**1. CODING AND DOCUMENTATION STANDARDS**

**1.1 Coding Guidelines**

**1.2 COBOL Coding and Documentation Standards**

**1.3 Example COBOL Program with Documentation**

**1.4 Assembler Coding and Documentation Standards**

**1.5 Example Assembler Program with Documentation**

**1.6 JCL Coding and Documentation Standards**

**1.7 Example JCL with Documentation**

# 1.1 Coding Guidelines

**The following is a list of strict coding guidelines for CSCI 465/680-J9. Follow them very carefully. Anyone found not following one or more of these guidelines will earn a score of zero on the current assignment. Repeat offenders will earn one letter grade lower than that determined by their scores at the end of the semester.**

* Never print blank lines. Use carriage control in both COBOL and Assembler.
* Unless otherwise noted, all report output must be spread across the entire 132 columns (bytes) available following the carriage control byte.
* Register 1 must be preset prior to EDMK.
* Do not use internal subroutines in Assembler.
* Set a return code of 0 before exiting all programs.
* Do not print first report headers prior to a read loop.
* Increment a counter at the top of the loop.
* All COBOL WRITE statements must use the AFTER 1, AFTER 2, AFTER 3 or AFTER PAGE carriage control phrase.
* All read loops must consist of a priming read prior to the top of the loop and a repeating read at the bottom of the loop.
* No instructions will be placed between a READ (COBOL) or GET (QSAM in Assembler) and the check for end-of-file.
* All packed decimal fields must be defined using a DC and be initialized to 0 (unless some other value is necessary).
* Assembler field lengths – especially in packed decimal instructions – must always be specified.

# 1. 2 COBOL Coding and Documentation Standards

The following standards will be used in CSCI 465; however, some will not be relevant until later in the course.

Flowcharts will not be required, but your instructor or your teaching assistant may suggest that you construct one to help with any programming problems you may encounter. Whenever you seek the assistance of an instructor or any teaching assistant for this course, you should bring your assignment copy and any manuals for the course that you may need to reference.

This course is designed for you to learn how to independently make use of manuals. Be sure to purchase all of the required manuals for this course because you will need them.

As usual, unstructured coding will not be acceptable. 'GO TO' statements are not allowed unless your instructor explicitly grants permission. Avoid using 'NEXT SENTENCE' in any IF statements.

**PROGRAM LISTING**

Each COBOL DIVISION should begin on a new page; however, it is okay to put the ENVIRONMENT DIVISION on the same page with the IDENTIFICATION DIVISION.

Do not use commas or semicolons in your code; they may be mistaken for periods in your code and cause confusion.

The program must be designed and coded in a modular fashion so that it is easy to maintain and modify.

**IDENTIFICATION DIVISION**

The AUTHOR statement is required.

A block of neatly formatted, easily readable documentation must be included and must provide the following information:

1. FUNCTION: State the function of the program in one or two sentences. Focus on WHAT the program does, not HOW it does it.
2. INPUT: List the names of any files used as input to the program. Include the DD name, file type, and logical record description associated with the file.
3. OUTPUT: List the names of any files or printed reports resulting from the program. Include the DD name, file type, and output description of each one separately.
4. ENTRY/EXIT CONDITIONS: List any parameters passed into and/or out of the program. State the name of the parameter as declared in the program and for what the parameter is used. Consider the following types of parameters:

INPUT PARM: A value is received but is not altered.

OUTPUT PARM: Has no initial value, but is expected to return a value.

VARIABLE PARM: A value is received, possibly altered, and then returned.

List any return codes set by the program and what each value indicates. State in what field the return codes are passed.

1. NOTES: Include any additional information that will be helpful (external routines or programs referenced, usage of linkage section, program limitations, complex formulas, etc.).

**ENVIRONMENT DIVISION**

The ENVIRONMENT DIVISION has no required documentation. It may be on the same page as the IDENTIFICATION DIVISION if both will fit.

**DATA DIVISION**

Organize common data items together under an 01-level group heading, e.g., 'ACCUMULATORS', 'FLAGS', 'PRINTER-CONTROLS', etc. Check with your teacher about 88-level conventions.

Each data item associated with a record description must have a unique prefix which identifies to which record it belongs, e.g., 'CUST-NAME', 'CUST-ADDRESS', and 'CUST-PHONE' may be data items in 'CUSTOMER-RECORD'. Use descriptive, self-documenting names for all data items. Use additional comments to supplement inexplicit data names. For example, 'EOF-FLAG' is self-defining if there is one input file; however, ERROR-FLAG is not self-defining if there is more than one possible error in the program. When in doubt, provide a brief explanation.

Number group items and their subordinates consistently, and use 2-digit values (e.g. 01, 05, 10).

Arrange attributes in uniform columns. For example, PICTURE clauses might all start in column 45 and VALUE clauses in column 60.

File descriptions must include the RECORDING MODE clause.

**PROCEDURE DIVISION**

All paragraph names should consist of a sequential number, a verb and its subject (e.g., '100-PRINT-HEADINGS', '200-READ-CUST-RECORD', '300-PROCESS-CATALOG-ORDER'). Each paragraph should be followed by an exit paragraph, which is named the same sequential number plus the word 'EXIT'. The only verb in this paragraph should be the verb 'EXIT'. For example, the exit paragraph for '100-PRINT-HEADINGS' should be '100-EXIT. EXIT.'. Note the exception here to the rule that there should only be one statement or paragraph name per line.

Some paragraphs names need not include the sequential number. These are the short paragraphs that must be placed in a procedure of their own because of syntax problems associated with nested conditionally imperative statements. For example, the READ...AT END statement may be one of several imperative statements within an IF statement. It is not clear which statements are associated with the IF condition and which statements are associated with the AT END condition. The READ...AT END statement must be performed in a separate procedure to avoid including the AT END imperatives with the IF imperatives. This separate procedure should follow the paragraph that references it.

Each numbered procedure must start on a new page in a printed listing, with the exception of short procedures (one to four lines of code) that may be printed two (maximum) per page. Each procedure must be preceded by a section of documentation including a statement of the FUNCTION of the procedure. The statement of FUNCTION should include any programming notes.

The IF statement and its corresponding ELSE must begin in the same column. Statements subordinate to an IF or ELSE must be indented.

On a CALL statement, the parameters must be lined up in the same column, one parameter per line. For example,

CALL SUBRTN USING PARM-1

PARM-2

PARM-3.

The examples on the following pages illustrate the points discussed above.

**COBOL IDENTIFICATION DIVISION Documentation Example**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* FUNCTION: THIS PROGRAM PRODUCES CUSTOMER CHECKING \*

\* ACCOUNT STATEMENTS. \*

\* \*

\* INPUT: CUSTOMER-FILE -- \*

\* A SEQUENTIAL FILE CONTAINING CUSTOMER \*

\* ACCOUNT AND TRANSACTION INFORMATION. \*

\* \*

\* OUTPUT: STATEMENT-FILE -- \*

\* CHECKING ACCOUNT STATEMENTS ARE PRODUCED \*

\* CONTAINING ITEMIZED TRANSACTION DATA FOR \*

\* EACH CUSTOMER. \*

\* \*

\* ENTRY CONDITIONS: \*

\* MONTH-LS -- INDICATES THE MONTH FOR \*

\* WHICH THE STATEMENTS ARE TO BE PRODUCED. \*

\* \*

\* EXIT CONDITIONS: NONE. \*

\* \*

\* NOTES: THE PARAMETER, MNTHPARM, IS PASSED INTO \*

\* THE PROGRAM VIA THE 'EXEC' STATEMENT. \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**COBOL DATA DIVISION Documentation Example**

WORKING-STORAGE SECTION.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* FIELD: DESCRIPTION: \*

\* \*

\* LINE-SUB SUBSCRIPTS USED TO ACCESS A TWO- \*

\* COLUMN-SUB DIMENSIONAL TABLE FOR BUILDING \*

\* TRANSACTION DETAIL LINES \*

\* \*

\* WORK-TRANS-DATE USED IN A MOVE STATEMENT TO UNPACK \*

\* TRANSACTION DATE \*

\* \*

\* TRANS-DATE-BREAK-DOWN USED TO ACCESS THE INDIVIDUAL \*

\* FIELDS OF THE TRANSACTION DATE \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

01 EOF-FLAG PIC X VALUE 'N'.

01 SUBSCRIPTS.

05 LINE-SUB PIC S9(4) BINARY SYNC.

05 COLUMN-SUB PIC S9(4) BINARY SYNC.

01 ACCUMULATORS.

05 TOTAL-WITHDRAWN PIC S9(9)V99 PACKED-DECIMAL

VALUE 0.

05 TOTAL-DEPOSITED PIC S9(9)V99 PACKED-DECIMAL

VALUE 0.

05 TOTAL-SERVICE-CHARGE PIC S9(5)V99 PACKED-DECIMAL

VALUE 0.

05 TOTAL-INTEREST-EARNED PIC S9(5)V99 PACKED-DECIMAL

VALUE 0.

05 NEW-ACCOUNT-BALANCE PIC S9(9)V99 PACKED-DECIMAL

VALUE 0.

05 TOTAL-CUSTOMERS-PROCESSED PIC 9(7) PACKED-DECIMAL.

01 WORK-FIELDS.

05 CONVERTED-TRANS-DATE PIC 9(6).

05 TRANS-DATE-BREAK-DOWN.

10 TRANS-MONTH PIC 99.

10 TRANS-DAY PIC 99.

10 TRANS-YEAR PIC 99.

**COBOL PROCEDURE DIVISION Documentation Examples**

1. Main routine documentation:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* THIS ROUTINE CONTROLS THE FLOW OF LOGIC TO PROCESS \*

\* THE CUSTOMER FILE AND PRINT THE CHECKING ACCOUNT \*

\* STATEMENTS. \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

2. Several subroutine documentation examples:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* THIS ROUTINE FORMATS AND PRINTS A DETAIL LINE OF \*

\* TRANSACTION DATA TO APPEAR WITHIN THE CHECKING \*

\* ACCOUNT STATEMENT OF ONE CUSTOMER. A COUNT OF ONE IS \*

\* ADDED TO A LINE COUNTER. \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* THIS ROUTINE CONTROLS THE LOGIC OF ACCUMULATING AND \*

\* PRINTING CUSTOMER CHECKING ACCOUNT STATEMENT \*

\* INFORMATION. \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* THIS ROUTINE PROCESSES A TRANSACTION AGAINST THE \*

\* ACCOUNT BALANCE. \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**COBOL VSAM DOCUMENTATION**

1. Follow the COBOL documentation standards above.
2. When referring to a file, be sure to describe what type of file it is. For example: VSAM KSDS, VSAM ESDS, or VSAM RRDS.

The following is an example of IDENTIFICATION DIVISION documentation for a COBOL VSAM program:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* PROGRAM: READDB \*

\* \*

\* FUNCTION: TO LOAD A FILE INTO A VSAM KSDS CLUSTER. \*

\* \*

\* INPUT: FILE ‑ FILE CONTAINING EMPLOYEE INFO. \*

\* \*

\* OUTPUT: KSDSFILE – EMPTY VSAM KSDS FILE. \*

\* \*

\* ENTRY/EXIT CONDITIONS: NONE. \*

\* \*

\* NOTES: NONE. \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# 1.3 Example COBOL Program with Documentation

Starting on the next page is an example of a complete COBOL program with documentation.

IDENTIFICATION DIVISION.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* PROGRAM NAME: GPA \*

\* \*

\* AUTHOR: TED PROGRAMMER \*

\* \*

\* FUNCTION: THIS PROGRAM COUNTS THE UPPER- \*

\* DIVISION STUDENTS WITH GPA >= 3.0 \*

\* AND THE LOWER-DIVISION STUDENTS \*

\* WITH GPA >= 3.0. \*

\* \*

\* INPUT: A DISK FILE CONTAINING ONE RECORD \*

\* PER STUDENT. EACH RECORD CONTAINS \*

\* THE STUDENT-ID, STUDENT-NAME, \*

\* CREDIT-HOURS COMPLETED, AND HONOR- \*

\* POINTS EARNED. \*

\* \*

\* OUTPUT: NO DETAIL LINES ARE PRODUCED. THE \*

\* ONLY THING PRINTED IS A SUMMARY \*

\* LINE WITH THREE TOTALS -- UPPER- \*

\* DIVISION STUDENTS WITH GPA >= 3.0, \*

\* LOWER-DIVISION STUDENTS WITH GPA >= \*

\* 3.0, AND THE GRAND TOTAL OF ALL \*

\* STUDENTS WITH GPA >= 3.0. \*

\* \*

\* NOTES: NONE \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

PROGRAM-ID. GPA.

ENVIRONMENT DIVISION.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT STUDENT-FILE ASSIGN TO STUDENT.

SELECT PRINT-FILE ASSIGN TO PRINTER.

DATA DIVISION.

FILE SECTION.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* STUDENT-FILE IS A DISK FILE USED AS INPUT. IT \*

\* CONTAINS AN UNKNOWN NUMBER OF RECORDS. EACH RECORD \*

\* CONTAINS THE STUDENT-ID, STUDENT-NAME, CREDIT HOURS \*

\* COMPLETED, AND HONOR POINTS EARNED. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

FD STUDENT-FILE

RECORDING MODE IS F.

01 STUDENT-RECORD.

05 STU-ID PIC X(11).

05 STU-NAME PIC X(20).

05 STU-CREDIT-HOURS PIC S9(3).

05 STU-HONOR-POINTS PIC S9(4).

05 FILLER PIC X(42).

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* PRINT-FILE CONTAINS A GENERIC PRINT LINE THAT IS \*

\* USED TO PRINT A HEADER LINE THAT LABELS THE TOTALS \*

\* AND THE SUMMARY LINE THAT CONTAINS THE TOTALS. \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

FD PRINT-FILE

RECORDING MODE IS F.

01 PRINT-LINE PIC X(132).

WORKING-STORAGE SECTION.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* VARIABLES: \*

\* \*

\* EOF-FLAG CHANGED TO 'Y' WHEN END OF FILE \*

\* OCCURS. \*

\* \*

\* UPPER-TOT USED TO COUNT THE UPPER-DIVISION \*

\* STUDENTS WITH HIGH GPA. \*

\* LOWER-TOT USED TO COUNT THE LOWER-DIVISION \*

\* STUDENTS WITH HIGH GPA. \*

\* GRAND-TOT USED TO CALCULATE THE TOTAL OF ALL \*

\* STUDENTS WITH HIGH GPA. \*

\* STUDENT-GPA CALCULATED BY THE PROGRAM (HONOR- \*

\* POINTS / CREDIT-HOURS) \*

\* \*

\* HEADER-LINE USED TO PRINT LABELS ABOVE THE \*

\* TOTALS. \*

\* SUMMARY-LINE USED TO PRINT THE TOTALS. \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

01 EOF-FLAG PIC X VALUE 'N'.

01 TOTALS.

05 UPPER-TOT PIC S9(4) PACKED-DECIMAL VALUE 0.

05 LOWER-TOT PIC S9(4) PACKED-DECIMAL VALUE 0.

05 GRAND-TOT PIC S9(5) PACKED-DECIMAL VALUE 0.

01 STUDENT-GPA PIC 9V9.

01 HEADER-LINE.

05 FILLER PIC X(34) VALUE SPACES.

05 FILLER PIC X(14)

VALUE 'UPPER DIVISION'.

05 FILLER PIC X(10) VALUE SPACES.

05 FILLER PIC X(14)

VALUE 'LOWER DIVISION'.

05 FILLER PIC X(12) VALUE SPACES.

05 FILLER PIC X(12)

VALUE 'ALL STUDENTS'.

05 FILLER PIC X(36) VALUE SPACES.

01 SUMMARY-LINE.

05 FILLER PIC X(39) VALUE SPACES.

05 SUM-UPPER PIC ZZZ9.

05 FILLER PIC X(20) VALUE SPACES.

05 SUM-LOWER PIC ZZZ9.

05 FILLER PIC X(20) VALUE SPACES.

05 SUM-GRAND PIC ZZZZ9.

05 FILLER PIC X(40) VALUE SPACES.

PROCEDURE DIVISION.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 0000-MAIN. THIS ROUTINE CONTROLS THE FLOW OF THE \*

\* PROGRAM. IT CALLS ROUTINES TO (1) READ THE FIRST \*

\* RECORD, (2) INCREMENT THE UPPER-DIVISION AND LOWER- \*

\* DIVISION TOTALS, AND (3) PRINT THE SUMMARY. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

0000-MAIN.

OPEN INPUT STUDENT-FILE

OUTPUT PRINT-FILE.

PERFORM 0100-READ.

PERFORM 0200-PROCESS-STUDENT UNTIL EOF-FLAG = 'Y'.

PERFORM 0300-PRINT.

CLOSE STUDENT-FILE

PRINT-FILE.

GOBACK.

0000-EXIT. EXIT.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 0100-READ. THIS ROUTINE READS A RECORD FROM THE \*

\* INPUT FILE. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

0100-READ.

READ STUDENT-FILE

AT END MOVE 'Y' TO EOF-FLAG

END-READ.

0100-EXIT. EXIT.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 0200-PROCESS-STUDENT. THIS ROUTINE IS USED TO \*

\* PROCESS EACH STUDENT RECORD. THE STUDENT'S \*

\* GPA IS CALCULATED. IF IT IS >= TO 3.0, THE \*

\* APPROPRIATE COUNT IN INCREMENTED. THEN, THE NEXT \*

\* RECORD IS READ. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

0200-PROCESS-STUDENT.

COMPUTE STUDENT-GPA ROUNDED =

STU-HONOR-POINTS / STU-CREDIT-HOURS.

IF STUDENT-GPA >= 3.0

IF STU-CREDIT-HOURS < 60

ADD 1 TO LOWER-TOT

ELSE

ADD 1 TO UPPER-TOT

END-IF

END-IF.

PERFORM 100-READ.

0200-EXIT. EXIT.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 0300-PRINT. THIS ROUTINE CALCULATES THE GRAND TOTAL, THEN \*

\* BUILDS AND PRINTS THE SUMMARY LINE. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

0300-PRINT.

ADD UPPER-TOT LOWER-TOT GIVING GRAND-TOT.

MOVE UPPER-TOT TO SUM-UPPER.

MOVE LOWER-TOT TO SUM-LOWER.

MOVE GRAND-TOT TO SUM-GRAND.

WRITE PRINT-LINE FROM SUMMARY-LINE AFTER 2.

0300-EXIT. EXIT.

# 1.4 Assembler Coding and Documentation Standards

The following documentation standards apply to all Assembler programs.

1. Columns

1. Labels and asterisks always begin in column 1.
2. CSECT and END statements, instruction mnemonics, LTORG, USING and DROP statements, macro names, and DS and DC for storage declarations all begin in column 10.
3. All operands, macro parameters, storage designators or, if present, multipliers for storage designators, and the name of the program on the END statement all begin in column 16.

2. White Space

* 1. Separate blocks or lines of code adding "white space" to make it easier to read. Do this by putting strategically-placed asterisks (\*) in column 1.
  2. For example, if you are going to code the following closely-related instructions:

LA 1,OUTAMT+11

MVC OUTAMT(13),=X'40206B2020206B2021204B2020'

EDMK OUTAMT(13),PAMT

BCTR 1,0

MVI 0(1),C'$'

Put an asterisk on the line before the LA and an asterisk on the line following the MVI.

3. Line Documentation

* 1. At least 90-95 percent of the instructions and storage statements should be documented.
  2. Line documentation may begin at least one space after the information found beginning in column 16. Wherever possible, try to begin documentation in the same columns on each line.
  3. Line documentation may extend up to and including column 71 but it is best to not extend past columns 65-70.
  4. Line documentation should be clear and concise and, if symbols are used (such as R2 -> TABLE), the meaning should still be very clear.
  5. Sections of code can also be documented using asterisks (\*) in column 1.
  6. Use line documentation in storage areas as well.

4. Documentation Box, or "Doc Box"

A block of comments outlined with a box of asterisks is often referred to as a 'DOC BOX', should

precede each CSECT and Should contain the following:

* 1. One sentence stating the function of the routine.
  2. Descriptions of the input and output including DDNAMES.
  3. Entry and exit conditions (parameters, return codes, etc.)
  4. Notes concerning any inherent limitations, tricky code, etc. If there are no notes, state 'NOTES: None.' in the DOC BOX.

5. NO Register Equates

* 1. Although they are sometimes used in professional Assembler programming environments, register equates are not recommended and not allowed in CSCI 465.

1. Register equates "muddy" up code and can cause other problems, especially in understanding how instructions are typed and subsequently encoded by the Assembler.

6. Standard Entry and Exit Linkage

* 1. ALL Assembler programs are to begin with standard entry linkage.
  2. ALL Assembler programs are to end with standard exit linkage.
  3. Although all Assembler programmers must understand what each instruction in standard entry and exit linkage does, the IBM macros XSAVE and XRETURN are allowed in place of the coded standard entry and exit linkage.

# 1.5 Example Assembler Program with Documentation

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \* \*

\* FUNCTION: THIS ROUTINE LISTS THE NAMES OF THOSE STUDENTS \*

\* WHO ARE BOTH COMPUTER SCIENCE AND MATHEMATICS \*

\* MAJORS. \* \* \*

\* INPUT: NONE. \* \* \*

\* OUTPUT: A PRINTED REPORT. \* \* \*

\* ENTRY CONDITIONS: R1 HOLDS THE ADDRESS OF A PARAMETER LIST. \* \* \*

\* 0(1) - ADDRESS OF TABLE 1 ( COMP SCI MAJORS ) \*

\* 4(1) - ENDING ADDRESS OF TABLE 1 \*

\* 8(1) - ADDRESS OF TABLE 2 ( MATH MAJORS ) \*

\* 10(1) - ENDING ADDRESS OF TABLE 2 \*

\* 12(1) - TOTAL NUMBER OF COMP SCI - MATH MAJORS \*

\* (TO BE ACCUMULATED) \* \* \*

\* EXIT CONDITIONS: R1 RETURNS THE ADDRESS OF SAME PARAMETER \*

\* LIST WITH 12(1) UPDATED WITH TOTAL NUMBER \*

\* OF COMP SCI - MATH MAJORS. \*

\* \*

\* NOTES: None \* \* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

EXAMPLE CSECT

PRINT NOGEN DO NOT EXPAND MACROS

STM 14,12,12(13) SAVE REGS IN CALLER'S SAVE AREA

LR 12,15 COPY CSECT ADDRESS INTO R12

USING EXAMPLE,12 ESTABLISH R12 AS THE BASE REG

\* WITH THIS CSECT AS THE BASE

\* ADDRESS

LA 14,SAVEAREA R14 POINTS TO THIS CSECT's SAVE

\* AREA

ST 14,8(,13) STORE ADDRESS OF THIS CSECT's

\* SAVE AREA (FORWARD LINKAGE)

\* IN CALLER'S SAVE AREA

ST 13,4(,14) STORE ADDRESS OF CALLER'S SAVE

\* AREA IN THIS CSECT's SAVE AREA

\* (BACKWARD LINKAGE)

LR 13,14 POINT R13 AT THIS CSECT'S SAVE

\* AREA

.

. *(more code here)*

.

ENDPGM SR 15,15 R15 = RETURN CODE OF 0

L 13,4(,13) POINT R13 TO CALLER'S REGS

\* IN THIS CSECT's SAVE AREA

L 14,12(,13) RESTORE REGISTER 14

LM 0,12,20(13) RESTORE R0 THRU R12

\* DO NOT RESTORE R15 SINCE IT HAS RC

BR 14 BRANCH BACK TO CALLER

\*

LTORG

\*

SAVEAREA DS 18F SAVE AREA FOR THIS PGM’S REGS

\*

END EXAMPLE

# 1.6 JCL Coding and Documentation Standards

JCL (Job Control Language) tells the system the programs to execute, the order in which they are to be executed, and the data to be used and/or created by each of the programs executed.

Because JCL is a cryptic, symbols-oriented language like Assembler, documentation is especially important. JCL will be studied in much more detail in Chapter 2.

* + 1. Outline all documentation with a box of asterisks.
    2. Include documentation immediately following the **JOB statement** that states the job's task(s). Comment on any JCL that appears before the first EXEC statement.
    3. Immediately after each **EXEC statement** explain the function of the step. List each **DD name** dentified with the step and state whether it labels input or output. Describe the file associated with each DD name.
    4. Document **temporary work data sets** in a separate box.
    5. Place any other explanatory comments in the JCL stream as needed.
    6. Separate some lines and blocks of somewhat "unrelated" JCL with empty comment lines by putting //\* in columns 1-3 and nothing following this sequence of symbols. This adds "white space" and makes it easier to follow.

VSAM IDCAMS DOCUMENTATION

1. Follow the JCL documentation standards above.
2. Line document each IDCAMS parameter.

Example of IDCAMS line documentation:

DEFINE CLUSTER ( /\* DEFINE VSAM KSDS CLUSTER \*/-

NAME (KC0nnnn.FA6.KSDS) /\* KSDS CLUSTER'S NAME \*/-

VOLUMES(ACA301) /\* RESIDES ON VOL=SER=ACA301 \*/-

TRACKS(1 1) /\* SPACE ALLOCATION \*/-

INDEXED /\* CLUSTER IS INDEXED \*/-

KEYS(9 0) /\* PRIME KEY LEN=9 DISPLACEMENT=0 \*/-

RECORDSIZE(80 80)) /\* RECORD LENGTH IS FIXED 80 BYTES \*/

# 1.7 Example JCL with Documentation

//KC0nnnnA JOB ,'your last name',MSGCLASS=H

//\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\* \*

//\* CSCI 465/565 - current semester, i.e., SPRING 2018 \*

//\* \*

//\* ASSIGNMENT n \*

//\* \*

//\* PROGRAMMER: your name \*

//\* \*

//\* DATE: due date \*

//\* TIME: due time \*

//\* \*

//\* THE PURPOSE OF THIS JOB IS TO COMPILE AND EXECUTE A \*

//\* COBOL PROGRAM TO PRODUCE CHECKING ACCOUNT STATEMENTS. \*

//\* \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*

//JSTEP01 EXEC PGM=IGYCRCTL,PARM=APOST

//\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\* \*

//\* JSTEP01 COMPILES THE COBOL SOURCE CODE USING V.5 OF THE \*

//\* COBOL COMPILER. \*

//\* \*

//\* DDNAME FILE DESCRIPTION \*

//\* \*

//\* SYSLIB INPUT: LIBRARY CONTAINING COBOL COPY BOOK \*

//\* SYSIN INPUT: SOURCE CODE TO BE COMPILED \*

//\* \*

//\* SYSLIN OUTPUT: THE RESULTING OBJECT MODULE \*

//\* SYSPRINT OUTPUT: SOURCE LISTING AND DIAGNOSTIC \*

//\* MESSAGES \*

//\* \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*

//SYSLIB DD DSN=KC0nnnn.CSCI465.COPYLIB,DISP=SHR

//\*

//SYSIN DD \*

*( COBOL source code goes here )*

/\*

//\*

//SYSLIN DD DSN=&&OBJMOD,SPACE=(TRK,(3,3)),

// DISP=(NEW,PASS,DELETE)

//\*

//SYSPRINT DD SYSOUT=\*

//\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\* \*

//\* THE FOLLOWING DATA SETS ARE REQUIRED BY THE COMPILER. \*

//\* \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*

//SYSUT1 DD SPACE=(CYL,(1,1))

//SYSUT2 DD SPACE=(CYL,(1,1))

//SYSUT3 DD SPACE=(CYL,(1,1))

//SYSUT4 DD SPACE=(CYL,(1,1))

//SYSUT5 DD SPACE=(CYL,(1,1))

//SYSUT6 DD SPACE=(CYL,(1,1))

//SYSUT7 DD SPACE=(CYL,(1,1))

//SYSUT8 DD SPACE=(CYL,(1,1))

//SYSUT9 DD SPACE=(CYL,(1,1))

//SYSUT10 DD SPACE=(CYL,(1,1))

//SYSUT11 DD SPACE=(CYL,(1,1))

//SYSUT12 DD SPACE=(CYL,(1,1))

//SYSUT13 DD SPACE=(CYL,(1,1))

//SYSUT14 DD SPACE=(CYL,(1,1))

//SYSUT15 DD SPACE=(CYL,(1,1))

//SYSMDECK DD SPACE=(CYL,(1,1))

//\*

//JSTEP02 EXEC PGM=HEWL,PARM='MAP,LET',COND=(0,LT)

//\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\* \*

//\* JSTEP02 CREATES A LOAD MODULE FROM THE INPUT OBJECT \*

//\* MODULE AND STORES IT AS A PROGRAM OBJECT. \*

//\* \*

//\* SYSLIB INPUT: LOAD MODULE LIBRARIES NEEDED BY \*

//\* THE LOADER FOR A COBOL OBJECT MODULE \*

//\* SYSLIN INPUT: THE OBJECT MODULE \*

//\* SYSLOUT OUTPUT: LOADER DIAGNOSTIC MESSAGES \*

//\* \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*

//SYSLIB DD DSN=CEE.SCEELKED,DISP=SHR

//\*

//SYSLIN DD DSN=&&OBJMOD,DISP=(OLD,DELETE,DELETE)

//\*

//SYSLOUT DD SYSOUT=\*

//\*

//JSTEP03 EXEC PGM=pgmname,COND=(0,LT)

//\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\* \*

//\* THE FOLLOWING DATA SETS ARE REQUIRED BY THE COBOL \*

//\* PROGRAM. \*

//\* \*

//\* STEPLIB INPUT: LOCATION OF THE PROGRAM OBJECT pgmname \*

//\* CUSTOMER INPUT: CUSTOMER TRANSACTION DATA \*

//\* \*

//\* STATEMNT OUTPUT: CHECKING ACCOUNT STATEMENTS \*

//\* \*

//\* SYSUDUMP OUTPUT: FOR A SYSTEM DUMP IF ABEND \*

//\* \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*

//STEPLIB DD DSN=KC0nnnn.CSCI465.LOADLIB,DISP=SHR

//\*

//CUSTOMER DD DSN=KC0nnnn.CUSTOMER.TRANS,DISP=SHR  
//\*

//STATEMNT DD SYSOUT=\*

//\*

//SYSUDUMP DD SYSOUT=\*

//