MH

UNPL 1426

Bell Laboratories

Cover Sheet for Technical Memorandum

Title: **cq — A Program for Testing C Compilers**

Date: **1979 May 14**

Other Keywords: **C**
Software Testing

TM: **79-2524-3**

Author(s)
F. T. Grampp

Location
MH 2C-253

Extension
3910

Charging Case: **70107-6**
Filing Case: **40125-3**

ABSTRACT

cq is a C program that performs a coarse check on the quality of a C compiler by comparing the behavior of the compiler to that which is advertised in the C Reference Manual. It is designed to run on almost any two's complement machine, and assumes no support from an underlying operating system, except for the availability of a *printf* function. This memorandum describes cq from a designer's point of view. Specifically, it is not a users' manual — that is an integral part of cq.

Pages Text: 7	Other: 0	Total: 7	
No. Figures: 0	No. Tables: 0	No. Refs.: 3	

E-1932-U (3-76)

SEE REVERSE SIDE FOR DISTRIBUTION LIST

BELL TELEPHONE LABORATORIES, INC.

TM-79-2524-3

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

ABAZI,OBI
+BARTON,M E
+BEYER,DONALD W
BROWN,W STANLEY
FRASER,A G
+FREEMAN,R DON
+JOHNSON,STEPHEN C
+KASPER,F J
+KERNIGHAN,BRIAN W
+LESK,MICHAEL E
+LONDON,THOMAS B
+MARTELLOTTO,N A
+MAST,C A
MCILROY,M DOUGLAS
+MITZE,ROBERT W
MORGAN,SAMUEL P
+PRIEVE,BARTON G
+REISER,JOHN F
+RIDDLE,GUY G
+RITCHIE,D M
+ROSLER,LAWRENCE
+SOVEGNO,HELEN D
+ROWLAND,BRUCE E
+SCANLON,J M
+SETHI,RAVI
TERRY,MILTON E
+USAS,ALAN M
+VACCARELLIS,ROBERT E
28 NAMES

COVER SHEET ONLY TO

CORRESPONDENCE FILES

4 COPIES PLUS ONE
COPY FOR EACH FILING
CASE

AAGESEN,JOHN
ACKERMAN,J T
AGARWALA,PRAMILA
AHO,ALFRED V
AHRENS,RAINER B
ALCALAY,C
ALMQUIST,R P
AMSTER,S J
ANDERSON,KATHRIN J

COVER SHEET ONLY TO

ANDERSON,MILTON M
APPELBAUM,MATTHEW A
ARMSTRONG,D B
ARNDT,DENNIS L
ARNOLD,GEORGE W
ARNOLD,JAMES C
ARNOLD,PHYLLIS A
ARNOLD,THOMAS F
BAGGA,YUDHVEER S
BAILY,DAVID E
BAIN,WILLIAM LAMAR,JR
BALLANCE,ROBERT A
BALLARD,E D,JR
BARCLAY,DAVID E
BARNHARDT,KARL R
BAROFSKY,ALLEN
BARON,ROBERT V
BASZIL,RICHARD J
BAUER,BARBARA T
BAUER,H C
BAUER,HELEN A
BAUER,WOLFGANG F
BEAUMONT,LELAND R
BEBLO,WILLIAM
BECKER,RICHARD A
BECKETT,J T
BECKNER,MARK W
BEGLEY,ALOYSIUS A
BEIGHLEY,KEITH A
BENISCH,JEAN
BENNETT,RAYMOND W
BENNETT,WILLIAM C
BERGLAND,G D
BERNSTEIN,DANIELLE R
BERNSTEIN,L
BERRYMAN,K D
BEYER,JEAN-DAVID
BEYLER,ERIC
BICKFORD,NEIL B
BILOMOS,R M
<BIREN,IRMA B
BISHOP,J DANIEL
BLANCHARD,JOETTE D
BLAZIER,S D
BLINN,J C
BLOSSER,PATRICK A
BLOZ,JAMES L
BLUMER,THOMAS P
BLUM,MARION
BODEN,F J
BOEHM,KIM R
BOGART,THOMAS G
BOITIE,RICHARD H
BOLSKY,MORRIS I
BORDELON,EUGENE P
BORISON,ELLEN A
BOULIN,D M
BOURNE,STEPHEN R
BOWYER,L RAY
BOYCE,R M
BOYER,PHYLLIS J

COVER SHEET ONLY TO

BOYLE,GERALD C
BRADLEY,M HELEN
BRENSKI,EDWIN F
BRESLER,RENEE A
BRIGGS,GLORIA A
BROER,H W
BRONZO,JOSEPH A
BROOKS,CATHERINE ANN
BROSS,JEFFREY D
BROWMAN,INNA
BROWN,ELLINGTON L
BROWN,W R
BULPER,A F
BURGESS,JOHN T,JR
BURG,F M
BURNETTE,W A
BURROFF,STEVEN J
BURROWS,THOMAS A
BUSCH,KENNETH J
BUTLETT,DARRELL L
BYERLEE,R W
BYRNE,EDGARD R
CAMPBELL,JERRY H
CANADAY,RUDD H
CANNON,LAINE W
CARR,RICHARD G
CARTER,DONALD H
CASPERS,BARBARA E
CASTELLANO,MARY ANN
CAVINESS,JOHN D
CERMAK,I A
CHAI,D T
CHAMBERS,B C
CHAMBERS,J M
CHANEY,F W
CHANG,JO-MEI
CHAPPELL,S G
CHENG,Y
CHEN,STEPHEN
CHERRY,LORINDA L
CHILDS,CAROLYN
CHIN,KATHLEEN E
CHI,M C
CHODROW,M M
CHOU,PAO-LO P
CHRISTENSEN,S W
CHRIST,C W,JR
CICHINSKI,STEVEN
CIESIELSKI,DEBRA F
CLARK,CONSTANCE E
CLARK,DAVID L
CLARK,EDNA L
CLAYTON,D P
CLINE,LAUREL M L
COBEN,ROBERT M
COCHRAN,ANITA J
COFFMAN,JAMES E
COHEN,SALVEY
COHEN,RICHARD L
COHOON,JAMES P
COLABBARO,CHRISTINE A

COVER SHEET ONLY TO

<COLE,LOUIS M
COLE,MARILYN O
COLLICOTT,R B
COLLINS,J P
CONKLIN,DANIEL L
CONNERS,RONALD E
COOK,DIANA
COOK,JOEL M
COOK,T J
COOPER,ARTHUR E
COPP,DAVID H
COTTRELL,JENNIE L
CRACOVIA,E V
CRAGUN,DONALD W
CRISTOFOR,EUGENE
CRUPI,JOSEPH A
CSENCSITS,BRENDA M
CUNY,THOMAS E
DAVISON,JOSEPH W
DAVIS,D E
DAVIS,R DREW
DE FAZIO,M J
DE GRAAF,D A
DE TREVILLE,JOHN D
DEAN,JEFFREY S
DENNY,MICHAEL S
DENSMORE,SUSAN
DENTON,E T
DESMOND,JOHN PATRICK
DEVLIN,SUSAN J
DI PIETRO,R S
DIB,GILBERT
DIESEL,MICHAEL E
DIMMICK,JAMES O
DINEEN,THOMAS J
DOEDLINE,BARBARA ANN
DOLATOWSKI,VIRGINIA M
DOLOTTA,T A
DOMANGUE,JAMES C,JR
DOMBROWSKI,P J
DOWDEN,DOUGLAS C
DOWDEN,IRIS S
DRAKE,LILLIAN
DREIZLER,H K
DROSEIKIS,FREDERICK C
D'ANDREA,LOUISE A
DUFFY,P P
DUGGER,DONALD D
DUMAIS,VALERIE
DUNCANSON,ROBERT L
DWYER,T J
DYER,MARY E
EDELSON,DAVID
EDMUNDS,T W
EICHORN,KURT H
EISELE,RONALD C
EITELBACH,DAVID L
ELLIS,DAVID J
ELY,T C
EPLEY,ROBERT V
ERICKSON,ROBERT L

COVER SHEET ONLY TO

ESCOLAR,CARLOS
ESPOSITO,VINCENT L
ESSERMAN,ALAN E
ESTVANDER,R A
EVANS,J G
FABISCH,M P
FABRICIUS,WAYNE N
FAIRCHILD,DAVID L
FAULKNER,ROGER A
FEAT,MARY R
FEDER,J
FELLIN,JEFFREY K
FERBER,NANCY L
FILDES,NEAL R
FINK,BERNICE A
FISCHER,HERBERT B
FISCHER,MICHAEL T
FISHER,EDWARD R
FISHMAN,DANIEL H
FLANDRENA,R
FLEISCHER,H I
FLEISLEBER,RAYMOND C
FLOHIN,DONALD H
FONG,K T
FORTNEY,V J
FOUGHT,B T
FOUNTOUKIDIS,A
FOWLER,BRUCE E
FOWLER,C F
FOWLER,GLENN D
FOWLER,H EUGENE
FOX,PHYLLIS A
FOY,J C
FRANK,AMALIE J
FREEMAN,K G
FREEMAN,MARTIN
FREMON,R C
FROST,H BONNELL
FRUCHTMAN,BARRY
GABBE,JOHN D
GARRISON,GARY A
GARVIN,JAMES B
GATES,G W
GAWRON,L J
GAY,FRANCIS A
GEARY,M J
GEERS,T J,JR
GEORGEN,MICHAEL R
GEPNER,JAMES P
GEYLING,F T
GIBB,KENNETH B
GIBSON,J L
GILKEY,THOMAS J
GILL,F J
GIMPEL,J F
GITHENS,J A
GITHENS,JAY L
GLASSER,ALAN L
GLUCK,F G
GOGUEN,N H
GOLABEK,RUTH T

...
706 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-DPH 252-DPH 2524

COVER SHEET TO:
COPPDB = COMPUTER PROGRAM DEBUGGING, MEASUREMENT AND TESTING
COPRSQ = COMPUTER SOFTWARE QUALITY

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.
4. INDICATE WHETHER MICROFICHE OR PAPER IS DESIRED.

aC CORRESPONDENCE FILES
HO 5C101

TM-79-2524-3
TOTAL PAGES 7

PLEASE SEND A COMPLETE

( ) MICROFICHE COPY ( ) PAPER COPY

TO THE ADDRESS SHOWN ON THE OTHER SIDE.

Bell Laboratories

subject: cq – A Program for Testing C Compilers
Case: 70107-6
File: 40125-3

date: 1979 May 14

from: F. T. Grampp
MH 2524
2C-253 x3910

TM: 79-2524-3

ABSTRACT

cq is a C program that performs a coarse check on the quality of a C compiler by comparing the behavior of the compiler to that which is advertised in the C Reference Manual. It is designed to run on almost any two's complement machine, and assumes no support from an underlying operating system, except for the availability of a *printf* function. This memorandum describes cq from a designer's point of view. Specifically, it is not a users' manual – that is an integral part of cq.

MEMORANDUM FOR FILE

The C programming language was originally developed for use in conjunction with the UNIX™ operating system. For a number of reasons, all of which can be summed up by saying that the language is both useful and highly portable, C has proliferated widely and rapidly both inside and outside Bell Labs. In the Bell System, it is the language in which most programs are written for UNIX systems (which now number in the hundreds) and it is available and widely used in many other environments, including Honeywell and IBM systems, ESS 3B processors, and a number of mini- and microcomputer applications.

The compilers we are presently using are, for the most part, either that which was written by Dennis Ritchie for the DEC PDP-11, or various implementations of Steve Johnson's Portable C Compiler. Also, we are aware of several C compilers that are being developed outside of the Bell System, and it is only a matter of time before these compilers start showing up in connection with products purchased from outside vendors.

It was fairly obvious that some way was needed to check the quality of the C compilers we use, both to guard against the propagation of unsuspected errors via new releases of our own compilers, and against substandard products being introduced from outside. It was felt that the latter possibility was particularly likely, if for no other reason than that such products would be relatively new, and hence not thoroughly shaken out.

Accordingly, it was decided that a program, cq, would be written to test C compilers. This program would be compiled and executed, and the results of execution would then be checked to see whether they were as expected. Failure of a compiler to successfully compile the program would be considered an error. During the course of execution, messages pertaining to deviation from expected behavior or to certain peculiarities of implementation would be published for the enlightenment of the user.

Certain characteristics of cq were abundantly clear very early in the design process:

1. C is a programming language. Although its development and the development of the UNIX operating system were closely intertwined, and people tend to associate one with the other, C and UNIX are quite independent. Provided that a compiler exists, one can write C programs to run under UNIX, or some other operating system, or for that matter, with no operating system at all. It follows from this that any test package for a C compiler would have to be as independent as possible from the target machine, the underlying operating system, and the implementation mechanics of a particular C environment. In particular, most assumptions about things like word size, all UNIX system calls, the use of library functions, and programming "tricks" based on things like the loading order of object modules or the layout of *a.out* files were to be strictly *verboten.*

2. C is currently being used to program microcomputers, and it is reasonable to expect this use to continue and to become more prevalent. The storage capacity of some of these machines is tiny, even when compared with a "minicomputer" like a PDP 11/70, yet provision had to be made for the tests to be run in such an environment. This would require a high degree of modularity in cq, so that the package could be taken apart and run in pieces on small machines. Consideration of possibly limited storage facilities and software support in the micro environment also supported previous feelings about dependence on operating system and library facilities.

3. Several C compilers are either available from, or are being written by sources outside of the Bell System. It was felt that a clean set of compiler tests would be extremely useful in cases where outside vendors wanted to sell their compilers to Bell System users. If the tests were to be used as a basis for the rejection of flaky compilers, it was essential that the tests be based only on information that was easily available to the general public and not on proprietary information. For this reason, the tests were based only on information contained in the book by Kernighan and Ritchie,[1] which is readily available to anyone through any bookstore. This approach has the disadvantage that it is difficult to respond rapidly to new C features developed at Bell Labs; the disadvantage, however, is outweighed by the fact that an outside vendor cannot use "lack of private information" as an excuse for a poor product.

4. It is expected that when the tests are needed, they will be needed quickly, probably by people who haven't used them previously and who may not have immediate or convenient access to someone who is familiar with them. Hence, the package must contain sufficient documentation for compiling and running, for "elective surgery" should this prove necessary, and for the interpretation of results.

It was also obvious that there were certain fundamental limitations that could not be circumvented and would have to be lived with:

1. For several reasons, none of which will be covered in detail here,[2] it is simply not possible to devise a set of tests to show the absence of bugs in a C compiler. Problems are going to slip by undetected, and if particularly troublesome cases arise in the future, it may be necessary to add to the tests to handle these cases. (One should be very careful about using words like "verification" and "correctness" in connection with cq, especially if there is any likelihood that such words might be interpreted in a formal sense.)

2. In addition to compiling correct programs, any compiler has an equally important job: that of rejecting incorrect programs. Compilers that ignore errors, or abort, or publish cryptic diagnostics, are unlovely beasts indeed, and should not be offered to the public as programming tools. It is unclear how to test that error detection is being handled properly, if, in fact, it is possible to make such a test at all. The output of C compilers is not defined for erroneous programs, other than that they should say "I didn't like that." in some fashion.

1. Kernighan, B. W., and Ritchie, D. M., *The C Programming Language*, Prentice-Hall, 1978.

2. See Hoare, C.A.R., and Allison, D.C.S, *Incomputability*, Computing Surveys, 1972 September, P169.

The number and content of diagnostics, and the format and destination thereof is unspecified, as is communication, if any, with the operating system, and with the control logic that determines whether processing will continue after an error has been encountered. Thus, while it may be possible to systematically test the error handling capabilities of a particular compiler running under a particular operating system, the best that can be done in the general case is to collect a body of incorrect programs and have a person feed them into the compiler, one at a time, and afterwards make a subjective evaluation of the suitability of the compiler's defenses. Since such a procedure is entirely unsatisfactory, and there appear to be no alternatives, this important aspect of compiler testing had to be ignored in the design of cq.

Program structure

The package consists of a number of files (currently 26) in a single UNIX directory. The files contain:

Documentation
A driver module
Some definitions
A single UNIX-specific module
A flock of test modules

The files are named so that they will be accessed in that order when they are referenced by UNIX commands. For example, pr * will print the files, in order of importance to a user, and in a form suitable for binding into an 8.5 x 11 inch operating manual. Alternatively, if the tests are to be run "in place" the user need only type in "cc *.c; a.out" or a suitable variant.

Documentation

A file called README contains general information about the package. It describes the overall structure of the program, and tells how to compile and run it, and what to do if modules fail to compile or execute. Finally, a detailed (and hopefully helpful) explanation is given for each of the error messages that can be produced by cq.

Documentation that is specific to individual modules appears in those modules.

The driver module

The file crm.c is the main() function for the package. Its purpose is to call the test modules in sequence, and to provide communication among modules.

The module consists of three sections. The first contains a couple of pages of commentary describing in detail how the program is put together, how test modules are called, and how to take the package apart or add to it. The second section contains a list of all test modules to be called and a structure defining the environment in which the modules are to operate. Finally, the half-dozen or so lines at the end contain the program logic, which, as expected, calls the modules in the list, accumulates return code information, and announces success or failure of the tests.

The interface between the driver module and the test modules has been designed to facilitate adding or deleting test modules. The list referred to above is actually a sequence of external function definitions. The driver module calls each test module in sequence by sequencing over the list with a *for* loop containing the funny-looking statement:

```
d0.rrc = (*sec[j])(pd0);
```

which effectively says "call the jth module in the list, passing it a pointer to a description of its operating environment, and then stuff its return code in a place called "d0.rrc". This is the only

really cryptic line of code in the entire package, and its justification is that it makes addition or removal of a module as simple as the insertion or removal of its name in a list.

Definitions.

The driver module contains a structure that defines the working environment of the package: sizes of primitive data types, precision of the floating point hardware, flags dictating the verbosity of the output, and so on. A pointer to this structure is passed to each module when the module is called. Since the structure is used by most of the modules in the package, it is defined in a file called "defs" which is then *#included* in each module that uses it. Should these definitions need to be changed, they can be changed in only one place.

UNIX-specific code

In any testing procedure, it is necessary to specify what is to be done, and to record certain particulars about the test, such as when and where the test was conducted. The easiest way to specify options is to include them in the command line that invokes the program. Similarly, the easiest way to record the time and date of a test is to have the program itself retrieve and print this information. Unfortunately, the facilities to do things like this are not part of the C language, but are supplied by the underlying operating system. While most operating systems supply such facilities, their use will probably not be the same from one system to the next; hence a program that depends on them is non-portable.

Since these features are very convenient, and since it was anticipated that cq would be heavily used under UNIX or in a UNIX-like environment, a single, non-portable module, "options.c" was provided to interpret options on a command line, and to print the time, date and system identification along with the rest of the output from cq. This routine is called once from the main() routine. The line that calls it can be removed in a non-UNIX environment, or if the routine causes trouble for any reason.

Test modules

Actual testing is done by a collection of test modules, all of which are called by crm.c. The organization of the modules is modeled after that of the C Reference Manual,[3] and the modules' names are suggestive of the corresponding manual sections. Thus module s22.c corresponds to section 2.2 in the manual. The correspondence is not isomorphic. There are some numbered sections of the manual for which there exist no corresponding modules (e.g., section 3, which describes the syntax notation used in the manual), and there are several cases in which tests based on a group of small, closely related manual sections have been combined into a single code module. Nevertheless, finding the text in the manual that corresponds to a given test, or vice versa, is quite painless.

When a module is invoked, it is passed a pointer to a structure that describes the environment in which the module is to operate. The structure contains flags that dictate the verbosity of the module, descriptions of the underlying machine hardware, and an eight-character field into which the module writes its name for the benefit of the calling program. When a module exits, it returns an *int* which is either zero if all went well, or the bitwise *or* of one or more flags indicating that something is not as advertised.

Limiting the number of distinct errors that can be passed back from a module to the number of bits that can be contained in an *int* may at first seem somewhat restrictive. It turns out, though, that the possibility of a module returning more than four or five distinct error messages is usually an indication either that the module is trying to do too much in one place and ought to be broken into smaller pieces, or that too much detail is being returned to the calling program. As an example of the latter case, one of the modules (s714.c), that tests, exhaustively, all possible combinations of assignment operators and primitive data types. If one or more of these tests fail, the module simply reports failure to the calling program. More extensive detail can be obtained by looking at local return codes printed by the module itself.

3. *Appendix A* of Kernighan and Ritchie, *op. cit.*

Error conditions are communicated to the user by means of terse messages that display the name of the module in which the error was detected and a number corresponding to the error. This is an ancient and ugly technique, and I have always regarded as obnoxious those programs that use it. In this case, it was unavoidable for two reasons. First, there is a big difference between being able to discover that something is wrong, and being able to devise an effective procedure capable of correctly diagnosing the causes for an anomaly. The former task is easy; the latter, impossible. In fact, the only reasonable strategy is to point the user of the package at the place where a problem surfaced, so that he can then do whatever sleuthing is appropriate, and the numbered messages do this quite well. Second, even if it were possible to construct a meaningful set of detailed diagnostics, these diagnostics would have to be stored somewhere. Storing them in the program is at odds with the goal of keeping modules small enough so that they can be run on tiny machines, while storing them externally requires that the program be dependent on some specific file system.

Most of the modules are fairly small. The two exceptions are s714.c and s7813.c which test, respectively, assignment and bitwise logical operators. These modules were for the most part generated by fairly simple-minded SNOBOL programs, and perform rather exhaustive tests. Like most machine generated programs, they are not particularly pretty to look at, although the commentary at the beginning of each eliminates the need for more than a quick peek.

The characteristic of the code in the test modules which is perhaps the most striking at first glance is that the behavior of the various types of C *statements* is explicitly checked in only a very few instances, and that the most attention is given to whether the *meaning* of data types, constants, conversions and operators is being interpreted properly. Making statements work right is easy — perhaps the easiest part of the compiler writer's job, whereas the other tasks are extremely difficult. Hence the amount of attention paid to the latter aspects is justified on account of their enormous potential for causing trouble. Moreover, if it were the case that something were seriously wrong with the way that statements were handled, it would be extremely unlikely that a program like cq with several thousand lines of code (or for that matter, most other programs) would run at all. Finally, in a few pathological cases, the C Reference Manual's authors have wisely opted for clarity rather than precision, and their description is not exact. For example,

```
for(expression_1; expression_2; expression_3) statement
```

is almost, but not quite equivalent to:

```
expression_1;
while(expression_2){
        statement
        expression_3;
}
```

but the fact that this description is inexact does not diminish its utility for most purposes. The fuzzyness of the language description at this level, however, precludes the construction of finer tests.

Tests of data objects, operators and conversions were constructed following the dictates of common sense. Specific examples that were given in the Manual were copied bodily. Where it was reasonable to do exhaustive testing, exhaustive testing was done. In cases where it was impractical to test something exhaustively, test conditions most likely to produce anomalies were used. Programs that were known to have caused problems in the past were also included.

It was recognized rather early that some minor (and tolerable) variations in the behavior of cq were to be expected, both because of the informal specification of the language, and also because of the hardware differences in the machines on which cq might be run. While such variations could not be considered errors, news of them would provide important information about both the subject compiler and the underlying machine. Accordingly, in addition to reporting errors, cq provides some commentary on the following:

1. The size and alignment of primitive data types.
2. The precision of floating point operations.
3. Whether or not sign extension occurs when chars or bit fields are promoted to objects of larger size.
4. The number of registers actually available for C's "register" storage class.

(Given the constraints under which cq must operate, getting the information for the last item turns out to be a fairly entertaining programming puzzle, and the reader is encouraged to try it. A solution can be had by looking in section s81 of cq.)

User Experience

The fact that tests may show the presence of bugs, but cannot show their absence makes it extremely difficult to evaluate a program like cq. The situation is further complicated by the fact that testing thus far has been limited to two compilers: Dennis Ritchie's cc compiler, and Steve Johnson's pcc (portable) compiler. Both of these compilers were well-worn in the sense that they had been used extensively for production purposes and thus had been more or less thoroughly shaken out by the time that cq was first run. (In the case of some implementations of pcc, the machine-dependent sections were relatively new.)

cq was tried on the following machine-compiler combinations:

DEC PDP 11/45 – cc
DEC PDP 11/45 – pcc
DEC PDP 11/70 – cc
DEC PDP 11/70 – pcc
DEC VAX 11/780 – pcc
Honeywell 6000 – pcc
IBM System/370 – pcc
Interdata 8/32 – pcc

with the following results:

1. No problems were experienced that could be attributed to nonportability of cq.
2. In no case did cq run correctly on the first try, in that at least one bug was found in each machine-compiler combination.
3. An impressive variety of bugs was uncovered. These included incorrect code generation, absence of templates for compiling various operator-operand combinations, wrong operator precedence, a couple of cases in which the target compilers aborted, and a floating-point hardware problem.

Thus, even in friendly and familiar surroundings, cq provided a simple way to find some bugs that could have been uncovered much more painfully by future users. It will be interesting to see what happens when *new* compilers appear on the scene.

Acknowledgement

In the course of designing and writing cq, I received extensive help in the form of suggestions, clarifications, encouragement, inspiration and sometimes even condolences from Lee Benoy, Steve Johnson, Brian Kernighan, Andy Koenig, Mike Lesk, Joe Maranzano, Dennis Ritchie, Larry Rosler and Ravi Sethi. To list all of their contributions here would take several pages. Let me simply say that I am very grateful.

14H-2524-FTG

F. T. Grampp