MTH 420 STATISTICS PACKAGE

GRADUATE PROJECT

BY

BOB SELLMAN
APRIL 23, 1988

COMMITTEE MEMBERS

Dr. Nadina Duran,
Chairman

Mr. Allen Bush,
Member

Mr. Steve Fant,
Member
ABSTRACT

Problem: Throughout the semester and as directed by a number of classroom assignments, CCSU MATH-420 students build a notebook of descriptive and inferential statistics based on a single data set. And although the values contained in the data set may be changed from semester to semester, the structure of the set and the nature of the assignments are relatively static. Students are given several options for calculating the required statistics:

- they may "hand crunch" the numbers;
- they may use a commercial spreadsheet to enter and manipulate the data;
- they may use an IBM microcomputer general statistical package to perform the calculations;
- or they may use any microcomputer to enter the BASIC code provided by the textbook for each calculation.

Many students taking MATH-420 fall into one of two broad categories: (1) CCSU science and technology- or social science-students who are computer-literate and familiar with the IBM PC; and (2) individuals pursuing education degrees whose computer background is either minimal or is limited to the Apple II series of computers. My project addresses both groups.

Solution: STAT-STUFF was conceived and developed to perform those calculations required by MATH-420 on Apple IIe/c computers. It is self contained: students using this program may (1) initialize data disks; (2) enter or edit the data set with a full feature line editor capable of validating each entry as it is made; (3) use the data to develop MATH-420 statistics; (4) dump the data to either the computer screen or to a printer; and (5) dump the results of all calculations to either screen or printer.

- Because STAT-STUFF is written in Pascal (UCSD), computer literate students will have an alternative to the BASIC source code provided by the textbook.
- Because STAT-STUF is tailored to the MATH-420 data set, it is much easier to learn than the more general microcomputer statistics package available on the IBM.
- Because STAT-STUF is tailored to the MATH-420 data set, very specific data validation routines can be employed.
- Because STAT-STUF is run on an Apple II, it allows a significant segment of MATH-420 students to operate in a more comfortably familiar environment. These students may spend less time adapting to the computer and more time addressing the significance of the statistics which they are developing.

Overall, STAT-STUFF provides MATH-420 students alternative methods of developing those statistics which are the focus of the course.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>INITIALIZATIONS</td>
<td>3</td>
</tr>
<tr>
<td>GLOBAL PROCEDURES &amp; MAIN PROGRAM</td>
<td>4</td>
</tr>
<tr>
<td>THE EDITOR</td>
<td>6</td>
</tr>
<tr>
<td>CALCULATIONS</td>
<td>8</td>
</tr>
<tr>
<td>OUTPUT</td>
<td>10</td>
</tr>
<tr>
<td>LIBRARY UNITS</td>
<td>11</td>
</tr>
<tr>
<td>SUMMARY &amp; CONCLUSIONS</td>
<td>13</td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td>14</td>
</tr>
<tr>
<td>USERS' GUIDE</td>
<td>APPENDIX A</td>
</tr>
<tr>
<td>PROGRAM LISTING</td>
<td>APPENDIX B</td>
</tr>
</tbody>
</table>
INTRODUCTION

This project consists of a menu-driven program titled STAT-STUFF and documentation which targets new or casual microcomputer users enrolled in CCSU MTH-420. It was designed to run on an Apple IIe or IIc with two 5 1/2-inch drives. Its use is limited to printers which internally generate a line feed after receiving a carriage return: this includes the Epson machines used by CCSU. It is written in Apple Pascal (UCSD II.2).

Although this project addresses specific MATH-420 course requirements, it was conceived and developed to meet a much more personal set of needs. I became very enthusiastic about Pascal programming in 1984 during CS 511. (This course employed Dale's Introduction to Pascal and Structured Design, Bowles Beginner's Guide for the UCSD Pascal System, and Pollack's Programming the IBM Personal Computer: UCSD Pascal) Given the high cost of IBM PC's and SoftTech or Microsoft Pascal compilers at that time, I acquired an Apple IIc and Apple Pascal. (Sure wish I had known what Borland was about to market!!!) I also acquired a small Pascal reference library, and pursued my own independent course of study. Ultimately, it was Dr. Duran's MTH-420 class which provided me with an opportunity to apply what I was learning about Pascal programming to a relatively large project.

The propendence of STAT-STUFF code addresses requirements other than statistics calculations. Data input, editing, validation and output (particularly screen control) were viewed during development as an opportunity to "exercise" the language and the operating system.

- SEGMENT PROCEDURES and compiler directives were used to regulate run-time memory utilization.

- Dynamic memory allocation was used to develop complex data structures (menus were coded as multilists in which each element was a member of three doubly linked rings). My primary source of information about Pascal pointers was Tennenbaum and Augenstein's Data Structures Using Pascal.

- Byte-level operations (SIZEOF, FILLCHAR, SCAN, MOVELeft, MOVERIGHT) were employed as very fast alternatives to conventional routines for initializing variables and manipulating arrays. Tom Swan explored the uses of these intrinsics in each of his three books listed in the bibliography.

- Free union variant records (variant records without tag fields) were used to effect "peeks" and "pokes" in order to monitor memory utilization during development and to rapidly load graphics when the screen was utilized in a bit-mapped mode. I was first introduced to these techniques by Hyde in his book P-Source. The application of these techniques to graphics manipulation was outlined by Swan in Pascal Programs for Games and Graphics.

- Sets were used extensively for data validation.
System error checking routines (range, I/O, etc.) were disabled, and my own routines were substituted.

6502 assembly language routines were incorporated into STAT-STUFF using the system assembler, the system linker, and EXTERNAL PROCEDURE and FUNCTION designations. These routines were gleaned from Sand's Advanced Pascal Programming Techniques.

Intrinsic units which had utility beyond this program were coded, compiled, and collected into a SYSTEM.LIBRARY. Credit for many of the routines in my library must go to Paul Sand and Tom Swan. The four works by these two authors cited in my bibliography were probably the ones I most intensly read and studied.

STAT-STUFF manipulates a data set which consists of a variable number of records, each describing the following attributes:

1. gender 3. height 5. perception of weight
2. age 4. weight 6. political orientation

Calculations performed upon this set include organization (grouped and ungrouped frequency distribution tables, contingency tables); measures of centrality (mean, median, mode); measures of variability (sum of squares, variance, and standard deviation); predictive instruments (correlation, regression); and measures of dependence (chi-square).

Programming constraints included:

1. The necessity of working with the Apple's relatively limited memory capabilities. A 128k (bank switched) Apple will accomodate up to 46k of compiled P-code and 41k of data and assembly-code.

2. The necessity of working with Apple's relatively limited disk capacity (143k).

3. The necessity of overcoming Apple Pascal's relatively slow text I/O routines. (A 35-record data set stored as a text file takes over a minute to fully upload using the READLN intrinsic.)

4. The problem of designing a system which could be used productively by a first-time computer user with only ten or fifteen minutes preparation. (This time contraint was my best guess as to what would be tolerable to a mildly computer-phobic person — an apt description of myself not too long ago.)

The first constraint was met by breaking the program into overlaying segments. The second constraint was met by stripping the main disk of all parts of the operating system not actually required by STAT-STUFF, and by giving the user the ability to prepare a raw data disk. The third was overcome using Sand's assembly language routines mentioned above. And the fourth constraint was met (1) by developing an easy to use editor which would validate and correct user input, and (2) by developing a light-bar menu system which would be familiar to folks who had used almost any of the currently popular Apple software.
INITIALIZATIONS

STAT-STUFF employs a relatively complex data structure as the basis for its menu selection procedure. The structure consists of records representing the string that appears on the screen, the row and column coordinates of that string, and pointers to other records in that same menu. These records constitute nodes in three doubly-linked rings: a ring ordered on the ASCII collating sequence of the first character of that node's string, a ring of nodes sharing a common row coordinate and ordered on each node's column coordinate, and a ring of nodes sharing a common column coordinate and ordered on each node row coordinate. And although the code for manipulating this data structure (i.e., making menu selections using arrow keys or character keys) is relatively compact, the code for linking these rings is substantial. Further, because these rings are static once linked, the code to perform the linking was relegated to an Apple Pascal SEGMENT PROCEDURE.

Apple Pascal SEGMENT PROCEDURES or FUNCTIONS differ from regular procedures or functions. Their code and data are in memory only while the procedure or function is actually running. Further, the RESIDENT (*R *) compiler directive is available to enhance SEGMENT utilization by temporarily overriding the machine's instructions to swap SEGMENTS in and out of memory. (This is essential if the SEGMENT is to be called from a loop.)

The obvious advantage of SEGMENT PROCEDURES or FUNCTIONS is the ability to fit large programs into available memory. Accordingly, SEGMENT PROCEDURE global_init is utilized by STAT-STUFF to build its menu data structure.

SEGMENT PROCEDURE global_init is completely original code. It contains nested procedures to both load and link menus. The linking routines in this procedure were created to link either "hard-coded" screen descriptions, or to read those descriptions from disk files into a sorted work array. The latter facility was used during program development, however, menus were ultimately "hard-wired" and sorting code stripped from STAT-STUFF in order to minimize initialization time.

The logically-linked ring is established by (1) inserting work array elements into a logical link list; (2) establishing back pointers for each node in that list; and (3) pulling the list into a ring by linking all end-node pointers.

The column and row linked rings are established by traversing the logical ring and building link lists for each unique row or column encountered, as appropriate. The lists are also back-linked and pulled into rings.

NOTE:

Since beginning my CS studies at CCSU, I've discovered that I absorb material more rapidly and thoroughly when I apply myself to relatively large projects. (This applies to learning programming languages and to learning computer applications.) This menu data structure was conceived after reading the Tenenbaum and Augenstein text as was my first stab at Pascal pointers.
GLOBAL PROCEDURES & MAIN PROGRAM

STAT-STUFF's main segment (always memory resident) serves 4 purposes. It calls the segment procedures necessary for global variable initializations; it houses routines to open and close the data (text) file and report the status of those operations; it contains the "three-ring menu" manipulation procedure (see INITIALIZATIONS); and it contains the code for invoking procedures to edit data, to process data, to display the data and calculation results, and to chain to a disk formatting program. The main segment also contains such cosmetic procedures as are necessary to display a cover screen and to support the graphic box and inverse label requirements all other parts of the program.

PROCEDURES hor_line, ver_line, drawbox, and print_inverse_label are straightforward implementations of their descriptive titles. They are invoked by the main program and all of its segment procedures.

PROCEDURES hilite and unhilite write strings to the screen in reverse and normal video. Both differ from the procedures addressed in the preceding paragraph in that they operate on menu nodes, and are passed pointers to those nodes as parameters. Similarly, the PROCEDURES show_menu and erase_menu operate on entire menu data structures, and are passed pointers to those structures.

PROCEDURE select_menu_item does just that! Menu's are displayed as a matrix of strings on the screen. Initial display may or may not include a string displayed in reverse video (light bar), depending on a BOOLEAN "accept default" parameter. Users may move the light bar by either typing the first character of the desired string, or by using the arrow keys. A highlighted selection may be confirmed by either pressing the RETURN key or by pressing the "first character" key a second time.

SAMPLE MENU

1. This one.
2. That one.
3. The other one.

For example, number 2 may be selected by typing either (2)(ret) or (2)(2).

PROCEDURES get_tfname, open_and_test_txtfile, open_output_file, output_txtarray_to_disk, input_txtarray_from_disk, and reporterror all handle disk I/O using special text file procedures which are located in the SYSTEM.LIBRARY unit TFSSTUFF (TF = text file). The decision to use Pascal text files instead of Pascal data files was made for three reasons:

1. I felt that the editor had a more complex relation with the "data file" than did the calculation procedure. The editor was more easily implemented using text strings than data records.
2. Early in the development process -- before the editor was fully realized -- I was able to use the P-system editor to build and manipulate my "data file". I could not have done this conveniently had the "data file" consisted of record-type data structures rather than strings.

3. After reading Paul Sands' Advanced Pascal Programming Techniques, I was looking for an excuse to try some of his slick, assembly-language routines.

PROCEDURE InitDiskChains to an EXEC file (P-system equivalent of an MS-DOS batch file) called FRMTR. That file, in turn, invokes the operating system's disk formatting utility. The EXEC file then restarts the STAT-STUFF program. I took this tack (1) in order to ease the formatting process for new users, and (2) in order to deny users direct access to the operating system and any of its facilities which might be used to destroy files on the program disk.

PROCEDURE MouseText disables an Apple ROM enhancement which displays special screen characters for ASCII values 128..255. Machines without this enhancement use these "high-bit-ON" values to display a reverse video image of their "high-bit-OFF" counterparts. STAT-STUFF makes extensive use of reverse video!

PROCEDURE ShowCoverScreen is a neat implementation of a technique I discovered in Swan's games and graphics book. Specifically, the Apple stores graphics screens as packed arrays of BOOLEAN values -- each value representing a screen pixel. By using free-union variant records to effect a "poke", and by using the low-level (fast) BLOCKREAD I/O intrinsic, I literally pick up a file off the disk and poke it directly into that part of the Apple's memory which it displays as its bit-mapped screen. This is the ultimate cosmetic waste of memory in the program, however, it was another area with which I wished to experiment. Of note is the fact that I created the graphics screen with a utility (PONTRIX) under a totally incompatible operating system (DOS 3.3). However, because both UCSD P-System and DOS 3.3 use the same disk format (15 sector, 35 track), I was able to use this same Pascal BLOCKREAD intrinsic to pick up the graphic off a DOS 3.3 disk and store it to a P-System disk.

In the PROCEDURE OutputTxtArrayToDisk, a crafty (and faster) method of tacking a character onto the end of a string was employed. Instead of the conventional approach of putting the character into a string(1) and concatenating, I turned off system range checking, bumped the length code in string(0), then made a simple assignment "string(new_last) := ch". I'd like to take credit for that one, however, I got the idea from the book P-Source, by Randall Hyde.

Finally, note that MAIN PROGRAM consists of only twenty-five lines. After initializations, the program enters an infinite loop. This also was done to deny the user access to the operating system.
STAT-STUFF incorporates a full-featured line editor with which the user may build and edit the data set file required by MATH-420. It employs several "word processing" features to make data entry and editing fast and fool-proof. The user is presented a three-part screen: the top third is reserved for menus, the bottom third functions as a "window" on five records in the file (the active record flanked by its two predecessors and two successors), and the middle third of the screen is used as a work and prompt area. A typical editor session is as follows:

Responding to a prompt for a file name, a user may create a new file or edit an existing file. Accordingly, a string array variable is initialized to nulls or loaded from the disk.

One of two menus is always visible in the top third of the screen: an "outer" editor FUNCTION MENU or an "inner" editor COMMAND MENU.

<table>
<thead>
<tr>
<th>FUNCTION MENU</th>
<th>COMMAND MENU</th>
</tr>
</thead>
<tbody>
<tr>
<td>add a line</td>
<td>( ret )</td>
</tr>
<tr>
<td>change a line</td>
<td>( down )</td>
</tr>
<tr>
<td>delete a line</td>
<td>( up )</td>
</tr>
<tr>
<td>insert a line</td>
<td>(Ctrl-Z)</td>
</tr>
<tr>
<td>describe self</td>
<td>(Ctrl-X)</td>
</tr>
<tr>
<td>scroll up</td>
<td>(Ctrl-I)</td>
</tr>
<tr>
<td>scroll down</td>
<td>(Ctrl-R)</td>
</tr>
<tr>
<td>quit</td>
<td>(Ctrl-C)</td>
</tr>
<tr>
<td></td>
<td>( esc )</td>
</tr>
</tbody>
</table>

Whether working with a new or an existing file, the user is initially presented the FUNCTION MENU. An add, change, delete, insert or describe-self selection will cause the FUNCTION MENU to be overwritten with the COMMAND MENU, and the user will be presented with the following display in the work area.

Gender ............
Age ............... 
Weight .............
Height .............
Wt Perception ..... 
Pol Orientation...

Data validation is performed as each "field" is entered into the "data record". Specifically, string entries (Gender, Wt Perception, and Pol Orientation) are checked as each character is entered, and characters which do not occur in an acceptable entry are "beeped" and rejected. Further, the actual string entries into the disk file are really generated according to the first letter of each entry: a disk file gender entry of FEMALE will be made if the user types FEMALE, FEMAIL, FXXXXX, or F. Similarly, numeric entries (Age, Weight and Height) are range checked. The approach does not to prevent erroneous entries, but it does prevent entries which cannot be processed!
Users are not permitted to exit and store a data set until they make an additional entry describing themselves. This entry is used in the SEGMENT PROCEDURE do_calcs to assign a percentile rank value to the user for each quantitative attribute in his or her record.

SEGMENT PROCEDURE addedit includes the nested PROCEDURES open_up_array and close_up_array. These are noted because they represent a departure from normal array handling. Specifically, I used the byte-level UCSD intrinsics MOVELEFT and MOVERIGHT to rapidly shift all array elements required to accommodate insertions or deletions.

Finally, the nested PROCEDURE select_procedure is offered as an example of circumventing UCSD Pascal's limitation on passing procedure or function names as parameters to other procedures. Program control is shuttled between the PROCEDURES get_gender, get_age, get_height, get_weight, get_wt_prctn, and get_politics by means of a procedure location variable procloc. An infinite REPEAT UNTIL loop housing a CASE statement for all possible procedure location values is the primary agent for transfer of program control.
CALCULATIONS

Because variable declarations in Apple Pascal simply reserve table space, SEGMENT PROCEDURE do_calcs begins with a number of nested initialization procedures. Simple data types are initialized with simple assignments: numerics (for the most part) are set to zero, sets are initialized as empty, string labels are set to their appropriate values, and arrays are filled with null characters or zeroes. To initialize arrays, I utilized the byte-level intrinsics FILLCHAR and SIZEOF as rapid alternatives to looping through the structures.

In order to avoid the repetitive use of Apple Pascal's relatively slow string intrinsics (POS, COPY, CONCAT, etc.), I load the text "data file" into a work array of true data records. During this process, I convert string images of numerics to integers (age, weight & height); and I convert string representations of gender, weight perception, and political orientation into user-defined enumerated simple data types. Consequently, contents of this work array can be processed rapidly as the various nested calculation routines loop through it.

Multiple passes through the work array are made for the various calculations. In some cases calculations require this technique (frequency distribution tables have to be built before median calculations can be accomplished). More often, multiple loops are employed for ease of program maintenance. In most cases, multiple loops are effected at the expense of program efficiency. (In addition to Pascal programming, this project taught me that program design and development is often a matter of striking some balance between competing design requirements.)

PROCEDURES init_ufds and init_gfds generate frequency distribution tables. GFD's are grouped according to criteria set forth in Jaccard's Statistics for the Behavioral Sciences.

PROCEDURES init_age_by_politics_table, init_sex_by_wt_table, and init_ht_by_wt_table generate contingency tables by using the common PROCEDURE bump_totals.

Because data set attributes may be multi-modal, I used Pascal sets to hold mode values. Separate procedures are required for nominal value modes, modes for ungrouped frequency distributions, and modes for grouped frequency distributions. Accordingly, multiple passes through the work array are made by PROCEDURES init_nominal_mode_sets, init_ufd_mode_sets, and init_gfd_mode_sets.

Procedures to calculate quantitative attributes' means and medians are nested inside PROCEDURES init_means and init_medians. Similarly, PROCEDURE init_variability_variables uses the nested PROCEDURES calc_ufd_stuff and calc_gfd_stuff to determine sum of squares, variance and standard
deviation for each of the quantitative attributes. As elsewhere, multiple
passes through the work array are made to perform these calculations.

PROCEDURE init_corr_n_regr_variables uses the cross-product method outlined in
Jaccard's book to develop the correlation coefficients and the slope,
intercept and constant components of simple regression formulae. While
developing this routine, I ran into interesting programming problem -- one
which also applies to Turbo Pascal. Both Apple (UCSD) and Turbo Pascal fail
to crash with an overflow error when the product of two integers exceeds
MAXINT. Instead, both implementations return a complimented value, i.e., the
product of two positive integers becomes negative. Should a subsequent line
of code use the SQRT function with this product as its argument, the system
will crash, but the subsequent diagnostic message, error offset, or editor
cursor position will provide no real clue as to the location of the actual
problem. (I bought that gem of information with about three hours of my
life!!!)

PROCEDURE init_prs calculates percential ranks for the quantitative attributes
in the user's description of her/himself. PROCEDURE init_chi2 determines
degrees of freedom, critical value, and chi square for the three contingency
tables. NOTE: Although not fully employed by this program, I have coded a
lookup procedure with a hard-wired critical value table for both .05 and .01
critical values. This statistics table lookup scheme would be equally
applicable to other statistical calculations and it might be of value to
anyone who uses STAT-STUFF as a jumping off point for their own project.

Finally, SEGMENT PROCEDURE docals takes several seconds to run. Therefore,
as each calculation is completed, a message to that effect is written to the
screen. This is for the benefit of users (like myself) who become worried
when a computer seems to break off a dialog and proceed on its own.
OUTPUT

STAT-STUFF employs two output procedures: SEGMENT PROCEDURES dumpcalcs and dumpdata. SEGMENT PROCEDURE dumpcalcs operates on the numerous simple and complex data structures initialized by SEGMENT PROCEDURE docalcs. SEGMENT PROCEDURE dumpdata operates directly on the text "data file". Both rely on the main program PROCEDURE open_outputfile, which initializes the global variable "outfile" to either the screen or the printer. PROCEDURE open_outputfile also sets the global BOOLEAN variable "screenflag", which can be tested by all output procedures in order to determine the max number of lines to be displayed at one time. For screen displays, both SEGMENT PROCEDURES also utilize a "continue_or_quit" routine in order to force a pause between screens. For printer operations, this flag acts as a signal for uninterrupted output.
LIBRARY UNITS

STAT-STUFF utilizes a number of library units residing in the file SYSTEM.LIBRARY. APPLLESTUFF, CHAINSTUFF, and TRANSCEND are part of the Apple Pascal development system. XTRASTUFF and TFSTUFF contain routines derived from various reference books. (See SUMMARY chapter comments regarding attribution.)

APPLESTUFF procedures and functions employed by STAT-STUFF are KEYPRESS and KEYFUNCTION. KEYPRESS is a boolean function which allows Apple's simple type-ahead keyboard buffer to be used somewhat as an interrupt driver: the keyboard is periodically polled and FUNCTION KEYPRESS returns TRUE when the buffer contains one or more characters. PROCEDURE NOTE is a machine language procedure which accepts integer parameters representing the pitch and duration of a tone to be produced by the computer's speaker. The use of APPLLESTUFF is limited to the startup routines in PROCEDURE global_inits and is associated with the display of the STAT-STUFF cover screen.

CHAINSTUFF permits program chaining similar to that found in many implementations of BASIC. Its use in STAT-STUFF is limited to the invocation of an EXEC file (FRMTR.TEXT) which executes a P-System utility to format new disks.

TURTLEGRAPHICS is a set of machine language routines which permit bit mapped screen manipulation using polar or cartesian coordinates. Its use in STAT-STUFF is confined to the display of the cover screen.

TRANSCEND contains transcendental math functions (intrinsic functions in Borland's implementation). STAT-STUFF uses the SQR and SQRT functions in its SEGMENT PROCEDURE do_calcs.

XTRASTUFF is a set of procedures and functions, some original - most derived, from Tom Swan's Pascal Programs for Business. Routines in this unit have the common theme of facilitating keyboard input. They convert strings and characters to reals and integers; they check keyboard input values and ranges, beeping and rejecting unwanted characters; and they invoke many screen control characters, giving them plain-language names. I made heavy use of Swan's PROCEDURE getresponse in STAT-STUFF's data editor.

TFSTUFF (TF = text file) was constructed from several routines gleaned from Paul Sand's Advanced Pascal Programming Techniques. For reasons cited in the EDITOR chapter, STAT-STUFF uses conventional Pascal text files as "data files". Unfortunately, UCSD Pascal text I/O is unacceptable slow. TFSTUFF contains routines which emulate conventional READLN's and WRITELN's by using the relatively fast UCSD Pascal intrinsics FUNCTION BLOCKREAD and FUNCTION BLOCKWRITE. Further, Sand described three routines in both Pascal and in 6502 assembly language: one to discriminate individual strings "blockread"
into an input buffer, one to append regular strings to an output buffer for
block transfer to disk, and one to encode leading spaces into the the DLE
format used by UCSD Pascal text files. (DLE = two bytes preceding each line
with leading spaces: ASCII 16 and (leading blanks + 32)(mod 256).) Benchmark
testing by both the author and myself of programs which applied TFSTUFF
routines to text files of varying length confirmed a 3:1 speed advantage using
the BLOCKREAD/WRIT and Sand's Pascal routines. This advantage increased to
10:1 when the Sand's assembly language routines were substituted for their
Pascal counterparts. Other Sand routines which I placed in TFSTUFF included
ones to create, open and close text files. To his routines, I added a Pascal
routine to return diagnostic strings to STAT-STUFF in the event of
BLOCKREAD/WRIT I/O error.
SUMMARY AND CONCLUSIONS

In most general terms, STAT-STUFF code consists of a call to an initialization procedure and than an infinite REPEAT UNTIL loop from which the user may call routines to initialize disks, enter data, process data, dump raw data, or dump processed data. Routines to perform these general tasks are grouped accordingly and placed in SEGMENT PROCEDURES which are swapped in and out of memory as required. Many routines employed by STAT-STUFF are housed in library units -- some of which I entered, and some of which were part of the development system which constitutes Apple Pascal.

STAT-STUFF reliably collects, processes and displays all data discussed in the preceding chapters. It has been used by a number of MATH-420 students to date with no reported bugs, miscalculations, or crashes. That STAT-STUFF could be more tightly coded is inarguable. The program was developed incrementally, as each statistic was studied: only later was it pulled together into an integrated unit. Further, this incremental structure lends itself to study by future MTH-420 students who might wish study a Pascal implementation of a particular calculation or who might wish to use this project as a departure point for their own. To the latter group I propose the following:

1. program implementation in Turbo Pascal;

2. development of routines for the calculation of T-tests for one and two sample groups;

3. development of routines for the calculation of Z-tests for one and two sample groups;

4. development of routines for the analysis of variance (ANOVA);

5. development of routines to generate those bar charts and histograms required of MATH-420 students.

Cumulatively, these suggestions would represent a substantial project. If accomplished, CCSU would possess a remarkable piece of custom software for its MATH-420 class.

As a final note, in order to facilitate proper attribution to those authors whose code I directly employed, committee members may use the following guide. Any code devised by other authors - even that which I modified - has been placed in the SYSTEM.LIBRARY and is submitted as a separate listing in the Appendix. All code in the main listing is original.
BIBLIOGRAPHY

Advanced Pascal Programming Techniques
Paul Sand
McGraw-Hill
1981

Advanced UCSD Pascal Programming Techniques
Eliakim Willner (Datronics, Inc.)
Barry Demchak (Software Construction Inc.)
Prentice-Hall Inc. Englewood Cliffs, New Jersey
1985

Algorithms + Data Structures = Programs
Niklaus Wirth (Eidgenossische Technische Hochschule, Zurich)
Prentice-Hall Inc. Englewood Cliffs, New Jersey
1976

Apple II Pascal 1.3 (Language System Reference Manual)
Apple Computer, Inc. Cupertino, California
1985

Apple Pascal Games
Douglas Hergert
Joseph T Kalash
SYBEX, Inc.
1981

Apple Pascal: A Hands on Approach
Arthur Luehrmann (University of California, Berkeley)
Herbert Peckham (Gavilan College)
McGraw-Hill
1981

Beginner's Guide for the UCSD Pascal System
Kenneth L. Bowles (University of California, San Diego)
BYTE Books Peterborough, New Hampshire
1980

Data Structures Using Pascal
Aaron M. Tenenbaum
Moshe J. Augenstein
Prentice-Hall Inc. Englewood Cliffs, New Jersey
1981

Introduction to Pascal and Structured Design
Nell Dale (University of Texas, Austin)
David Orshalick (University of Texas, Austin)
D.C. Heath & Co. Lexington, Massachusetts
1983
P-Source: A Guide to the Apple Pascal System
Randell Hyde
Reston Publishing Company, Inc. Reston Virginia
1983

Pascal plus Data Structures, Algorithms, and Advanced Programming
Nell Dale (University of Texas, Austin)
Susan C. Lilly
D.C. Heath & Co. Lexington, Massachusetts
1985

Pascal Programming for the Apple
T. G. Lewis (Oregon State University)
Reston Publishing Company, Inc. Reston Virginia
1981

Pascal Programs for Business
Tom Swan
Hayden Book Company Hasbrouck, New Jersey
1983

Pascal Programs for Data Base Management
Tom Swan
Hayden Book Company Hasbrouck, New Jersey
1984

Pascal Programs for Games and Graphics
Tom Swan
Hayden Book Company Hasbrouck, New Jersey
1983

Pascal Programs for Scientists and Engineers
Alan R. Miller (New Mexico Institute of Mining & Technology)
SYBEX, Inc.
1981

Perfect Pascal Programs
Robert Platt (Washington Apple Pi (users' group))
Tab Books, Inc. Blue Ridge Summit, Pennsylvania
1985

Programming Concepts: A Second Course
William B. Jones (California State University)
Prentice-Hall Inc. Englewood Cliffs, New Jersey
1982

Programming Expert Systems in Pascal
Brian Sawyer
Dennis T. Foster
John Wiley & Sons, Inc.
1986

15
Programming in Pascal, 2nd Edition
Peter Grogono (Concordia University)
Addison-Wesley Publishing Co. Reading, Massachusetts
1983

Programming the IBM Personal Computer: UCSD Pascal
Seymour V. Pollack (Washington University, St.Louis)
Holt, Rinehart and Winston
1983

Some Common Pascal Programs
Gregory Davidson
Brad Hallman
McGraw-Hill
1982

Statistics for Social Scientists
Michael Holliday (Loughborough University of Technology)
Louis Cohen (Loughborough University of Technology)
1982

Statistics for the Behavioral Sciences
James Jaccard (New York State University, Albany)
Wadsworth Publishing Co. Belmont, California
1983

The UCSD Pascal Handbook: A Reference Guidebook for Programmers
Randy Clark
Stephen Koehler
Prentice-Hall Inc. Englewood Cliffs, New Jersey
1982