1976

A Program to Count Operators and Operands for ANSI—FORTRAN Modules

Karl J. Ottenstein

Report Number:
76-196

A PROGRAM TO COUNT OPERATORS AND OPERANDS FOR ANSI-FORTRAN MODULES

Karl J. Ottenstein

Computer Sciences Department
Purdue University
West Lafayette, Indiana 47907

CSD-TR 196

June 1976

This paper describes a program which scans an arbitrary number of ANSI-FORTRAN modules, counting the basic software science parameters [Halstead 1972] for each module. There is no syntax checking code in the analyzer since conformance to the ANSI standard is easily verified by existing compilers. It is essential, therefore, that all input follow the standard for accurate results. Most of the analyzer itself is written in ANSI-FORTRAN, allowing not only portability, but also the ability to run the program on itself as a test procedure.

Only the following statements effect the counts of operands and operators:

- assignment
- ASSIGN
- CALL
- DO
- GO TO
- IF

Most counts are accumulated as described by [Bulut 1974a]:

- All arithmetic, boolean, and replacement operators are counted.
- Function (subroutine) names are operators.
- Each GO TO to a unique label is a unique operator.

- The ASSIGN statement is counted as an assignment.

- IF is an operator.

- Statement labels in GO TO()L1,L2,L3 are counted as GO TO L1, GO TO L2, GO TO L3.

- Bracketing is an operator occurring as parenthetical grouping or a (DO,DO-foot) pair.

- Subscripting as indicated by parentheses is counted as such.

- All commas are operators.

- The implied EOS (end of statement) operator is counted.

The following differences (additions) are present:

- Parameters are treated as variables.

- No attempts are made at correcting impurities in the programs [Bulut 1974b]. Bulut had resolved local variable ambiguity and had detected certain common subexpressions related to array addressing.

- Each computed (assigned) GO TO counts as a unique "special" GO TO since there is a unique (implied) jump into a jump table.

- All constants are mapped into real numbers. The integers are handled trivially. The boolean constants .TRUE. and .FALSE. are mapped to 1.0 and 0.0 respectively. Hollerith constants are ignored since they are only permitted in DATA statements.

- Statement functions are ignored.

Input Format

Input to the analyzer consists of any number of ANSI-FORTRAN modules terminated by a card with an asterisk (*) in column one.

Transporting the Analyzer

Bringing up the analyzer on a new machine requires two things if that machine is not a 60-bit CDC machine:

- changing certain global parameters,
- rewriting certain machine dependent routines.

The analyzer consists of the following ANSI modules:

ANALYZE	- driver
CONSCH	- searches the constant table for a given value
DIMCHK	- scans dimensioning statements, recording array sizes
EXPR	- scans and counts an expression
GOTO	- searches goto-table for a given label
INITM	- initializes for a new module
INITP	- initializes program
NULINE	- increments lines-per-page count and pages on overflow
NUSTMT	- reads and packs one FORTRAN statement
PAGE	- performs page ejection and titling
PARSE	- determines statement type, counting everything except expressions
PMATCH	- searches input for a matching right parenthesis
PRSTAT	- prints out statistics at the end of a module
REALNO	- scans a number and returns its real value
SCAN	- scans one token from the input stream
SKIP	- skips blanks in source statement
SUMARY	- prints a summary of the statistics for each module in the run (provided that more than one module was scanned).

The constants that require changing are in INITP:

- PDEPTH, page depth (in lines)
- SYSIN, logical input unit number
- SYSOUT, logical output unit number
- SYSSUM, logical unit number for a scratch summary file.

The non-ANSI modules that may require rewriting are:

CODE	- converts a character to an integer on the range 0 to 63 according to the CDC internal representation (display code)
DIGIT	- returns true if its input is the character code for a digit
LETDIG	- returns true if its input is the character code for a letter or a digit
LETTER	- returns true if its input is the character code for a letter
LOOKUP	- searches the hash table for a given symbol
NUMCVT	- converts a numeric character string to its integer representation
WRTNAM	- writes the name of a symbol from the hash table and its frequency of occurrence.

Implementation Notes

Much of the slowness and awkwardness of the program is caused by the conformance to the ANSI-standard (specifically, the one character per word restriction). A P-register trace showed that most execution time was spent doing I/O and removing blanks from the input statements. If efficiency is not satisfactory, the loop in NUSTMT which handles blank removal might be coded in machine language.

The analyzer will stop with an informative message if any of the various tables should overflow. An alphabetic variable glossary in the source listing will aid in determining both the table name and the variable indicating its size. The size variables are set in INITP and should be changed to reflect any new dimensioning in the common blocks.

REFERENCES

[Bulut 1974a] Bulut, Nedet, Halstead, M. H., and Bayer, Rudolf. Experimental validation of a structural property of FORTRAN algorithms. CSD TR 115. Purdue University (April 1974).

[Bulut 1974b] Bulut, Neodet, and Halstead, Maurice H. Impurities found in algorithm implementations. CSD TR 111. Purdue University (1974).

[Halstead 1972] Halstead, M. H. Natural laws controlling algorithm structure? ACM SIGPLAN Notices 7,2 (February 1972) 19-26.

NOTE: The program source is not included as part of this technical report any longer. It has been substantially modified since 1976. Contact the Software Metrics Research Group at Purdue for information about the current version and other counters.