TEACHERS' INVESTMENT AND HOUSING CO-OPERATIVE

PROGRAMMING STANDARDS MANUAL

TO: G. Matus
M. Lee
D. Everett
B. Sameshima
A. Akizuki

The standards in this manual should be regularly reviewed to ensure they are up to date, realistic and useful.

Comments and suggestions should be directed to B. Sameshima.

TEACHERS' INVESTMENT AND HOUSING CO-OPERATIVE

PROGRAMMING STANDARDS MANUAL

INDEX

Teachers' Investment and Housing Co-operative
Programming Standards

Computer Department Testing and Approval Standards

In future, before any new or modified program(s) are moved to production, the following procedure is to be followed in addition to the required sign-off sheet:

- a meeting to review the new and/or changed program(s) consisting of the following:
    - programmer(s) involved.
    - either User Representative(s).
    - Systems Analyst and DP Manager or both.
    - Operations supervisor.

Also, after researching a request, the programmer(s) should list the steps required to complete the assignment and timing sequence of the steps, if applicable.

As a final check off, the User Department should examine the production output for as long as deemed necessary to ensure the integrity of the ouput.

On jobs other than normal production which modifies any master file, you must run a verification program before and after on that master file(s).

On jobs which updates the master files such as (SAVUP1) backups must be taken prior to execution when the program is moved into live production.

Teachers' Investment and Housing Co-operative
Programming Standards

Modification Request Form

All programming request must go through the modification request form but naturally if an existing program crashes and production is held up till the program is fixed then the program must be fixed.

If an existing program is modified then the attached description changes will be made along with an necessary documentation.

"Unit test" means a set of programs have to be fully tested even though only 1 program in the set is changed. Example, report program needs changes then strip must be run then file sorted for report program to be tested.

"System test" will involve whole off-line processing if major update program is modified or if major online program is modified then corresponding offline must be run.

As the modified program becomes tested the user rep will have to sign off that the proper modifications were made and complete system is still in tack.

Once a program is fully debugged and tested and ready to go into production, operations will move the loadable into the live file and also copy the source/copy/loadable into the saved revision backup tape set.

If programs effect systems, then the systems flowchart for operations must be also be changed.

On maintenance, the programmer will use the existing JCL that is in production but one must remember to exectute the job with the "R=T" GBL command. Also make sure JCl will work on live run including memory size.
If too many changes are neccessary then we must create a temporary JCL stream.

On new programs, the programmers must create the initial JCL using the "// PROD" and other necessary PROC elements.
Running the job will mean only keying in RU NEWPGM,,R=T.
Thought must be given for where the new program will execute in the existing JOB STREAM for the daily and monthly operations of the production systems.

Teachers' Investment and Housing Co-operative
Programming Standards

Program Assignment And Program Completion

1. New programs or modifications are assigned only by the Systems Analyst or by the D.P. Manager. If a new program, a unique program number will be assigned.

2. After researching a request, the programmer must submit on the modifiaction request form:

a) an estimated time and completion date.

b) list the steps required to complete the assignment and timing sequence of when the program is moved to production, if applicable.

3) The program numbers will be used in the following ways:

a) for source program ID.

b) to identify the program in the Source Program Library.

c) display on the top right-hand corner of any report, the preceding page number.

4) A program will not be considered complete until:

a) it has been thoroughly unit tested.

b) it has been thoroughly system tested on a current backup of live files.

c) if a changed program, it has been run in parallel with the production program, if necessary (necessity and number of parallel runs to be determined on the situation).

d) written operating procedures completed for operations and approved by operations.

e) program and systems documentation updated.

f) final source listing must be filed.

g) balancing procedures have been fully codumented and explained to operations and/or user rep.

h) a meeting has taken place with the programmer involved, user rep, systems/analyst and D.P. Manager to review the changes to be made.

Teachers' Investment and Housing Co-operative
Programming Standards

Program Assignment And Program Completion (Con't)

4. i) user rep has signed modification request form.

j) operations has signed off modifications request form.

k) programmer immediately posts program(s) to the "Program Library Changes, Additions and Deletions" sheet.

The operations department will move the program(s) to production only after the above steps have been completed.

Teachers' Investment and Housing Co-operative
Programming Standards

MODIFICATIONS REQUEST FORM

DATE:

SYSTEM NAME:

REQUESTED BY:

DATE REQUIRED:

DESCRIPTION AND REASON FOR CHANGE - ATTACH DETAILED DESCRIPTION.

PROGRAMMER

APPROXIAMATELY SCHEDULED START DATE AND ESTIMATED TIME:

SCHEDULE BY D.P. MANAGER AND SYSTEM ANALYST:

OPERATIONS ACCEPTANCE SIGN-OFF:	JCL	OPERATIONS	DATE
ATTACH IF APPLICABLE COPIES OF:		instructions	moved
Job control - new/changes			
Operator instructions			
Log of successful run			
Memory size of programs			
On-line programs			

PROGRAMMING STANDARDS ACCEPTANCE

UNIT TEST	SYSTEMS TEST	PROGRAM DOC.	SYSTEMS DOC.

Teachers' Investment and Housing Co-operative
Programming Standards

Testing Procedures

1. Most testing will be performed by the operations department.

2. Each test shot must be accompanied by a test "spec" sheet.

3. The test spec sheet must include the following information:

a) program number.

b) programmmer's initials.

c) estimated time to run.

d) current date of run.

e) type of test - ie. -program or systems test.
-on-line or off-line test.

f) run date requred by test.

g) charge code - for new development(mortgages, estimated time for compiles on a daily basis.)
- for test, operations to total run logs.

h) disc used for test.

i) tapes used for test.

j) identify read only files (ie use // LBL *SAVFILE)

k) special instructions;
-sufficiently detailed to enable the operator to perform a test requiring operator intervention. The programmer must specify all protected files which may be safely deleted during the test run.

4. Upon completion of a test run, operations will perform the following functions:

a) record appropriate comments.

b) record the actual time.

c) intitial the test spec sheet.

d) return output to the programmer.

5. Testing should be kept as short as possible without reducing the quality of the testing. The operating characteristic should be strongly considered when setting up tests.

Teachers' Investment and Housing Co-operative
Programming Standards

MODULAR PROGRAMMING

A modular approach to Cobol programs is desirable for the following reasons:

1. The problem can be divided into logical self-contained routines.

2. Each routine can be programmed separately.

3. A single "main line" routine controls the overall program logic. This main line provides branches to and from the detailed subroutines by means of a "PERFORM" instruction. The "main line" routine will correspond closely to the program flowchart.

4. Program debugging and program maintenance are facilitated as the programmer can readily isolate the routine in error. Any major logic error in file handling is easier to detect in a compact main line than in non-modular programs.

5. If core size is an important factor, the rouintes can be written as Cobol sub-programs; the OVERLAY features can then be used.

The following flowcharing considerations will aid in designing a modular program:

1. The flowchart main line follows the path of the average, or most straight forward item of data.

2. The downward path of the main line will generally correspond to the "no" or "equal to" branch from a decision function.

3. Detailed subroutines can be charted on the same sheet, or can be referenced and drawn on a separate sheet, dependent upon program size and complexity.

4. The mainline will then provide an overall view of the program's file-handling and main processing logic.

5. The auxiliary detailed "legs" will correspond to the various self-contained routines referred to by the mainline.

Teachers' Investment and Housing Co-operative
Programming Standards

MODULAR PROGRAMMING (Con't)

The following coding considerations will help produce a readable comprehensive Cobol program:

1. Coding will follow the mainline first.

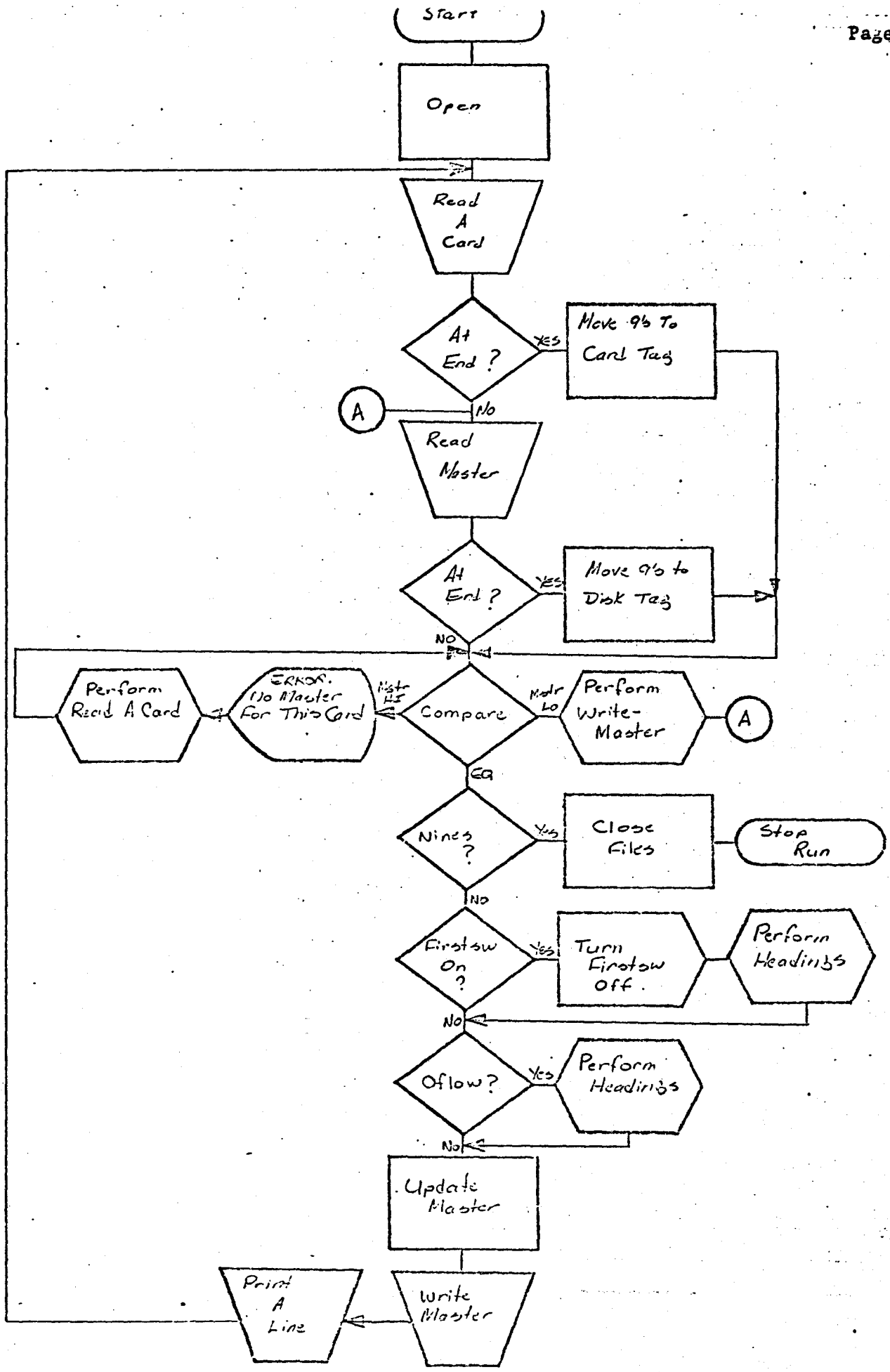
2. Subroutines referenced by the flowchart mainline are designated by the "PERFORM" verb.

3. After completion of the mainline coding, the various subroutines should be coded separately.

4. Procedures names should by as descriptive as possible. Do not be overly concerned with the length of the procedure name.

5. Note statements should be used to give a brief description of routines, referenced by a PERFORM statement, when the purpose is ambiguous.

An example of a simple file update using these concepts follows. The flowchart or draw up main logic and procedure division are shown.


Start
Open
Read A Card
At End ?
YES
Move 9's To Card Tag
A
NO
Read Master
At End ?
YES
Move 9's to Disk Tag
NO
Perform Read A Card
ERROR. No Master For This Card
Mstr LT
Compare
Mstr LO
Perform Write-Master
A
EQ
Nines ?
Yes
Close Files
Stop Run
No
Firstsw On ?
Yes
Turn Firstsw Off.
Perform Headings
No
Oflow ?
Yes
Perform Headings
No
Update Master
Write Master
Print A Line

COBOL SAMPLE PROCEDURE DIVISION:

```
PROCEDURE DIVISION.
    OPEN INPUT READER, MASTER-FILE-IN,
        OUTPUT PRINTER, MASTER-FILE-OUT.
READ A CARD.
    READ READER: AT END,
         MOVE 9999 TO C-TAG,
         GO TO COMPARE.
READ MASTER.
    READ MASTER-FILE IN AT END MOVE 9999 TO D-TAG.
COMPARE.
    IF D-TAG IS LESS THAN C-TAG,
         PERFORM WRITE MASTER,
         GO TO READ-MASTER.
    IF D-TAG IS GREATER THAN C-TAG,
         DISPLAY C-TAG, "NO MASTER FOR THIS CARD" UPON CONSOLE,
         PERFORM READ-A-CARD,
         GO TO COMPARE.
*   NOTE EQUAL COMPARE
    IF D-TAG IS EQUAL TO 9999,
         AND C-TAG IS EQUAL TO 9999,
         GO TO EOJ.
*   NOTE ABOVE BOTH FILES AT-END.
    IF FIRSTSW IS EQUAL TO "0",
         MOVE "1" TO FIRSTSW,
         PERFORM HEADINGS.
    IF OFLOW PERFORM HEADINGS.
    PERFORM UPDATE-MASTER.
*   NOTE UPDATE-MASTER ADDS DETAIL TO CUMULATIVE AND REPLACES
*   CURRENT WITH DETAILS.
WRITE-MASTER.
    MOVE MASTER-IN TO MASTER-OUT.
    WRITE MASTER-OUT.
MOVE-TO-PRINT.
    MOVE D-TAG TO P-TAG.
    MOVE D-NAME TO P-NAME.
    MOVE D-AMT TO P-AMT.
PRINTING.
    WRITE RPT AFTER CNTRL.
    MOVE SPACES TO RPT.
EXIT.
    GO TO READ-A-CARD.
*   END OF MAINLINE - SUBROUTINES TO FOLLOW.
HEADINGS.
    MOVE HEAD1 TO RPT. PERFORM PRINTING. MOVE HEAD2 RPT. PERFORM
    PRINTING.
UPDATE MASTER.
    ADD C-AMT TO D-AMT. MOVE C-AMT TO D CUR-AMT.
EOJ.
    CLOSE READER, MASTER-FILE-IN, PRINTER, MASTER-FILE-OUT.
    STOP RUN.
```

COBOL SAMPLE PROCEDURE DIVISION:(Con't).

```
    PERFORM T3-ROUTINE.


T3-ROUTINE SECTION.
BEG-SECTION.
    PERFORM T2-ROUTINE.
    MOVE T3-TOTAL TO PR-TOTAL.
    MOVE ZERO TO T3-TOTAL.
    PERFORM PRINTING.
T3-EXIT.
    EXIT.
T2-ROUTINE SECTION.
BEG-SECTION.
    PERFORM T1-ROUTINE.
    ADD T2-TOTAL TO T3-TOTAL.
    MOVE ZERO TO T2-TOTAL.
T2-EXIT.
    EXIT.
T1-ROUTINE SECTION.
BEG-SECTION.
    ADD T1-TOTAL TO T2-TOTAL.
    MOVE ZERO TO T1-TOTAL.
T1-EXIT.
    EXIT.
```

Teachers' Investment and Housing Co-operative
Programming Standards

COBOL CODING SHEETS

Certain rules for alignment of COBOL are necessary if easily readable source programs are to be produced. With referenece to the COBOL program sheet, consider the following:

1. DIVISION-NAMES begin in cc 8 and are followed by a period. The remainder of the line shall be left blank.

2. SECTION-NAMES begin in cc 8 and are followed by a period. The remainder of the line shall be left blank.

3. PARAGRAPH-NAMES begin in cc 8 and are followed by a period. For clarity the rest of the line shall be left blank.

4. FD, 01, and 77 level entries will begin in cc 8.

5. Sentences shall begin in cc 12. Continuation of the sentence shall begin in cc 16.

6. Because of possible lengthy data names or a number of ascending level entries, it may not always be feasible to line up the PICTURE, VALUE, or COMPUTATIONAL clauses as indicated. However, to facilitate keypunching and for the most comprehensive coding, these clauses should begin in the indicated column when possible.

Teachers' Investment and Housing Co-operative
Programming Standards

RECOMMENDATIONS FOR CODING COBOL EFFICIENTLY:

1. Convert fields requiring arithmetic to packed or binary format. Univac 90/30 uses binary calculations for all arithmetic even packed decimal fields.

2. Study decimal requirements of established files, then align decimals of related fields.
example.
SENDING-FIELD: 01 FIELDA PIC S999V99 COMP-3.
RECIEVE-FIELD: 01 FIELDB PIC S999V9 COMP-3.
By adding one more decimal place to FIELDB the need for alignment instructions to be generated in the Procedure Division is eliminated. Univac 90/30 will use 3 bytes for both fields.

3. Write literals with the same number of decimal positions as the receiving field.
example;
instead of ADD 1 TO FIELDA.
USE ADD 1.00 TO FIELDA.
Only one byte is added to the literal area but 18 bytes are required for alignment of decimal points are saved.

4. Specify Packed fields with an ODD number of digits.

5. Avoid mixed modes when possible. To avoid multiply conversion, move DISPLAY fields to working storage fields defined as packed or binary.

6. Specify a sign with the picture if apllicable but not mandatory.

7. Declare unsigned Display fields (ie. part number employee number), as alpha numeric when these fields are used in IF statements.

8. Using different size fields in IF statements requires extra core.

9. Use indexing with literals, when possible. Binary indexing results in more efficient coding.

10. Keep statements simple.
example:

```
 instead of  IF PACKAGE IS SMOKING OR TICKING,
                GO TO ACTIONA.
             IF PACKAGE IS READ AND GREEN
                GO TO ACTIONA ELSE GO TO ACTIONB.
         ACTIONA.
             MOVE PACKAGE TO BOMBREPORT, GO TO ACTIONC.
         ACTIONB.
             MOVE PACKAGE TO SAFEREPORT.
   USE       IF PACKAGE = SMOKING OR TICKING
                MOVE PACKAGE TO BOMBREPORT, GO TO ACTIONC
             ESLE MOVE PACKAGE TO SAFEREPORT.
```

Teachers' Investment and Housing Co-operative
Programming Standards

RECOMMENDATIONS FOR CODING COBOL EFFICIENTLY:(Con't)

11. In general, on commercial applications and on the Univac 90/30 it is safer to use numeric fields in packed format for readibilty but if fields are greater than 6 intergers and space is to be considered then binary is more efficient.
example

length PIC	packed BYTES	binary BYTES
S9	1	2
S9(3)	2	2
S9(5)	3	4
S9(6)	4	4
S9(7)	4	4
S9(8)	5	4
S9(9)	5	4
S9(10)	6	8

12. Put many comment lines in throughout the program (ie. cc 7 *)

13. Re-define I/O areas whenever possible.

14. DO redefine the print area in performing print routine.

```
01 PRINT-RECORD.
01 DETAIL-LINE REDEFINES PRINT-RECORD.
   02 NAME-PRT                  PIC X(40).
   02 FILLER                    PIC X(05).
   02 AMT PRT                   PIC ZZ,ZZZ,ZZZ.99-.
   02 FILLER                    PIC X(73).
```

Using this and print copy routine PRINTCTL.

15. Perform the I/O operations whenever possible. Cobol READ, WRITE, ACCEPT AND DISPLAY macros are large.
eg. each WRITE takes 118 bytes.
PERFORM takes 20 bytes.

16. At Teachers' memory core is not a problem for offline programs so the use of ACCEPT is good for data rather than reading a CARD file.

17. Use the same data name in all files for like data. Prefix the data name in each file with a file identifier.
example:

```
 instead of  N-BRANCH          PIC S9(03).
             SDT-BR-NO         PIC S9(03).
    USE      N-BRANCH          PIC S9(03).
             SDT-BRANCH        PIC S9(03).
```

Teachers' Investment and Housing Co-operative
Programming Standards

RECOMMENDATIONS FOR CODING COBOL EFFICIENTLY:(Con't)

18. Use the same data name in all files for like data. Prefix the data name in each file with a file identifier.
example:

```
 instead of  01  SREC.
                 02  N-BRANCH          PIC S9(03).
             01  SDT-REC.
                 02  SDT-BR-NO         PIC S9(03).
    USE      01  SREC.
                 02  N-BRANCH          PIC S9(03).
             01  SDT-REC.
                 02  N-BRANCH          PIC S9(03).
      PROCEDURE DIVISION.
          MOVE CORR SREC TO SDT-REC.
```

If using qualifiers then computations can be done with corresponding clause as shown above.

19. The COMPUTE verb is recommended when a number of arithemtic operations are being accomplished.
example:

```
 instead of  ADD JAN, FEB, MAR GIVING FIRST-QTR-TOTAL.
             DIVIDE FIRST-QTR-TOTAL BY 3 GIVING QTR-AVERAGE.
    USE      COMPUTE QTR-AVERAGE = (JAN + FEB + MAR) / 3.
```

Often the use of arithmetic symbols is more intelligible than the wordiness of several statements.

20. Optionally use figurative constants or literals but be consistent within the program.
example:

```
 instead of   MOVE SPACES TO PRINTLINE.
              MOVE " "    TO PRINTLINE.
              MOVE ZERO   TO COUNTER.
              MOVE ZEROES TO COUNTER1.
              MOVE ZEROS  TO COUNTER2.
              MOVE 0      TO COUNTER3.
   USE        MOVE SPACES TO PRINTLINE.
              MOVE ZERO   TO COUNTER COUNTER1 COUNTER2 COUNTER3.
```

21. Avoid comparison of group items; if necessary move items to be compared into fields defined as elementary items.

22. Avoid constants for numeric numbers although in Cobol dumps the defined constants are easily found.
example:

```
 instead of   77  CONSTANT-1  VALUE 1 PIC S9 COMP-3.
                  ADD CONSTANT-1 TO PRINT-CTR.
    USE           ADD 1 TO PRINT-CTR.
```

CODING STANDARDS

No attempt will be made to set concise standards for creating data names. However, in order to preserve the meaningful nature of the Cobol language, paragragh names, item names, and names for program switches must not be ambiguous; for this reason it is preferable to have a name that is too long rather than one that is no clear. In the following examples, data names are given for some of the common procedures.

example:

```
 instead of   77  C1SW                    PIC S9   COMP-3.
              77  C2SW                    PIC S9   COMP-3.
      PROCEDURE DIVISION.
          MOVE 1 TO C1SW  C2SW.
      A100-BGN.
          PERFORM C1RTN.
          PERFORM C2RTN.
      C1RTN.
          PRINT C1TOT.
      C2RTN.
          PRINT C2TOT.

    USE         77  CONTROL-1-SW          PIC S9  COMP-3.
                77  CONTROL-2-SW          PIC S9  COMP-3.
      PROCEDURE DIVISION.
          MOVE 1 TO CONTROL-1-SW CONTROL-2-SW.
      A100-MAIN-LINE.
          PERFORM CONTROL-LEVEL-1-RTN.
          PERFORM CONTROL-LEVEL-2-RTN.
          STOP RUN.
      CONTROL-LEVEL-1-RTN SECTION.
      BEG-SECTION.
          PRINT CONTROL-1-TOTALS.
      END-SECTION.
          EXIT.
      CONTROL-LEVEL-1-RTN SECTION.
      BEG-SECTION.
          PRINT CONTROL-2-TOTALS.
      END-SECTION.
          EXIT.
```

example 2:
Setting up a table could use descriptive data names.

```
       77  BRANCH-NO              PIC S9(03)   COMP-4.
       01  TOTAL-DOLLARS-BY-BRANCH.
           05  BRANCH-ENTRY    OCCURS 12 TIMES.
               10  BRANCH-AMT   PIC S9(9)V99 COMP-4.
```

Teachers' Investment and Housing Co-operative
Programming Standards

REPORT FORMAT

1. The eight postion date as indicated in the standard report headings is standard output for the Coop users.

2. That date shall be written on the right corner of the report preceding the page number.

3. The program ID number shall be written on the top left corner of all reports.

4. The program run date in the form mm/dd/yy extracted form the systems date shall be written two spaces to the right of the program ID number.

5. The first line will contain also the bank name which is naturally Teachers' Investment and Housing Coop and the report title.

6. The second line will contain the branch number and name.

example:

```
MIS901   MM/DD/YY   BANK NAME          REPORT NAME        PAGE 999
     BRANCH  999 BRANCH NAME
```

Teachers' Investment and Housing Co-operative
Programming Standards

DISC OPTIMIZATION:

Univac 90/30

8418 are sectorized discs which means all I/O's are done in sectos which are multiples of 256.

Single density 8418:

256 bytes per record (sector).
40 sectors per track.
10240 bytes per track.
404 tracks per surface.
7 active surfaces.
71680 bytes per cylinder.
28,958,720 bytes per pack.

Double density 8418:

256 bytes per record (sector).
40 sectors per track.
10240 bytes per track.
808 tracks per surface.
7 active surfaces.
71680 bytes per cylinder.
57,917,440 bytes per pack.

Teachers' Investment and Housing Co-operative
Programming Standards

DISC OPTIMIZATION:(Con't)

The records are read by blocks and since the discs are setorized, all I/O is done by accessing shole sectors (256 bytes). Therfore, block / record sizes should be less than or equally divisable by 256. A block size could be 10 k long but keep the block sizes smaller for the program file memory constraints. If a record needs 250 bytes then make the record 256 bytes long with a 6 byte filler for future expansion in the record. Since the record is evenly divisalbe by 256 then no bytes are wasted. If the block and record was left at 250 bytes every I/O would waste 6 bytes.

Examples of file optimization:

- record is 50 bytes long.

- If you choose:
a) 5 records per block, would waste 6 bytes per block throught the file.

b) 100 record per block, no bytes would be wasted.
50 bytes per record, 25600 bytes per block which is 100 sectors per read or write of a block.

Teachers' Investment and Housing Co-operative

Programming Standards

Programming Documentation

1 Copy Module Documentation.

a) In writing copy modules one must have as the firtst 3 lines.
```
* @W SREC,$LOKO7C,TIHC91 SEQ '000001' BY 1
*SOE@SY C SREC,$LOKO7C,TIHC91 SAVINGS MASTER RECORD LAYOUTS
*  LAST UPDATED BY BRIAN SAMESHIMA ON 1979 JAN 17.
```

b) For all copy modules the last line must be:
```
*  [][][] END OF SREC MODULE [][][]
```

c) All files layouts will be in copy code therefore documentations will be complete as to field by field documentation and file description such as block, record sizes. See example SREC in copy library.

d) All file descriptions must be in copy code. See example, SVFILCPY in copy library.

e) All I/O routines must be in copy code. This applies for programs which accesses the IMS files. Example is reading savings file.

f) If any code will possible be used again then put them in well documented copy code.

2 Program documentation.

a) In writing programs one must have the following 4 lines:
```
*SOE@W CISO01,$LOKO6S,TIHC91 SEQ'000001' BY 1
*SOE@SY C CISO01,$LOKO6S,TIHC91,CISO01 DETAIL BALANCING PRGM
*SOE@SY RU ONR(DET),,N=CISO01,O=C,S=H
*  LAST UPDATED BY BRIAN SAMESHIMA ON 1979 MAY 05.
```

b) For all programs the last line must be:
```
* [][][] END OF CISO01 PROGRAM [][][]
```

c) All special information needed in programs should be put in JCL this will include parameters needed by programs for example special dates which are needed. This will be beneficial for operations documentation.

d) All one shot programs must still be fully documented.

3 Coding Standards.

a) Follow structured coding practices:
Reference - A Simplified Guide To Structured Programming
by Daniel McCracken.

b) Follow the KISS rule (KEEP IT SIMPLE STUPID).

Teachers' Investment and Housing Co-operative

Programming Standards

Programming Documentation (Con't)

c) In modules have only 1 entry and 1 exit point.

d) If procedure portion could be used elsewhere take the time now and make a reusable copy module.

e) Try for table driven logic which will make maintenance more easily done.

f) The following rules must be followed for new programs: 1 All file record layouts must be in copy code. 2 For all programs I/O routines must be in copy code. 3 For online programs, one must make sure the transaction do not take advantage of the Audconf Facility. We must do all editing and checking before any records are written. Automatic rollback will not occur if a transaction cancells in a database. 4 All programs LFD's must match standard LFD's for IMS files. New programs must match LBL to LFD's. The standards are:

```
SELECT PRINT-FILE   ASSIGN TO PRNTR   PRINTER.
SELECT PRINT-FILE-1 ASSIGN TO PRNTR-1 PRINTER.
SELECT RPINT-FILE-2 ASSIGN TO PRNTR-2 PRINTER.
SELECT CARD-FILE    ASSIGN TO RDR     CARD-READER.
DATEFILE LFD IS DATEFL.
DJFILE   LFD IS DJFILE.
DSFILE   LFD IS DSFILE.
NAFILE   LFD IS NAFILE.
RSPKEY   LFD IS RSPKEY.
RSPTRL   LFD IS RSPTRL.
SAVFILE  LFD IS SAVFILE.
SDTFILE  LFD IS SDTFILE.
STFILE   LFD IS STFILE.
BTFILE   LFD IS BTFILE.
SVHSTFLE LFD IS HSTFLE.
SVTRLFLE LFD IS SAVTRL.
SDTEXTXX LFD IS SDTEXTXX.(Where XX is the number)
```

4 JCL standards.

a) As previously mentioned LFD must be consistant.
b) On testing, updates of master files remember to use 'R=T as gobal parameter in keying job control'.
c) Now because of '// PROD' proc loadable must be stated '// EXEC CCS001,LOD'.
d) Standard // DVC' to '// VOL' statements are as follows:

```
// DVC RES    for release pack REL052
// DVC 51  // VOL TIHC02
// DVC 54  // VOL TIHC90
// DVC 53  // VOL TIHC91
// DVC 54  // VOL TICH04
// DVC 55  // VOL TIHC07
// DVC 56  // VOL MICRS5
```

Teachers' Investment and Housing Co-operative
Programming Standards

Systems Documentation

- Systems documentation consists of:

1 User manuals created by user department.
a) operation of user screens.
b) reports for various systems.

2 Operations manuals created by operations.
a) daily, monthly and yearly running of jobs consisting of I/O and special parameters needed.

3 Programmers' manuals created by programmers:
a) cross reference for job-program sequence
program-job sequence
job-run sequence
run-job sequence
source-copy sequence
copy-source sequence.
b) list of programs with comments.
c) list of JCL with comments.
d) Remarks section of programs.
e) file layouts with field documentation.
f) online documentation.
g) systems flowcharts of daily, monthly and yearly JCL.
h) RRSP/THOSP documentation.
i) Investment/terms documentation.
j) Managers documentation consisting of systems overview and drawbacks.

- Programmers documentation is quite automatic. All cross references, list of file/JCL/programs and list of remarks are JCL runs. The rest must be updated in $LOKO8D,TIHC91 library and on the AES diskettes. But if record formats change or systems flowcharts changes they must be changed accordingly.

- All programs must include detail documentation for the remarks section. In the remarks section is where Teachers' computer department will keep all programs documentation including:
1 Input/output files
2 program descriptions
3 Special notes for programming and operations.
example is shown below, note what's in brackets is explanation only.

```
 Remarks.
*CRT500                Teachers'
*Revnote            Transaction report
*
*          Revision number :        8.0
*          Original author :     Glen Matus
*          Input files     : (files, cards and console in)
*             Nafile       - Name/address file
*             Rspkey       - Rspkey file used to point
```

Teachers' Investment and Housing Co-operative
Programming Standards

Program Requirements (Con't)

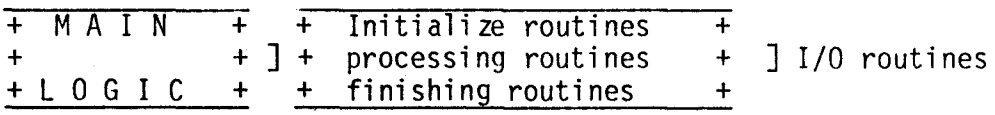
```
*           Beginning date - submitted by operator via console
*           Ending date    - submitted by operator via console
*           Bank no request- submitted by operator via console.
*
*       I/O file: (files only)
*           Savfile        - THOSP master file
*
*       Output file: (files, printer and console)
*           Printer        - THOSP tax receipts.
*           Console message- msgs to inform operator validity
*                            of beginning and ending dates and
*                            request bank.
*
*       Program descriptions: (briefly program main logic.)
*           The program will request which bank to run (1 or 2).
*           Next beginning and ending dates are requested thru
*           the console.  The pgm seeks to the THOSP (class 16)
*           part of the RSPKEY file, reads it sequentially while
*           also randomly reading the SAVFILE, NAFILE and
*           printing the tax receipts for each THOSP account.
*            when a control break is detected in the RSPKEY
*           file, a total page is printed showing the number of
*           records read, unreceipted amt and receipted amt.  If
*           another bank was to have tax receipts then the pgm
*           seeks to the bank in the RSPKEY file and begins
*           processing for that bank.  When the end of the
*           RSPKEY is detected the program will terminate.
*
*       Programming notes: (special information regarding pgm)
*           This program will be subject to changes yearly due
*           to government changes.
*
*       Operational notes: (parameter cards, option thru
*           console and program to program flow.)
*           This program must be run at year-end (Dec 31) or
*           month-end (Feb 28) depending on the government
*           regulations, since it updates Savfile.
*
*       Program enhancements:
*           Module documentation needed.
```

- In one shot programs the following must be followed:
1 The program must be validated by other programmers and user rep for program structure and checking logic. The patch program must display the fields changed giving before and after images for the first 100 records. In testing, run printing of master records before and after patch run. In live run, execute verification of of all monetary fields (totals).
eg) RUN VERSAV RUN PRTTOT RUN PATCH PGM RUN PRTTOT RUN VERSAV.

Teachers' Investment and Housing Co-operative
Programming Standards

Programming Requirements (con't)

- In writing new programs documentation must be completed as the program is being written and/or updated. This means the remarks section and the program main logic and tricky routines must be fully documented. Remember that programmers will have 80 % maintenance work so documenting modules before coding ensures that ample thought must be given to the logic and will lead to good structured programming practices and avoid spagetti programs. ie) main logic will call a processing routine which will call I/O routines.



- In maintenance of programs where documentation was ommittted the time must be taken then to fully document the program. Namely, the remarks section and the progrm main logic and tricky routines.

- All temporary files must be prefixed with TEMP_____.

Teachers' Investment and Housing Co-operative
Programming Standards

Programming Error Messages

Dealing with programmers' error messages:

1 Systems errors such as reading NAFILE, there should be a valid record but isn't so the error message is very important and should give the bad NAFILE key and the Programmer should be given a call.

2 Not to critacal errors such as name too long then error message should reflect just name but processing should continue and programmers won't be contacted unnecessarily.

3 All errors where termination of program occurs must use cancel macro.

- Also in 1st heading line top left hand corner must contain program name.
eg)

```
Savrpt                Teachers' Coop   Date:May 05, 1979  Page  1
Account #     Name                      Balance     Accured int
999999-9/99 Sameshima Brian         999,999.99-  999,999.99999
```

Teachers' Investment and Housing Co-operative
Programming Standards

Programs Compiler Options:

To do Extended Cobol Compiles:
RU OFF,,N=program,O=?,E=?,S=? Offline compiles
RU ONR,,N=program,O=?,E=?,S=? Online reentrant compiles
RU ONS,,N=program,O=?,E=?,S=? Online serial compiles
where N is option: program name with no default but only 6 chars
where O is option: O=N null,S source,C cross,X alphabetic cross,
O object and I source no copy expansion.
defaulting to O=S for source.
where E is option: E=Y yes link,N no link defaulting to Y yes link
where S is option: S=P print immediately,H hold in spool for look
by RSP defaulting to H hold in spool.

To do Assembler Compiles:
RU ASMOFF,,N=program,E=? Assembler compile to object code
RU ASMON,,N=program Assembler compile to online subroutine
RU LNKOFF,,N=program Assembler link edit of object compile.
where N is option: program name with no default but only 6 chars
where E is option: E=Y yes link,N no link defaulting to N no link

To do RPGII Compiles:
RU RPGII,,N=program,O=? RPGII compiles
where N is option: program name with no default but only 6 chars
where O is option: O=N null,S source,C cross defaulting N null.

Teachers' Investment and Housing Co-operative
Programming Standards

Programming Coding standards

Structured Programming

Another feature of structured programming is the use of the state variables. Examples of a state-variables would be:

```
05  WHERE-IN-FILE          PIC 9  COMP-4.
88  AT-FIRST-RECORD        VALUE  1.
88  AT-A-MIDDLE-RECORD     VALUE  2.
88  AT-LAST-RECORD         VALUE  3.

05  RRSP-READ-sw           PIC 9  COMP-4.
88  RRSP-END-OF-FILE       VALUE  1.
88  RRSP-READ-GOOD         VALUE  2.
88  RRSP-READ-BAD          VALUE  3.
```

You can then separate out the logic that sets these state variables from the logic that tests them. Changing the logic of the program becomes much simpler as the setting of the switch is normally done in only one place. You do not have to insert new code all over the place. Reading debug dumps is very easy. By looking at the values of all the state variables, you can easily figure out where you were when the job bombed. By finding a state variable that does not look right, you have a very good idea where to look for the bug.

There are a number of other requirements of structured programming but they all boil down to KEEP IT SIMPLE. Avoid the GO TO except for GO TO LOOP-START, GO TO LOOP-END, GO TO LOOP-EXIT, GO TO END-SECTION and GO TO END-OF-JOB. Use only simple loops -- absolutely no rats' nests where control wanders all over the map. It should flow top to bottom with the exception only of loops (clearly marked), and performs. One of the main reason for the max 60 lines per section rule is so that the beginning and end of a loop will always appear on the same page along with any code that makes you jump out of the loop. If code is "performed" in a loop, and it wishes to terminate the loop, UNDER NO CIRCUMSTANCES MUST IT JUMP OUT BY ITSELF. IT MUST SET A SWITCH THAT IS TESTED BY THE CALLING MODULE THAT DOES THE JUMPING OUT OF THE LOOP. If you follow this rule, you will never be surprised by program flow. On a module by module basis, its all there. You do not need to look at the lower levels.

Teachers' Investment and Housing Co-operative
Programming Standards

Programming Coding standards

Structured Programming (Con't)

Break your code in COBOL sections -- no more than 60 lines of code per section. The mainline logic should be a simle set of performs -- less than 1 page. This makes the overall flow dramatically clear. Detailed level flow is usually self evident in COBOL programs; it is the overall flow that gives trouble.

At the front of each section should be a few lines of comments telling what the module does. Try to make each section as self-contained as possible. In other words, if you were to change the code in a module, it should have no side effects on other modules. If there are side effects, document them!!! This will also make it easier to replace modules with more efficent versions as time permits.

Whenever you write code, turn it into a copy module if there is even the remotest chance that it could be used somewhere else. This will encourage documenting the method of use of the module. It will discourage the use of non-general purpose code.

Debug your code on a module (section) by module basis. By writing special test driver code, you can thoroughly test your modules. There is absolutely no way to exhaustively test a complete system. As our systems get larger the probabilty of them being bug free gets very remote. We must be more and more careful as our system gets more and more complex.

"CODING is 10 % of the work; MAINTENANCE is 90%"

WE must code with following priorities in mind:
1 Ease of maintenance.
2 Reliability.
3 Speed.

To this end we must use:
1 Structured programming.
2 Copy code for all I/O and common functions.
3 Documentation embedded in the program code and JCL.
4 Table-driven logic.

Programming Coding standards

Table Driven Logic

Table driven code is an absolute must for programs that behave differently for different transaction codes. Table driven code is 100 % easier to debug. It is much less likely to have bugs in the first place. It is much less likely to develop bugs when changes (such as new tran codes or changed processing on some tran code) are made. New tran codes can be incorporated in minutes -- without any understanding whatsoever of the COBOL code.

For a simple example lets say that we have tran codes 1 thru 4. Tran codes 1 and 3 add to the balance, tran code 2 has no effect and tran code 4 subtracts from the balance. Tran codes 1 and 2 require a password whereas tran code 3 and 4 do not.

We describe this with a table in working storage. It can be read off cards, or it can be built directly with value clauses. Table entries look like:

```
       E   P
001    P   Y
002    Z   Y
003    P   N
004    N   N
E - P=positive effect on balance
    N=negative effect on balance
    Z=zero effect on balance.
P - Y=yes password required
    N=no password required.
```

Procedure Division.
Logic will be like:

```
IF EFFECT = 'P' ADD AMT TO BALANCE
ELSE IF EFFECT = 'M' SUBTRACT AMT FROM BALANCE
ELSE IF EFFECT = 'Z' NEXT SENTENCE
ELSE GO TO BUG-IN-TABLE.

IF PASSWORD-REQUIRED = 'Y'
    IF PASSWORD-GIVEN NEXT SENTENCE
    ELSE MOVE 'NEED A PASSWORD' TO ERROR-MSG GO TO BAD
ELSE IF PASSWORD-REQUIRED = 'N' NEXT SENTENCE
ELSE GO TO BUG-IN-TABLE.
```

You can also see at a glance exactly what processing has been chosen for each tran code. This means no-EDP people can look at your code and help you debug without exhaustive systems tests.

Teachers' Investment and Housing Co-operative
Programming Standards

Programming Considerations

For any program that does massive printing the following standards should be observed so that it will be easy to farm out the printing to a service bureau should the need arise. The program should be written in COBOL. It should access only a single sequential file of input data. Namely it should not read any of the on-line data files. If any branch addresses etc are needed they should be read through the job control stream. Any dates required should come through control cards and not through the "SYSDATE" feature. Avoid the use of "COMP-3" fields in the input file. This will allow the print program to be easily modified to run on a foreign machine reading a single input tape.

Any new design work should be walk-through tested after the screens are designed, but before any code is written. This will test some editing and should catch "OH, COULD YOU ADD THIS FIELD".

Also all screens and reports must be signed off and approved by management and the users then this will catch "OH, WHAT ABOUT THIS FIELD".

Little thought has been given to RE-RUNS. Inordinate use has been made of update-in-place. At present if a single run bombs because of power-surges, disc problems etc, the whole off-line system must be re-run from scratch. Proper consideration for operational restart procedures must be given in order to rerun or restart jobs. We should attempt to use sequential-copy-type updates from pack to different pack as much as possible. If a job bombs all we have to do rerun that job, not the whole off-line process. At present with our disc drive shortage this is impossible but this must be considered in the future if disc space becomes available.

A definite plus is sequential verses random processing. Read the savings file sequentially whenever possible -- bypassing the use of the Rspkey file as an example. Use "reserve 1 alternate area" and "apply cylinder-index area of xxx indices" speeds up ISAM sequential reads.

All screens should have separate CDA's. Fields with similar names can be accidentally used. The problems may not show up for years because usually the two similiar fields are overlayed on each other. An example is MMCDA2 which will be changed when time permits. Here "AB" and "NS" and "MM" screens all use "MMCDA2".

It may help to recompile the world every once in a while to ensure no source modules have been destroyed. Better find out now rather than after backup has been deleted. This will also prevent incompatibilities between modules compiled under different releases. Already we have encountered problems "DTECK2" source has been lost! Hopefully if we recompile the world on a release change then we should have the new improved advantages of the COBOL compiler.

Teachers' Investment and Housing Co-operative
Programming Standards

OPERATING PROCEDURE REQUIREMENTS:

1. All jobs must have an operating procedure. Until a written copy of the operating procedure has been approved, the run shall be the responsibility of the programmer.

2. Each page of the procedure write-up shall show the following information:
a) The area and the application.

b) The frequency of processing.

c) Programs - show the program number and the program names in order of execution. Give each program, including utilities, an item number. Start each page with item number 1. The item number will be used as a cross reference in the following sections.

d) Forms - Define by item number the forms used with each program.

e) Cards - Define by item number the cards used with the program.

f) Discs - Define by item number the discs used wiht each program.

g) Process - Define by item number and in condensed form the function of each program.

h) Notes - Indicate by item number any information pertaining to the program.

i) The maximum record capacity of a file shall be shown in this section wheneveer an initial sort, merge or reorganize takes place.

3. Related programs comprising one Job step should be included together on one page. The operating procedures for each program must be complete on one page.

4. A written copy must be made of all operating procedures. The copy is included in the appropriate manual of operating procedures (run book) kept in the operating area.

Teachers' Investment and Housing Co-operative
Programming Standards

BALANCING PROCEDURE:

1. All jobs shall have a detailed balancing procedure if the control clerk will have any contact with the job.

2. The balancing procedures shall have attached all reports the control clerk will come in contact with while balancing or auditing the job.

3 The balancing procedures shall show the following:
a) The name of the report to which the procedure pertains.

b) Detailed balancing instructions.

c) The frequency of the report.

d) The distribution of the report.

e) Any special instructions.

Teachers' Investment and Housing Co-operative
Programming Standards

Programming Number Composition

JCL for offline will match program names except extracts will remain the same as daily and monthly runs. Sorts, data utilities and librarian JCL will have S, D & L respectively.
eg) RU JCRR505 Coop RRSP statement strip
RU JCRR505S Sort
RU JCRR506 Coop RRSP statement print.

Program standard for names are:

Bank	System	Sub-System	Program number
C-Coop	S-Savings	I-Investent	000-499 online
T-Trust		T-Term	500-999 offline
		U-Special	
M-System	C-CIF	F-CIF	
	R-RRSP	R-RRSP	500-699 Daily
		T-THOSP	700-799 Monthly
	M-Mortgage	M-Member	800-899 Yearly
		C-Commercial	900-999 Demand
	A-Account	G-General L	
	I-System	S-System	

Print programs savings	prefix	CSS900.
Savings (Inv/Terms)	prefix	CSS901.
Name/address	prefix	CCF500.
Investment	prefix	CSI900.
Terms	prefix	CST900.
Special	prefix	CSU900.
RRSP	prefix	CRR900.
THOSP	prefix	CRT900.
System all banks	prefix	MIS900.
System Coop	prefix	CIS900.
System Coop RRSP	prefix	CRS900.
Member mortgages	prefix	CMM900.
Commercial mortgage	prefix	CMC900.
Accounting	prefix	CAG900.

Teachers' Investment and Housing Co-operative
Programming Standards

LOGBOOKS:

Program library log:

1. The "Program Library Changes and Additions" sheet shall be coded immediately for any one of the following reasons:

a) Before revising a program.

b) Upon establishing a new program number.

c) To signify an existing program is obsolete.

2. The "Program Library Changes and Additions" sheets are the input to the "Program Library Logbook" which contains a listing of all programs showing the date the program was originally written, the author's initials, the date of the last revision, the number of revisions, and the last revising programmer's initials.

CROWN ZELLERBACH CANADA LIMI

DATA CENTRE • PROGRAM LIBRARY CHANGES & ADDITIONS

CHARGE CODE

PROGRAM NUMBER	FUNC-TION	PGM CODE	INDEX CODE	AUTHOR	DATE		REV. PGM'R.	PROGRAM DESCRIPTION
					MO.	YR.		

~~PROGRAMMER CODE: 1 = 1401~~
~~2 = 1410~~

FUNCTION CODE: 1 = REVISION
2 = ADDITION
3 = OBSOLETE

Teachers' Investment and Housing Co-operative
Programming Standards

Programs Library

Volume #	Abbr	Library	Pack Job	Contains
TIHC91	ON	$LOKO1ONLOD	PKO1ON	Production online load mod.
RES	OFF	$LOKO2OFFLOD	PKO2OFF	Production offline load mod
TIHC91	TON	$LOKO3TONLOD	PKO3TON	Test online load modules
TIHC91	TOFF	$LOKO4TOFFLOD	PKO4TOFF	Test offline load modules.
TIHC91	I	$LOKO5I	PKO5I	IMS source online.
TIHC91	S	$LOKO6S	PKO6S	Offline source.
TIHC91	C	$LOKO7C	PKO7C	Copy code.
TIHC91	D	$LOKO8D	PKO8D	Documentation library.
TIHC91	P	$LOK1OP	PK1OP	Procs
TIHC91	B	$LOK11B	PK11B	Batch IMS.
TIHC91	TB	$LOK13B	PK13B	Test batch IMS.
RES	J	YJCS	PK1	Job control.
		RES/SYSTEM	PACKRES	All system libraries.
		TIHC91/USER	PKWORLD1	All user libraries on TIHC91
		RES/USER	PKWORLD2	All user libraries on RES.
		EVERYTHING	PKWORLD	Packs everything.

In addition any library can be packed with:
RU PK1,,V=XXXXX,F=XXXXX
where V is Volume defaulting to RES
and F is file nmae defaulting to YJCS