I wrote most of this material for the Douglas Aircraft Co. (DAC) manual between 1953 and 1955. It is the first section of the complete manual which describes the DAC machine-language coding system (symbolic, developed in 1952 for the 701, unique to DAC & perhaps one or two other firms) which centered on the assembler/reassembler which I also developed. (If you are interested, I can send you specs from that section of the manual. It allowed symbolic corrections to partially assembled code in 1953!) All work was fixed point or (slow!) interpretive floating point. Alpha was expensive.

The procedures & standard conventions of this section (along with many others of the manual) were rigidly enforced — especially those dealing with pre- and post- coding documentation — some procedurally, some by software checks.

I find the sections "Outline For Programming", "Outline for Program Checking", "Minimum Program Requirements", "Vellum Problem Statements", "Region Specification Form", "Problem Notebook" and "CETM" particularly relevant to the "Structured Programming" discussion: This was indeed structured programming, as these weren't merely words. J.R. Lowe saw to that, if he did nothing else!

As the years passed - the 701's lasted to the late 50's — the goals of systems development were always directed toward more standardization and structuring,

with many "fungible" routines, methods, etc. A frequent admonition was "Programming is an exercise in discipline, not self expression."

I suspect that this all came about because the precedents were taken from the Tabulating Shop, where such things were taken for granted.

Some translations:

Region xx : Subroutine, with more formal constraints & Restrictions [except Regions 1 & 2]

Region 10 : Mainprogram; again, relatively restricted as to form. Some "standard" Region 10's existed, e.g. for P.E. solving; Matrix work, etc

Region 1 : Temporary storage for calculations

Region 2 : Sorta like blank common

Region 3 : "3" used as a "local" designator, so that "Region 3" is always "this (contextual) region". So to's ~~mostly~~ probably restricted to Region 3 except subroutine linkages in standard form.

Load Sheet : Input form for application program

It turns out that the goals at the middle and bottom of Page 03.02 (next to last) were indeed largely met: programs were reassignable (the concept of a "pickup" is an old tab room standby) and operations did take over & the programmer got out of the loop.

Once again, I attribute most of the success of DAC (and her alumnae) to J.R. Lowe. He wasn't lovable (or loved) but the work got done-right.

N.B. Period pieces always have more appeal to those who lived through the "good old days" and even for whom a little jargon evokes copious memories, well filtered by time.

INTRODUCTION TO THE MANUAL

The Computing Engineering Manual has been prepared to make accepted computing methods, procedures, and routines readily available. The material presented in this manual results from the organization and standardization of tested experience and proven data which has been accumulated by the Douglas Santa Monica Computing Engineering Section since early 1951.

The success of the application of these methods and procedures has been recognized throughout the industry. This fact dictates that the material specified must be followed if a high standard of quality is to persist. On the other hand, it is recognized that progress depends upon the flexibility of procedures. Recommendations for improvement are invited; but deviation from the established practices should not be made without proper authorization.

FUNCTIONS OF THE COMPUTING ENGINEERING SECTION

Introduction

The Computing Engineering Section provides computational and mathematical services to the remainder of the Engineering Department. The computational service is restricted to problems requiring the use of high-speed computers, either analog or digital. The mathematical service is normally that associated with programming of problems to be solved on the Section's computing equipment but it is not necessarily so restricted.

Organization

The Section is composed of five groups:

1. Analysis
2. General Computing
3. Analog
4. 701 Operating
5. Key Punch

While the activities of these groups are largely distinct, there is, and should be, overlap in their interests. This manual is primarily for the use of the Analysis and 701 Operating groups. However, the interests of efficiency and personal advancement dictate that the personnel of these two groups be very much aware of the activities, problems, and capabilities of the other groups.

Responsibilities

A. Service. The basic function of the Section is to provide computational and mathematical service to the remainder of the Engineering Department. To be effective, this service must have three characteristics: It must be accurate, timely, and orderly.

Accuracy must be safeguarded at two basic levels - the program itself and the machine. Pointers on avoiding programming errors occur elsewhere in this manual, particularly in the section on "Standard Operating Procedure." However, in the final analysis, the question of whether or not a program is adequately checked depends upon the professional judgment of the programmer. A further responsibility of the programmer is to incorporate in his program, or in associated procedures, definite safeguards against machine errors which may arise in production running. Again, determining a reasonable compromise between adequate checks and wasting machine time by superfluous checks is a matter of programmer judgment. Perhaps in these two areas of determining adequate checks of accuracy must one differentiate between the outstanding, as opposed to the merely mediocre, programmer.

The nature of the aircraft business is such that considerable time must elapse between the final freezing of a design and its being used in the field. The state of the art is advancing so rapidly that the original design is inevitably, in some measure, obsolete by the time it is in use. Furthermore, during this lag time potential profits from the sale of a new article are being lost. For these and many other reasons it is most important that the elapsed time to

FUNCTIONS OF THE COMPUTING ENGINEERING SECTION

Responsibilities (cont'd)

engineer a new article be minimized. Computing Engineering shares this burden with other engineering specialties. Furthermore, since time is very much of the essence in our total effort, scheduling becomes vitally important - a responsibility which we must again share by giving accurate estimates of the time required to accomplish a given task.

A basic requirement of all engineering is to leave accurate records of work accomplished. These records, in our case, take the form of adequate, neat, and orderly program books, problem statements, and Computing Engineering Technical Memoranda. The contents and format of these are discussed elsewhere in this Manual.

B. Liaison and Consultation. An important responsibility of Computing Engineering is to provide expert consultation in matters concerned with computing, data processing, and data reduction. This involves acquainting the rest of the Engineering Department with the facilities and methods which are available and endeavoring to see that computing plays its proper role. In part, this implies that each problem must be examined to insure that it is of the proper scope, i.e., that neither too much nor too little of the total picture is included. Including too much implies that our equipment and service are being employed inefficiently; including too little implies that time and effort are being wasted elsewhere.

An important function of Computing Engineering, more by accident than design, is to act as a liaison agent between other engineering groups. The proper discharge of this responsibility insures that programming effort will not be duplicated and that all groups can take advantage of what has been done for any group. Furthermore, it is often feasible by such liaison efforts to make certain programs much more useful by incorporating the ideas of several groups in them.

C. Equipment and Methods. The tools necessary to provide the foregoing services include computing machines and methods for using them and numerical and computational methods and techniques. Therefore, an important responsibility of the Section is to be entirely aware of the latest developments in all these areas and to insure that its tools are the best possible.

OUTLINE FOR PROGRAMMING

I. ANALYSING THE PROBLEM

A. Discuss the problem with the engineer to determine the following:

1. Input
   - a. What are the input quantities?
   - b. Should we generate some of the input quantities?
   - c. How accurate is the input data?
   - d. What are the dimensions or units of input quantities?
2. Output
   - a. What is the form of the output?
   - b. How will the engineer use the output data?
   - c. Should we further process the output for the engineer?
3. Ranges of Variables
   - a. Input data
   - b. Intermediate quantities
   - c. Output data
4. Physical Checks
5. Possible Pitfalls
   - a. Physical
   - b. Mathematical
   - c. Numerical
6. Possible changes (and modifications)
7. Size, frequency, and number of runs
8. Are there other engineers who might be interested in the problem?
9. What similar problems have been done or are being considered?
10. Is this a modification of an existing program or should it be coded completely?

B. Analyse the Problem Numerically with Respect to:

1. Simplification of equations
2. Methods of solution
3. Methods of checking
4. Arithmetic system

C. Restate the Problem on Vellum Showing:

1. Problem description
2. Input
3. Procedure of solution
4. Output
5. Ranges of variables
6. Equations in sequence of calculation

II. PROGRAMMING THE PROBLEM

A. Storage Assignment

1. Plan a rough storage assignment of auxiliary storage.
2. Assign Region 2 storage.

B. Design Load Sheets

1. Discuss with the engineer the form and arrangement of the input data.
2. Form a rough draft of load sheets.
3. Have coordinator and machine room leadman check load sheets.
4. Make vellum load sheets (or have them drawn by artist). Always try bluelines of the load sheets for a few production runs before having printed ones made.
5. Obtain machine room leadman's final approval of completed load sheets.

C. Arithmetic Regions

1. Draw a rough flow chart of arithmetic calculation.
2. Specify regions in logical sequence.

3. Draw region flow charts where necessary.

4. Code Regions

5. Check Regions

D. Control Regions

1. Draw a rough flow chart of control regions with an eye to:

a. Reloading
b. Restarting
c. Changes
d. Assembly
e. Ease of check out.

2. Specify control regions

3. Code control regions

4. Check control regions

5. Correct the flow charts

III. TESTING THE PROBLEM

A. If practical, test out logical blocks of regions independently. Always use realistic and non-degenerate data.

B. Prepare an overall test case of the problem. If feasible, have the customer prepare at least an order-of-magnitude test case.

C. Make the pilot runs.

D. Reconcile the test answers with the engineer.

E. Give consideration to conditions not tested.

IV. MAKING THE FINAL REPORT

A. Correct the vellum statement of the problem.

B. Write the operating instructions and organize the notebook.

C. Explain the problem to the machine room leadman.

D. Write a Computing Engineering Technical Memorandum (C.E.T.M.) if the problem is a production problem.

OUTLINE FOR PROGRAM CHECKING

I. What to do while coding the problem.

A. Specify the regions & data storage so that:

1. Unless storage is tight, all results of a region are saved in region 2.

2. It will be relatively easy to force the program through all logical branches of a program.

3. If necessary, a region may be checked out independently without an elaborate test region 10.

B. Include extra coding helps to simplify check-out, such as:

1. Special print-outs, by means of sense switches, for portions of problem not ordinarily printed.

2. Provisions for saving data to be printed out by auxiliary programs.

3. A normal print routine as a separate region so that control may be transfered to it in case of difficulty.

4. Provision to by-pass certain regions, expecially time consuming ones, when their results are input from cards.

5. Any programmed stop should have an address such that pushing the START will continue the program even if using bad data.

II. What to know before getting on the machine.

A. Always have definite purpose well in mind before getting on the machine; e.g.

1. To check certain arithmetic computations.

2. To check logical branches of the program.

3. To run a new set of data to check new scaling or logical conditions.

4. To try a revised method of numerical computation to shorten time or increase accuracy.

Note: When correcting a previously found mistake, try to check as many other conditions as possible at the same time; i.e., assume that the correction does fix the mistake.

B. Have a listing or notation, preferably on the "Octal Storage Assignment" sheet, so as to be able to determine readily if

DOUGLAS

OUTLINE FOR PROGRAM CHECKING

II. What to know before getting on the machine (cont'd)

a program stop, divide check, loop, etc. is in:

1. Data storage.
2. Region 1.
3. A subroutine from tape (be able to tell which one).
4. A region of your own coding (be able to tell which one).
5. Any locations modified by octal changes.

C. Know what loading sequence to use and what sequence of events to expect; e.g.

1. Reset & clear, load, for program deck.
   a. Pattern of lights when loading.
   b. Skip card cycles at reader while loading binary cards to drum.
   c. Stop at 7764_8 if sequence or check sum error.
   d. Stop at 0000 with accumulator equal to H 2+, H 3+ from standard load card.
   e. Tape 4 rewinding during the loading process.
2. Load for octal changes.
   a. Pattern of lights characteristic of DAC 003.
   b. Stop at 0000 with C(P,Q, acc) = 2000 . . . when finished loading.
3. Start for program.
   a. Reading of subroutines from tape and pattern of lights when doing this.
   b. Reading of header cards.
   c. Skip card cycles at reader while loading headers to drum.
   d. Reading of input data. Know the pattern of lights for a DPBC error in the decimal reading program.
   e. Pause for calculating. Know how long this should take before using an input-output device.

OUTLINE FOR PROGRAM CHECKING

II. What to know before getting on the machine (cont'd)

f. Printing of results.

g. Reading of more data or copy check when trying to do so.

D. Know how to use the vaious diagnostics.

1. M.P.O.

a. Switch settings for normal use.

b. Method of skipping portions of memory.

c. Saving particular drums while printing.

d. Restoring memory and proceeding after M.P.O.

e. Use of DAC 009 to reduce printing time.

2. T.P.O.

a. Switch settings for proper tapes. Do units need to be interchanged to print tapes 5 & 6?

b. Loading without rewinding.

c. Printing various ways, e.g. decimal, octal, etc.

d. Printing headings only, partial records, skipping records.

e. Use of TPO for printing drums in decimal (or memory, after loading memory to drum with M.P.O.).

f. Changing to next unit after printing from the first.

III. What to do before getting on the machine

A. Check the set-up of the deck for:

1. Proper combining of multiple assemblies.

2. Binary cards in sequence with 12-80 card (or 12-46 if necessary).

3. DAC 003 in proper place (before permanent headers?)

4. Octal changes properly punched.

a. Always have someone check unverified keypunched cards.

b. Avoid making octal changes on changes, and reassemble when number is great enough to warrant it.

DOUGLAS

OUTLINE FOR PROGRAM CHECKING

III. What to do before getting on the machine (cont'd)

5. Header cards in proper place and sequenced.

6. Data correctly punched, in sequence, and with 12-80.

a. Always have someone check unverified keypunched cards.

B. Take the necessary information to the machine room with you. (see II B above)

1. Assembly listing.

2. Data load sheets.

3. Previous run for comparison purposes.

C. Tell the machine room staff the following:

1. How long you expect to be on. It may be that in case of trouble it will only run a few minutes, but may go a long time. Ask the machine room staff if this is O.K.

2. What tape units need special tapes put on and what tapes need to be free for writing.

D. Be sure that you are ready to go on the machine and that you have what you want to accomplish well in mind. If not, wait a while. Never rush to get on the machine.

IV. Running the Program

A. Check the console for:

1 Address entry keys set to 0000.

2. Automatic-Manual switch to automatic.

3. Proper sense switch setting.

4. Load selector to cards.

5. Printer, Punch and Reader unselected.

B. Load the program. While it is loading:

1. Get the input-output units ready.

a. Printer with proper board, on ready, with paper restored.

b. Punch with proper color cards, on ready, with correct switch settings.

OUTLINE FOR PROGRAM CHECKING

IV. Running the Program (cont'd)

2. Check the operation of the machine, comparing the sequence of events with that anticipated in II-C above.

a. Note that all expected actions take place.

b. Watch for unexpected events, such as tape rewinding, too many cards read, etc.

C. If program is running properly, tell the machine room staff how long you expect to be.

D. If the program does not run, the difficulty and remedy may be one of the following:

1. Program stop.

a. Due to improper loading sequence? Load with proper sequence. Check-sum error? Fix card with DAC 011.

b. D.P.B.C. error? Fix improper card and re-read it.

c. In input-output subroutine? Proper tape on unit? Proper cards in reader?

d. In own coding? Stop programmed as check on machine operation? Stop due to improper scaling or calculation? Check location of stop and check arithmetic on Friden (Partial M.P.O.). Push START button and continue. Unprogrammed? Search for store with DAC 008.

e. In data storage? Does accumulator say RA XXXX+? Check for improper transfer at XXXX+1. Does L(Halt)-1 say halt? Search for T L (Halt) with DAC 007.

f. At loc. 0000? Check for region with exit address not set up.

g. If possible, push start to continue after Halt.

2. Divide check.

a. In input routine? Check for proper scaling of input on cards.

b. In subroutine? Check exit address of subroutine to determine at what point it is being used. Proper calling sequence?

c. In own coding? Division by 0? Check for data not entered or results of previous calculation not stored correctly.

OUTLINE FOR PROGRAM CHECKING

IV. Running the Program (cont'd)

d. Caused by improper scaling? Note amount by which C(acc.) exceeds C(memory reg.).

e. On most divide checks press START and continue.

3. Copy Check.

a. In input-output routine? What unit was being used? Reader? 12-80 last card? Tape? Proper tape on unit? Proper calling sequence? Tape in proper status?

b. In own coding? Was proper unit selected? Too much time used? End of file or end of record test made incorrectly? Too much time between copies?

c. In region 2? All subroutines entered from tape? Does accumulator say RA XXXX? Look for improper transfer at XXXX+1. Search for improper transfer to L (copy check) - 1 with DAC 007.

4. Tight loop. Put on manual and use multi-step key to determine extent of loop (if stop is in subroutine, put on automatic, start, and try again to stop in own program). Check for:

a. Proper set-up and test words for loop.

b. Iteration which is not converging. If almost converged, transfer to exit from iteration.

5. Input output select. Check address in instruction register.

a. Printer or punch? Put on ready.

b. Card reader selected? Run-out switch off? Card feed stop? Fix card and run in again.

c. Tape? Tape ready? In proper status? Write-Non-Write switch to non-write?

d. Does C(inst. register) say C XXXX? Unit selected is not ready.

e. Does C(instr. register) contain a legitimate input-output instruction? A previously given input-output instruction was not executed. Trying to rewind tape? Put tape in ready. Drum selected? No copies given after Read or Write drum instruction.

OUTLINE FOR PROGRAM CHECKING

IV. Running the Program (cont'd)

f. Does C(instr. register) contain a non-legitimite input-output instruction? Address set pu improperly? Executing data or instructional constants? Search for transfer with DAC 007. Not programmed? Search for store with DAC 008.

E. When one of the above difficulties occurs:

1. If the error is very easily corrected, do it at the console and continue with the program.

2. If the error is correctable, but will require new octal changes, new data, etc., do not try to correct the trouble at the machine. Get off, fix the trouble, and get on later.

3. If it appears that the trouble will be easy to locate, write down the contents of the console and get off without using diagnostics.

4. When the error appears to be obscure, use one or more diagnostic programs.

a. M.P.O. Print as little information as you can. DAC 009 should be used when possible, and subroutines in region 2 should be skipped.

b. T.P.O. Print as little information as you can. Be sure to know desired q for decimal, etc. Print only part of each record (Heading, first and last line) unless entire record is needed.

c. Drum print-out. Use T.P.O. for decimal information, M.P.O. for octal or instructions.

d. Use the search memory cards (DAC 007 and 008 to find unwanted transfers or stores.

5. Always have a reason for doing something at the machine. Do not print memory, for example, just because you think it might be useful.

a. Remember - When in doubt, get off the machine.

V. When getting off the machine.

A. Always try to warn the machine room staff well in advance so that the next person may be called.

B. Take all your material with you.

DOUGLAS

OUTLINE FOR PROGRAM CHECKING

V. When getting off the machine. (cont'd)

1. Run cards out of reader and band them immediately so that the next person can load his cards.

2. Replace diagnostic programs in the card file.

3. Remove any cards from the punch, band and label them.

4. Restore the page at the printer and remove any listings. Date them with the time stamp if they can be confused with other listings.

5. Tell the machine room staff which tapes to save, if any, and be sure to label them. Replace them in the tape files.

6. Take all your material away from the console, printer and punch immediately. Throw unwanted material away.

7. Be sure your time card is filled in and stamped out.

C. Do not put your name on the board for more machine time right away. You may be able to find several errors by a more critical examination of the results obtained.

VI. Checking the results.

There are as many methods of trying to find errors as there are errors. In general, error detection should be broken into two phases.

A. Program control and logic.

1. Check to see that the exit address of each region has been set up. If not, either that region has not been executed, or the exit has not been set properly.

2. Check the calling sequence for each region.

a. Was α RA α + used?

b. Does the transfer go to the right place? Are one instruction regions correct?

c. Are the right number of halt instructions present? Was allowance for end of record and end of file exits made?

d. Are subroutines entered in the proper place in region 2?

e. Are the orders to set the exit address, etc. correct?

OUTLINE FOR PROGRAM CHECKING

VI. Checking the results. (cont'd)

3. Do 2.0, 2.1, 2.2, and 2.3 still contain 0,1,2, and 3? These are used to compute transfer addresses.

4. Within a region, do conditional transfers all go to the correct place? Were initial addresses and test words computed correctly?

B. Program Arithmetic

1. Check computations in the order in which they occur.

a. Has correct data been entered?

b. Is scaling correct?

c. Have intermediate answers been put in the proper place?

d. Is round-off error a factor?

e. Do incorrect answers seem to be a function of the binary number system? e.g., off by a power of two, or related to all 1's?

2. When wrong answers are calculated, use these wrong results as input for further calculations and continue with test case.

VII General Notes

A. Try to get program control working first. It is easier to check for wrong answers when the program runs through completely.

B. Check all arithmetic thoroughly, with a range of input data. Don't rely on one test case.

C. Use a simple test case for early runs. This will save machine time and the test case will be easier to calculate. Work up to the complicated test cases slowly.

D. Check all portions of the program as completely as possible. Don't rely on the simple test case for the final check-out.

E. Note any difficulties due to data. These same troubles may occur after the program is in production.

F. Save difficulties with getting pretty print-out format, headers, etc., until last. It may pay to use a different print-out form (e.g., full words instead of half words) during check-out and change this at the final reassembly.

G. GOOD LUCK ! ! !

701 IDENTIFICATION NUMBERS

Three kinds of identification numbers are used for 701 work: a program number, a job number, and a case or run number.

Program Number

A four decimal digit number, consisting of three whole numbers and a dash number, is assigned to each program or subroutine coded for the 701, and should completely identify the program with regard to its mathematical or logical function. The dash number is used primarily for variants of a given program. A complete index of these program numbers is maintained in the Computing Engineering Job Log.

Job Number

A four digit number is assigned to each job brought to Computing Engineering for solution by the 701. This job number is the accounting charge number to be used on the daily time slips within the Group. Together with the case number (see below), it will identify the data for a given job. A job number index is available, cross-referenced to program number, in the Job Log.

Case or Run Number

Six digits of case or run number are used to identify a particular set of data for a given job, and are normally assigned by the engineer. The six digits may be grouped into fields, if necessary, to provide complete identification.

Identification Card Columns

The first eight card columns are available for identifying 701 cards, and have been assigned as follows:

Type of Cards

Instruction Cards-symbolic, $\alpha\beta\gamma$, or binary.

Data-decimal or binary.

1	2	3	4	5	6	7	8
Sequence Number			N	Program Number			Dash
Job Number*				Case or Run Number			

*Last two digits

Notes:

N is an assembly identification field; when a binary program deck consists of several assemblies, each assembly is assigned an identification symbol, which may be alphabetic or numeric. If the program requires only one file or

701 IDENTIFICATION NUMBERS

Notes: (cont'd)

The three sequence number positions are used for the decimal sequencing of all complete binary program decks, including fixed header cards, and all $\alpha\beta\gamma$ cards beginning with 000. Additionally, all $\alpha\beta\gamma$ cards have a common X-1 punch. Symbolic cards are not sequenced in columns (1 - 3).

When punching cards from 701, depending upon the output routine being used, up to eight of the identification columns may be punched under program control.

PERMANENT OCTAL OR DRUM CHANGES

1 2 3	4 5 6 7	8 9 10
Sequence Number	Program Number	Block Number

All octal changes or drum changes which will perform special functions with a binary program deck and become part of the permanent file should be identified according to the card form described below.

Sequence numbers will begin with 000 for each block of octal or drum changes.

Program number will be that of the program number affected by the changes. In addition, there should be a common X punch in c.c. 8 of each change.

A new block number will be used for each set of changes and a summary should be provided in the problem notebook to indicate the function of each different block number. The maximum block number is 89.

DATA LOAD SHEETS

Each data load sheet for a particular program should contain the program number of its associated binary program deck, spaces for the job and case numbers, and an indication of the card columns into which these numbers are to be punched.

701 OUTPUT FORMS

If at all possible, each sheet of output printed from the 701 should include, in addition to a descriptive title, the job number and case number of the run.

ASSIGNING JOB AND PROGRAM NUMBERS

When a new program is started, an identifying program number must be assigned to it, as well as a job number which will determine the accounting charge for programming time.

Program numbers are assigned sequentially from the Program Number Index of the Computing Engineering Job Log. Job numbers are also assigned sequentially, and entered in the Job Log, together with the corresponding program number, and the following information, obtained from the engineer requesting the program:

Shop Order
Engineering Work Order
Operation
Model
Group Name and Number
Engineer
Job Title

LOAD SHEET DESIGN

The load sheet provides a means of communication between the engineer and computer. It must, at one time, satisfy the needs of both. This demands that careful consideration be given to its design to ensure that this important function is carried out in an efficient manner. The principal requirements are:

a. General The load sheet must be complete and accurate. This is best proved by processing a complete test case whose input data has been prepared on the final load sheet. A check list has been included below to aid the analyst in checking for completeness.

Design should allow for minimum computer input time. Considerable savings can sometimes be realized by a change in format or the introduction of a preliminary standard equipment procedure.

Whenever possible, the load sheet should be designed to allow for standard equipment or 701 input data checking; e.g., sequence checking or card counting.

b. Keypunch Requirements The load sheet must be designed to allow a straightforward and routine punching pattern. When a complicated input format occurs, a moderate change in the program or the introduction of a preliminary standard equipment card processing procedure can be justified.

Sufficient keypunching instructions must be recorded on each document. Separate keypunch and preparation instructions to achieve clarity.

All card columns must be clearly defined. Keypunching procedures should be forwarded to the Keypunch Group for all non-standard card formats. Employ as few input formats for any one problem as possible.

The keypunch operators cannot be expected to be intimately familiar with each job.

c. User Requirements. Considerable attention must be given to the user. Data field assignment should allow for ease of completion and checking. Preparation procedures should follow a logical pattern consistent with standard engineering procedures. Knowledge of the source and nature of the data will be of valuable assistance.

DOUGLAS

LOAD SHEET DESIGN

CHECK POINTS

1. All load sheet designs will be sketched in full for approval by the problem coordinator and the machine room leadman before final forms are released.

2. The following information will be recorded at the top of each load sheet:

   a. Program number
   b. Job number blank
   c. Run or case number blank
   d. Input format number
   e. Card color
   f. Program title
   g. Name blank
   h. Date blank
   i. Page number (when required)

3. Indicate keypunch instructions for zero values and blank fields.

4. Denote all decimal points.

5. Record all symbols. Include units only when required for clarity.

6. Indicate all 12 - 80 punches. When a 12 - 80 punch is always made, do not use question type signal. Place 12 - 80 punch instructions as close to data fields as possible.

7. Use "pips" in all data fields.

8. Do not use arrows or ditto marks for repeated values.

9. Define multi-color card arrangements only when absolutely required. The standard data card color is salmon.

10. When using non-standard card formats (e.g. input #8) label all card columns. Column numbers will be recorded at the beginning of each new field at the top of the respective column.

11. Do not permanently record information that may change on the master sheet; e.g., job number.

12. When a preliminary standard equipment data preparation procedure has been defined, check card layout for minimum card processing.

MINIMUM PROGRAM REQUIREMENTS

In an effort to facilitate certain standard operating procedures, the following minimum program requirements are established:

ALL PROGRAMS SHOULD:

1. Provide a method to skip reading permanent headers subroutines from tape.
2. Print at least one heading with any printed output.
3. Provide a method for printing complete "blocks" of data on a single page.
4. Use sense switches for error or restart procedures only.
5. Use control codes(input) for altering procedures.
6. Provide a method for running multiple cases, runs, jobs, etc. without external control.
7. Provide a method for starting or restarting a case, run, job, etc. without reloading the program.
8. Provide a method for breaking long cases, runs, jobs, etc. such that computation can be continued at a later time.
9. Provide a method for checking during use of tapes or drums. If checks fail, re-read and check N times before halting.
10. Provide internal checking of input for sequence, proper data, volume, etc. when large quantities of data are required, or when a single case runs excessively long.
11. Indicate progress (with sense lites or printing) on excessively long running programs.
12. Provide a maximum and exit on iteration problems.

VELLUM PROBLEM STATEMENTS

A problem statement is a statement of the mathematical problem and the proposed method of solution. It may also include a statement of the associated physical problem wherever this aids in the understanding of the mathematical problem. By means of the problem statement, the problem is "frozen" after a certain amount of discussion between the customer and the programmer. This is important, because the programmer usually cannot assign storage, decide on scale factors, and determine the most efficient sequence of calculations until the problem is fixed in detail.

A problem statement can also be regarded as an informal contract between the customer and the analyst. It specifies what the customer's requirements are and what the analyst has committed Computing Engineering to produce.

The problem statement should include the following:

1) Problem description
   - a) physical (if necessary)
   - b) mathematical

2) Definitions of quantities and symbols

3) Ranges of variables and parameters (give units)

4) Input
   - a) list of given quantities (classify as to constant, parameter, or variable)
   - b) indication of how many significant figures in the given quantities, if this information is important
   - c) indication of number of cases, if possible
   - d) proposed load sheet(s), if possible

5) Procedure
   - a) numerical methods used
   - b) equations, arranged in a logical order (indicate whatever tests are to be made, including tests for errors or faulty data)

6) Checks on results
   - a) checks in the program itself
   - b) checks made after the program is run (e.g., plotting results)

7) Output
   - a) list of computed quantities
   - b) indication of the form in which they will appear, if possible

See the problem statement notebooks for examples.

REGION SPECIFICATION FORM

JOB NO. PAGE NO.
ANALYST PROGRAM NO.
DATE PROGRAM TITLE

REGION XX

I.	FUNCTION:	A brief description of what the region does.
II.	INPUT:	A list of flags and counters, data tables, and individual quantities needed for calculation with their respective q's and locations. Any necessary information about card, tape, or drum input.
III.	OUTPUT:	A list of the output quantities, including flags, counters and data tables with their respective q's and locations. If necessary, information on card, tape, or drum output.
IV.	EQUATIONS:	List of the calculations to be performed.
V.	PROCEDURE:	An ordered list of the computations, tests, and any non-arithmetic procedures to be done in this region.
VI.	CALLING SEQUENCE:	Unless the region is entered by basic linkage, give the calling sequence and explain each item in it.
VII.	TEMPORARY STORAGE:	The region one or region two storage available for use by this region.
VIII.	SUBROUTINES USED:	A list of all regions used by this region. Miscellaneous and standard subroutines should have their locations in the manual specified. For others, furnish calling sequence and a brief description.
IX.	ERROR PROCEDURE:	Type of errors for which to test and the procedure to follow in case the error occurs.
X.	MISCELLANEOUS:	Any information about the purpose of the region, about the mathematical procedures involved, about any unusual nomenclature, systems, or coding techniques, etc., that might be an aid to the coder.

PROBLEM NOTEBOOK

The machine room notebook is a detailed, technical report on the program. It has two functions: first, it is the source of information for machine room personnel on the operation of the program, and second, it is a repository of all the useful information about the program.

In line with these two functions, the notebook should contain:

1) All the essential information for running the program.

    a) Use the standard two-page form (with added pages if necessary) to indicate loading instructions, sense switch settings, error halts, special instructions for processing the data, etc.

The goal is to enable the machine room personnel to quickly and accurately grasp the steps in running the program.

2) All other useful information about the program

    a) A copy of the problem statement
    b) An up-to-date assembly listing
    c) Storage assignments (symbolic and octal)
    d) Flow charts
    e) A list of the regions and their functions
    f) Region specifications
    g) Notes on special mathematical or numerical methods used
    h) The original symbolic coding, if in a useful form
    i) Test case(s) with input data and results
    j) Card forms when necessary

The goal here is to have the notebook both complete and logically arranged, so that another programmer can study it and understand the program completely in minimum time.

COMPUTING ENGINEERING TECHNICAL MEMORANDUM (C.E.T.M.)

A C.E.T.M. is a report which introduces the program to prospective users. Its purpose is to inform people outside our group of the existence of the program and to give them enough information about it to enable them to decide whether or not it can be useful to them.

A C.E.T.M. usually consists of a one-page memorandum and a number of appendices. The appendices comprise a blue-line copy of the problem statement, sample load sheet(s) filled in with sample data and typical identification (plus any required explanations), and one or more pages of output, corresponding to the sample input data (with explanations if required).

The covering memorandum usually gives a brief introductory statement of the purpose of the program and what it does. It also should include as many of the following points as are relevant:

1) Any important limitations on the use of the program
2) Significant differences between this program and other similar programs
3) If the program is a revision of an older existing program, the reason for the revision
4) Summary information on filling out load sheets, including an explanation of identification numbers
5) An indication of running time (for a typical run, or for the sample case given in the appendices, or both)

The standard heading for the memorandum is as follows:

To: (Section Chief)

From: J. R. Lowe, A-250

Subject: (Title of program), Program No. xxx.x

Copies to: (List of all people who require copies)

Copies are sent to the customer, the customer's supervisor, other people the customer feels may be interested, and to the supervisors of the Computing Engineering groups at the Company locations in El Segundo, Long Beach, Tulsa and Missiles Computing Engineering at Santa Monica.

See the C.E.T.M. file for examples.