AD-A184 492

NAVAL OCEAN SYSTEMS CENTER, SAN DIEGO, CA
UNIX BENCHMARK SYSTEM BY: T BREWER

1 OF 1
NOSC TD 1111
UNCLASSIFIED
JULY 87

END
DATE
FILMED

NOSC TD 1111

NOSC
NAVAL OCEAN SYSTEMS CENTER San Diego, California 92152-5000

Technical Document 1111
July 1987

UNIX Benchmark System

T. Brewer
Integrated Systems Analysts, Inc.

NAVAL OCEAN SYSTEMS CENTER

San Diego, California 92152-5000

E. G. SCHWEIZER, CAPT, USN
Commander

R. M. HILLYER
Technical Director

ADMINISTRATIVE INFORMATION

This work was performed for Code 914, NAVOCEANSYSCEN. Contract N66001-84-D-0053 was carried out by Integrated Systems Analysts, Inc., Marina Gateway, 740 Bay Boulevard, Chula Vista, California 92010-5254.

Released by
R.L. Broersma, Head
System Software Branch

Under authority of
J. Gilbreath, Head
Computer Sciences and
Simulation Division

MA

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY 2b DECLASSIFICATION DOWNGRADING SCHEDULE	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S) NOSC TD 1111
6a NAME OF PERFORMING ORGANIZATION Integrated Systems Analysts, Inc. 6b OFFICE SYMBOL (if applicable)	7a NAME OF MONITORING ORGANIZATION Naval Ocean Systems Center
6c ADDRESS (City, State and ZIP Code) Marina Gateway 740 Bay Blvd. Chula Vista, CA 92010-5254	7b ADDRESS (City, State and ZIP Code) Systems Software Branch San Diego, CA 92152-5000
8a NAME OF FUNDING SPONSORING ORGANIZATION Naval Ocean Systems Center 8b OFFICE SYMBOL (if applicable) Code 914	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N66001-84-D-0053
8c ADDRESS (City, State and ZIP Code) System Software Branch San Diego, CA 92152-5000	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO: VARI PROJECT NO: VARI TASK NO: 914-VA01 AGENCY ACCESSION NO: DN788 736

11 TITLE (include Security Classification)
UNIX Benchmark System

12 PERSONAL AUTHOR(S)
T. Brewer

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Final	FROM Nov 85 TO May 86	July 1987	61

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	block read, sort, random numbers, matrix inversion, polynomial roots, prime numbers, bench, kernel, test and services, sieve

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

A benchmark suite has been consolidated to test and evaluate a variety of computer systems and compare the results. VAX 11/780, under UNIX 4.3 BSD, has been selected as the baseline system to which each target system (procurement candidates) would be compared. The program, BENCH, collects and stores test results from all the target systems, and produces two reports. The first report compares any two systems that the user selects. The second report summarizes all the test data into a single report.

The suite presently has 18 tests and the user can specify which test may be used by modifying an ASCII file called BENCHLIST. Additional tests may be added to the suite by modifying the BENCHLIST and supplying the appropriate code.

20 DISTRIBUTION AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL R. Broersma	22b TELEPHONE (Include Area Code) (619)225-2148 22c OFFICE SYMBOL Code 914

DD FORM 1473, 84 JAN

83 APR EDITION MAY BE USED UNTIL EXHAUSTED
ALL OTHER EDITIONS ARE OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

DD FORM 1473, 84 JAN

CONTENTS

Introduction

In March 1986, NAVOCEANSYSCEN tasked ISA to prepare a comprehensive test suite to systematically test and evaluate a variety of computer systems and compare the results. NAVOCEANSYSCEN supplied ISA with their Pre-Award test suite, with instructions to also include certain published tests, such as the Dhrystone and Whetstone benchmarks. It was ISA's task to consolidate all the designated tests into one easily-runnable program which can be used by either Government or Contractor personnel to test all computer systems that are candidates for procurement.

NAVOCEANSYSCEN selected the VAX 11/780 under UNIX 4.3 BSD as the baseline system to which each target system (procurement candidates) would be compared. Our program, *bench*, collects and stores test results from the baseline system, collects and stores results from all the target systems, and produces two reports. The first report compares any two systems which the user selects, and the second report summarizes all the test data in one report. The user tells the program which of these tasks he wishes performed through the use of options on the input line. All input is in standard UNIX format. For example,

```
% bench -b
```

would execute the test suite and store the results as the baseline data. See the enclosed man page for a list of all available options.

The suite includes eighteen tests at present. The user is able to specify which of the eighteen tests will be used by modifying an ASCII file called *benchlist*. *Benchlist* includes the names of all the tests. The user adds or deletes the comment indicator to tell the program to include or exclude that particular test. Additional tests may be added to the suite by modifying the *benchlist* and supplying the appropriate code.

Descriptive Summary

The following is a very brief description of the tests in the benchmark suite.

Fortran Tests

1. Prime Numbers

   This program generates the prime numbers from 0 to 8192 (optionally printing out the results).

2. Calling Sequence and Argument Passing

   This program initializes nine variables, passes them to a subroutine, which in turn has four assignments. This sequence is repeated one million times.

3. Random Numbers

   This program tests the random number generator by generating 12,800 random numbers and checking the randomness.

4. Fast Fourier Transform

   This program performs fast fourier transform using the decimation-in-time method (optionally printing out the results).

5. Matrix Inversion

   This program performs matrix inversion using the Gauss-Jordan Reduction (optionally printing out the results).

6. Polynomial Roots

   Roots of polynomials are calculated using the Bairstow's method (optionally printing out the results).

Sieve Tests

7. C Sieve

   C version of the Sieve of Eratosthenes prime number program.

8. Fortran Sieve

   Fortran version of the Sieve of Eratosthenes prime number program.

9. Pascal Sieve

   Pascal version of the Sieve of Eratosthenes prime number program.

General Tests

10. Whetstone

    A C version of the original Algol benchmark, "A Synthetic Benchmark" by H. J. Curnow and B. A. Wichman. Compiler optimization and floating point performance are tested.

11. Dhrystone

    This program contains a distribution of statements which are considered to be representative: 53% assignment, 32% control statements, and 15% procedure and function calls.

Pre-Award Tests

12. Block Write

    This program creates a very large file by writing 8K byte blocks one after the other.

13. Block Read

    This program reads the file created by the block write program. The reads are executed in 8K byte blocks.

14. Sort

    A shell script to test the section 1 sort call. A file is sorted on a particular column and the result is compared to a presorted file to test the results of the sort.

15. Integer Arithmetic

    Addition, subtraction, multiplication, and division are performed on integer variables. The group of operations is executed 2.9 million times.

16. Real Arithmetic

    Similar to the integer arithmetic, this program performs addition, subtraction, multiplication, and division on real variables. This group of operations is executed 600,000 times.

17. Large Data Space

    This program references a data area larger than real memory making 20,000 references to random locations.

18. Compile

    This script compiles two C code files and loads the two object files into a single output file.

Database Format

The benchmark test data is stored in a series of files which reside in the current working directory. The file containing the baseline data is called *baseline*. A file is created for each system tested and is called *targetXXX*, where XXX is a 3-digit number assigned by the benchmark program and which is unique to each system. The format for both the baseline file and all the target files is identical.

The format of the database files illustrated on the following page.

```
DEC
VAX 11/780
4.3 BSD
5
1 1 1
20.08 5.18 0.36
20.04 5.09 0.25
20.04 5.06 0.30
0.51 0.05 0.10
0.19 0.03 0.08
0.50 0.05 0.07
2.18 1.71 0.07
1.99 1.67 0.06
1.97 1.69 0.06
1.71 1.33 0.05
1.61 1.30 0.04
1.61 1.30 0.07
2.46 2.09 0.08
2.36 2.08 0.05
2.43 2.02 0.09
10.05 5.26 0.38
10.07 5.13 0.27
10.09 5.24 0.47
1.02 0.02 0.03
0.58 0.05 0.10
1.28 0.01 0.14
7.41 1.76 0.10
5.20 1.69 0.04
7.78 1.70 0.09
6.25 1.35 0.08
4.32 1.29 0.03
6.56 1.37 0.08
7.86 2.12 0.06
5.73 2.10 0.05
8.66 2.16 0.07
10.04 5.11 0.30
10.08 5.09 0.28
10.04 5.13 0.23
0.71 0.02 0.09
0.76 0.01 0.11
0.76 0.07 0.06
7.45 1.70 0.05
6.49 1.69 0.03
6.70 1.75 0.01
5.49 1.35 0.03
4.85 1.27 0.03
5.03 1.30 0.04
9.09 2.04 0.07
8.40 2.12 0.06
8.21 2.01 0.06
```

EXECUTED ONE AT A TIME

EXECUTED ALL AT ONCE (NO LOAD)

EXECUTED ALL AT ONCE WITH EXTRA LOAD

SYSTEM MANUFACTURER

SYSTEM MODEL NUMBER

OPERATING SYSTEM VERSION

NUMBER OF TESTS, INCLUDING ONE FOR THE COMPOSITE

THREE FLAGS (0 OFF, 1 ON) FOR THE THREE WAYS TO EXECUTE THE SUITE: SINGLY, ALL AT ONCE, ALL AT ONCE WITH A LOAD

EACH LINE CONTAINS THREE NUMBERS:
1. REAL TIME
2. USER TIME
3. SYSTEM TIME

EACH GROUP OF THREE LINES REPRESENTS THE RESULTS FROM ONE TEST AND EACH LINE REPRESENTS ONE REPETITION OF THE TEST. THIS IS SET IN THE C CODE (CURRENTLY REP = 3)

Reports

Bench produces two reports: a comparison report based on two systems of the user's choice, and a summary report which includes all systems tested.

The comparison report is invoked when the user specifies the *-p* option on the command line. *Bench* displays a list of those systems in its database, and prompts the user to choose two systems from the list. The comparison report displays elapsed time, user time, system time, and percent usage for each test and each system chosen. A composite is also displayed. The composite is a sum of all systems chosen and represented as if it were a separate test. The elapsed time is the total amount of time that is consumed; the "clock" time. The user time is the amount of time the process spent executing nonprivileged instructions (e.g., arithmetic calculations, sorting, searching, etc.). System time is the time the process spent executing privileged (kernel) commands, such as system calls, plus some system-level overhead. The percent usage is that portion of the elapsed time that is actually spent executing the command. It is calculated thusly:

$$\text{percent usage} = \left(\frac{\text{system time + user time}}{\text{elapsed time}} \right) \times 100$$

The lowest elasped time for each test for each system is indicated on the report by an asterisk (*). A separate column is displayed for the elapsed ratio. The first figure in the elapsed ratio column is the lowest time ratio, which is the ratio of the lowest elapsed time of the second system to the lowest elapsed time of the first system , or

$$\text{lowest time ratio} = \frac{\text{lowest elapsed time of second system}}{\text{lowest elapsed time of first system}}$$

The second figure of the elasped ratio is the average time ratio. The average time ratio is the average elapsed time of the second system divided by the average elapsed time of the first system, or

$$\text{average time ratio} = \frac{\text{average elapsed time of second system}}{\text{average elapsed time of first system}}$$

At the bottom of the report, an average elasped ratio is given, based on all tests except the composite. The average elapsed ratio is the average of each lowest time ratio (first figure) and the average of each average time ratio (second figure).

The summary report displays elapsed time averages for all systems tested. The test results are normalized to the baseline system. The summary report is invoked when the user specifies the *-e* option on the command line.

Sample output for both comparison report and the summary report follow.

Comparison Report

Test executed one at a time with no extra load
(Expressed in seconds)

	DIGITAL VAX 11/780 4.3 BSD					DIGITAL VAX 11/730 4.3 BSD					
Type	Test Number	Elapsed Time	User Time	System Time	Percent Usage	Test Number	Elapsed Time	Elapsed Ratio +	User Time	System Time	Percent Usage
composite	1	20.08	5.18	0.36	28	1	20.07*		5.26	0.42	28
	2	20.04	5.09	0.25	27	2	20.08		5.18	0.60	29
	3	20.04*	5.06	0.30	27	3	20.09		5.29	0.58	29
	avg	20.05					20.08	[1.00, 1.00]			
uptime	1	0.51	0.05	0.10	29	1	0.76*		0.04	0.11	20
	2	0.91	0.03	0.08	12	2	0.81		0.05	0.11	20
	3	0.50*	0.05	0.07	24	3	1.42		0.02	0.09	8
	avg	0.64					1.00	[1.52, 1.56]			
whetstone	1	2.18	1.71	0.07	82	1	2.66*		1.67	0.09	66
	2	1.99	1.67	0.06	87	2	3.24		1.71	0.16	58
	3	1.97*	1.69	0.06	89	3	3.07		1.77	0.15	63
	avg	2.05					2.99	[1.35, 1.46]			
integer arith	1	1.71	1.33	0.05	81	1	2.97		1.44	0.09	52
	2	1.61	1.30	0.04	83	2	2.62		1.33	0.14	56
	3	1.61*	1.30	0.07	85	3	2.41*		1.38	0.11	62
	avg	1.64					2.87	[1.50, 1.62]			
real arith	1	2.46	2.09	0.08	88	1	2.75*		2.10	0.11	80
	2	2.36*	2.08	0.05	90	2	4.01		2.09	0.15	56
	3	2.43	2.02	0.09	87	3	3.58		2.12	0.18	64
	avg	2.42					3.45	[1.17, 1.43]			
							Average Elapsed Ratios **	[1.11, 1.21]			

* Marks lowest elapsed time for the particular test
** Averages of all the test ratios
\+ Ratios are displayed as the [lowest time ratio, average time ratio]

Comparison Report

Test executed all at once with no extra load
(Expressed in seconds)

	DIGITAL VAX 11/780 4.3 BSD					DIGITAL VAX 11/730 4.3 BSD					
Type	Test Number	Elapsed Time	User Time	System Time	Percent Usage	Test Number	Elapsed Time	Elapsed Ratio +	User Time	System Time	Percent Usage
composite	1	10.05*	5.26	0.38	56	1	10.09		5.40	0.57	59
	2	10.07	5.13	0.27	54	2	10.08		5.31	0.44	57
	3	10.09	5.24	0.47	57	3	10.07*		5.12	0.42	55
	avg	10.07					10.08	[1.00, 1.00]			
uptime	1	1.02	0.02	0.09	11	1	2.57		0.08	0.07	6
	2	0.58*	0.05	0.10	26	2	1.67		0.02	0.11	8
	3	1.28	0.01	0.14	12	3	1.01*		0.02	0.11	13
	avg	0.96					1.75	[1.74, 1.82]			
whetstone	1	7.41	1.76	0.10	25	1	6.89		1.72	0.19	28
	2	5.20*	1.69	0.04	33	2	7.05		1.76	0.13	27
	3	7.78	1.70	0.09	23	3	6.20*		1.70	0.08	29
	avg	6.80					6.71	[1.19, 0.99]o'P			
integer arith	1	6.25	1.35	0.08	23	1	6.26		1.46	0.08	25
	2	4.32*	1.29	0.03	31	2	5.85		1.36	0.05	24
	3	6.56	1.37	0.08	22	3	5.34*		1.30	0.10	26
	avg	5.71					5.82	[1.24, 1.02]			
real arith	1	7.86	2.12	0.06	28	1	7.78		2.13	0.17	30
	2	5.73*	2.10	0.05	38	2	7.35		2.17	0.08	31
	3	8.66	2.16	0.07	26	3	6.73*		2.09	0.10	33
	avg	7.42					7.29	[1.17, 0.98]			
						Average Elapsed Ratios **		[1.07, 0.96]			

* Marks lowest elapsed time for the particular test
** Averages of all the test ratios
\+ Ratios are displayed as the [lowest time ratio, average time ratio]

Comparison Report

Test executed all at once with an extra load
(Expressed in seconds)

	DIGITAL VAX 11/780 4.3 BSD					DIGITAL VAX 11/730 4.3 BSD					
Type	Test Number	Elapsed Time	User Time	System Time	Percent Usage	Test Number	Elapsed Time	Elapsed Ratio +	User Time	System Time	Percent Usage
composite	1	10.04	5.11	0.30	54	1	15.11		5.20	0.42	37
	2	10.08	5.09	0.28	53	2	10.11		5.39	0.47	58
	3	10.04*	5.13	0.23	53	3	10.11*		5.21	0.51	57
	avg	10.05					11.78	[1.01, 1.17]			
uptime	1	0.71*	0.02	0.09	15	1	0.80*		0.08	0.05	16
	2	0.76	0.01	0.11	16	2	1.03		0.04	0.10	14
	3	0.76	0.07	0.06	17	3	1.01		0.07	0.06	13
	avg	0.74					0.95	[1.13, 1.27]			
whetstone	1	7.45	1.70	0.05	23	1	9.83		1.72	0.06	18
	2	6.49*	1.69	0.03	27	2	7.92		1.80	0.07	24
	3	6.70	1.75	0.01	26	3	7.43*		1.76	0.10	25
	avg	6.88					8.39	[1.14, 1.22]			
integer arith	1	5.49	1.35	0.03	25	1	7.92		1.29	0.11	18
	2	4.85*	1.27	0.03	27	2	6.45		1.36	0.12	23
	3	5.03	1.30	0.04	27	3	5.43*		1.34	0.09	26
	avg	5.12					6.60	[1.12, 1.29]			
real arith	1	9.09	2.04	0.07	23	1	10.04		2.11	0.14	22
	2	8.40	2.12	0.06	26	2	9.23		2.16	0.12	25
	3	8.21*	2.01	0.06	25	3	8.52*		2.04	0.17	26
	avg	8.57					9.26	[1.04, 1.08]			
							Average Elapsed Ratios **	[0.89, 0.97]			

* Marks lowest elapsed time for the particular test
** Averages of all the test ratios
\+ Ratios are displayed as the [lowest time ratio, average time ratio]

Summary Report

Elapsed time averages normalized to the baseline

	(baseline) DIGITAL VAX 11/780 4.3 BSD	DIGITAL VAX 11/730 4.3 BSD	DEC VAX 8600 4.3 BSD	DEC VAX 11/750 4.1 bsd
composite				
one at a time	1.0	1.0	1.0	1.0
all at once, no load	1.0	1.0	1.0	1.7
all at once, w/ load	1.0	1.2	1.0	1.3
uptime				
one at a time	1.0	1.6	1.1	3.2
all at once, no load	1.0	1.8	0.7	2.5
all at once, w/ load	1.0	1.3	1.5	6.4
whetstone				
one at a time	1.0	1.5	1.1	1.4
all at once, no load	1.0	1.0	0.8	1.8
all at once, w/ load	1.0	1.2	1.2	1.3
integer arith				
one at a time	1.0	1.6	1.1	1.5
all at once, no load	1.0	1.0	0.8	1.6
all at once, w/ load	1.0	1.3	1.3	1 5
real arith				
one at a time	1.0	1.4	1.3	1.6
all at once, no load	1.0	1.0	0.8	1.8
all at once, w/ load	1.0	1.1	1.0	1.2

NAME

bench - benchmark driver and result comparison generator

SYNOPSIS

bench -b [-v] [-s] [-f] [-l] [-n "make, model, version"]
bench -a [-v] [-s] [-f] [-l] [-o *outfile*] [-n "make, model, version"]
bench -p [-s] [-f] [-l] [-o *outfile*]
bench -e [-s] [-f] [-l] [-o *outfile*]

DESCRIPTION

Bench is a benchmark driver program to time the execution of a suite of tests specified in the ASCII file *benchlist*. The defaults to **bench** are designed to allow a user with little or no understanding of the options to establish a baseline system and create comparisons of other systems with the baseline.

The user can override the defaults by using the options. For example: data can be collected without generating a comparison report; an output filename can be specified for the comparison report; a summary table of all the system results can be generated; and selected groups of tests can be executed without running the complete suite.

The available flags are:

-b Execute the test suite and add the results to the database as the baseline from which comparisons will be produced.

-a Execute the test suite and add the results to the database. Unless used with the -o option, no comparison report will be generated.

-p Prepare a comparison report between two systems of the user's choice. May be used with the -o option; default is to the line printer.

-e Generate a summary table containing normalized elapsed times for all systems in the database. If no system has been assigned as the baseline, the user will be prompted for a system to use as a baseline. Default is to the line printer.

-v Verbose: causes output to be generated to standard output. This information is helpful when trying to follow the progress of the driver. Default is off.

-n "make, model, version"
Use the make, model, and version of the system to identify the results. This option is useful when executing the driver in a batch mode. If this is not specified on the command line, the user will be prompted for make, model, and version.

-o *outfile*
Name the formatted output file *outfile*. By default the output file is created by adding the last three digits of the process id to */tmp/bench*.

The presence of any of the -s, -f, and -l flags cause the execution to be limited to only what is specified. (If -s, -f, or -l are not specified, the default sets all three flags.)

-s This flag causes tests to be executed one after the other with no extra load added to the system.

-f This flag causes simultaneous execution of the tests with no extra load added to the system.

-l This flag causes simultaneous execution of the tests with extra load added to the system at the same time.

EXAMPLES

Execute the test suite on the current machine and store the results as the baseline.

% *bench -b*

Execute the test suite on the current machine and store the results. Also input the make, model, and version from the command line.

% *bench -a -n "DEC, VAX 8600, 4.3 BSD"*

Print a summary of current database into outfile.

% *bench -e -o outfile*

FILES

/tmp/benchXXX	formatted output of the comparison
baseline	result data of baseline system
benchlist	the path of test and the printable name
targetXXX	result data of system to be compared to baseline

BUGS

Instructions for Data Collection Using the Bench Program

Before *bench* can be used on any system, instructions 1 through 4 must be completed. All *tar* instructions are assuming 1600 bpi on drive 0.

1. Mount tape on drive 0 at 1600 bpi.
2. Change to a working directory with at least 3,000 blocks free.
3. To unload the tape, type:

   % *tar xv*

4. To compile the driver program and test suite, type:

   % *make*

   It may be necessary to edit the *bench.mk* file to alter the names for the different compilers with optimizers on.

After completing steps 1-4 above, any of the remaining sections can be followed to collect data or display previously collected data.

To establish a baseline and store the results on the tape:

1. Type:

   % *bench -b*

   a. Enter the make, model, and version of the system when prompted by the program.
   b. Sit back and relax.
2. To store the baseline file on the tape, type:

   % *tar u baseline*

3. Remove all working files and directories from the disk if desired.

To add a target system to the database:

1. Type:

   % *bench -a*

   a. Enter the system description when prompted by the program.
   b. Sit back and relax.
2. To store target system results on tape, type:

   % *tar u target**

   If a print-out is desired, skip to one of the last two sections.
3. Remove all working files and directories if desired.

To print a comparison report between two systems in the database:

1. Type:

   % *bench -p*

   A list of systems in the database will appear preceded by a number. The system will prompt you for two numbers to indicate the two systems to be compared. The output will be sent to the line printer.

2. Remove all working files and directories if desired.

To print a summary report of all systems in the database:

1. Type:

   % *bench -e*

   The output will be sent to the line printer.

2. Remove all working files and directories if desired.

Prime Numbers

```
C
C prime.f
C
C PROGRAM TO GENERATE PRIME NUMBERS
C
C Compile by: fort -O prime.f -o prime
C
      PROGRAM PRIME
C
      COMMON/DAT/VALUE(8192)
C
C INITILIZE DATA STRUCTURES
C
      ILUN=6
      IPRT=0
      ICNT=512
      CUR=2.
      TOP=3.
      I=1
C
C CHECK REMAINDER
C
5     IF(AMOD(TOP,CUR).EQ.0.)GO TO 10
      CUR=CUR+1.
      IF(CUR.LT.TOP)GO TO 5
C
C IF WE SCAN FROM 2 THRU TOP, THEN TOP IS A PRIME NUMBER
C
      VALUE(I)=TOP
      I=I+1
C
C SET UP FOR NEXT PRIME NUMBER
C
10    TOP=TOP+2.
      CUR=2.
      IF(I.LE.ICNT)GO TO 5
C
C PRINT THE PRIME NUMBERS WEVE GENERATED
C
      IF(IPRT.EQ.0)STOP
      DO 15 I=1,ICNT,8
      WRITE(ILUN,9010)(VALUE(J),J=I,I+7)
9010  FORMAT(8F10.0)
15    CONTINUE
      STOP
      END
```

Calling Sequence and Arguments Passing

```
C calseq.f
C
C PROGRAM TO TEST CALLING SEQUENCE AND ARGUMENT PASSING
C
C Compile by:    fort -O calseq -o calseq
C
C
        PROGRAM CALSEQ
C
        Z=0.
10      I=0
        J=1
        K=2
        L=3
        A=0.
        B=1.
        C=2.
        D=3.
        CALL CALSEQ1(A,I,B,J,C,K,D,L)
        Z=Z+1.
        IF(Z.LT.1.E6)GO TO 10
        STOP
        END
C
        SUBROUTINE CALSEQ1(A,I,B,J,C,K,D,L)
        A=D
        B=C
        I=J
        K=L
        RETURN
        END
```

```
C
C rndsk.f
C
C PROGRAM TO PERFORM A CHECK OF THE RANDOM NUMBER
C GENERATOR BY PERFORMING DIRECT ACCESS TO A DISK FILE.
C THE SUBROUTINE WILL USE A RANDOM NUMBER FROM 1 TO 256
C AS THE KEY TO READ A RECORD, INCREMENT THE VALUE READ,
C AND WRITE THE NEW VALUE.
C
C Compile by: fort -O rndisk.f -o rndsk
C
        PROGRAM RNDISK
C
        ILUN=6
        IPRT=0
        ICNT=128
        FCNT=FLOAT(ICNT)
        FCHK=FCNT*100.
        IRAN=0
        isize = 4
        B=rand(IRAN)
        OPEN(ACCESS='DIRECT',
     1FILE='TEST',
     2FORM='UNFORMATTED',
c    3MAXREC=ICNT+1,
     4RECL=isize*2,
     5STATUS='UNKNOWN',
     6UNIT=4)
C
C CREATE FILE WITH EACH RECORD CONTAINING ALL ZEROS
C
        DO 10 I=1,ICNT
        IREC=I
        WRITE(4,rec=IREC)FLOAT(IREC),0.
10      ;:CONTINUE
C
C GENERATE ICNT*100 RANDOM NUMBERS
C
        A=0.
20      IREC=IFIX(FCNT*rand(IRAN))+1
        IF(IREC.GE.1.AND.IREC.LE.ICNT)GO TO 25
        WRITE(ILUN,9010)IREC
9010    FORMAT(' RANDOM NUMBER OUT OF RANGE',I6)
25      I=IREC
        READ(4,rec=I)RNUM,COUNT
        COUNT=COUNT+1.
        I=IREC
        WRITE(4,rec=I)RNUM,COUNT
        A=A+1.
        IF(A.LT.FCHK)GO TO 20
C
C READ FILE, GET MIN, MAX AND AVERAGE OF RANDOM NUMBER GENERATOR
C
        AMIN=9999.
        AMAX=0.
```

```
         AVE=0.
         DO 30 I=1,ICNT
         IREC=I
         READ(4,rec=IREC)RNUM,COUNT
         IF(COUNT.GT.AMAX)AMAX=COUNT
         IF(COUNT.LT.AMIN)AMIN=COUNT
         AVE=AVE+COUNT
30       CONTINUE
         CLOSE(UNIT=4)
         AVE=AVE/FCNT
         IF(IPRT.EQ.0)STOP
         WRITE(ILUN,9000)AMIN,AMAX,AVE
9000     FORMAT(3F15.0)
         STOP
         END
```

Fast Fourier Transform

```
C fft.f
C
C PROGRAM TO PERFORM A FAST FOURIER TRANSFORM USING THE
C DECIMATION-IN-TIME METHOD.
C
C Compile by: fort -O fft.f -o fft
C
        PROGRAM FFT
C
        COMMON/DAT/A(4096)
        COMPLEX A,U,W,T
C
C INITILIZE
C
        ILUM=6
        IPRT=0
        DO 25 LOOP=1,10
        M=12
        ICNT=2**M
        PER=FLOAT(ICNT/16)
        PI=3.141592653589793
        DO 1 I=1,ICNT
        B=SIN(2.*PI*FLOAT(I)/PER)
        A(I)=CMPLX(B,0.)
1       CONTINUE
        N=2**M
        NV2=N/2
        NM1=N-1
        J=1
        DO 7 I=1,NM1
        IF(I.GE.J)GO TO 5
        T=A(J)
        A(J)=A(I)
        A(I)=T
5       K=NV2
6       IF(K.GE.J)GO TO 7
        J=J-K
        K=K/2
        GO TO 6
7       J=J+K
        PI=3.141592653589793
        DO 20 L=1,M
        LE=2**L
        LE1=LE/2
        U=(1.,0.)
        W=CMPLX(COS(PI/FLOAT(LE1)),SIN(PI/FLOAT(LE1)))
        DO 20 J=1,LE1
        DO 10 I=J,N,LE
        IP=I+LE1
        T=A(IP)*U
        A(IP)=A(I)-T
10      A(I)=A(I)+T
20      U=U*W
25      CONTINUE
        IF(IPRT.EQ.0)STOP
```

```
        DO 30 I=1,128,4
        WRITE(ILUN,9000)(A(J),J=I,I+3)
9000    FORMAT(4G15.6)
30      CONTINUE
        STOP
        END
```

```
C
C matrix.f
C
C MATRIX INVERSION USING GAUSS-JORDAN REDUCTION
C INVERTED MATRIX OVERLAYS ORIGINAL MATRIX IN MEMORY
C PARITAL PIVOTING IS NOT USED
C
C Compile by; fort -O matrix -o matrix
C
      PROGRAM MATRIX
C
      COMMON/DAT/A(15,15)
      DOUBLE PRECISION A
      ILUM=6
      IPRT=0
      DO 10 LOOP=1,10000
      N=4
      A(1,1)=1.
      A(1,2)=1.
      A(1,3)=1.
      A(1,4)=1.
      A(2,1)=4.
      A(2,2)=5.
      A(2,3)=6.
      A(2,4)=7.
      A(3,1)=6.
      A(3,2)=10.
      A(3,3)=15.
      A(3,4)=21.
      A(4,1)=12.
      A(4,2)=30.
      A(4,3)=60.
      A(4,4)=105.
C
C CALCULATE ELEMENTS OF REDUCED MATRIX
C
      DO 6 K=1,N
C
C CALCULATE NEW ELEMENTS OF PIVOT ROW
C
      DO 4 J=1,N
      IF(J.EQ.K)GO TO 4
      A(K,J)=A(K,J)/A(K,K)
4     CONTINUE
C
C CALCULATE ELEMENT REPLACING PIVOT ELEMENT
C
      A(K,K)=1./A(K,K)
C
C CALCULATE NEW ELEMENTS NOT IN PIVOT ROW OR PIVOT COLUMN
C
      DO 5 I=1,N
      IF(I.EQ.K)GO TO 5
      DO 5 J=1,N
      IF(J.EQ.K)GO TO 5
```

```
      A(I,J)=A(I,J)-A(K,J)*A(I,K)
5     CONTINUE
C
C CALUCLATE REPLACEMENT ELEMENTS FOR PIVOT COLUMN-EXCEPT PIVOT ELEMENT
C
      DO 6 I=1,N
      IF(I.EQ.K)GO TO 6
      A(I,K)=-A(I,K)*A(K,K)
6     CONTINUE
10    CONTINUE
C
C OUTPUT INVERTED MATRIX
C
      IF(IPRT.EQ.0)STOP
      WRITE(ILUN,8)((A(I,J),J=1,N),I=1,N)
8     FORMAT(4F16.4)
      STOP
      END
```

```
C
C roots.f
C
C ROOTS OF POLYNOMIAL BY BAIRSTOWS METHOD
C
C Compile by: fort -O roots.f -o roots
C
        PROGRAM ROOTS
C
        DIMENSION A(30),B(30),C(30)
        ILUN=6
        IPRT=0
        IF(IPRT.EQ.0)GO TO 200
        IPRT=0
        JPRT=1
200     DO 100 LOOP=1,10000
        IF(LOOP.NE.10000)GO TO 220
        IF(JPRT.EQ.0)GO TO 220
        IPRT=1
220     UI=0.
        VI=0.
        EPSI=1.E-6
        N=5
        A(1)=-3.
        A(2)=-10.
        A(3)=10.
        A(4)=44.
        A(5)=48.
C
C SEE IF N=0,1, OR GREATER THAN 1
C
40      IF(N-1)100,5,7
5       P=-A(1)
        Q=0.
        IT=1
        IF(IPRT.NE.0)WRITE(ILUN,6)N,P,Q,IT
6       FORMAT(' X(',I2,') =',2X,F8.4,6X,F8.4,10X,I3)
        GO TO 100
C
C SEE IF N=2 OR IF N IS GREATER THEN 2
C
7       IF(N.EQ.2)GO TO 8
        GO TO 13
8       U=A(1)
        V=A(2)
        IT=1
9       P=-U/2.
        RAD=U**2-4.*V
C
C CHECK THE SIGN OF U**2-4.*V
C
        IF(RAD.GT.0.)GO TO 12
        RAD=-RAD
        Q=SQRT(RAD)/2.
        IF(IPRT.NE.0)WRITE(ILUN,6)N,P,Q,IT
```

```
      N=N-1
      Q=-Q
90    IF(IPRT.NE.0)WRITE(ILUN,6)N,P,Q,IT
10    N=N-1
C
C CHECK TO SEE IF N IS GREATER THEN ZERO
C
      IF(N.LE.0)GO TO 100
      DO 11 I=1,N
11    A(I)=B(I)
      GO TO 40
12    Q=SQRT(RAD)/2.
      W=P
      Z=Q
      P=P+Q
      Q=0.
      IF(IPRT.NE.0)WRITE(ILUN,6)N,P,Q,IT
      N=N-1
      P=W-Z
      GO TO 90
13    U=UI
      V=VI
      IT=1
C
C CALCULATE THE B VALUES
C
50    B(1)=A(1)-U
      B(2)=A(2)-B(1)*U-V
      DO 14 K=3,N
14    B(K)=A(K)-B(K-1)*U-B(K-2)*V
C
C CALCULATE THE C VALUES
C
      C(1)=B(1)-U
      C(2)=B(2)-C(1)*U-V
      M=N-1
      DO 15 K=3,M
15    C(K)=B(K)-C(K-1)*U-C(K-2)*V
C
C CALCULATE DELU AND DELV
C
      IF(N.GT.3)GO TO 17
      DENOM=C(N-1)-C(N-2)**2
      IF(DENOM.EQ.0.)GO TO 30
      DELU=(B(N)-B(N-1)*C(N-2))/DENOM
16    DELV=(C(N-1)*B(N-1)-C(N-2)*B(N))/DENOM
      GO TO 18
17    DENOM=C(N-1)*C(N-3)-C(N-2)**2
      IF(DENOM.EQ.0)GO TO 30
      DELU=(B(N)*C(N-3)-B(N-1)*C(N-2))/DENOM
      GO TO 16
C
C CALCULATE NEW U AND V VALUES
C
```

```
18      U=U+DELU
        V=V+DELV
        SUM=ABS(DELU)+ABS(DELV)
C
C STORE THE FIRST SUM CALCULATED
C
        IF(IT.EQ.1)GO TO 19
        GO TO 20
19      STORE=SUM
        GO TO 21
20      IF(IT.EQ.50)GO TO 28
        IF(IT.GE.200)GO TO 26
21      IF(SUM.LE.EPSI)GO TO 9
        IF(IT.EQ.100)GO TO 23
22      IT=IT+1
        GO TO 50
23      IF(IPRT.NE.0)WRITE(ILUN,24)
24      FORMAT(' CONVERGENCE IS SLOW')
        IF(IPRT.NE.0)WRITE(ILUN,25)U,V
25      FORMAT(' U=',E14.7,' V=',E14.7)
        GO TO 22
26      IF(IPRT.NE.0)WRITE(ILUN,27)
27      FORMAT(' STOPPED AFTER 200 ITERATIONS')
        IF(IPRT.NE.0)WRITE(ILUN,25)U,V
        GO TO 100
C
C SEE IF SUM AFTER 50 ITERATIONS EXCEEDS FIRST SUM STORED
C
28      IF(SUM.LT.STORE)GO TO 21
        IF(IPRT.NE.0)WRITE(ILUN,29)
29      FORMAT(' DIVERGENCE OCCURRING')
        IF(IPRT.NE.0)WRITE(ILUN,25)U,V
        GO TO 100
30      IF(IPRT.NE.0)WRITE(ILUN,31)
31      FORMAT(' DENOMINATOR IS ZERO')
        GO TO 100
100     CONTINUE
        STOP
        END
```

C Sieve

```c
/*
 * sieve.c
 *
 * Eratosthenes Sieve Prime Number Program in C */
 *
 * Compile by: cc -O sieve.c -o csieve
 *
 */

#define true 1
#define false 0
#define size 8190

        char flags[size + 1];

main() {
        int i,prime,k,count,iter;

        printf("100 iterations\n");
        for(iter = 1;iter <= 100;iter ++) {
            count=0;
            for(i = 0;i <= size;i ++)
                flags[i] = true;
            for(i = 0;i <= size;i ++) {
                if(flags[i]) {
                    prime = i + i + 3;
                    for(k=i+prime;k<=size;k+=prime)
                            flags[k] = false;
                    count++;
                }
            }
        }
        printf("%d is largest of %d primes.\n",prime,count);
}
```

Fortran Sieve

```
c
c sieve.f
c
c eratosthenes sieve with Knuth's optimization
c
c Compile by: fort -O sieve.f -o fsieve
c
        integer i,j,k,iter,prime,count
        logical flags(8191),last

        write(6,10)
10      format (' 100 iterations')
        do 20 iter = 1, 100
           count = 0
           do 30 i = 1, 8191
30            flags(i) = .true.
           last = .false.
           do 40 i = 1, 8191
                if (.not. flags(i)) go to 50
                   prime = i + i + 1
                   count = count + 1
c                  write(6,11) prime
11                 format (1x,i6)
                   if (last) go to 50
                      k = (prime*prime - 1) / 2
c                     k = i + prime
                      do 60 j = k, 8191, prime
60                       flags(j) = .false.
                      if (prime .ge. 127) last = .true.
50              continue
40         continue
20      continue
        write(6,12) count
12      format (1x, i6, ' primes')
        end
```

Pascal Sieve

```
(* sieve.p *)

(* Eratosthenes Sieve Prime Number Program in Pascal *)

(* Compile by: pi sieve.p *)

program prime(output);

const
  size = 8190;

var
  flags : array [0..size] of boolean;
  i,prime,k,cnt,iter : integer;

begin
  writeln('100 iterations');
  for iter := 1 to 100 do begin
        cnt := 0;
        for i := 0 to size do
                flags[i] := true;
        for i := 0 to size do
                if flags[i] then begin
                        prime := i+i+3;
                        k := i + prime;
                        while k <= size do begin
                                flags[k] := false;
                                k := k + prime
                                end;
                        cnt := cnt + 1
                        end;
        end;
  writeln(cnt,' primes')
end.
```

Whetstone

```
/*      whet.c
 *
 *      Whetstone benchmark in C.  This program is a translation of the
 *      original Algol version in "A Synthetic Benchmark" by H.J. Curnow
 *      and B.A. Wichman in Computer Journal, Vol  19 #1, February 1976.
 *
 *      Used to test compiler optimization and floating point performance.
 *
 *      Compile by:             cc -O -s -o whet whet.c
 *      or:                     cc -O -DPOUT -s -o whet whet.c
 *      if output is desired.
 */

#define ITERATIONS      10 /* 1 Million Whetstone instructions */

#include "math.h"

double          x1, x2, x3, x4, x, y, z, t, t1, t2;
double          e1[4];
int             i, j, k, l, n1, n2, n3, n4, n6, n7, n8, n9, n10, n11;

main()
{

        /* initialize constants */

        t    =    0.499975;
        t1   =    0.50025;
        t2   =    2.0;

        /* set values of module weights */

        n1   =    0 * ITERATIONS;
        n2   =   12 * ITERATIONS;
        n3   =   14 * ITERATIONS;
        n4   =  345 * ITERATIONS;
        n6   =  210 * ITERATIONS;
        n7   =   32 * ITERATIONS;
        n8   =  899 * ITERATIONS;
        n9   =  616 * ITERATIONS;
        n10  =    0 * ITERATIONS;
        n11  =   93 * ITERATIONS;

/* MODULE 1:  simple identifiers */

        x1 =  1.0;
        x2 = x3 = x4 = -1.0;

        for(i = 1; i <= n1; i += 1) {
                x1 = ( x1 + x2 + x3 - x4 ) * t;
                x2 = ( x1 + x2 - x3 - x4 ) * t;
                x3 = ( x1 - x2 + x3 + x4 ) * t;
                x4 = (-x1 + x2 + x3 + x4 ) * t;
        }
```

```
#ifdef POUT
        pout(n1, n1, n1, x1, x2, x3, x4);
#endif


/* MODULE 2:  array elements */

        e1[0] =  1.0;
        e1[1] = e1[2] = e1[3] = -1.0;

        for (i = 1; i <= n2; i +=1) {
                e1[0] = ( e1[0] + e1[1] + e1[2] - e1[3] ) * t;
                e1[1] = ( e1[0] + e1[1] - e1[2] + e1[3] ) * t;
                e1[2] = ( e1[0] - e1[1] + e1[2] + e1[3] ) * t;
                e1[3] = (-e1[0] + e1[1] + e1[2] + e1[3] ) * t;
        }
#ifdef POUT
        pout(n2, n3, n2, e1[0], e1[1], e1[2], e1[3]);
#endif

/* MODULE 3:  array as parameter */

        for (i = 1; i <= n3; i += 1)
                pa(e1);
#ifdef POUT
        pout(n3, n2, n2, e1[0], e1[1], e1[2], e1[3]);
#endif

/* MODULE 4:  conditional jumps */

        j = 1;
        for (i = 1; i <= n4; i += 1) {
                if (j == 1)
                        j = 2;
                else
                        j = 3;

                if (j > 2)
                        j = 0;
                else
                        j = 1;

                if (j < 1 )
                        j = 1;
                else
                        j = 0;
        }
#ifdef POUT
        pout(n4, j, j, x1, x2, x3, x4);
#endif

/* MODULE 5:  omitted */

/* MODULE 6:  integer arithmetic */
```

```
        j = 1;
        k = 2;
        l = 3;

        for (i = 1; i <= n6; i += 1) {
                j = j * (k - j) * (l -k);
                k = l * k - (l - j) * k;
                l = (l - k) * (k + j);

                el[l - 2] = j + k + l;          /* C arrays are zero based */
                el[k - 2] = j * k * l;
        }
#ifdef POUT
        pout(n6, j, k, el[0], el[1], el[2], el[3]);
#endif

/* MODULE 7:  trig. functions */

        x = y = 0.5;

        for(i = 1; i <= n7; i +=1) {
                x = t * atan(t2*sin(x)*cos(x)/(cos(x+y)+cos(x-y)-1.0));
                y = t * atan(t2*sin(y)*cos(y)/(cos(x+y)+cos(x-y)-1.0));
        }
#ifdef POUT
        pout(n7, j, k, x, x, y, y);
#endif

/* MODULE 8:  procedure calls */

        x = y = z = 1.0;

        for (i = 1; i <= n8; i +=1)
                p3(x, y, &z);
#ifdef POUT
        pout(n8, j, k, x, y, z, z);
#endif

/* MODULE9:  array references */

        j = 1;
        k = 2;
        l = 3;

        el[0] = 1.0;
        el[1] = 2.0;
        el[2] = 3.0;

        for(i = 1; i <= n9; i += 1)
                p0();
#ifdef POUT
        pout(n9, j, k, el[0], el[1], el[2], el[3]);
#endif
```

```
/* MODULE10:  integer arithmetic */

        j = 2;
        k = 3;

        for(i = 1; i <= n10; i +=1) {
                j = j + k;
                k = j + k;
                j = k - j;
                k = k - j - j;
        }
#ifdef POUT
        pout(n10, j, k, x1, x2, x3, x4);
#endif

/* MODULE11:  standard functions */

        x = 0.75;
        for(i = 1; i <= n11; i +=1)
                x = sqrt( exp( log(x) / t1));

#ifdef POUT
        pout(n11, j, k, x, x, x, x);
#endif
exit (0);
}

pa(e)
double e[4];
{
        register int j;

        j = 0;
      lab:
        e[0] = (  e[0] + e[1] + e[2] - e[3] ) * t;
        e[1] = (  e[0] + e[1] - e[2] + e[3] ) * t;
        e[2] = (  e[0] - e[1] + e[2] + e[3] ) * t;
        e[3] = ( -e[0] + e[1] + e[2] + e[3] ) / t2;
        j += 1;
        if (j < 6)
                goto lab;
}


p3(x, y, z)
double x, y, *z;
{
        x  = t * (x + y);
        y  = t * (x + y);
        *z = (x + y) /t2;
}
```

```
p0()
{
        e1[j] = e1[k];
        e1[k] = e1[l];
        e1[l] = e1[j];
}

#ifdef POUT
pout(n, j, k, x1, x2, x3, x4)
int n, j, k;
double x1, x2, x3, x4;
{
        printf("%6d%6d%6d  %5e  %5e  %5e  %5e\n",
                n, j, k, x1, x2, x3, x4);
}
#endif
```

```
/*      dry.c
 *
 *      "DHRYSTONE" Benchmark Program
 *
 *      Version:        C/1.1, 12/01/84
 *
 *      Date:           PROGRAM updated 01/06/86, RESULTS updated 02/17/86
 *
 *      Author:         Reinhold P. Weicker,  CACM Vol 27, No 10, 10/84 pg. 1013
 *                      Translated from ADA by Rick Richardson
 *                      Every method to preserve ADA-likeness has been used,
 *                      at the expense of C-ness.
 *
 *      Compile:        cc -O dry.c -o drynr                    : No registers
 *                      cc -O -DREG=register dry.c -o dryr      : Registers
 *
 *      Run:            drynr; dryr
 *
 *
 *
 *      The following program contains statements of a high-level programming
 *      language (C) in a distribution considered representative:
 *
 *      assignments                     53%
 *      control statements              32%
 *      procedure, function calls       15%
 *
 *      100 statements are dynamically executed.  The program is balanced with
 *      respect to the three aspects:
 *              - statement type
 *              - operand type (for simple data types)
 *              - operand access
 *                      operand global, local, parameter, or constant.
 *
 *      The combination of these three aspects is balanced only approximately.
 *
 *      The program does not compute anything meaningfull, but it is
 *      syntactically and semantically correct.
 *
 */
```

```
/* Accuracy of timings and human fatigue controlled by next two lines */
/*#define LOOPS 50000            /* Use this for slow or 16 bit machines */
#define LOOPS   500000           /* Use this for faster machines */

/* Compiler dependent options */
#undef  NOENUM                   /* Define if compiler has no enum's */
#undef  NOSTRUCTASSIGN           /* Define if compiler can't assign structures */

/* define only one of the next two defines */
#define TIMES                    /* Use times(2) time function */
/*#define TIME                   /* Use time(2) time function */

/* define the granularity of your times(2) function (when used) */
#define HZ      60               /* times(2) returns 1/60 second (most) */
/*#define HZ    100              /* times(2) returns 1/100 second (WECo) */

/* for compatibility with goofed up version */
/*#define GOOF                   /* Define if you want the goofed up version */

#ifdef GOOF
char    Version[] = "1.0";
#else
char    Version[] = "1.1";
#endif

#ifdef  NOSTRUCTASSIGN
#define structassign(d, s)      memcpy(&(d), &(s), sizeof(d))
#else
#define structassign(d, s)      d = s
#endif

#ifdef  NOENUM
#define Ident1  1
#define Ident2  2
#define Ident3  3
#define Ident4  4
#define Ident5  5
typedef int     Enumeration;
#else
typedef enum    {Ident1, Ident2, Ident3, Ident4, Ident5} Enumeration;
#endif

typedef int     OneToThirty;
typedef int     OneToFifty;
typedef char    CapitalLetter;
typedef char    String30[31];
typedef int     Array1Dim[51];
typedef int     Array2Dim[51][51];

struct  Record
{
        struct Record           *PtrComp;
        Enumeration             Discr;
        Enumeration             EnumComp;
```

```
        OneToFifty              IntComp;
        String30                StringComp;
};

typedef struct Record   RecordType;
typedef RecordType *    RecordPtr;
typedef int             boolean;

#define NULL            0
#define TRUE            1
#define FALSE           0

#ifndef REG
#define REG
#endif

extern Enumeration      Func1();
extern boolean          Func2();

#ifdef TIMES
#include <sys/types.h>
#include <sys/times.h>
#endif

main()
{
        Proc0();
        exit(0);
}

/*
 * Package 1
 */
int             IntGlob;
boolean         BoolGlob;
char            Char1Glob;
char            Char2Glob;
Array1Dim       Array1Glob;
Array2Dim       Array2Glob;
RecordPtr       PtrGlb;
RecordPtr       PtrGlbNext;

Proc0()
{
        OneToFifty              IntLoc1;
        REG OneToFifty          IntLoc2;
        OneToFifty              IntLoc3;
        REG char                CharLoc;
        REG char                CharIndex;
        Enumeration             EnumLoc;
        String30                String1Loc;
        String30                String2Loc;
        extern char             *malloc();
```

```
#ifdef TIME
        long                        time();
        long                        starttime;
        long                        benchtime;
        long                        nulltime;
        register unsigned int       i;

        starttime = time( (long *) 0);
        for (i = 0; i < LOOPS; ++i);
        nulltime = time( (long *) 0) - starttime; /* Computes o'head of loop */
#endif
#ifdef TIMES
        time_t                      starttime;
        time_t                      benchtime;
        time_t                      nulltime;
        struct tms                  tms;
        register unsigned int       i;

        times(&tms); starttime = tms.tms_utime;
        for (i = 0; i < LOOPS; ++i);
        times(&tms);
        nulltime = tms.tms_utime - starttime; /* Computes overhead of looping */
#endif

        PtrGlbNext = (RecordPtr) malloc(sizeof(RecordType));
        PtrGlb = (RecordPtr) malloc(sizeof(RecordType));
        PtrGlb->PtrComp = PtrGlbNext;
        PtrGlb->Discr = Ident1;
        PtrGlb->EnumComp = Ident3;
        PtrGlb->IntComp = 40;
        strcpy(PtrGlb->StringComp, "DHRYSTONE PROGRAM, SOME STRING");
#ifndef GOOF
        strcpy(String1Loc, "DHRYSTONE PROGRAM, 1'ST STRING");   /*GOOF*/
#endif
        Array2Glob[8][7] = 10;  /* Was missing in published program */

/*****************
-- Start Timer --
*****************/
#ifdef TIME
        starttime = time( (long *) 0);
#endif
#ifdef TIMES
        times(&tms); starttime = tms.tms_utime;
#endif
        for (i = 0; i < LOOPS; ++i)
        {

                Proc5();
                Proc4();
                IntLoc1 = 2;
                IntLoc2 = 3;
                strcpy(String2Loc, "DHRYSTONE PROGRAM, 2'ND STRING");
                EnumLoc = Ident2;
```

```
                BoolGlob = ! Func2(String1Loc, String2Loc);
                while (IntLoc1 < IntLoc2)
                {
                        IntLoc3 = 5 * IntLoc1 - IntLoc2;
                        Proc7(IntLoc1, IntLoc2, &IntLoc3);
                        ++IntLoc1;
                }
                Proc8(Array1Glob, Array2Glob, IntLoc1, IntLoc3);
                Proc1(PtrGlb);
                for (CharIndex = 'A'; CharIndex <= Char2Glob; ++CharIndex)
                        if (EnumLoc == Func1(CharIndex, 'C'))
                                Proc6(Ident1, &EnumLoc);
                IntLoc3 = IntLoc2 * IntLoc1;
                IntLoc2 = IntLoc3 / IntLoc1;
                IntLoc2 = 7 * (IntLoc3 - IntLoc2) - IntLoc1;
                Proc2(&IntLoc1);
        }

/*****************
-- Stop Timer --
*****************/

#ifdef TIME
        benchtime = time( (long *) 0) - starttime - nulltime;
        printf("Dhrystone(%s) time for %ld passes = %ld\n",
                Version,
                (long) LOOPS, benchtime);
        printf("This machine benchmarks at %ld dhrystones/second\n",
                ((long) LOOPS) / benchtime);
#endif
#ifdef TIMES
        times(&tms);
        benchtime = tms.tms_utime - starttime - nulltime;
        printf("Dhrystone(%s) time for %ld passes = %ld\n",
                Version,
                (long) LOOPS, benchtime/HZ);
        printf("This machine benchmarks at %ld dhrystones/second\n",
                ((long) LOOPS) * HZ / benchtime);
#endif

}

Proc1(PtrParIn)
REG RecordPtr   PtrParIn;
{
#define NextRecord      (*(PtrParIn->PtrComp))

        structassign(NextRecord, *PtrGlb);
        PtrParIn->IntComp = 5;
        NextRecord.IntComp = PtrParIn->IntComp;
        NextRecord.PtrComp = PtrParIn->PtrComp;
        Proc3(NextRecord.PtrComp);
        if (NextRecord.Discr == Ident1)
        {
```

```
		NextRecord.IntComp = 6;
		Proc6(PtrParIn->EnumComp, &NextRecord.EnumComp);
		NextRecord.PtrComp = PtrGlb->PtrComp;
		Proc7(NextRecord.IntComp, 10, &NextRecord.IntComp);
	}
	else
		structassign(*PtrParIn, NextRecord);

#undef	NextRecord
}

Proc2(IntParIO)
OneToFifty	*IntParIO;
{
	REG OneToFifty		IntLoc;
	REG Enumeration		EnumLoc;

	IntLoc = *IntParIO + 10;
	for(;;)
	{
		if (Char1Glob == 'A')
		{
			--IntLoc;
			*IntParIO = IntLoc - IntGlob;
			EnumLoc = Ident1;
		}
		if (EnumLoc == Ident1)
			break;
	}
}

Proc3(PtrParOut)
RecordPtr	*PtrParOut;
{
	if (PtrGlb != NULL)
		*PtrParOut = PtrGlb->PtrComp;
	else
		IntGlob = 100;
	Proc7(10, IntGlob, &PtrGlb->IntComp);
}

Proc4()
{
	REG boolean	BoolLoc;

	BoolLoc = Char1Glob == 'A';
	BoolLoc |= BoolGlob;
	Char2Glob = 'B';
}

Proc5()
{
	Char1Glob = 'A';
	BoolGlob = FALSE;
```

```
}

extern boolean Func3();

Proc6(EnumParIn, EnumParOut)
REG Enumeration EnumParIn;
REG Enumeration *EnumParOut;
{
        *EnumParOut = EnumParIn;
        if (! Func3(EnumParIn) )
                *EnumParOut = Ident4;
        switch (EnumParIn)
        {
        case Ident1:    *EnumParOut = Ident1; break;
        case Ident2:    if (IntGlob > 100) *EnumParOut = Ident1;
                        else *EnumParOut = Ident4;
                        break;
        case Ident3:    *EnumParOut = Ident2; break;
        case Ident4:    break;
        case Ident5:    *EnumParOut = Ident3;
        }
}

Proc7(IntParI1, IntParI2, IntParOut)
OneToFifty      IntParI1;
OneToFifty      IntParI2;
OneToFifty      *IntParOut;
{
        REG OneToFifty  IntLoc;

        IntLoc = IntParI1 + 2;
        *IntParOut = IntParI2 + IntLoc;
}

Proc8(Array1Par, Array2Par, IntParI1, IntParI2)
Array1Dim       Array1Par;
Array2Dim       Array2Par;
OneToFifty      IntParI1;
OneToFifty      IntParI2;
{
        REG OneToFifty  IntLoc;
        REG OneToFifty  IntIndex;

        IntLoc = IntParI1 + 5;
        Array1Par[IntLoc] = IntParI2;
        Array1Par[IntLoc+1] = Array1Par[IntLoc];
        Array1Par[IntLoc+30] = IntLoc;
        for (IntIndex = IntLoc; IntIndex <= (IntLoc+1); ++IntIndex)
                Array2Par[IntLoc][IntIndex] = IntLoc;
        ++Array2Par[IntLoc][IntLoc-1];
        Array2Par[IntLoc+20][IntLoc] = Array1Par[IntLoc];
        IntGlob = 5;
}
```

```
Enumeration Func1(CharPar1, CharPar2)
CapitalLetter   CharPar1;
CapitalLetter   CharPar2;
{
        REG CapitalLetter       CharLoc1;
        REG CapitalLetter       CharLoc2;

        CharLoc1 = CharPar1;
        CharLoc2 = CharLoc1;
        if (CharLoc2 != CharPar2)
                return (Ident1);
        else
                return (Ident2);
}

boolean Func2(StrParI1, StrParI2)
String30        StrParI1;
String30        StrParI2;
{
        REG OneToThirty         IntLoc;
        REG CapitalLetter       CharLoc;

        IntLoc = 1;
        while (IntLoc <= 1)
                if (Func1(StrParI1[IntLoc], StrParI2[IntLoc+1]) == Ident1)
                {
                        CharLoc = 'A';
                        ++IntLoc;
                }
        if (CharLoc >= 'W' && CharLoc <= 'Z')
                IntLoc = 7;
        if (CharLoc == 'X')
                return(TRUE);
        else
        {
                if (strcmp(StrParI1, StrParI2) > 0)
                {
                        IntLoc += 7;
                        return (TRUE);
                }
                else
                        return (FALSE);
        }
}

boolean Func3(EnumParIn)
REG Enumeration EnumParIn;
{
        REG Enumeration EnumLoc;

        EnumLoc = EnumParIn;
        if (EnumLoc == Ident3) return (TRUE);
        return (FALSE);
}
```

```
#ifdef  NOSTRUCTASSIGN
memcpy(d, s, l)
register char   *d;
register char   *s;
register int    l;
{
        while (l--) *d++ = *s++;
}
#endif
```

```
/*
 * blockwrite.c
 *
 * This program creates a very large file.
 *
 * Compile by: cc -O blockwrite.c -o blockwrite
 */

#define NAME     "BLOCKWRITE"
#define FNAME    "bigfile"
#define BSIZE    8096     /* 8K block */
#define BLOCKS   128      /* number of 8K blocks in file (total 1 Mbyte) */

#include <sys/file.h>

int main() {

        int     fileflags = O_CREAT O_TRUNC|O_APPEND|O_WRONLY;
        int     filemode = 0777;
        int     f;
        int     lcount = 0;
        char    buffer[BSIZE];
        int     i;

        printf("%s: beginning (%d bytes in file))\n", NAME, BSIZE * BLOCKS);
        fflush(1);
        for(lcount = 0; lcount < 23; lcount++) {
            if ((f = open(FNAME, fileflags, filemode)) <= 0) {
                printf("%s: unable to create '%s'\n", NAME, FNAME);
                exit(1);
            }
            for (i=1;i<=BLOCKS;i++) write(f, buffer, BSIZE);
            close(f);
        }
        printf("%s: complete (%d bytes in file))\n", NAME, BSIZE * BLOCKS);
        fflush(1);
}
```

Block Read

```c
/*
 * blockread.c
 *
 * This program reads a very large file.
 *
 * Compile by: cc -O blockread.c -o blockread
 *
 */

#define NAME    "BLOCKREAD"
#define FNAME   "bigfile"
#define BSIZE   8096    /* 8K block */
#define BLOCKS  128     /* number of 8K blocks in file (total 1 Mbyte) */

#include <sys/file.h>

int main() {

        int     fileflags = O_RDONLY;
        int     filemode = 0444;
        int     f;
        char    buffer[BSIZE];
        int     i;
        int     lcount;

        printf("%s: beginning (%d bytes in file))\n", NAME, BSIZE * BLOCKS);
        fflush(1);
        for(lcount = 0; lcount < 39; lcount++) {
                if ((f = open(FNAME, fileflags, filemode)) <= 0) {
                        printf("%s: unable to open '%s'\n", NAME, FNAME);
                        exit(1);
                }

            for (i=1;i<=BLOCKS;i++) read(f, buffer, BSIZE);
            close(f);
        }
        printf("%s: complete (%d bytes in file))\n", NAME, BSIZE * BLOCKS);
        fflush(1);
}
```

Sort (Part 1)

```
#csh script to run timing on sort
#
# sorttest
# sortfile - input to sort
# sortstandard - presorted output for checking
#
echo Start of sort
sort -4 +5 ./preaward/sortfile > sortout
echo End of sort.  Start of compare.
diff ./preaward/sortstandard sortout
echo End of compare.
rm -f sortout
```

```
.de sh
.br
.ne 5
.PP
\fB\\$1\fR
.PP
..
.if n .ds ua \o'^|'
.if t .ds ua \(ua
.if n .ds aa '
.if t .ds aa \(aa
.if n .ds ga `
.if t .ds ga \(ga
.if t .tr *\(**
.TH CSH 1 "18 July 1983"
.UC 4
.SH NAME
csh \- a shell (command interpreter) with C-like syntax
.SH SYNOPSIS
.B csh
[
.B \-cef\^instvVxX
] [
arg ...
]
.SH DESCRIPTION
.I Csh
is a first implementation of a command language interpreter
incorporating a history mechanism (see
.B "History Substitutions)"
job control facilities (see
.B Jobs)
and a C-like syntax.
So as to be able to use its job control facilities, users of
.I csh
must (and automatically) use the new tty driver fully described in
.IR tty (4).
This new tty driver allows generation of interrupt characters
from the keyboard to tell jobs to stop.  See
.IR stty (1)
for details on setting options in the new tty driver.
.PP
An instance of
.I csh
begins by executing commands from the file `.cshrc' in the
.I home
directory of the invoker.
If this is a login shell then it also executes commands from the file
`.login' there.
It is typical for users on crt's to put the command ``stty crt'' in their
.I \&.login
file, and to also invoke
.IR tset (1)
there.
.PP
```

```
In the normal case, the shell will then begin reading commands from the.
terminal, prompting with `% '.
Processing of arguments and the use of the shell to process files
containing command scripts will be described later.
.PP
The shell then repeatedly performs the following actions:
a line of command input is read and broken into
.IR words .
This sequence of words is placed on the command history list and then parsed.
Finally each command in the current line is executed.
.PP
When a login shell terminates it executes commands from the file `.logout'
in the users home directory.
.sh "Lexical structure"
The shell splits input lines into words at blanks and tabs with the
following exceptions.
The characters
`&' `|' `;' `<' `>' `(' `)'
form separate words.
If doubled in `&&', `\|\|', `<<' or `>>' these pairs form single words.
These parser metacharacters may be made part of other words, or prevented their
special meaning, by preceding them with `\e'.
A newline preceded by a `\e' is equivalent to a blank.
.PP
In addition strings enclosed in matched pairs of quotations,
`\*(aa', `\*(ga' or `"',
form parts of a word; metacharacters in these strings, including blanks
and tabs, do not form separate words.
These quotations have semantics to be described subsequently.
```

.

.

.

etc.

```
        $       last argument
        &       Repeat the previous substitution.
        0       first (command) word
        10  ex write.c
        11  cat oldwrite.c
        12  diff *write.c
        [1| 1234
        ■*(ua   first argument,  i.e. `1'
        ■-■fIy■fR       abbreviates `0■-■fIy■fR■|'
        ■09  write michael
        ■fIn■fR ■fIn■fR■|'th argument
        ■fIx■fR■|*      abbreviates  ■fIx■fR■|■-$'
        ■fIx■fR■|■-■fIy■fR      range of words
        d       directory
        e       existence
        f       plain file
        o       ownership
        r       read access
        s/■fIl■fR■|/■fIr■fR■|/  Substitute ■fIl■fR for ■fIr■fR
        w       write access
        x       execute access
        z       zero size
$#name
$$
$*
$0
$<
$?0
$?name
$name
$name[selector]
$number
${#name}
${?name}
${name[selector]}
${name}
${number}
(As in
(Both
(See the description of
(The
(The words
(as in
(e.g. `$shell').
(second form).
..
.B  ■-V
.B  ■-X
.B  ■-c
.B  ■-e
.B  ■-f
.B  ■-i
.B  ■-n
.B  ■-s
.B  ■-t
```

Sort (Part 111)

```
.B  !-v
.B  !-x
.B  alias
.B  alloc
.B  break
.B  breaksw
.B  breaksw
.B  breaksw
.B  cd
.B  chdir
.B  continue
.B  default:
.B  default:
.B  else
.B  else
.B  end
.B  end
.B  end
.B  endif
.B  endif
.B  endsw
.B  endsw
.B  exit
.B  history
.B  login
.B  logout
.B  nice
```

.

.

.

etc.

Integer Arithmetic

```c
/*
 * integer.c
 *
 * This program does integer arithmetic.
 *
 * Compile by: cc -O integer.c -o integer
 *
 */

#define NAME    "INTEGER"
#define COUNT   2900000 /* number of iterations */

main() {
        long    i;                /* iteration counter */
        long    a, b, c, d;       /* integer variables for arithmetic */

        a = 1234; b = 2345; c = 3456; d = 4567;
        printf("%s: beginning (%d iterations)\n", NAME, COUNT);
        for ( i = 0; i<COUNT; i++) {     /* do some arithmetic */
            a = b + c - d;
            b = a * b / d;
        }
        printf("%s: complete (%d iterations)\n", NAME, COUNT);
}
```

Real Arithmetic

```
/*
 * real.c
 *
 * This program does real arithmetic.
 *
 * Compile by: cc -O real.c -o real
 *
 */

#define NAME    "REAL"
#define COUNT   600000  /* number of iterations */
float aa,bb,cc,dd;

int ii,jj,kk;

main()
{
        printf("%s: beginning (%d iterations)\n", NAME, COUNT);
        for(ii = 1; ii < COUNT; ii++) {
                aa = ii;
                bb = aa * aa;
                cc = (bb - aa - .137526)/aa;
        }
        printf("%s: complete (%d iterations)\n", NAME, COUNT);
}
```

Large Data Space

```
/*
 * largedata.c
 *
 * This program has a data space larger than real memory.
 *
 * Compile by: cc -O largedata.c -o largedata
 *
 */

#define NAME    "LARGEDATA"
#define COUNT   20000   /* number of iterations */

#define BLOCK   1024    /* 1K block */
#define BSIZE   4000    /* large buffer size (blocks) */
#define ADDR    0xe000  /* base address of array */

main() {
        register char   *curptr;        /* current pointer */
        register long   i;              /* iteration counter */
        register long   pagecount;      /* number of new pages */
        register long   limit;          /* number of references */
        register long   size;           /* size of array */

        limit = COUNT;
        size = BSIZE;

        sbrk(ADDR + BSIZE * BLOCK);     /* increase data space */
        srand(1);                       /* init random generator */
        i = 0;
        pagecount = 0;
        printf("%s: beginning (%d iterations, size %d)\n",
            NAME, COUNT, size * BLOCK);
        while ( ++i < limit ) { /* make COUNT memory references */
            curptr = (char *)(ADDR + ((rand() % size) * BLOCK));
            if ( *curptr == 0 ) {
                pagecount++;    /* increase new page count */
                *curptr = 1;
            }
        }
        printf("%s: complete (%d pages referenced, %d for the first time)\n",
            NAME, COUNT, pagecount);
}
```

Compile

```
# Compile and load of to routine
#
# compiletest
# to.c - C source
# subs.c - C source
#
echo cc -O -c preaward/to.c
cc -O -c preaward/to.c
echo cc -O -c preaward/subs.c
cc -O -c preaward/subs.c
echo cc -O -o to to.o subs.o
cc -O -o to to.o subs.o
rm -f to.o subs.o
```

END
DATE
FILMED

8-12-87