To: MTB Distribution

From: M. D. Schroeder

Date: January 7, 1974

Subject: Cost Comparison of TSO and Multics

The attached report by Harry Forsdick of experiments he has done to compare the cost of using TSO on the Computation Center's 370/165 and of using Multics may be of interest to the Multics development community. The experiments were done as a project for the System Performance Measurements Seminar I taught this fall. In general, Harry found roughly comparible costs for doing the same jobs on both systems. Compared are the costs of editing, compiling PL/I, loading, running PL/I-compiled programs, printing, and running-off a document. A comparison of FORTRAN use on the two systems is in the works, and should be interesting, since FORTRAN is reported to be much more efficient than PL/I on the IBM system.

TSO and Multics: A Cost Comparison

Harry Forsdick
6.845 - 12/17/73

"You can't do that."
-- anonomous staff member of Information Processing Center, MIT when told of my intention to compare TSO with Multics. (I forget whether the emphasis was on the first, second, third or forth word.)

"Comparing TSO with Multics is like comparing a Volkswagen with a Ferarri: you can get to the same palce using either vehicle, however in one you get there with more class."
-- Tom van Vleck, Information Processing Center, MIT.

I. Introduction

There are at least two general purpose interactive computer systems that are available to the general MIT public: TSO (Time-Sharing Option of OS-MVT running on an IBM 370/165 computer) and Multics (Multiplexed Information and Computing Service running on a Honeywell 6180 computer). These two systems have similar goals and thus comparison between them is inevitable. The goal of this study is to perform such a comparison in a rigorous way.

There are several areas that are comparable for these two systems. A first cut comparison might be to contrast the <u>response times</u> of the two systems: the utility of an interactive computer system is best measured by the amount of time users have to wait before their computations are completed. If users had unlimited funds to spend on computer services, this would indeed be the best measure. In reality, this is not the case. At this point, <u>cost</u> appears to be a good measure: an accurate indicator of the utility of a computer system is how inexpensively a computation can be performed. This turns out to be a popular measure because everyone can relate to a monetary comparison. There are, however, other considerations. One system would be useless if the computation could not be performed because of, say, size restrictions on the amount of storage that is available for use. The question of whether a system has the ability to solve a task must be considered. Another aspect is user pleasure: if a system is extremely frustrating to use, the utility of that system decreases. Finally, the speed of a system can be important. One system may be cheaper, but so much slower that the task cannot be finished in a reasonable amount of time.

These five areas -- response time, cost, ability to solve a task, user pleasure and speed -- are all useful measures in a comparison of TSO and Multics. I will consider all of these measures, but will concentrate on cost because it is a measure with which no one can take exception: there is a well defined cost assigned to each interaction on both computer systems.

To further define the comparison, I will state certain assumptions and attitudes that I have assumed in this study. The person that is interacting with each of the computer systems is assumed to be a new user of the system who has a good knowledge of PL/I. In addition, I assume a certain "innocence" about this user: the actions of this user are not influenced by the implementation of the computer system. There is no attempt to optimize programs or uses of commands so that cost will be minimized. Instead, the user interacts in whatever ways feel comfortable. In assuming that the user takes no account of the implementation, I am showing the effect of the limited resources used in this study. Perhaps another study could be made to see how inexpensively the tasks I will measure could be done on each of the machines studies if the implementation were taken into consideration.

Finally, there is a conception of the "virtual" computer system that is responding to the commands being typed at an interactive terminal. In this study, I view a terminal as an input device to a computer system with the following subsystems:

1) a PL/I subsystem.
   a) a preprocessor called a compiler.
   b) a set of utility routines called a library.
   c) a runner that actually does the computation.

2) a file subsystem.
   a) naming conventions, file formats.
   b) list contents.
   c) cost of storage.

3) a text editing subsystem.

4) a text file printing subsystem.

5) a text file formatting subsystem.

6) a program interrupter.

7) administrative functions.
   a) logging on to the system, logging off.
   b) computing how much an interaction costs.

Certainly this virtual system is influenced by the resources that were available on the two systems under consideration. It is part of the intersection of the facilities available on both systems.

One final assumption will be made. The rates for service that will be considered are the rates in effect for a weekday between 9 am and 5 pm. This has a tendency to work in the favor of TSO because rates for interactive service on TSO never vary; at times, the rates for the processor service on Multics are one half the amount they are on weekdays. The weekday rates for the two systems as of December 14, 1973 are found in Table 1. Processor charges are for the time spent by the processor of the computer system executing instructions in response to the user's commands. Memory charges are for the memory resources used in response to the user's commands; this is measured in different ways on the two systems since they have different implementations of multiplexing one physical memory amoung multiple users. Connect charges are for the communication link from the terminal to the computer system. I/O charges on TSO are for input/output operations performed between the memory and external devices (like disks). Finally, Storage charges are for online storage of files on secondary storage devices.

	Multics	TSO
Processor	$ .075/second	$ .116/second
Memory	$ .015/memory unit	$ .021/Kbyte-second
Connect	$ .021/minute	$ .021/minute
I/O	----	$ .001/ I/O operation
Storage	$ .50/page/month (32,768 bits)	$ .50/track/month (104,240 bits)

Table 1: Charges for Multics and TSO

One point should be made about charges verses capacities. Looking at Table 1 and comparing the processor charges for the two machines one might be tempted to next look at the speeds of the actual processors and then derive some feeling for the cost of doing a computation on the systems. As results later in this paper will show, this would be wrong because there is no reflection, for example, of the machine instructions produced by the compiler is response to a particular PL/I construct. What should really be considered is the cost of a system that includes not only hardware but also software; each system distributes charges differently over its various parts and the final cost is the sum of the costs for all parts.

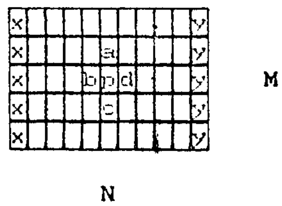
In the next three sections I will present a description of the experiment performed, the results of that were observed and finally, conclusions about the comparison of the two systems.

II. The Experiment

To arrive at a characterization of the relative performance of TSO and Multics with respect to cost, three computations were performed on both systems and costs as well as certain of the other characterizations described above were measured. As much as possible, the input to the two systems was identical; thus, for example, an attempt was made to write PL/I programs that would run on both machines. The three computations were: a processor bound task written in PL/I, an I/O bound task written in PL/I and a text formatting task written in the language understood by "Runoff" (for Multics) and "Nscript" (for TSO). In writing, inputting, compiling, running and debugging the programs to solve these tasks, a central part of each interactive system was used. By recording the cost of each interaction with the systems, comparisons can be made about relative costs.

The first task, referred to as "Relax", is intended to be processor bound. Imagine a pipe N units long with an M unit circumferance. If an ideal temperature source of X degrees is connected at one end and a second ideal temperature source of Y degrees is connected to the other, then along the length of the pipe, there will be a temperature gradient. The task of Relax is to compute this gradient. (This is a classic example that is usually given to show the power of Illiac IV, a machine which has parallel processors to perform this computation efficiently.) The data structure used to represent the pipe is an N+2 by M array where an element of the array is used to represent the temperature on a one square unit of the pipe as shown in Figure 1. The first row is assumed to be adjecent to the M^{th} row. The 0^{th} column of the array is held constant at X degrees and the $N+1^{st}$ column is held constant at Y degrees.

The computation is done by assigning as the value of each element the average of its immediate neighbors. This operation is repeated over the whole array until the values for one iteration are within 0.5 percent of the values of the previous iteration. This task can be made to use arbitrarly large amounts of processor time and memory space by altering the sizes of N and M. Also, no matter how arrays are linearized in the implementation of PL/I, the averaging of the four adjacent neighbors will cause references to relatively distant memory locations. Appendix A contains a listing of the program Relax that performs this computation; Relax consists of approximately 55 PL/I statements.



p = (a+b+c+d)/4

Figure 1: Computation done in Relax

The second task, called Record, is aimed at being an I/O bound computation. In Record, the user is prompted for information about a student's grades and that information is stored and later printed out in a neat report. (I do not advocate such a practice with student's grades since the confidentially of such information is not guarenteed even on a system like Multics). There is a lot of interaction with the user at the terminal and most of the program text is concerned with putting numbers out in the correct columns. Appendix B contains a listing of the program Record. Record consists of approximately 60 PL/I statements.

The third task, called Knuth, is a computation which is aimed at reproducing on the terminal in a pleasantly readable form the first page of The Art of Computer Programming, volume 1, by Donald Knuth. This task was chosen because each system has a program that is aimed at formatting text and I wanted to see how compatable the two services were. Appendix C contains a listing of the output of this task.

With these tasks in mind, there are a number of interesting costs to measure. In each of the descriptions of the subparts of the experiment, I will indicate in parentheses a capitalized name for the subpart. First, each of the programs will have to be entered in character form through a text editor to the computer systems (EDIT). I will compare the cost in terms of dollars and user time (connect time) needed to do the editing for one of these tasks, Knuth. Next, the two programs Relax and Record will have to be preprocessed before they produce any results. This preprocessing consists of compiling the PL/I statements into a language that is understood by the processor of the computer system (COMPILE) and linking the result of that computation together with externally defined routines that will handle such tasks as typing results on the terminal (LINK). The final step is running the program and receiving results at the terminal (RUN).

In the process of programming it is useful to print the contents of a file on the terminal (a listing of Relax, for example) (PRINT). In addition, it is interesting to find out how much overhead is involved in envoking a program (NOTHING -- a PL/I program consisting of a "procedure" statement immediately followed by an "end" statement.); this would be, for example, the cost charged for loading the program into the primary memory for execution. Finally, charges are made for the space in the file system occupied by programs and data (STORAGE); not only the rates, but also the amount of storage required to do the same tasks are interesting to compare. Table 2 contains a listing of the subparts and the commands or programs that are relevant to the specific systems.

In the course of this study, I have made a number of observations about the two systems that are difficult to attach a cost to but which certainly influence the attitude of a person interacting with the system. This group of observations includes such items as the nature of error messages, response time, ease of getting data from parts other than my own, etc. At the end of the results, I will attempt to express some of these observations.

Sub-part name	Multics **	TSO **
EDIT	edm	edit
COMPILE	pl1 (version II)	pli (optimizing)
LINK	"program name" (dynamic linking)	link
RUN	"program name"	call "program name"
PRINT	print	list
NOTHING	"null program" written in PL/I	"null program" written in PL/I
STORAGE	stored in segments	TSO on-line storage

** all program versions are those that were installed from 12/1/73 to 12/15/73.

Table 2: Commands that were used.

III. Results

Before any numerical results are stated, I will describe how costs were computed. On TSO, there is a command "eb" that types on the terminal the "estimated bill" for the current terminal session. This bill is broken up into charges for usage of the processor, memory, I/O channel and communication links. The cost of running eb is $.09 and this figure was subtracted out from all cost measurements to get the true cost of an interaction. The actual resource usage figures were derived from the cost figures and the rates for services. On Multics, the cost of an interaction was derived from the "ready" message. The total cost is the sum of the cost of the processor usage, the memory usage and the communication line usage (the time the terminal was connected to the computer system).

Perhaps the most disturbing result of this study is the variability of charges on Multics. The cost of doing, say, a PL/I compilation on Multics can vary by a factor of at least three. This is due to variations in system load, the multiplexing of memory by paging and the charging algorithms. The actual measurement given by the ready message is quite easy to work with since the incremental charges for each interaction are specified explicitly. TSO produces charges that are relatively constant for identical tasks, however the method of getting at these figures is awkward. The command "eb" produces a running total of charges, not an incremental one and must be called explicitly each time the cost of an interaction is needed.

As a final preface to the discussion of the numerical results, I must disclaim any efforts to extrapolate the results of this paper to general results about either system. The results of this study are a suggestion, but certainly not the final answer to the question "Which system should I use?"

Tables 3 and 4 contain measures of the cost broken down into components for the eight subparts of the experiment. Most of the notation is self explanatory except the parenthesized numbers under the subpart RUN. The size of the array for the Relax task is noted as "(N,M) K", where N and M are the dimensions of the array and K is the number of iterations over the array that were computed. Where K is not specified, the computation completed to convergance.

For EDIT, Multics appears to be cheaper. The two editors, "edm" and "edit" are quite similar and one gets used to working with either editor quite quickly. There are perhaps some defaults on the TSO editor that I find questionable (all input is mapped into upper case as a default; the "asis" condition must be specified if lower case is desired), however, this may be a subjective opinion. Line numbering is available in the TSO editor and can be a help or a hindrance depending on whether the user can remember which mode he/she wants. It is interesting to note that the "wall times" for inputting the text for the FORMAT task were almost identical for the two systems.

Multics

		processor		memory		connect		b s p f	total
Sub-Part		sec	$	units	$	min	$	number	$
EDIT									
	knuth	1.9	0.55	18.8	0.28	51	1.07	1074	1.90
COMPILE									
	relax	7.2	0.55	50.5	0.75	0	0.00	765	1.30
		11.7	0.88	134.8	2.02	1	0.02	1346	2.92
	record	5.8	0.44	23.5	0.35	0	0.00	288	0.79
		8.3	0.62	72.1	1.08	1	0.02	1209	1.73
LINK									
	relax	0.1	0.01	6.2	0.09	0	0.00	177	0.10
	record	0.4	0.03	6.3	0.10	0	0.00	108	0.13
RUN									
	relax								
	size: (0,0)	0.3	0.03	2.8	0.04	0	0.00	76	0.07
	(20,20)	7.3	0.54	4.5	0.07	2	0.04	133	0.65
	(50,50) 100	40.4	3.03	6.2	0.09	4	0.08	165	3.21
	(180,180) 1	58.9	4.40	176.6	2.65	14	0.29	2995	7.34
	record	5.7	0.43	29.4	0.44	8	0.17	779	1.03
FORMAT									
	knuth	1.9	0.15	5.6	0.08	3	0.06	82	0.29
PRINT									
	relax	1.2	0.09	4.1	0.06	3	0.06	84	0.22
	record	1.4	0.11	4.5	0.07	3	0.06	97	0.23
NOTHING									
	nothing	0.1	0.01	0.6	0.01	0	0.00	35	0.02
STORAGE									
	relax	relax pl1		1 record at $0.50/month/record					0.50/month
		relax		1 record at $0.50/month/record					0.50/month

Table 3: Costs of Experiments Performed on Multics

TSO

Sub-Part		processor sec	processor $	memory KByteHr	memory $	i/o Kops	i/o $	connect min	connect $	total $
EDIT										
	knuth	6.5	0.75	74.7	0.56	0.7	0.75	50	1.03	3.00
COMPILE										
	relax	3.2	0.37	33.3	0.25	0.3	0.34	0	0.00	0.99
		3.6	0.42	38.7	0.29	0.4	0.35	1	0.02	1.07
	record	3.2	0.37	33.3	0.25	0.3	0.34	0	0.01	0.93
		3.2	0.37	36.0	0.27	0.3	0.34	0	0.01	0.99
LINK										
	relax	1.4	0.16	34.7	0.26	0.3	0.33	0	0.01	0.75
	record	1.6	0.19	28.0	0.21	0.3	0.32	0	0.00	0.71
RUN										
	relax									
	size: (0,0)	WOULDN'T RUN FOR (0,0) CASE								
	(20,20)	3.3	0.38	26.7	0.20	0.2	0.19	2	0.04	0.81
	(50,50) 100	30.6	3.55	93.3	170	0.2	0.19	2	0.05	4.49
	(180,180) 1	5.1	0.59	81.1	0.61	0.2	0.21	1	0.03	1.44
	record	1.1	0.13	41.3	0.31	0.4	0.26	9	0.18	1.00
FORMAT										
	knuth	0.6	0.07	5.7	0.04	0.1	0.11	4	0.09	0.31
PRINT										
	relax	0.8	0.09	14.7	0.11	0.2	0.15	4	0.09	0.45
	record	NO DATA								
NOTHING										
	nothing	0.4	0.05	4.0	0.03	0.1	0.07	0	0.00	0.16
STORAGE										
	relax	relax pli 1 track at $.50/track/month								$0.50/month
		relax.obj 3 tracks at $.50/track/month								$1.50/month
		relax.load 24 tracks at $.50/track/month								$12.00/month

Table 4: Costs of Experiments Performed on TSO

For a short (~60 statement) program, the COMPILE part of the experiment shows that the TSO PL/I compiler is cheaper to run and will respond faster under all observed loads (the highest observed load on TSO was 28 out of 42 possible users while on Multics the figure was 67 out of 70 possible users). Under a heavy load, the Multics PL/I compiler responds like a compiler running on a good batch system. The lesson here is that PL/I compilations should be avoided on Multics during the peak load periods (2 pm to 5 pm). The slowness of the PL/I compiler on Multics is offset by the following consideration: a fair amount of the "work" done in compiling, linking and running a program on Multics is done in the PL/I compiler. If the worst times for compiling, linking and running Relax for the (50,50) case are added, the sum cost for Multics is $6.23 while the sum cost for TSO is $6.31. This analysis is an indication of the different ways Multics and TSO distribute tasks.

It is possible to write programs in PL/I that with little alteration will compile on either system (TSO requires "options(main)" on the main procedure, Multics does not allow this declaration). An interesting comparison of the compilers can be made by attempting to compile programs that have deliberate syntactic and semantic errors and seeing how the compiler responds. The Multics PL/I compiler makes no attempt to analyze syntactic errors. It just notes that a syntax error occurred and prints out the statement that is in error. The TSO PL/I compiler (I used the Optimizing compiler because it was described as the system standard) attempts to determine the nature of syntax errors and in doing so, often produces messages that are difficult to understand; several times, messages that were obviously supposed to have fields filled in had them in their "raw" state. In general the information content of error messages produced by the compilers for semantic errors is about the same; Multics is perhaps a little kinder to the user of a slow printing terminal by abbreviating error messages on their second and later appearances in a compilation; the TSO compiler makes no attempt to do this.

The LINK part of the experiment shows that the Multics dynamic linking facility is relatively inexpensive when compared to the TSO "link" command. The cost of dynamic linking was determined by envoking the program immediately after it was compiled and interrupting the execution after all callable routines had been called (there are several interactions that stop the computation so that this is feasible). Then the same thing was done a second time. The difference in cost between the two interactions is the cost of dynamic linking. On TSO, linking is relatively expensive and at least during debugging, linking is done quite frequently -- at least as many times as compiling. Also, when considering the high cost of storing the output of the link operation on TSO, the "load" module, linking a program each time it is to be run might be more cost effective than linking once and saving the load module. The command "loadgo" was not studied, but I suspect it would produce results more favorable to TSO.

In the section of the experiment concerning the running of the PL/I programs (RUN), Multics and TSO are reasonably close for the I/O bound task, Record. The differences in the implementation of the interactive parts of PL/I are present, but not bothersome (TSO produces a prompting character which slows the process of inputting data). When running the program Relax, the generality of the

Multics system allowed for larger values for N and M than the TSO system. In fact, for some unexplained reason, the TSO version of Relax caused a "system error 0c5" for the case N = 0, M = 0! While TSO could not allocate enough storage for the (200,200) case of Relax, the Multics version ran into high paging rates when the system had 67 out of 70 user and 45 out of 100 user loads.

In the tests of the formatting routines "Runoff" and "Nscript", the costs are almost identical, however, when coupled with the higher cost of editing on TSO, Multics looks like the more desirable system. When printing the text of the PL/I program Relax, an interesting effect was noted (originally pointed out by Art Evans): Multics prints the character " " (space) in whatever case the Selectric typewriter happens to be in while TSO always prints space in lower case. The "wall time" for printing the text of the program Relax on Multics was 3 minutes and 35 seconds while on TSO, it was 4 minutes and 25 seconds! Apparently the time for shifting cases on a Selectric typewriter is substantial. The cost of running the NOTHING program on TSO was higher than on Multics. This results probably from the large amount of PL/I operator routines that are always bound by the "link" command on TSO, but which were never referenced by the null program. This result lead Mike Schroeder to say, "NOTHING costs less on Multics." Who's on first?

Finally, the disparity between the <u>amount</u> of storage required to store compiled programs on TSO and on Multics is remarkable. The output of the PL/I compiler on TSO takes nine times the amount of storage that is required by the output of the Multics PL/I compiler for the program Relax. If load modules are saved, tremendous storage costs can be incurred on TSO.

IV. Conclusions

In conclusion, I propose a scheme for programming in PL/I in the MIT community. If the 370/165 could be connected to the APRA Network or even if a set of special purpose communication lines could be connected between the 370/165 and the 6180, then program preparation and debugging could be done on Multics and then the text of the debugged program could be shipped over the communication link to TSO. After one compilation and linking, the actual full size runs could be done on TSO or even submitted from TSO to run on the Job Processing System in batch mode. For highly interactive jobs, Multics is easier and more pleasant to use, but for speed and efficiency, TSO is hard to beat.

Appendix 1: Listing of Relax

```
     relax.pl1   12/15/73  1012.0 est Sat

relax: proc;

     dcl         (maxi,maxj,i,j,converge,im1,ip1,jm1,jp1,maxiter) fixed;
     dcl         (sysin,sysprint) file;
     dcl         (temp) float;
     dcl         response char(100) varying;
     dcl         (mod,abs) builtin;

     dcl         not_stable bit(1);
w: proc(msg); dcl msg char(*);
     put edit(msg) (a);
     put skip;
    end;

     call w("How many cells in X and Y directions?");
     get list(maxj,maxi);

     call w("How many iterations?");
     get list(maxiter);

     (nounderflow): begin;
          dcl a(maxi,0:maxj+1) float;

          call w("What is value for A(*,0)?");
          get list(a(1,0));
          a(*,0) = a(1,0);
          put edit("What is value for A(*,",maxj+1,")?") (a,f(4),a);
          put skip;
          get list(a(1,maxj+1));
          a(*,maxj+1) = a(1,maxj+1);

          do j = 1 to maxj; a(*,j) = 0.0; end;

          converge = 1;
 restart:
          not_stable = "1"b;

          do converge = converge to converge+maxiter-1 while (not_stable);
               not_stable = "0"b;
               do i = 1 to maxi;
                    do j = 1 to maxj;
                         jp1 = j + 1;
                         ip1 = mod(i,maxi) + 1;
                         jm1 = j - 1;
                         im1 = mod(i-2,maxi) + 1;

                         temp = ( a(im1,j) + a(i,jp1) + a(ip1,j) + a(i,jm1) ) / 4.0;

                         if abs(temp - a(i,j)) > 0.05*abs(a(i,j))
                              then not_stable = "1"b;
```

```
                        a(i,j) = temp;
                    end;
                end;
            end;

            if not_stable then
              do;
                put edit("After ",converge-1," iterations, still " ||
                    "no convergence.  Continue?") (a,f(4));
                put skip;
                get edit(response) (a(3));
                if response = "yes" then goto restart;
              end;
            else do;
                put skip; put edit("Relaxation",converge," iterations.")
                        (a,f(6),a);
                call w("  Print results?");
                get edit(response) (a(3));
                if response ^= "yes" then goto fin;

                put skip;

                do i = 1 to maxi;
                    put skip;
                    do j = 0 to maxj+1;
                        put edit(a(i,j)) (f(5,1),x(1));
                    end;
                end;
              end;
fin:
            end;

        return;
    end;
```

Appendix 2: Listing of Record

```
        record.pl1  12/15/73  1008.6 est Sat

 record: proc;


    dcl         sysin stream input ;
    dcl         sysprint stream output print;
    dcl         1 r(35),
                     2 name char(20),
                     2 ps(11) fixed,
                     2 quiz(2) fixed,
                     2 final fixed;
    dcl         response char(20) varying;
    dcl         num_students fixed init(0);
    dcl         (i,j,tquiz,tps,tnonzero) fixed;
    dcl         tpercentnonzero float;

w: proc(msg); dcl msg char(*);
    put file(sysprint) edit(msg) (a); put file(sysprint) skip;
    return;
   end;

read: proc(i); dcl i fixed;
    call w("name");
    get list(r(i).name);
    if r(i).name = "end" then return;
    call w("problem sets (11)");
    get list(r(i).ps(*));
    call w("quizzes (2)");
    get list(r(i).quiz(*));
    call w("final");
    get list(r(i).final);
    return;
      end;


    open file(sysprint) linesize(110);

    call w("type in records with name = 'end' as last entry");

    do i = 1 to 35;
        call read(i);
        if r(i).name = "end" then goto summary;
        num_students = num_students + 1;
    end;

summary:
    call w("index name      ps1 ps2 ps3 ps4 ps5 ps6 ps7 ps8 ps9 p10 " ||
            "p11 qz1 qz2 fin q+f +ps =ps");
    put file(sysprint) skip; put file(sysprint) skip;
```

```
    do i = 1 to num_students;
         put file(sysprint) edit(i,r(i).name,r(i).ps(*),
                  r(i).quiz(*),r(i).final)
            (f(3),col(7),a(20),col(30),14(f(3),x(1)));

         tquiz = r(i).quiz(1) + r(i).quiz(2) + r(i).final;

         tps = 0;
         do j = 1 to 11; tps = r(i).ps(j) + tps; end;

         tnonzero = 0;
         do j = 1 to 11; if r(i).ps(j) ^= 0 then tnonzero = tnonzero + 1;
         end;

         tpercentnonzero = (float(tnonzero)/11.0)*100.0;

         put file(sysprint) edit(tquiz,tps,tpercentnonzero) (3(f(3),x(1)));
         put file(sysprint) skip;
    end;

    put file(sysprint) skip;
    call w("modifications?");
    get list(response);
    if response = "yes" then
       do;
redo:           call w("index?");
         get list(i);
         if i = 0 then goto summary;
         call read(i);
         goto redo;
       end;

    return;
  end;
```

Appendix 3: Listing of Formatting operation, Knuth, vol. 1

Chapter One

Basic Concepts

"Many persons who are not conversant with mathematical studies imagine that because the business of (Babbage's Analytical Engine) is to give its results in numberical notation, the nature of its processes must consequently be arithemetical and numerical, rather than algebraical and analytical. This is an error. The engine can arrange and combine its numerical quantities exactly as if they were letters or any other general symbols; and in fact it might bring out its results in algebraical notation, were provisions made accordingly."
-- ADA AUGUSTA, Countess of Lovelace (1844)

"Wherever the term 'computer' or 'digital computer' appears throughout the text, replace it by the term 'Data Processor.'"
-- from a list of errata for a digital computer reference manual (1957)

1.1. ALGORITHMS

The notion of an algorithm is basic to all of computer programming, so we should begin with a careful analysis of this concept.

The word "algorithm" itself is quite interesting; at first glance it may look as though someone intended to write "logarithm" but jumbled up the first four letters. The word did not appear in Webster's New World Dictionary as late as 1957; we find only the older form "algorism" with its ancient meaning; i.e., the process of doing arithmetic using Arabic numerals. In the middle ages, abacists computed on the abacus and algorists computed by algorism. Following the middle ages, the origin of this word was in doubt, and early linguists attempted to guess at its derivation by making combinations like <u>algiros</u> (painful) + <u>arithmos</u> (number); others said no, the word comes from "King Algor of Castile." Finally, historians of mathematics found the true origin of the word algorism: it comes from the name of a famous Aribic textbook author, Abu Ja'far Mohammed, son of Moses, native of Khowārizm. Khowārizm is today the small Soviet city of Khiva. Al-Khowārizmī wrote the celebrated book <u>Kitab al jabr w'al-muqabala</u> ("Rules of restoration and reduction"); another word, "algebra," stems from the title of his book, although the book wasn't really very algebraic.