modifications to the form of the program may make it more difficult to compare to the original if there's a problem—so use your judgement. Anything between quotes must be typed exactly as shown; including spaces, and upper and lower case letters.

The latter present another problem. Action! allows you to type in whatever case you want, but also offers the option of being case sensitive. This lets you have two or more variables of the same name, with different letters in upper case to distinguish them.

D:CHECK in Action! offers the same option. If the article doesn't specify otherwise, you may use either case. If the program must be compiled with the case-sensitive option, the words Case Sensitive will appear with the checksum data. You must type all letters in the case in which they're listed. Because it must be able to handle other case-sensitive programs, D:CHECK in Action! must also be able...
to compile under this option, so remember to type each letter properly.

**Using it.**

First, type in Listing 1 and save it. Because the program can't check itself until it compiles successfully, you must be careful keying it in. Don't forget that the program is case sensitive.

Try compiling. If there are any compile errors, fix them as you normally would. Once D:CHECK in Action! compiles, run it. If the program says there are no problems, you're ready to use it for other programs printed in ANALOG Computing.

If it tells you there are problems, follow the directions below. The program cannot guarantee the locations it gives for errors in its own source file, because typing errors may be causing problems with its error-locating routines. Be sure to save a final copy after you've fixed all the errors, as D:CHECK.ACT (cassette users may save it as C:).

To check another program containing the checksum numbers, type it in and save a copy. With the newly typed program still in memory, go to the monitor. If the article says the program is case sensitive, use the monitor option command to set it for this. Otherwise set this option to "no."

When this is done, type R"R:D:CHECK.ACT" (cassette users, type R"C:). The program will load from disk, compile, then run. It must be uncompiled when stored, because it needs to relocate itself for each program checked.

If the program being checked isn't too large (about 100 disk sectors), time can be saved by loading D:CHECK.ACT into the second editing window. It can then be compiled and run without accessing the disk each time.

To do this, type in the program to be checked and save it. Enter window 2 by pressing CTRL-SHIFT-2, read D:CHECK.ACT, then enter the monitor and use C and R to compile and run.

To rerun D:CHECK.ACT after a problem's corrected, just enter window 2 (CTRL-SHIFT-2) before returning to the monitor, then compile and run as before. If you don't move the cursor to window 2 before entering the monitor, Action! will attempt to compile the program in window 1, and D:CHECK.ACT won't be executed.

If you get an out-of-memory error when loading or compiling D:CHECK.ACT, make sure your source program is saved and use the monitor boot command to reset Action! Then reload your program and run D:CHECK.ACT from disk or tape as explained previously.

If the D:CHECK.ACT program says there are no problems, you should be able to compile the program. If it does find a problem, it will display the checksum lines on-screen, with one number highlighted. It will ask if that sum was typed correctly.

Check the highlighted value against the magazine listing. If they don't match, press N in response to its question. Return to the editor. The cursor will be on the first digit of the incorrect sum. Retype the sum, return to the monitor, and repeat the command to run D:CHECK.ACT.

If the highlighted sum matches that printed in the magazine, press Y. The program will tell you to return to the editor and check the line containing the cursor, plus a certain number of lines following it. The number of lines to check depends mostly on line length, and blank lines aren't counted.

Find any mistakes you can on these lines, correct them, return to the monitor, and rerun D:CHECK.ACT. Once you've found all errors and are given a clean check, save a final copy, then compile and run it, according to the directions in the related article.

It's possible that a program which checks out all right won't compile. If this happens, return to the editor and make sure you didn't insert or delete a space, which would confuse the compiler. Check the word the cursor's on and change it to look exactly as it does in the magazine. This should be the only thing to cause a problem in a properly checked program.

Now you should be able to enjoy printed Action! programs without becoming frustrated by cryptic compiler errors, or having a program compile and not perform as expected. If you remember the difference D:CHECK made in the time it took to get a BASIC program running, you'll type in D:CHECK in Action! as soon as possible.
Listing 1.
Action! listing.

```plaintext
; CHECK.ACT
; Steven Yates
; 12/02/85

; CHECKSUM DATA
; 7E 43 81 20 74 DD 87 7C
; 33 38 5F 7A CA 69 77
; AD AE 96 4F 8F 99 39
; ED 7

BYTE StartChar, CurChar, Count, X,
    Flag=10, Character, Sum, ISUM,
    Product, Key, Case=101, Column=1152,
    Sensitive=11, Lines, SUMLine,
    WrongLine, Segment=10, STRING=11,
    Substring=12, Space=132
CARD StartLine, Line=1160,
FirstLine=1166
BYTE ARRAY SUMs(256)
CARD ARRAY SUMLine(12)
BYTE POINTER Length,
CARD POINTER CurLine, NextLine,
    WrongSUM

DEFINE is="=", not="()", Done="Flag=1"

PROC End_Of_Line()
    DO
        CurLine=NextLine+
            CurChar=7
            Length=CurLine+5
            NextLine=CurLine+4
            IF CurLine=0 THEN
                EXIT
            FI
            UNTIL Length<>0
    OD
RETURN

PROC Quotes()
    IF Segment is String THEN
        Segment=0
    ELSE
        Segment=String
    FI
RETURN

PROC Ignore()
    X=-1
    Count=-1
    Character=0
RETURN

PROC Check_Line()
    Character=peek (CurLine+CurChar)
    IF Segment is SubString THEN
        Segment=0
    ELSEIF Character="" THEN
        Quotes()
    ELSEIF Segment is String THEN
```

---

**ATARI**
800, 800XL, 65XE,
100XE

**MORE! GRAPHICS**
for The Print/Shop™

Over 120 Graphics!

10 Screen Magic Scenes!

Pierstorf

THE PIERSTORFF CO. $19.95
1311 MAIN ST.
WOODBURY, CA 95695
CLASSIC SUPPLY (916) 666-3550 ORDERS
California residents add 64 tax
add 32 shipping
CHECK C.O.D. VISA MASTERCARD
THE PRINTSHOP IS A TRADEMARK OF SOFTBRAIN SOFTWARE
NOT AFFILIATED WITH THE PIERSTORFF CO
CLEANS INKS ONE STEP

CIRCLE #115 ON READER SERVICE CARD

ANALOG COMPUTING JULY 1986 / PAGE 25
PROC D_Check()
Initialize()
Get_Sums()
ISUM=0
DO
  IF Done THEN EXIT
  FI
  SUM=0
  FOR Count = 0 TO 127 DO
    IF Count = 0 THEN EXIT
    START Char = CurChar
    Check_Char = START Char + 6
    IF Char = NULL THEN Check_Sum
    IF Done THEN EXIT
    FI
    IF No_Problem THEN EXIT
    FI
    Lines = Lines + 1
    FI
  OD
  IF Done THEN EXIT
  FI
  Check_Sum()
  OD
Close(1)
RETURN

WANT TO SUBSCRIBE?
It's worth it.

CALL TOLL FREE
1-800-345-8112
In Pennsylvania
1-800-662-2444

ANALOG COMPUTING

JULY 1986 / PAGE 27
ELSE 
   End_Of_Line()
FI 
IF CurLine=0 THEN 
   Done 
   EXIT 
FI 
OD 
CurChar=8 
Character=Peek(CurLine+CurChar) 
IF Character is 'I' THEN 
   EXIT 
FI 
IF Done THEN 
   PutE("Listing does not") 
   PutE("contain checksums.") 
   PutE() 
   PutE("Cannot CHECK!") 
   EXIT 
FI 
End_Of_Line() 
OD 
RETURN 

PROC Get_Sums() 
BYTE I 
BYTE ARRAY Hex(I) 

Find_Sums() 
SumLine=0 
DO 
   IF Done THEN 
      EXIT 
   FI 
   SumLines(SumLine)=CurLine 
   CurChar=9 
   FOR ISum=0 TO 7 DO 
      FOR I=0 TO 7 DO 
         Hex(I)=Peek(CurLine+CurChar+I) 
         IF Hex(I)=="A" THEN 
            Hex(I)==="A-9-1" 
         FI 
         Hex(I)==="0" 
         OD 
         SUM=(Hex(8) LSH 4)+Hex(I) 
         SUM=SUMLine+ISum 
         IF Peek(CurLine+CurChar+3) is 'I' THEN 
            Done 
            EXIT 
         FI 
         CurChar=+3 
         OD 
      End_Of_Line(
      IF Done THEN 
         Flag=0 
         EXIT 
      FI 
      SumLine=+1 
   OD 
RETURN 

PROC Mistyped() 
Line=StartLine 
Column=StartChar 
PrintE("Return to editor and check") 
Print("the ") 
PrintB(Lines) 
PrintE(" lines following the line") 
Print("the cursor is on for a ") 
PrintE("typo.") 
RETURN 

PROC Bad_Sum() 
BYTE I 
IF Case is Sensitive THEN 
   Print("If article does not ") 
   PrintE("specify") 
   Print("case sensitive, use ") 
   PrintE("option") 
   Print("command to set this to ") 
   PrintE("no.") 
   PutE() 
   PutE() 
FI 
WrongLine=ISum/8 
ISum=ISum%7 
WrongSum=SumLines(WrongLine)+9+Sum%3 
WrongSum=+%8000 
FOR I=0 TO SumLine DO 
   PrintE(SumLines(I)+6) 
   OD 
WrongSum=+%8000 
PutE() 
Print("Is highlighted sum correct?") 
Key=GetB(I) 
Key=#32 
PutE() 
PutE() 
IF Key='y' THEN 
   Mistyped() 
ELSE 
   Print("Return to editor and ") 
   PrintE("correct") 
   PrintE("mistyped sum.") 
   Line=SumLines(WrongLine) 
   Column=(5Sum+1)%3 
FI 
RETURN 

PROC No_Problems() 
   PrintE("Program CHECKS out fine.") 
   PrintE("Save program and use") 
   PrintE("according to directions") 
   PrintE("in the article.") 
   Done 
RETURN 

PROC Check_Sum() 
   IF Sum()%Sum(ISum) THEN 
      Bad_Sum() 
      Done 
      ISum=+1 
   RETURN 

PROC Initialize() 
   IF Peek(126)=255 THEN 
      Case is Sensitive 
      FI 
      X=1 
      CurLine=FirstLine 
      Length=CurLine+6 
      NextLine=CurLine+4 
      IF Length=0 THEN 
         End_Of_Line() 
      FI 
      CurChar=7 
      Close(II) 
      Open(II,"k:",4,0) 
   RETURN