1102

Bell Laboratories

Cover Sheet for Technical Memorandum

The information contained herein is for the use of employees of Bell Laboratories and is not for publication (see GEI 13.9-3)

MH

Title- Some Statistics on the Usage of the C Language

Date- November 24, 1975

TM- 75-1273-13

Other Keywords- Computer Languages
Programming Language C
UNIX

Author(s)	Location and Room	Extension
Hania Gajewska	MH 2C-572	7667
Responsible Engineer: S. C. Johnson	MH 2C-572	3968

Charging Case- 39199

Filing Case- 39199-11

ABSTRACT

The question of how people use computer languages is interesting for several reasons. A collection of statistics on the use of a language may help the compiler writer to produce better code with less effort. It may help designers of programming languages to design languages which are easy to use because they include constructs preferred by their users. Finally, programmers may be interested in such information for stylistic reasons.

This paper describes a program that analyzes programs written in the C language. Some results were obtained on approximately 35000 lines of code.

Pages Text 6 Other 8 Total 14

No. Figures 0 No. Tables 0 No. Refs. 6

Address Label

DATE - FILE COPY

BELL TELEPHONE LABORATORIES INCORPORATED

E-1932-C (6-73)

SEE REVERSE SIDE FOR DISTRIBUTION LIST

BELL TELEPHONE LABORATORIES, INC.

TM-75-1273-13

DISTRIBUTION
(REFER GEI 13.9-3)

COMPLETE MEMORANDUM TO

CORRESPONDENCE FILES

OFFICIAL FILE COPY
PLUS ONE COPY FOR
EACH ADDITIONAL FILING
CASE REFERENCED

DATE FILE COPY
(FORM E-1328)

10 REFERENCE COPIES

<AHO,A V
ALMQUIST,R P
ARNOLD,S L
<BECKER,R A
BECKETT,J T
BERRYMAN,R D
BIREN,MRS IRMA B
BLUE,J L
BOLSKY,MORRIS I
BOWYER,L RAY
BOYCE,W M
BOYLE,GERALD C
BROEK,HOWARD W
BROWN,W STANLEY
BROWN,WILLIAM R
CAMPBELL,STEPHEN T
CRUME,LARRY L
DAVIS,R D
DENTON,R T
DEVLIN,MRS SUSAN J
ESTVANDER,ROBERT A
<FLEISCHER,HERBERT I
FOUGHT,B T
<FRASER,A G
FREEMAN,K GLENN
GEARY,M J
GEYLING,F T
<GOLDSTEIN,A JAY
GRAHAM,R L
GUNDERMAN,K
HALL,ANDREW D JR
<HAMILTON,PATRICIA
HAMMING,R W
<HANNAY,N B
IVIE,EVAN L
<JOHNSON,STEPHEN C
<KEESE,W M
<KERNIGHAN,BRIAN W
KORNEGAY,R L
<LUDERER,GOTTFRIED W R
LYONS,STEVEN H
<MARANZANO,JOSEPH F
<MC GILL,ROBERT
MC GONEGAL,MISS C A
MC ILROY,M DOUGLAS
+MCDONALD,H S
MENON,P R
MORGAN,S P
OPITZ,R
OSSANNA,J F JR

COMPLETE MEMORANDUM TO

PARA,P S
PATTON,GARDNER C
PINSON,ELLIOT N
+PRIM,ROBERT C
<RALEIGH,THOMAS M
RIDDLE,GUY G
>RITCHIE,DENNIS M
ROBERTS,CHARLES S
RODRIGUEZ,ERNESTO J
ROSLER,LAWRENCE
SATZ,L R
SCHLEGEL,C T
SEMMELMAN,C L
SHORTER,J W
SIPES,J D
SMITH,K H
SMITH,LESLIE T
SPIRES,R J
STROHECKER,CARY A
TERRY,M E
THOMPSON,K
<THOMSON,M-L
TRACY,MRS C E
WARNER,JACK L
WARNER,TERRY L
WASSERMAN,MRS Z
WATSON,D S
WEBBER,SUSAN A
WITKOWSKI,MISS JOAN D
WOLFE,ROBERT M
<YAMIN,MRS E E
YEN,T C
ZISLIS,P M
83 NAMES

COVER SHEET ONLY TO

CORRESPONDENCE FILES

4 COPIES PLUS ONE
COPY FOR EACH FILING
CASE

ACKERMAN,A F
AHERN,PETER L
AHRENS,RAINER B
ALCALAY,DAVID
ALLEN,JAMES R
ARMSTRONG,DOUGLAS B
ARNDT,DENNIS L
ARNOLD,GEORGE W
ARNOLD,THOMAS F
ATAL,B S
BADURA,DENNIS C
BAKER,BRENDA S
BALLARD,EDWIN C JR
BASEIL,RICHARD J
BAUGH,C R
BERNSTEIN,LAWRENCE
BEYER,JEAN-DAVID

COVER SHEET ONLY TO

BICKFORD,N B
BILOWOS,RICHARD M
BIRCHALL,R H
BLINN,JAMES C
BLUM,MRS MARION
BLY,JOSEPH A
BODEN,F J
BOURNE,STEPHEN R
BOWERS,J L
BOWLES,J B
BOYLE,W S
BRADY,PAUL T
BRANDT,RICHARD B
BRAND,JOE E
BROOKS,CATHERINE A
BROWN,JAMES W
BULFER,ANDREW F
BURNETTE,W A
BURROWS,T A
BYRNE,EDWARD R
CANADAY,RUDD H
CASPERS,MRS BARBARA E
CASTELLANO,MRS M A
CAVINESS,JOHN D
CHAFFEE,N F
CHAMBERS,J M
CHAMBERS,MRS B C
CHEN,STEPHEN
CHEN,T L
CHERRY,MS L L
CHRIST,C W JR
CLAYTON,D P
CLOUTIER,J E
COBEN,ROBERT M
COHEN,HARVEY
<COLE,LOUIS M
COLE,M O
COLLINS,J P
COOPER,A E
COPP,DAVID H
COSTANTINO,B B
CRACOVIA,E V
CRAGUN,D W
CUTLER,C CHAPIN
D ANDREA,MRS LOUISE A
DESENDORF,JUDITH
DESMOND,J P
DEUTSCH,DAVID N
DI PIETRO,RICHARD S
DICKMAN,B N
DIMMICK,JAMES O
DOLOTTA,T A
DOMBROWSKI,F J
DOWDEN,D C
DRAKE,MRS L
DUFFY,FRANCIS P
DWYER,T J
EDDY,MICHAEL R
EDELSON,D
EDMUNDS,T W
EITELBACH,DAVID L
ELLIOTT,R J

COVER SHEET ONLY TO

ELY,T C
EMERSON,R S
ERDLE,K W
ESSERMAN,ALAN R
FABISCH,MICHAEL P
FAULKNER,R A
FEDER,J
FELDMAN,STUART I
FELS,ALLEN M
FISCHER,H B
FLANDRENA,R J
FLOMENHOFT,MARK J
FONG,KENNETH T
FOUNTOUKIDIS,A
FOWLER,BRUCE R
FOWLER,C F
FOX,PHYLLIS
FOY,J C
FRANK,MISS A J
FRANK,RUDOLPH J
FREEMAN,R DON
FREIDENREICH,MRS B
FROST,H BONNELL
FULTON,ALAN W
GARCIA,R F
GATES,G W
GAY,FRANCIS A
GEPNER,JAMES R
GIBB,KENNETH R
<GILLETTE,DEAN
GIMPEL,JAMES F
GITHENS,JOHN A
GLASSER,ALAN L
GLUCK,F
<GNANADESIKAN,R
GOLABEK,MISS R
GORDON,BRIAN G
GORMAN,JAMES E
GOTTDENKER,ROBERT G
GREENE,MRS DELTA A
GREENWALGH,H WAIN
GROSSO,I
GROSS,ARTHUR G
GUERRIERO,JOSEPH R
GUIDI,PIER V
GUMMEL,HERMANN K
HAFER,E H
HALE,A L
HALFIN,SHLOMO
HALL,MILTON S JR
HALL,W G
HANNAH,JUDY R
<HARKNESS,C J
HARRISON,NEAL T
HARTWELL,WALTER T
HARUTA,K
HASZTO,EDWARD D
HAUSE,A D
<HAWKINS,DONALD T
HAWKINS,RICHARD B
HEATH,SIDNEY F III
HELD,RICHARD W

COVER SHEET ONLY TO

HEROLD,JOHN W
HESS,MILTON S
HILL,DOUGLAS W
HINDERKS,I W
HOEHN,MISS MARIE J
HOLTMAN,JAMES P
HONIG,W L
HOYT,WILLIAM P
HO,MS J
HUANG,VICTOR K L
HUDSON,E T
HUNNICUTT,CHARLES F
HUPKA,MRS FLORENCE
<HYMAN,B
HYMAN,G M
IERLEY,W H
IPPOLITI,O D
IRVINE,M M
JACKOWSKI,D J
JACKSON,JAMES H
JACOBS,H S
JAMES,J W III
<JENSEN,PAUL D
JESSOP,WARREN H
JONES,M B
<JUDICE,CHARLES N
<KAISER,J F
KALRO,ASHOK L
KAMLET,ARTHUR S
KANE,J RICHARD
KANE,MRS ANNE B
KAPLAN,M M
KATZ,MURRAY J
KAUFELD,J C JR
KAUFMAN,LARRY S
KAYEL,R G
KELLY,L J
KENNEDY,ROBERT A
KERTZ,DENIS R
KILLMER,JOEN C JR
KLAPPROTH,F F
KNOWLTON,KENNETH
KNUDSEN,DONALD B
KUBITSKY,MRS G
LANDOLINA,W C JR
LANG,A
LAYTON,RICHARD L
LEGENHAUSEN,S
LEHRMAN,WILLIAM
LERNER,E M
LESK,MICHAEL E
LESSER,PETER V
LESTER,THOMAS E
LEVINSON,EDWARD
LICWINKO,J S
LIEBERT,THOMAS A
LINDERMAN,J
<LOGAN,MRS V G
LOMUTO,N
LOYD,D G
LUTZ,KENNETH J
<LYCKLAMA,HEINZ
...
402 TOTAL

+ NAMED BY AUTHOR > CITED AS REFERENCE < REQUESTED BY READER (NAMES WITHOUT PREFIX WERE SELECTED USING THE AUTHOR'S SUBJECT OR ORGANIZATIONAL SPECIFICATION AS GIVEN BELOW)

MERCURY SPECIFICATION.....

COMPLETE MEMO TO:
127-SUP

COPRDB = COMPUTER PROGRAM DEBUGGING, MEASUREMENT AND TESTING

COVER SHEET TO:
12-DIR 13-DIR 127

COCSHE = HUMAN ENGINEERING OF COMPUTER SYSTEMS

HO CORRESPONDENCE FILES
HO 5C101

TM-75-1273-13
TOTAL PAGES

TO GET A COMPLETE COPY:

1. BE SURE YOUR CORRECT ADDRESS IS GIVEN ON THE OTHER SIDE.
2. FOLD THIS SHEET IN HALF WITH THIS SIDE OUT AND STAPLE.
3. CIRCLE THE ADDRESS AT RIGHT. USE NO ENVELOPE.

PLEASE SEND A COMPLETE COPY TO THE ADDRESS SHOWN ON THE OTHER SIDE.

DATE FILE COPY
BELL TELEPHONE LABORATORIES
INCORPORATED

E-1328A (5-69)

Bell Laboratories

subject: Some Statistics on the Usage of the C Language

File 39199-11

date: November 24, 1975

from: Hania Gajewska

(Responsible Engineer: S. C. Johnson)

TM 75-1273-13

MEMORANDUM FOR FILE

Introduction

The question of how people use computer languages is interesting for several reasons. A collection of statistics on the use of a language may help the compiler writer to produce better code with less effort. It may help designers of programming languages to design languages which are easy to use because they include constructs preferred by their users. Finally, programmers may be interested in such information for stylistic reasons.

Statistical analyses of programming languages have previously been performed for FORTRAN (Knuth [1]) and ALGOL (Wichman [2]) programs. This paper describes a program that similarly analyzes programs written in C [3]. Some results obtained using this program are presented and compared with the FORTRAN results described by Knuth.

The issue of whether to collect statistics on what the programmer does or on what the compiler sees has come up at many points. We have tried to compromise: on the one hand we treat subscripts as they are written by the programmer; on the other, we could not resist the temptation to reduce constant expressions.

Unfortunately, results of this sort raise as many questions as they answer. In anticipation of this, an effort was made to code the statistics collection routines so that they could be easily modified to collect additional information. This goal was largely met; however, adding new counts is still not trivial.

The following two sections describe, respectively, the method used to collect the data and a summary of the results. Detailed tables of results may be found in Appendix B.

DATE FILE COPY
BELL TELEPHONE LABORATORIES
INCORPORATED

Method

The backbone of the system is a lexical analyzer, a parser, and symbol table routines, written by S.C. Johnson. The lexical analyzer segments the input into a sequence of basic components called tokens. In our case the input is a C program and it is separated into C keywords, operators, identifiers, and constants.

The tokens are passed to the parser. The parser imposes a structure on the tokens according to a set of rules (a grammar). Our grammar describes the language C, so the parser accepts C programs.

The lexical analyzer and the parser are automatically generated by the utilities Lex [4] and Yacc [5], respectively. Both these utilities allow the user to get control inside the programs they generate: the user may associate a program segment (action) with any of the specification rules for Yacc or Lex. This action will be executed when its associated rule is applied to the input.

In our case, the grammar contains actions which maintain a symbol table. For each identifier this table contains its type and class; it also contains offsets into other tables where more information about the identifier (e.g. its size) can be found.

Taking this program as a starting point, we added actions to the grammar which build trees for arithmetic expressions. The interior nodes of the trees represent binary and unary operations; the leaves represent identifiers and constants. In the first case, the data structure used to represent a node consists of the operation (+, %, sizeof, etc.) and pointers to operands. In the second case, it consists of the type (integer, identifier, float, etc.) and the value or symbol table offset.

To each production of the form

<expression> -> ...

we added an action to build a tree for the expression. If the expression represents an operation on integer constants, it is collapsed , i.e. replaced by a single node representing an appropriately valued constant.

The counts are collected into an array. Each element of the array contains the count for a different item. All array indices are either simple manifest constants, or can be obtained by adding (manifest) offsets to manifest indices. This makes the data structure flexible and the program easy to modify.

We collect information at many different places. Key-

words are counted as they are recognized by the lexical analyzer. Types and classes of variables of internal scope are counted at the end of each function parsed; types and classes of external variables are counted at the end of each file.

The rest of the counting takes place during parsing. To grammar rules involving operators we added actions to count operators; grammar rules which accumulate lists (of arguments or statements) count the list sizes.

Perhaps the most important of all is the data which describes the complexity of expressions. For each expression, three measures of complexity are computed: the number of operators, the number of tokens, and a modified Sethi-Ullman number (an estimate of the number of registers required to compute the expression with no stores). Details of the modifications to the Sethi-Ullman algorithm are described in Appendix A.

We keep complexity data separately for the various positions in which expressions can occur: a while loop, three places in a for statement, and so on. The computations are performed as actions associated with rules of the form

<statement> -> ... <expression> ... .

Complexity counts for <expression> are accumulated into a temporary data structure during a recursive traversal of the tree for the expression. They are later added into the count array. This approach is easily extensible: to compute additional information about the expression one needs only to add an extra field to the temporary data structure, and extra code to the recursive routine.

When the program finishes running, we have the accumulated complexities for the expressions at each position. We also have the number of expressions at each position; thus we can calculate the average complexity.

The output of the counting program is simply the counts, printed four numbers to the line. This output is then usually processed by another program (called output) to make it more readable. Output is based on Lex and Yacc. It accepts any number of output files from the counting program (count files) and a format file. It merges the contents of the count files, adding the corresponding counts. It then prints selected results according to a pattern given in the format file.

The format file consists of text, which is simply copied, and count expressions, which are evaluated and printed. Count expressions consist of count names (listed in Appendix C), together with the usual arithmetic opera-

DATE FILE COPY
BELL TELEPHONE LABORATORIES
INCORPORATED

tions +,-,*, and /, and parentheses (). Thus

```
N_LINES/N_FUNCS
```

would print the average number of lines per function. If a format file is not specified, format commands are accepted from standard input. This means that output can be used interactively to interrogate a file of counts.

Experiments and Results

We ran the program on roughly 35000 lines of C code contained in the Murray Hill Unix directories /usr/source/s1 and /usr/source/s2. These directories contain most of the Unix commands (such as the C compiler and the editor). Although these programs may not represent the average existing C program, they are probably the programs which get used most frequently. The most interesting results of these experiments are discussed below; complete tables appear in Appendix B.

The 35000 lines consisted of approximately 1000 functions. Although these figures might suggest that an average C function consists of 35 lines (including comments), the existence of external definitions in the code would make the actual average smaller. Perhaps a more significant measure of function size, then, would be the number of statements -- 18.5 per function. As we shall see, however, the concept of "statement" in C is not well defined.

We obtained an estimate of the number of statements as follows: we summed up the keyword counts for return, goto, if, switch, break, continue, for, and while (note that the number of whiles gives the total number of while and do-while constructs). To this we added the number of null statements (;), and expression statements (defined as <expression> ; as in [3]) for the total number of statements. By this convention the construct

```
if (i) a=b; else a=c;
```

consists of three statements. Note also that labels, cases, and defaults are not counted, so that

```
switch (i) {
     case 1:  a=b; break;
     case 2:  a=c; break;
     default:  a=d;
}
```

consists of six statements. In what follows, we will use the word "statement" as described above. Similarly, "while statement", "if statement", "expression statement", etc. will have the obvious meanings. The formal definitions can

be found in [3].

By far the most commonly used C statement is the expression statement: it constitutes roughly 57% of the 18500 statements. This number cannot be compared directly with the results obtained by Knuth for FORTRAN programs (where 41% of all statements were ASSIGNMENTs) because it includes calls to subprograms (similar to CALL statements in FORTRAN), as well as calls to input and output routines (which perform the function of FORTRAN READs and WRITEs). The total percentage of Knuth's ASSIGNMENT, CALL, READ, and WRITE statements was a suggestive 58%.

if statements constitute 20% of the C statements examined, representing a higher percentage than that found by Knuth (14.5%). Where 13% of FORTRAN statements are GOTOs, the percentage was 3% in our C code (as should be expected due to the availability of {}, elses, and while loops in C).

Return statements constitute 7% of all C statements, with 76% of them returning expressions. There are 1.3 returns per function.

We define a basic block in C to be a sequence of expression statements. The average size of a basic block is 2.2. There are, on the average, four statements in a compound statement in C.

The average user-written C function has 1.2 arguments. The average function call has 1.5 arguments. Of the 6200 function calls made, 18% had no arguments, 37% had one, 30% had two, 13% had three, and 2% had four or more.

The expression complexity of the average C expression varies from a Sethi-Ullman number of 1.65 for expressions controlling while loops, to 1.00 for goto expressions. The Sethi-Ullman number for expressions in expression statements is 1.46. The average such expression contains 2.04 operators and 4.71 tokens (for example: a=b op c). This last result appears to differ from Knuth's -- he found that most FORTRAN ASSIGNMENT statements are simple replacements (of the form A=B).

The Sethi-Ullman number for expressions passed to functions was surprisingly small (1.03). In addition, 79% of all arguments were trivial (either constants, strings, or simple variables). This last result indicates that the average Sethi-Ullman number of a non-trivial argument is 1.22 -- still very small. Thus it may be efficient to pass as many arguments in registers as possible.

The most popular operator in C is, not surprisingly, the assignment operator: it occurs approximately 6200 times (.34 times per C statement as opposed to .36 times per card

DATE FILE COPY
BELL TELEPHONE LABORATORIES

of FORTRAN code). Next come two operators unknown in FORTRAN: the increment, either prefix or postfix (.14/statement), and indirection through a pointer (.11/statement). These were followed by the equality comparator (.09/statement); the counts for the FORTRAN .EQ. do not appear in Knuth's study. Other operators occur much less frequently.

Summary

We briefly described a program for collecting statistical information about C programs. We presented some data collected using this program and discussed how they compare with similar data for FORTRAN. Unfortunately, due to the very different natures of FORTRAN and C, it is difficult to base significant conclusions on such comparisons.

There is a variety of additional data that our program could be modified to collect. The amount of sharing of identifiers in a basic block seems most interesting. Also of interest would be the average number and complexity of arguments passed to printf.

The data collected are static, i.e. they represent the numbers of occurrences of certain constructs in code. It would be interesting to see how these numbers compare with dynamic results, based on counting the constructs as they get executed.

Acknowledgements

I would like to thank Steve Johnson for suggesting the problem, and for his continued help and guidance. I am also grateful to other members of Center 127 for their interest and encouragement.

MH-1273-HG-sc

Hania Gajewska

Hania Gajewska

Att.
References
Appendices A,B,C

References

[1] D.E. Knuth, Empirical Study of FORTRAN Programs, Software -- Practice and Experience, 1:2, 105-133 (1971)

[2] B. A. Wichman, Some Statistics from ALGOL Programs, National Physical Laboratory, Central Computer Unit Report, 1970.

[3] D. M. Ritchie, C Reference Manual, TM 74-1273-1

[4] M. E. Lesk, Lex -- A Lexical Analyzer Generator, TM 75-1274-15

[5] S. C. Johnson, Yacc: Yet Another Compiler-Compiler, Computing Science Report No. 32, 1975, Bell Laboratories.

[6] R. Sethi and J. D. Ullman, The Generation of Optimal Code for Arithmetic Expressions, J. ACM 17:4, 715-728.

DATE FILE COPY
BELL TELEPHONE LABORATORIES
INCORPORATED

Appendix A -- Computing the Sethi-Ullman Number

We want to measure the complexity of an arithmetic expression by determining the number of registers required to evaluate the expression with no stores. An algorithm to do this for a machine model with binary memory-register or register-register operators has been described by Sethi and Ullman [6]. We modify this algorithm to better approximate the number of registers required on the PDP-11, where certain operations (such as multiplication) always require two registers.

The original algorithm was:

> Given a binary expression tree, label a leaf "1" if it is a root or the left son of its father; label all other leaves "0". For each interior node n with sons with labels i and j, label n with max(i,j) if i≠j; label n with i+1 otherwise. Let su (n) be the value of the label of n.

The modifications are as follows:

1. Unary operators -- label a node n representing a unary operator (i.e. with no right son) with the label of its left son.
2. Operators requiring two registers -- label a node n representing "*", "/", or "%" with max(su (n), 2).
3. "&&", "||", and "?", ":" -- label a node n representing any of these operators with max(i,j), where i and j represent the labels of the sons of n (i.e. do not add 1 if i=j).

We treat function calls as if they required only one register. This is under the assumption that function calls would be evaluated first and their values left in registers.

In building the tree for a[e], we consider "[" a binary operator with sons a and e, and we ignore the "]". The complexity of a subscripted variable is computed from such a tree as if "[" were a binary operator. Thus the modified Sethi-Ullman number (msu) for a[e] is equal to msu(e) if msu(e)>1; is 2 if msu(e)=1; and is 1 if msu(e)=0 (i.e. when e is represented by a leaf).

It should be stressed that the msu can be treated only as an approximation. There are many instances where the actual number of registers required may vastly differ from this approximation. For example, we do not take into account any hidden multiplications that may be necessary when operating on arrays of structures, and we do not require two registers for floating point numbers.

DATE FILE COPY
BELL TELEPHONE LABORATORIES

Appendix B -- Tables of Results

operators	occurrences
==	1797.00
!=	842.00
<	975.00
>	464.00
<=	287.00
>=	321.00
&&	585.00
||	299.00
&	407.00
|	92.00
^	1.00
<<	58.00
>>	84.00
unary &	619.00
unary *	2219.00
unary -	436.00
!	181.00
~	49.00
sizeof	154.00
--	469.00
++	2573.00
*	254.00
/	117.00
+	791.00
-	481.00
%	60.00
=	6427.00
=-	97.00
=&	63.00
=*	21.00
=+	276.00
=/	11.00
=%	8.00
=|	75.00
=^	7.00
=<<	13.00
=>>	6.00
a?b:c	92.00
a.b	885.00
a->b	877.00

DATE FILE COPY
BELL TELEPHONE LABORATORIES
INCORPORATED

keywords	occurrences
int	1995.00
char	1750.00
float	73.00
double	33.00
struct	375.00
auto	2.00
static	55.00
extern	95.00
register	681.00
long	71.00
return	1349.00
goto	517.00
if	3739.00
else	458.00
switch	156.00
break	378.00
continue	486.00
while	954.00
do	88.00
for	568.00
default	65.00
case	917.00

expression complexities

exp location	average number of operators	tokens	Sethi-Ullman number
func args	0.29	1.49	1.04
array ind	0.17	1.26	1.01
if exp	2.30	5.25	1.31
for exp 1	1.22	3.36	1.31
for exp 2	1.70	4.13	1.41
for exp 3	1.06	2.19	1.30
initexp	0.00	1.01	1.00
while exp	3.21	6.18	1.65
do exp	2.16	4.50	1.40
switch exp	1.26	2.96	1.26
return exp	0.58	2.04	1.11
goto exp	0.00	1.00	1.00
case exp	0.00	1.00	1.00
exp stat	2.05	4.72	1.46

function arguments

args per func call	1.45
args per user-written func	1.23
funcs with 0 args	0.18
funcs with 1 arg	0.38
funcs with 2 args	0.30
funcs with 3 args	0.12
funcs with 4 or more args	0.02
simple args:	
total	0.79
constants	0.22
strings	0.17
simple ident	0.39

other data

number of lines	35491.00
number of stmts	19099.00
number of funcs	1055.00
function size -- lines	33.64
function size -- stmts	18.10
basic block size	1.62
cmpd stmt size	4.08
labels	241.00
null stmts	187.00
simple fors (for(;;))	69.00

DATE FILE COPY
BELL TELEPHONE LABORATORIES
INCORPORATED

classes	internal occurrences	external occurrences
automatic	2242.00	0.00
ext decl	3232.00	3.00
ext def	0.00	2608.00
static	68.00	0.00
label	241.00	0.00
mem of struct	105.00	780.00
parameter	1295.00	0.00
struc name	1.00	87.00
register	0.00	0.00
field	0.00	0.00

types	occurrences	occur. of func ret	occur. of array of	occur. of pointer to
char	281.00	1.00	361.00	1163.00
int	2844.00	4155.00	405.00	201.00
float	91.00	11.00	4.00	201.00
double	39.00	9.00	3.00	0.00
struct	161.00	0.00	44.00	192.00
long	76.00	7.00	2.00	8.00
label	241.00	0.00	0.00	0.00
function		0.00	0.00	17.00
array		0.00	7.00	0.00
pointer		25.00	99.00	211.00

type of statement	number of occurrences	fraction of all stats.
expression stmt	10765.00	0.56
if stmt	3739.00	0.20
return stmt	1349.00	0.07
while stmt	954.00	0.05
for stmt	568.00	0.03
continue stmt	486.00	0.03
goto stmt	517.00	0.03
break stmt	378.00	0.02
null stmt	187.00	0.01
switch stmt	156.00	0.01
do stmt	88.00	0.00

Appendix C -- Count Names

The following are the count names recognized by output:

TEQ	the number of	==	operators
TNE	" " "	!=	"
LT	" " "	<	"
GT	" " "	>	"
TLE	" " "	<=	"
TGE	" " "	>=	"
TANDAND	" " "	&&	"
TOROR	" " "	||	"
EQMIN	" " "	=-	"
EQAND	" " "	=&	"
EQSTAR	" " "	=*	"
EQPLUS	" " "	=+	"
EQSLASH	" " "	=/	"
EQMOD	" " "	=%	"
EQOR	" " "	=|	"
EQCAR	" " "	=^	"
EQLS	" " "	=<<	"
EQRS	" " "	=>>	"
ASSIGN	" " "	=	"
AND	" " "	&	"
OR	" " "	|	"
CAR	" " "	^	"
POINT	" " "	&	" (unary)
AT	" " "	*	" (unary)
CHSIGN	" " "	-	" (unary)
NOT	" " "	!	"
COMPL	" " "	~	"
TSIZEOF	" " "	sizeof	"
TDECR	" " "	--	"
TINCR	" " "	++	"
TSTREF	the number of	a->b	constructs
ST_TERM	the number of	a.b	constructs
MULT	the number of	*	operators
DIVIDE	" " "	/	"
ADD	" " "	+	"
SUB	" " "	-	"
MOD	" " "	%	"
TINT	the number of	int	keywords
TCHAR	" " "	char	"
TFLOAT	" " "	float	"
TDOUBLE	" " "	double	"
TSTRUCT	" " "	struct	"
TAUTO	" " "	auto	"
TSTATIC	" " "	static	"
TEXTERN	" " "	extern	"
TREGISTER	" " "	register	"
TLONG	" " "	long	"
TSHORT	" " "	short	"
NULL	the number of	null statements	
LABELS	the number of	labels	

DATE FILE COPY
BELL TELEPHONE LABORATORIES
INCORPORATED

TRETURN	the number of return keywords
TGOTO	" " " goto "
TIF	" " " if "
TELSE	" " " else "
TSWITCH	" " " switch "
TBREAK	" " " break "
TCONTINUE	" " " continue "
TWHILE	" " " while "
TDO	" " " do "
TFOR	" " " for "
TDEFAULT	" " " default "
TCASE	" " " case "
NOARGFUNCS	the number of function calls with no args
ARGFUNCS	the number of function calls with args
FUNCARGS	the number of function arguments
ARARGS	the number of array indices
FOREXP1	the number of for exps in the first position
FOREXP2	" " " for " in the second position
FOREXP3	" " " for " in the third position
INITEXP	" " " intializing expressions
WHIEXP	" " " while "
DOEXP	" " " do "
SWEXP	" " " switch "
RETEXP	" " " return "
GOTOEXP	" " " goto "
CASEEXP	" " " case "
EXPSTAT	" " " expression statements
BLOCK	the number of basic blocks
COMPOUND	the number of compound stmts
QUESTCOL	the number of ?: operators
TLS	" " " << "
TRS	" " " >> "
N_LINES	the number of lines
N_FUNCS	the number of functions
IFEXP	the number of if expressions
SFOR	the number of simple for stmts
STAT	the number of statements