IBM

SYSTEM/360 COBOL

Writing Programs in COBOL

Text

Programmed Instruction Course

IBM

SYSTEM/360 COBOL

Writing Programs in COBOL

Text

Programmed Instruction Course

Copies of this publication can be obtained through IBM Branch Offices.
Address comments concerning the contents of this publication to:
IBM DPD Education Development, Education Center, Endicott, New York

PREFACE

The general objective of this book is to teach students to compose original programs in System/360 COBOL. The major topics discussed are: the COBOL program sheet -- its format and the rules for using it; the notation system used to describe entry formats; the formats of the four divisions; file descriptions, record descriptions, and item descriptions; ways of naming data items; and the formats of the most commonly used procedural words.

The student gets practice in recognizing correctly written entries, recognizing faulty entries and correcting them, and writing original entries. He uses the entries in complete programs.

Not all possible entry formats are discussed; rather, a selection of important entries is presented. Also, many of the formats have been simplified or abbreviated to make them easy to learn, and to spare the student from being buried in a heap of details. For example, when arithmetic verbs are discussed, considerations such as truncation, rounding, size errors, etc. are not mentioned; these topics are taken up in the next course in this series.

Omitted, too, is a discussion of the entries that are used for processing non-sequential (random) files. This book concentrates on presenting a subset of System/360 COBOL which is adequate for processing sequential files.

This textbook is designed to be studied in conjunction with the Writing Programs in COBOL reference handbook (Form R29-0211). This book serves as a study guide, and is meant to be re-used. All technical information is contained in the reference handbook, which is kept by the student when he completes the course.

The student is expected to have completed the previous course in this series, COBOL Program Fundamentals. The publications for that course are a programmed instruction textbook (Form R29-0205) and a reference handbook (Form R29-0206). The student should have the reference handbook from the previous course, and he must also be provided with a pad of COBOL program sheets (Form X28-1464).

ACKNOWLEDGEMENT

The following information is reprinted from COBOL-61 EXTENDED, published by the Conference on Data Systems Languages (CODASYL), and printed by the U. S. Government Printing Office.

This publication is based on the COBOL System developed in 1959 by a committee composed of government users and computer manufacturers. The organizations participating in the original development were:

Air Materiel Command,
United States Air Force
Bureau of Standards,
Department of Commerce
David Taylor Model Basin,
Bureau of Ships, U.S. Navy
Electronic Data Processing Division, Minneapolis-Honeywell Regulator Company
Burroughs Corporation
International Business Machines Corporation
Radio Corporation of America
Sylvania Electric Products, Inc.
Univac Division of Sperry-Rand Corporation

In addition to the organizations listed above, the following organizations participated in the work of the Maintenance Group:

Allstate Insurance Company
Bendix Corporation, Computer Division
Control Data Corporation
DuPont Company
General Electric Company
General Motors Corporation
Lockheed Aircraft Corporation
National Cash Register Company
Philco Corporation
Royal McBee Corporation
Standard Oil Company (N.J.)
United States Steel Corporation

This manual is the result of contributions made by all of the above-mentioned organizations. no warranty, express or implied, is made by any contributor or by the committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection therewith.

It is reasonable to assume that a number of improvements and additions will be made to COBOL. Every effort will be made to insure that the improvements and corrections will be made in an orderly fashion, with due recognition of existing users' investments in programming. However, this protection can be positively assured only by individual implementors.

Procedures have been established for the maintenance of COBOL. Inquiries concerning procedures and methods for proposing changes should be directed to the Executive Committee of the Conference on Data Systems Languages.

TABLE OF CONTENTS

STUDENT INSTRUCTIONS

1. This is the second in a series of programmed instruction courses on System/360 COBOL. The previous course, entitled COBOL Program Fundamentals, is a prerequisite to this course.

2. Be sure to read the Preface of this book, which explains the overall goal of this course.

3. Besides this book, you must have:

   - the reference handbook (Form R29-0211) for this course.
   - the reference handbook (Form R29-0206) from the previous course in this series.
   - a pad of COBOL program sheets (Form X28-1464).

4. All reading assignments given in this textbook are in reference handbook R29-0211. However, you are expected to take the initiative in reading reference handbook R29-0206 whenever you need to review background information.

5. The reference handbook is yours to keep, and you can write notes in it if you wish. The textbook, on the other hand, will be used by other students, so you are not to fill in any of the blanks or make any notes in this book.

6. The format of this book is exactly the same as that used in the previous programmed textbook in this series. As before, topics of study are presented in a series of frames, with most frames requiring you to choose an answer or to formulate an answer mentally. The correct answers are given right after each question, and you should use a card or a sheet of paper to cover up the correct answer until you have had a chance to formulate your own response to the question.

7. If the meanings of symbols like bracket and braces (as they are used in frames) are fresh in your mind, you may begin Lesson 1; otherwise, read the information on the next page.

HOW TO STUDY THIS BOOK

1. Each lesson is broken up into a number of frames, which are simply convenient instructional steps that are to be studied in sequence. Most frames have two parts: the first part usually asks a question or requires you to take some action; the second part gives the correct answer to the question. The end of the first part is marked by a group of three dots printed in the center of the page. If the frame asks a question, the correct answer is printed on the same page, below the three dots.

2. As you study each frame, you must use an ordinary sheet of paper or a card to hide the correct answer from yourself. You will learn the subject best by working out the answers, not by just reading words.

3. Start each page by putting your "hider" sheet or card at the top. Then slide your sheet down until you just uncover a group of three dots. This will allow you to read the first part of a frame, and to formulate your answer to the question or problem it poses. When you have your answer clearly in mind, slide the "hider" sheet down to the next group of three dots. This will not only reveal the correct answer, but also uncover the first part of the next frame.

4. Most frames require you to formulate an answer mentally. Your answer may sometimes be different from the printed answer, but it should mean the same. If your answer is wrong, study the question again with the correct answer in mind.

5. On the whole, the course is composed of reading assignments and questions. When a frame gives you a reading assignment, be sure to complete the assignment before going on to the next frame. The frames that follow a reading assignment may ask questions about what you have read, or ask you to apply what you have read; they may also provide additional information about the topic. You will find instructions, remarks, and the author's opinions printed in italics in a few frames.

6. Whenever a frame asks a question based on information in the reference handbook, and you cannot remember the information, you should reread that topic in the reference handbook.

7. When you come to a blank _____ in a frame, you are to think of one or more words that complete the sentence. The length of the blank space is always the same, so it is not a clue to the length of the answer. Do not write your answer in the book.

8. Some frames present a choice of answers, from which you are to select the one best answer. The choices are stacked in { } braces.

9. Other frames present a choice of answers, from which you are to select all correct answers. All of the choices may be correct; more than one, or just one may be correct; or none may be correct. It is therefore necessary for you to examine every choice. Each choice of this kind is enclosed in brackets [].

LESSON 1

1. *In the previous course (COBOL Program Fundamentals), you studied many sample COBOL entries and programs, all of which were written on program sheets. COBOL programs are normally written on such sheets, which are sometimes called "coding sheets"; however, this is not absolutely necessary, since programs might be written on just about anything -- even scratch paper. (Some companies that use COBOL have printed their own special coding forms, and a few companies have done away with coding forms altogether by punching COBOL cards directly from flowcharts, decision tables, and record layouts!) Most COBOL users write on the standard type of program sheet, though, and you will be using the standard sheet throughout this course.*

It will be worth your while to remember which columns of the program sheet are used for program entries, and which columns are used for other purposes. Make sure that you learn what "margin A" and "margin B" are.

Reading assignment: PROGRAM SHEET FORMAT

●●●

2 The program sheet has space for _____ columns of information.

●●●

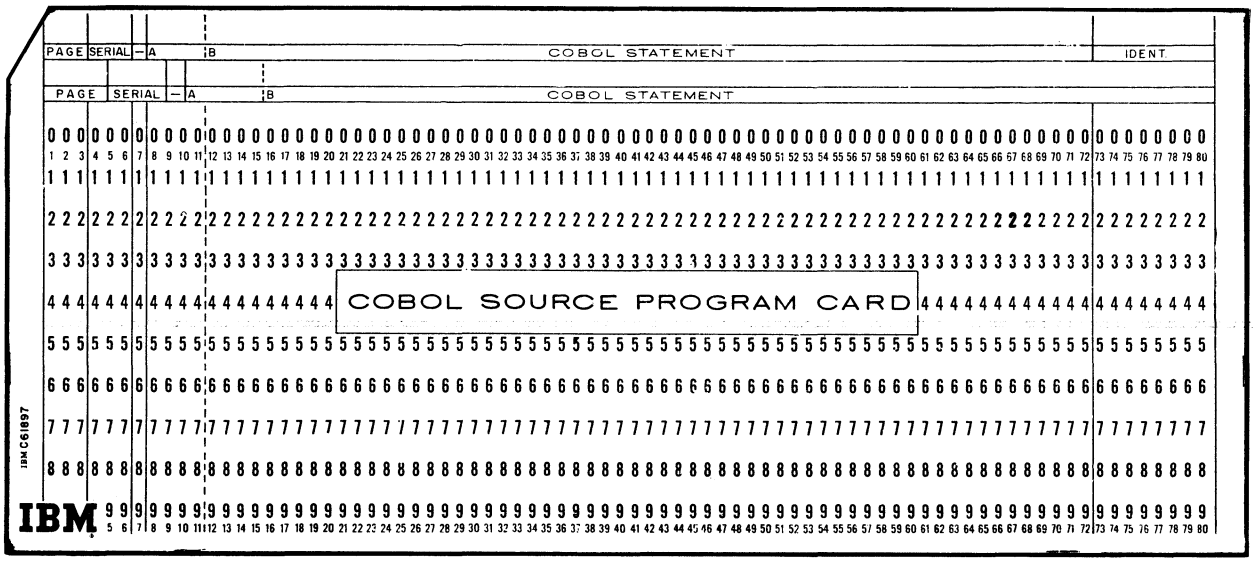
80

3 The program sheet has 80 columns because _____.

●●●

it serves as an input document to a card punching operation

4 This is what the COBOL card looks like.



Although columns 8-72 have no title on the program sheet, the corresponding field in the card is named _____.

●●●

COBOL STATEMENT

5 A separate card is punched for

{each separate program sheet / each column on the program sheet / each line on the program sheet} .

●●●

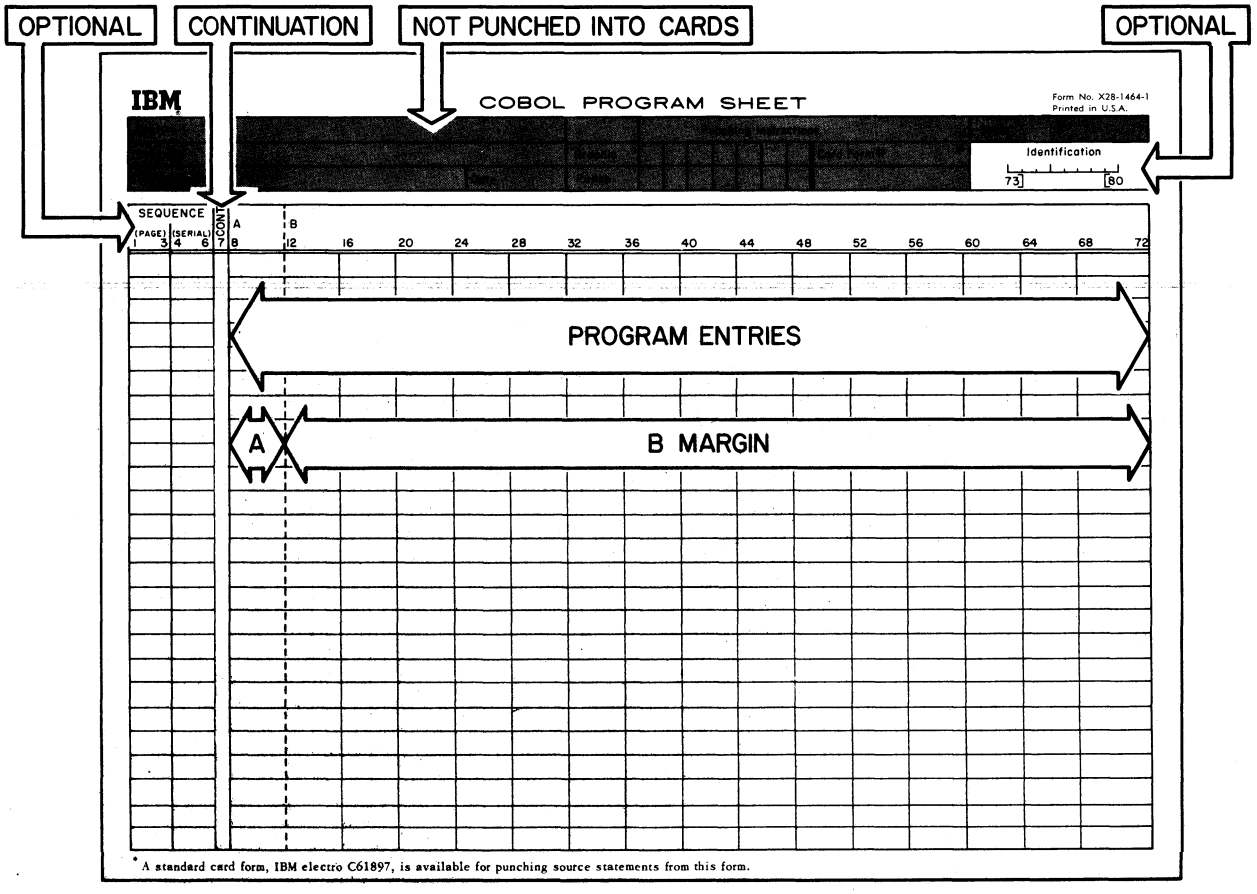
each line on the program sheet

6 Information entered at the _____ of the sheet is not punched into cards.

●●●

top

7 Examine the illustration below, which sums up the information that you read in the reference manual.



Columns 1-6 and 73-80 are marked "optional", which means that

those columns are not printed on some program sheet forms
something must be written in one field or the other, but not both
neither the identification nor the sequence number is mandatory
the programmer can disregard his boss's orders to fill them in

●●●

neither the identification nor the sequence number is mandatory

However, both of these fields are generally filled in, with each computer installation setting its own standards for identification codes and sequence numbers -- which means that the programmer fills in those fields the way his boss wants it done -- regardless of the fact that the compiler technically doesn't require anything in them. In this book, we will leave the optional columns blank, in order to concentrate on program entries.

8 A hyphen is written in column 7 to signify the continuation of _____.

●●●

non-numeric literals

Complete rules for continuing non-numeric literals will be discussed a little later in this lesson.

9 COBOL entries are written in columns _____ through _____.

●●●

8, 72

10 The program entry columns of the program sheet are divided into two "margins". Notice that a "margin" in this case is an area that you are supposed to write in; a rather different use of the word from the usual notion of a margin as a narrow border around the edges of the paper -- a space that you are not supposed to write in.

The names of the margins are certainly simple and easy to remember -- they are margin _____ and margin _____.

●●●

A, B,

11 A broken line has been printed between columns 11 and 12 to mark the boundary between margin A and margin B, and little letters A and B have been printed above columns 8 and 12 to identify the margins.

Don't let those little letters mislead you. They are there to remind you where each margin begins, and they do not mean that each margin is just one column. Actually, margin A comprises columns _____ through _____, while margin B comprises columns _____ through _____.

●●●

8, 11; 12, 72

12 Suppose a rule states that a certain entry must "begin in the B-margin". Following the rule, that entry

{ must be started in column 12
might be started in column 16
can be started in column 8, 9, 10, 11, or 12. } .

●●●

might be started in column 16

We will see that the most likely starting point for such entries is column 12 -- but that this is not a "must".

13 Suppose that the rule for another kind of entry states that the entry must "begin in the A-margin". One such entry is the paragraph header "REMARKS". If a programmer were to begin this entry in column 8, he could only write REMA in the A-margin. If he followed the rule, you would then expect him to

{ write RKS. in columns 12-15 of the B-margin
write RKS. in the A-margin of the next line
abbreviate the entry to RMK. so it would fit in the A-margin
omit the remainder of the entry } .

●●●

write RKS. in columns 12-15 of the B-margin

The rule specifies a beginning place, but does not require the entry to be completely contained within the A-margin.

14 *There really are rules like those referred to above, and we will turn our attention to them now. First, we will look at the rules for writing elements in entries. From the previous course, you will recall that an "element" is the basic unit of the COBOL language; reserved words, programmer-supplied names, symbols, literals, level numbers, and pictures are the six "elements". You will find that there are just two major rules for writing elements in entries, but that each rule has an exception. You will want to study the exception carefully, of course, but make sure you understand the general rule first.*

Reading assignment: HOW ELEMENTS ARE WRITTEN IN ENTRIES
Elements written on program sheets
Spacing between elements

●●●

15 The first rule that you have just read states that, except for non-numeric literals, no {elements / entries} may be split or divided between lines.

●●●

elements

You will discover shortly that entries can be continued on more than one line. The distinction between an element and an entry should be clear in your mind. Remember that an "entry" consists of two or more elements, the last of which is a period.

16 *COBOL systems for some other computers do permit elements such as names to be divided; however, experience has shown that this makes a program harder to read, and makes errors more likely. So the rule for System/360 COBOL is that no elements except non-numeric literals can be divided.*

Which is the correct way to write the name ACCOUNTS-RECEIVABLE on a program sheet, if there are not enough spaces left on a line to write out the whole name:

[The name may be broken at a syllable boundary, so ACCOUNTS-RECEIV might be written on one line, and ABLE on the next line.]
[The name can be broken at the hyphen, so ACCOUNTS- would be written on one line, and RECEIVABLE on the next line.]

●●●

NEITHER of these ways is correct.

The characters form one name (one element) and so they must not be broken at any point. If there is not enough room left on a line to write the entire name, then it must be written entirely on the next line.

17 There is a good reason for permitting non-numeric literals to be divided. No matter how small you print, there are still only 61 spaces in the B-margin, and a non-numeric literal can be up to _____ characters long.

If you don't remember, look this information up in the COBOL Program Fundamentals reference handbook (Form R29-0206).

●●●

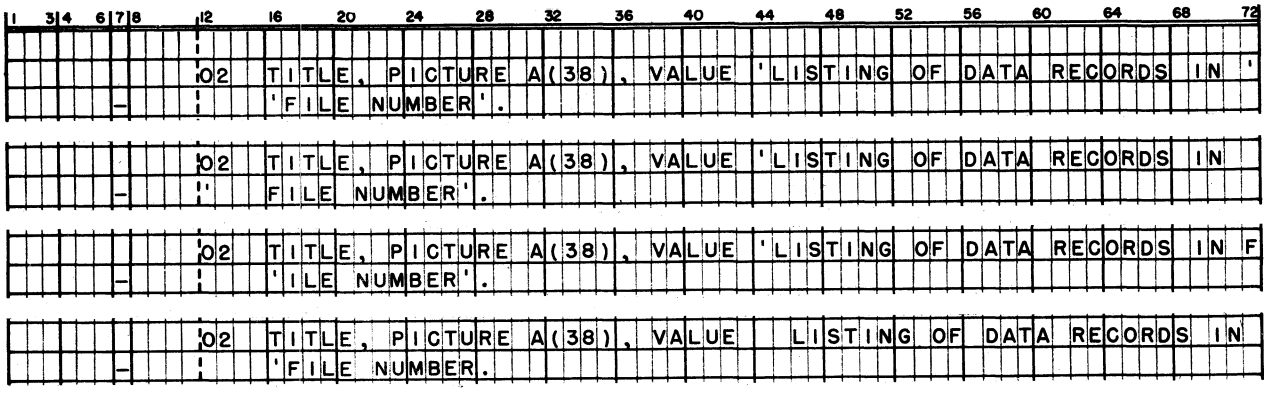
120

18 Doesn't this same reasoning apply to numeric literals? Shouldn't it be permissible to divide a numeric literal that is, say, 80 digits long?

●●●

Numeric literals are limited to 18 digits, so they need not (and must not) be divided.

19 Which of the examples below is a correct way to divide the literal LISTING OF DATA RECORDS IN FILE NUMBERS between two lines?

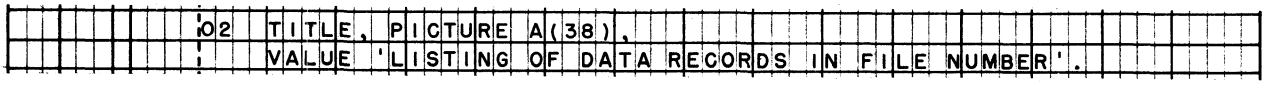


●●●

The third entry is the correct one. In it, the characters of the literal are written all the way out to column 72, then an extra quotation mark is written in the B-margin of the next line, followed by the remaining characters of the literal.

The first entry contains two non-numeric literals instead of one, which in this case, is an error; the mistake is the quotation mark at the end of the first line. In the second entry, the characters of the literal are not written to the end of the first line, nor do they follow right after the extra quotation mark in the second line; as a result, there would be four more spaces than were wanted in the literal, and the compiler would take it to be LISTING OF DATA RECORDS IN FILE NUMBER. In the fourth entry, the quotation marks that must enclose the literal have been omitted.

Even though the third entry is correct, its awkward breaking of the word FILE makes it hard to read. Actually, it would have been possible to write this entry without dividing the literal at all, as shown below. The "common sense" rule is: Don't divide non-numeric literals, if you can avoid it.



20 You have also read that each element must be separated from the next element by at least one space. The exception in this rule applies to certain _____.

●●●

symbols

21 *Specifically, the exception to the spacing rule applies to parentheses, quotation marks, commas, semicolons, and periods. This exception (which says that no spaces are to be left between those symbols and other elements in some cases) should not seem unreasonable to you; in fact, each 'exception' case can be found in the punctuation of the sentence that you are now reading.*

If the parallel between ordinary English punctuation and the exceptions to the spacing rule did not occur to you before, it would be wise for you to reread the rule and to look for this parallel.

To be sure , a sentence ' punctuated ' like this one (I hope you noticed that it is different) is not particularly hard for you to read ; however , a program ' punctuated ' like this sentence would not be acceptable to the COBOL compiler .

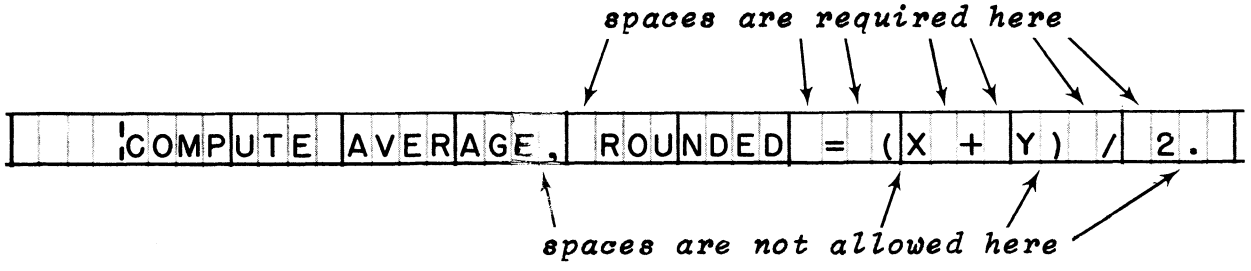
●●●

22 Keep in mind that the spacing exception applies only to certain symbols. The remaining symbols follow the general rule -- that is, they must be separated from neighboring elements by at least one space.

The entry below violates this rule several times. See if you can eliminate all the errors. Rewrite the entry on a COBOL program sheet.

```
    !COMPUTE AVERAGE,ROUNDED=(X+Y)/2.
```

●●●



23 *Now that you know the rules regarding the division and spacing of elements to make up entries, let's go one step further and look at the rules for writing entries on the program sheet.*

Reading assignment: RULES FOR PROGRAM ENTRIES
Entries that begin in margin A
Entries that begin in margin B
New line required
Spacing between entries
Continuation of entries

●●●

24 According to the rules for using the COBOL program sheet, it is all right for a line of the form to be [filled in completely] [left partly blank] [left entirely blank].

●●●

ALL of these possibilities are allowed. A line may be filled in completely OR left partly blank OR left entirely blank.

Incidentally, if lines are given sequence numbers (columns 1-6), the blank lines are numbered too. There is a distinction, of course, between lines left entirely blank to improve the readability of the program listing, and unused lines on the program sheet.

25 An entry may be continued on the next line or lines

{whether or not it could be written on one line / only if it is too long to fit on one line}.

●●●

whether or not it could be written on one line

26 Most entries are written entirely within margin _____.

●●●

B

27 A few entries are required to begin in margin A. What this amounts to is that major headings are brought out to the left for emphasis. Although these entries may start in columns 8, 9, 10, or 11, it makes sense to make them stand out as far as allowed.

Accordingly, these entries are usually started in column {8 / 11} .

●●●

8

28 A division header (such as IDENTIFICATION DIVISION.) must begin in the A-margin, and

{no part of it may be written in the B-margin / most of it will be written in the B-margin} .

●●●

most of it will be written in the B-margin

29 Below is a sample file description entry. This entry serves to illustrate that the continuation of an entry is written in margin _____, even when the entry begins in margin _____.

```
FD  SALES-MASTER-FILE,
    BLOCK CONTAINS 30 RECORDS,
    LABEL RECORDS ARE STANDARD,
    DATA RECORD IS SALES-MASTER-RECORD.
```

●●●

B; A

30 In the sample entry given in the previous frame, two spaces were left between FD and SALES-MASTER-FILE. Would it have been equally correct to leave just one space, as shown below?

```
FD SALES-MASTER-FILE,
```

●●●

No, only the level indicator (FD) may appear in the A-margin; the rest of the entry must be written in the B-margin.

31 You know that the elements in an entry must, generally, be separated from each other by at least one space. A special case exists when an element ends in column 72, as shown below.

```
1  3|4  6|7|8   12   16   20   24   28   32   36   40   44   48   52   56   60   64   68   72
           MULTIPLY AMOUNT OF CLIENT-PURCHASES BY TRADE-DISCOUNT, GIVING
```

Which of the lines below shows a correct way of completing this entry?

```
     -REDUCTION.
       REDUCTION.
           REDUCTION.
     -           REDUCTION.
```

●●●

ONLY the third line

If an element ends in column 72, it is treated (by the compiler) as if it were followed by a space. Therefore, the continuation of such an entry may be written right at the beginning of the B-margin.

The first two choices show the continuation of the entry written in the A-margin, which is illegal. And the first and fourth choices have hyphens in column 7, also illegal. The fourth choice is correct, except for the hyphen in column 7.

32 *Here is another application of the "column 72" rule. On the first line below, an entry goes all the way to the end of margin B; on the second line, the next entry starts right at the beginning of margin B. The requirement that the period ending an entry must be followed by a space is satisfied by the"assumed space" after column 72.*

```
1  3|4  6|7|8   12   16   20   24   28   32   36   40   44   48   52   56   60   64   68   72
           DISPLAY 'LAST CHECK NUMBER USED--' CHECK-NUMBER UPON CONSOLE.
           CLOSE PAYROLL-FILE, CHECK-FILE.
```

●●●

33 You can see that, in most cases, the "column 72" rule works to our advantage; in the case below, however, it does not. The entry below is wrong. See if you can figure out why it is wrong, and what can be done to correct it.

```
1 3 4 6 7 8   12  16  20  24  28  32  36  40  44  48  52  56  60  64  68  72
              READ SERVICE-CALL; AT END, CLOSE SERVICE-CALLS-FILE, STOP RUN
              .
```

●●●

The period used to end an entry must not be preceded by a space. Since the reserved word RUN ends in column 72, it is treated as if it were followed by a space -- and that space precedes the period written on the next line. The simplest correction is to write the word RUN on the second line instead of the first, and write the period directly after it.

34 *In some cases, programmers mistakenly assume that the "column 72" rule applies, when it actually does not. The following is such a case.*

In this entry, the programmer thought he was writing this literal: ORDER NUMBER IS NOT VALID. The word NUMBER ended in column 72, so he assumed that the compiler would insert a space after it. However, when the program was compiled, the result was: ORDER NUMBERIS NOT VALID.

```
1 3 4 6 7 8   12  16  20  24  28  32  36  40  44  48  52  56  60  64  68  72
              IF NEXT-ITEM IS NOT EQUAL TO PREVIOUS + 1, STOP 'ORDER NUMBER
            - 'IS NOT VALID.'.
```

What was the fallacy in this programmer's thinking? How can his error be corrected?

●●●

The "column 72" rule applies only when an element ends in column 72. In this case, the word NUMBER is not an element; it is part of an element -- part of a non-numeric literal. Correction is easy; insert the desired space between the extra quotation mark on the second line and the word IS, or preferably, write the entire STOP statement on the second line, thereby avoiding the problem of continuing the non-numeric literal.

35 *Throughout the remainder of this book, you will be called upon to apply the rules that you read in this lesson. Whenever you are asked to write an entry, and there is any question in your mind about how to enter it on the program sheet, be sure to refer to the rules in your reference handbook.*

●●●

LESSON 2

36 *Now that you know the rules for making entries, the next step is to learn the formats of the entries. But there are scores of possible entries -- too many to commit to memory; so, you will have to resign yourself to making liberal use of reference materials for a while, at least until writing some of the more common entries becomes second nature to you.*

In COBOL, there is a standard system of notation that is used to describe entry formats in reference materials. The system is used in the reference handbook for this course, and it is also used in reference manuals and other books about COBOL, so it is important that you become familiar with it. We will now devote a little time to the notation system, before continuing to study the actual entry formats for the Identification division.

Reading assignment: SYSTEM OF NOTATION USED TO DESCRIBE ENTRY FORMATS
Sample entry format

●●●

37 One of the main distinctions that is made in this system of format notation is between words printed in capital letters and words printed in small letters. What is the difference between them?

●●●

Words printed in capital letters are reserved words; words printed in small letters represent information to be supplied by the programmer.

38 This distinction deserves a little further clarification. Words in capital letters will actually be written as such within an entry. Words in small letters will never be written as such within an entry.

Now and then, a beginner misses the point when he reads an entry format such as "GO TO procedure-name", and writes:

```
    GO TO PROCEDURE-NAME.
```

Actually, the programmer was expected to write GO TO followed by the programmer-supplied name of the _____ to which the program was to branch.

●●●

procedure

Technically, PROCEDURE-NAME is a valid programmer-supplied name, even though it is not especially meaningful; this GO TO would work if there were a procedure whose name was PROCEDURE-NAME. But imagine the mess that results when a misguided programmer writes PROCEDURE-NAME in every entry that calls for a procedure name!

39 Here is a variation on the misunderstanding of format words printed in small letters. Suppose that an entry had the format "DISPLAY literal", and a programmer wrote:

```
    DISPLAY LITERAL 'END OF PHASE 4'.
```

If he was trying to display the message, END OF PHASE 4, the programmer should have

- written only DISPLAY LITERAL, and stored the message as a constant
- omitted the word LITERAL and written DISPLAY 'END OF PHASE 4'
- written the word "literal" in small letters, as shown in the format
- omitted the word DISPLAY, since the underline means it is optional

.

●●●

omitted the word LITERAL and written DISPLAY 'END OF PHASE 4'.

40 Reserved words are required in an entry when they are underlined. However, the information represented by words printed in small letters

{ may be omitted if the programmer is tired of writing
  is optional regardless of whether the words are underlined
  is required even though the words are never underlined
  is never written as such within an entry } .

●●●

is required even though the words are never underlined

41 In certain cases, "required" words and information may be part of an optional portion of an entry. Optional portions of a format are enclosed in (what symbols?) _____.

●●●

[] (brackets)

42 The notion that some "required" parts of an entry can be "optional" is not really as paradoxical as it sounds. What we mean is that a portion of an entry must be included under certain circumstances, and omitted under other circumstances; and that portion is required to contain certain words and information when it is included.

It's like saying that it is optional to rent a car to get where you're going, but that if you decide to rent the car, you are required to pay the rental charge.

Let's take a COBOL example. Here is an optional portion taken from the item description entry format:

[VALUE IS literal]

This clause is used when you want to specify the initial value of an item in working-storage. If you decide to use the clause, what must you write?

●●●

You must write the word VALUE and a literal. (If you wish, you may write the word IS after VALUE.)

43 When an entry provides a choice of optional portions, they are stacked within a pair of brackets, like this:

```
[UNIT ]
[UNITS]
```

In the above example, the programmer

{may write UNIT, or UNITS, or neither, in the entry
must write either UNIT or UNITS in the entry
may write UNIT, or UNITS, or both, in the entry} .

●●●

may write UNIT, or UNITS, or neither, in the entry

44 When an entry provides a choice of required portions, they are stacked **within** a pair of braces, like this:

```
{POSITIVE}
{NEGATIVE}
{ZERO    }
```

In this case, the programmer

{must write one, and only one, of the words
must write at least one of the words, and may write all three
may write one, or none, of the words} .

●●●

must write one, and only one, of the words

45 This is the format of a class test, which is used in IF entries:

```
data-name  IS  [NOT]  {NUMERIC   }
                      {ALPHABETIC}
```

Can you explain which parts of this format are always required? Sometimes required? Never required?

●●●

A data name, and either NUMERIC or ALPHABETIC are always required. The word NOT is sometimes required (when you want to test for the opposite of NUMERIC or ALPHABETIC). The word IS is never required.

46 *You will get plenty of additional practice on the COBOL format notation system as we examine the entries, division by division, beginning with the Identification division. Needless to say, being able to interpret the format notation is just a small part of the game. The larger part is knowing when and why to write an entry in your program; sometimes the rules are clear-cut, leaving you no choice in the matter, but most of the time (alas!) you will be required to know what it is that you are trying to accomplish.*

To help you with the "whens" and "whys", the reference handbook summarizes the function of each entry or set of entries. And because the notation system is not foolproof, and is sometimes open to various interpretaions, examples of the entries, and notes about the formats, are also given. Be sure to study the function, example, and notes, in addition to the format.

Reading assignment: IDENTIFICATION DIVISION

●●●

47 *Keep the reference handbook open to the page on the Identification division while we examine the entries that make up the division.*

How many required entries are there in the Identification division?

●●●

Three (division header; Program-Id paragraph header; and program name entry)

48 Each entry is required to end with (what symbol?) _____.

●●●

a period

49 What other punctuation is required in the division?

●●●

Quotation marks are required around the program name.

Remember that required symbols are printed in the format, though not underlined. Optional symbols, such as commas, that may be used in entries are not printed in the format at all.

50 Program name may be up to _____ characters long.

●●●

eight

51 *Programmers sometimes ask, "How will I know what the program name is?" I suppose the best answer is that it all depends on the type of job situation you are working in. If you are one of several programmers who are implementing a large system, then the system designer will probably have specified the name of each program in the system. If you are working independently on a project of your own, it will no doubt be up to you to invent a name for your program; but be sure to find out whether standards have been set up for classifying and naming programs in your computer installation. If you are a student, and the program is a class exercise, you can most likely dream up any name you please; or perhaps your teacher will come up with a choice name -- or two -- for your program.*

●●●

52 The Program-Id paragraph is required in every program; all of the other paragraphs are _____.

●●●

optional

53 Since the format for each optional paragraph is enclosed in a separate pair of brackets, you can conclude that

{ you may choose to write one, or none, of the optional paragraphs
you must write one or more of the optional paragraphs
you must write one, and only one, of the optional paragraphs
you may write any number, or none, of the optional paragraphs } .

●●●

you may write any number, or none, of the optional paragraphs

The paragraphs are not stacked within brackets; each is a separate option. Note, though, that your installation very likely has firm standards for documentation that must be provided in the Identification division. Obviously, such standards override the fact that the compiler allows you to omit all of the optional paragraphs.

54 Each optional paragraph consists of a paragraph header and "entry...". What does "entry..." mean?

●●●

One entry is required, and additional entries are permitted.

55 Are there any restrictions on what may be written in the entries in optional paragraphs?

●●●

No, except that each entry must be terminated by a period.

These entries are simply comments, and the compiler has been programmed to ignore their contents. Therefore, you may write anything -- reserved words, quotation marks, numbers, parentheses, asterisks -- you name it. Whatever you write will be printed in the listing of the source program, so that your explanatory comments will always accompany the rest of the program. Needless to say, your comments should be concise and pertinent; don't use the Identification divison as a vehicle for getting your short novels into print, or for publishing sonnets inspired by the cute programmer in Dept. 983B.

56 *You are supposed to have a pad of COBOL program sheets; if you don't, get one at this time.*

On a COBOL program sheet, write a complete Identification division for a hypothetical program named EXPENSES. The program is designed to produce a weekly listing of all operating expenses, by department, of our mythical company, Dynamic Data Devices, Inc. Naturally, the output and the program itself are strictly company-confidential, and available only to authorized personnel. The programmer, Charles Brown, himself an unauthorized person, wrote the program on November 9, 1965; and compiled and tested it, blindfolded, the very next day.

From this description, see if you can sort the pertinent information into the proper paragraphs. Most important, be certain that you observe the rules for making entries on the program sheet.

●●●

The solution for this frame is printed on the next page.

57 *The Identification division that you have written should resemble this:*

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
    'EXPENSES'.
AUTHOR.
    CHARLES BROWN.
INSTALLATION.
    DYNAMIC DATA DEVICES, INC.
DATE-WRITTEN.
    NOVEMBER 9, 1965.
DATE-COMPILED.
    NOVEMBER 10, 1965.
SECURITY.
    COMPANY-CONFIDENTIAL; AVAILABLE TO
    AUTHORIZED PERSONNEL ONLY.
REMARKS.
    PRODUCES A WEEKLY LISTING OF ALL
    OPERATING EXPENSES, BY DEPARTMENT.
```

Check your work on each of the following points:

1. *Did you begin every header entry in margin A (preferably in column 8)?*

2. *Did you keep all other entries within the B-margin?*

3. *Did you terminate every entry with a period?*

4. *Did you remember to enclose the program name in quotation marks?*

If you made any mistakes, take a moment to correct them.

●●●

58 Paragraph header entries do not have to be written on separate lines. Which line below shows another correct way in which the Program-Id paragraph might have been written?

```
PROGRAM-ID 'EXPENSES'.
```

```
PROGRAM-ID. 'EXPENSES'.
```

```
PROGRAM-ID.'EXPENSES'.
```

```
PROGRAM ID. 'EXPENSES.'
```

●●●

```
PROGRAM-ID. 'EXPENSES'.
```

The first choice is incorrect because there is no period following PROGRAM-ID. The third choice has a period, but the mandatory space after the period has been forgotten. The fourth choice has two mistakes; the hyphen in PROGRAM-ID is missing, and the period has improperly been written inside the quotation marks.

59 *Whether or not you will choose to have the paragraph headers on separate lines is something for you or your company to decide. Here are two arguments in favor of using separate lines: (1) program listings are easier to read when all of the non-header entries are aligned; (2) all fixed header entries can be prepunched into cards, and the deck of header cards can be reproduced as required, thereby eliminating all future writing and keypunching of header entries.*

●●●

60 *The Environment division is almost as easy as the Identification division. Its format is a bit more rigid, though, since each entry has a specific meaning for the compiler. (By contrast, you will recall that the compiler ignores the contents of most Identification division entries.)*

You will first study the overall format of the Environment division, and later study two particular entries in detail. Don't read about the SELECT and APPLY entries yet.

Reading assignment: ENVIRONMENT DIVISION

●●●

61 *Refer to your handbook to answer these questions.*

What are the two sections of the Environment division?

●●●

Configuration section and Input-Output section

62 The Input-Output section is an optional portion of the division's format. This section

{may be included or omitted at the whim of the programmer
must be included in any programs that process data
must be included when there are input or output files}.

●●●

must be included when there are input or output files

Data "files" are defined and discussed in the reference handbook for the previous course.

63 When an Input-Output section is written, the _____ paragraph must be included, but the _____ paragraph may be omitted if no special techniques or conditions are defined.

●●●

File-Control; I-O Control

64 The Configuration section, containing the Source-Computer and Object-Computer paragraphs, is required in every program. Each of these paragraphs may contain the reserved word _____ and the _____ of the computer.

●●●

IBM-360; model number

65 If your company has more than one System/360, you will probably want to specify which computer will be used to compile and execute your program. The model number that you write must consist of a letter representing the storage size, followed by the actual System/360 model number.

The letters that represent storage capacities are C for 8K; D for 16K; E for 32K; F for 64K; G for 128K; H for 256K; and I for 512K. As was explained in the previous course, these "K" capacities are only rough approximations of the number of bytes of storage; exact figures are given in the previous reference handbook.

If your computer is a System/360 Model 30 with a storage capacity of 128K, which of these would be the correct model number to write in your COBOL program:

{360/30G
30/G
30G
G30}

●●●

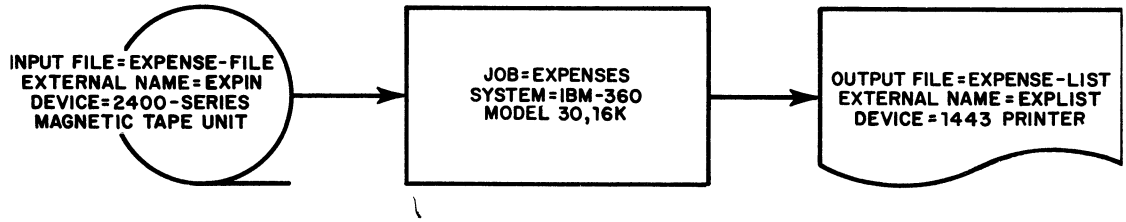
G30

66 On a COBOL program sheet, write the first half of the Environment division for the hypothetical "Expenses" program for which you coded an Identification division earlier. That is, write the division header and a complete Configuration section. The program is to be executed on a 16K Model 30, but it will be compiled on a 64K Model 40.

●●●

A B 8 12 16 20 24 28 32 36 40 44 48
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.
IBM-360 F40.
OBJECT-COMPUTER.
IBM-360 D30.

67 *This system flowchart gives us additional information about the computer environment in which the "Expenses" job is going to be run.*



From the flowchart, it is clear that the job involves data files; therefore, an Input-Output section is required in the program, and you will have to learn the format of SELECT entries for the File-Control paragraph.

(Note that the printed report is treated as a file; each line printed on the form constitutes one record of the file.)

Reading assignment: SELECT

●●●

68 Apply the information that you have just read, by coding the first part of the Input-Output section for the "Expenses" program (section header and complete File-Control paragraph). The flowchart in the previous frame contains all of the programmer-supplied information that you need. You may, of course, look back at the "ENVIRONMENT DIVISION" page of the reference handbook, to see the overall format of the section.

●●●

```
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT EXPENSE-FILE, ASSIGN TO 'EXPIN',
        UTILITY 2400 UNIT.
    SELECT EXPENSE-LIST, ASSIGN TO
        'EXPLIST', UNIT-RECORD 1443 UNIT.
```

Points to check: (1) The input file must be assigned to a UTILITY device, since it is a magnetic tape file. (2) The output file must be assigned to a UNIT-RECORD device, since it is a printed report. (3) External names must be enclosed in quotation marks. (4) Make sure you have hyphens and periods where required.

69 *The SELECT entries that you have coded may not be exactly the same as the entries printed in the preceding frame. As you can tell from the format, "device-number" is optional (and so are the words TO and UNIT, and the commas).*

Let's make this clear: you must let the computer know, sooner or later, what specific devices are going to be used for input and output. If the devices are not specified in the source program, they will have to be specified on job control cards at the time that the object program is executed. In some cases, we may wish to be able to change device assignments each time the job is run; in such cases, we will make our program "device independent" by not specifying devices in the SELECT entries.

In most cases, though, our programs are written with specific devices in mind. Then, it is just as well to indicate the device numbers right in the source program. This is true of our practice problem; the system designer has definitely told us which input and output devices are going to be used.

●●●

70 The rules for creating an external name for a file are the same as the rules for creating a _____ for the Identification division.

●●●

program name

The comments dealing with "where program names come from" apply equally to external file names.

71 *The only thing needed to complete our practice Environment division is an I-O-Control paragraph. You know that this paragraph is omitted when no special techniques or conditions are needed; however, we must include the paragraph in this program, because we have a special condition that must be specified -- the form overflow condition on the printed report. (Form-overflow is the only one of the "special techniques and conditions" that will be discussed in this course.)*

Reading assignment: APPLY

●●●

72 *Keep in mind that the overflow condition name that is defined in the I-O-Control paragraph will be used in the Procedure division. There we may write an entry such as:*

```
    IF LAST-LINE, WRITE EXPENSE-LINE
       AFTER SKIPPING-TO-NEXT-PAGE.
```

On your program sheet, finish the Environment division. Define the name LAST-LINE to represent the form-overflow condition on the printed report.

●●●

```
I-O-CONTROL.
    APPLY LAST-LINE TO FORM-OVERFLOW ON
       EXPENSE-LIST.
```

Points to check: (1) Did you remember the paragraph header? (2) The file name of the report is required in the format, not the external name of the file; don't confuse the two.

73 *It should be apparent to you that the "APPLY overflow-name" entry is used only when you have printed output. If our job had called for magnetic tape output as well as magnetic tape input, this APPLY entry would have been omitted -- in fact, we would have omitted the entire I-O-Control paragraph.*

●●●

74 *To conclude our look at Environment division entry formats, let's make just one more point: when you write a division, you must make your entries in the sequence in which they are indicated in the format; and you are allowed to make only those entries shown in the format. There is no provision, for instance, for making explanatory comments in the Environment division. You are not allowed to create a new paragraph, nor to "borrow" the Remarks paragraph from the Identification division.*

The moral of the story is: follow the formats religiously; don't deviate or improvise. After all, when you write a source program, you are preparing input to be processed by a computer program (the compiler); hence, you must adhere to the specifications for that program's input -- or else, your entries will be diagnosed as errors, and rejected as non-processable.

●●●

LESSON 3

75 *As you discovered in the previous course, the Data division is probably the most complex of the four divisions. Certainly, it demands much more of the programmer than the Identification and Environment divisions do. To be sure, the complexity of the coding depends on the complexity of the data itself; and the programmer's job is easier if he is intimately familiar with the layouts of the records and files which he is trying to describe in COBOL.*

Another thing that makes the job easier is that the programmer can give his complete attention to describing files and records, without being concerned with the procedures that will process the data. That is, the task of describing the data has been logically separated from the task of processing the data; this separation is an important feature of COBOL. In practice, this means that each record will be described once, and the same record description will be used in every program that processes the record. It also means that all programmers will use the same names to refer to data items. And it means that we are justified in studying about the Data division without worrying about the Procedure division at the same time.

We have tried not to duplicate information that you studied in the previous course. As a result, it is taken for granted that you recall the basic definitions and concepts; if you don't, you will want to re-read the appropriate topics in the previous reference handbook.

We will first work on file descriptions, then on record structures and descriptions, and lastly on item descriptions.

Reading assignment: FILE DESCRIPTION

●●●

76 The only portion of a file description entry that is written in the A-margin is _____.

●●●

FD (the "level indicator")

77 Each clause of a file description {must be / may be} written on a separate line.

●●●

may be

Having each clause on a separate line makes the entry easier to read and easier to correct or update.

78 Besides FD and the file name, which clauses are required in every file description?

●●●

LABEL RECORD and DATA RECORD clauses

79 The file name given in a file description must correspond to a file name specified in a _____ entry in the _____ division.

●●●

SELECT; Environment

80 *In the next five frames, we will deal with files of punched cards.*

The recording mode of a punched card file must be _____, because _____.

●●●

F, because all records in the file are the same length (80 characters), and there are no record-length control fields.

Recording modes are discussed in detail in the reference handbook for the previous course.

81 For a card file, the BLOCK CONTAINS clause would be omitted, because

{each block contains only one record / there are no blocks in a card file} .

●●●

each block contains only one record

By definition, all files contain blocks, since a "block" is the unit of data that is transferred to or from storage at one time by the input-output device. A card file is read or punched one card at a time; hence each card is a block, and contains just one record.

82 Every record in a card file contains 80 characters, regardless of how many card columns are punched. (Unpunched columns contain the character "blank".) Thus, each record description for a card record must account for all 80 characters, and the file description entry

[should state RECORD CONTAINS 80 CHARACTERS]
[should account for only the columns that are actually punched]
[may be omitted].

●●●

EITHER should state RECORD CONTAINS 80 CHARACTERS, OR may be omitted

83 Every card in a file is a data record. This includes even those cards, such as leader or trailer cards, which contain accounting or control information about the file.

From this you can conclude that label records are {standard / omitted} in card files.

●●●

omitted

Let me restate the idea: all cards must be defined as data records, even though certain cards in the file may serve the same function as label records would in tape or disk files.

84 On a COBOL program sheet, write the file description entry for a card file called MAILING-FILE. The file contains three types of cards -- ACTIVE-CUSTOMER, INACTIVE-CUSTOMER, and PROSPECT cards.

●●●

```
FD  MAILING-FILE;
    RECORDING MODE IS F;
    LABEL RECORDS ARE OMITTED;
    DATA RECORDS ARE ACTIVE-CUSTOMER,
        INACTIVE-CUSTOMER, PROSPECT.
```

You might also have written a BLOCK CONTAINS clause after the RECORDING MODE clause: BLOCK CONTAINS 80 CHARACTERS. Make certain that you wrote the entire FD-entry as one entry; that is, make sure you have just one period -- at the very end. The other punctuation shown above (semicolons and commas) might have been omitted.

85 *Files of records that are going to be printed are treated exactly the same as files of cards. The recording mode is F, since each output line has a fixed length equal to the capacity of the printer. Each block contains only one record, since lines are printed one at a time. And there are no label records.*

On another program sheet, write the FD-entry for the EXPENSE-LIST file (the output file of the EXPENSES program). Two kinds of lines are to be printed in the report; so there are two types of data records in the file. The lines are named EXPENSE-DETAIL and DEPARTMENT-TOTAL.

●●●

```
FD  EXPENSE-LIST;
    RECORDING MODE IS F;
    LABEL RECORDS ARE OMITTED;
    DATA RECORDS ARE EXPENSE-DETAIL,
         DEPARTMENT-TOTAL.
```

86 *Now let's turn to files recorded on tape or disk. Here the recording mode can be V, F, or U. Each block generally contains more than one record. Usually, the file contains label records.*

On still another program sheet, write the file description of EXPENSE-FILE (the input file of the EXPENSES program). You will recall that this file is on magnetic tape. The file has standard label records, and one type of data record, called EXPENSE-RECORD All data records in the file are a fixed length, and each record is preceded by a record-length control field. There are 20 records per block.

●●●

```
FD  EXPENSE-FILE,
    RECORDING MODE IS V,
    BLOCK CONTAINS 20 RECORDS,
    LABEL RECORDS ARE STANDARD,
    DATA RECORD IS EXPENSE-RECORD.
```

The recording mode must be V, because there are record-length control fields. (The RECORDING MODE clause might therefore have been omitted.)

87 *The format of the File section requires that each file description must be followed by one or more record descriptions -- one record description for each type of record in the file. Record descriptions were discussed in detail in the previous course, and we will review them only briefly here. The important points are that a record description is a set of item descriptions; that it shows the sequence of items in a record; and that the items are arranged into levels to show the structure of the record.*

Reading assignment: RECORD STRUCTURE
An illustration of levels of data items
AN EXAMPLE OF A RECORD DESCRIPTION

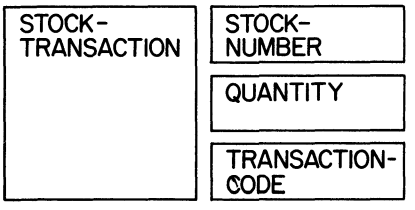
●●●

88 Level number _____ is always assigned to the record as a whole.

●●●

01

89 Suppose that we have a tape record named STOCK-TRANSACTION, which contains just three items, as pictured below.



What level number would you assign to:

STOCK-TRANSACTION?
STOCK-NUMBER?
QUANTITY?
TRANSACTION-CODE?

●●●

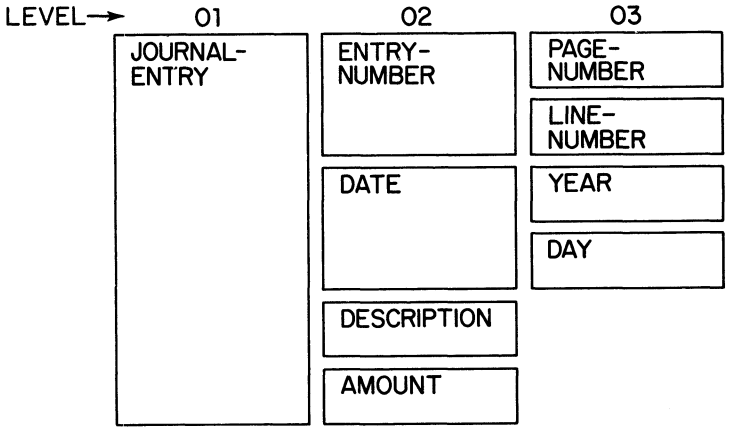
Level 01 must be assigned to STOCK-TRANSACTION, which is the record as a whole. The other three items would be assigned level number 02 (or any number greater than 02, but not greater than 49).

Normally, the levels are numbered consecutively -- 01, 02, 03, etc. -- and we will use this convention in this book. However, your firm may have adopted another numbering convention. The important thing to recognize about the record pictured in this frame is that STOCK-NUMBER, QUANTITY, and TRANSACTION-CODE are all at the same level; what the level number is doesn't matter -- as long as all three have the same level number.

90 Now let's envision a slightly more complicated record, called JOURNAL-ENTRY. It consists of ENTRY-NUMBER, DATE, DESCRIPTION, and AMOUNT, in that order. Some of these items are broken down further: ENTRY-NUMBER is made up of PAGE-NUMBER and LINE-NUMBER and DATE is made up of YEAR and DAY.

See if you can diagram the structure of this record. Also, indicate the level numbers. (Just sketch this roughly on a piece of scratch paper.)

●●●



91 Next, check your understanding of the order in which items are entered on the program sheet. Using a program sheet, write the level number and name of each item in the record diagrammed in the preceding frame.

Put each item on a separate line. (Don't be concerned about making complete entries, with PICTURE clauses, and so on -- just write the level numbers and names.)

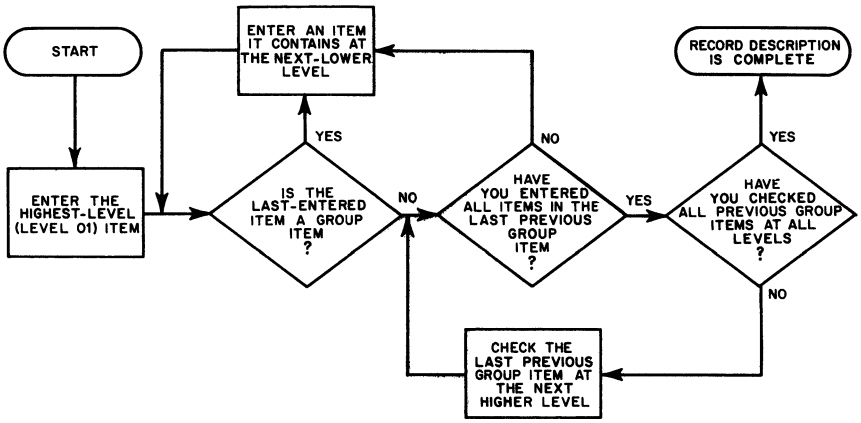
●●●

```
01  JOURNAL-ENTRY
    02  ENTRY-NUMBER
        03  PAGE-NUMBER
        03  LINE-NUMBER
    02  DATE
        03  YEAR
        03  DAY
    02  DESCRIPTION
    02  AMOUNT
```

92 *Check these points on your program sheet: (1) Indenting is highly desirable, but not required. The convention is to indent each level to the next column whose number is a multiple of 4 (these columns are marked by heavier lines on the program sheet). So, if you began with level 01 in column 8, all level-02 entries would begin in column 12, all level-03 entries would begin in 16, and so on. (2) Remember this program sheet rule: level numbers may be written in the A-margin, but <u>not</u> data names or any other elements of an item description entry. This means that a level number may be written in columns 8 and 9, but that the name of the item must not start prior to column 12 -- the name <u>must not</u> begin in column 11. (3) Did you enter the level numbers and names in precisely the order shown? If you did, skip the next two frames. If you did not, and you are puzzled about the order in which items are to be entered in a record description, go right on to the next frame.*

●●●

93 *This flowchart depicts the steps that a programmer follows when making entries in a record description. Study it briefly and then proceed to the next frame, in which we will apply this procedure to the exercise that you just completed.*



●●●

94 *In order to understand this procedure, you must recognize that the highest-level item has the smallest level number (01). And that larger level numbers designate lower levels. With this in mind, let's trace the steps taken in writing the record description for the JOURNAL-ENTRY record.*

1. *We start by entering the level-01 item:*

```
01  JOURNAL-ENTRY
```

2. *JOURNAL-ENTRY is a group item, since it is subdivided into smaller items; therefore, we enter the first item it contains at the next-lower level:*

```
    02  ENTRY-NUMBER
```

3. *ENTRY-NUMBER is itself a group item, so we enter the first item that it contains at the next-lower level:*

```
        03  PAGE-NUMBER
```

4. *PAGE-NUMBER is not further subdivided; so it is an elementary item, not a group item. But we have not yet entered all of the items that are part of the last previous group item (ENTRY-NUMBER); we now enter the next level-03 item that it contains:*

```
        03  LINE-NUMBER
```

5. *LINE-NUMBER is not a group item, and we have entered all of the items that make up ENTRY-NUMBER. Now we must check to see if we are finished with the last previous group item at the next higher level; that would be level 01, and we have not yet entered all of the items in JOURNAL-ENTRY. Its second item at the next-lower level (02) is:*

```
    02  DATE
```

6. *Since DATE is a group item, we enter its first sub-item:*

```
        03  YEAR
```

7. *YEAR is not a group item, but there is another sub-item in DATE:*

```
        03  DAY
```

8. *We are through with DATE, but there are more items in JOURNAL-ENTRY, and the next is:*

```
    02  DESCRIPTION
```

9. *DESCRIPTION is not a group item, so we enter the next component of JOURNAL-ENTRY:*

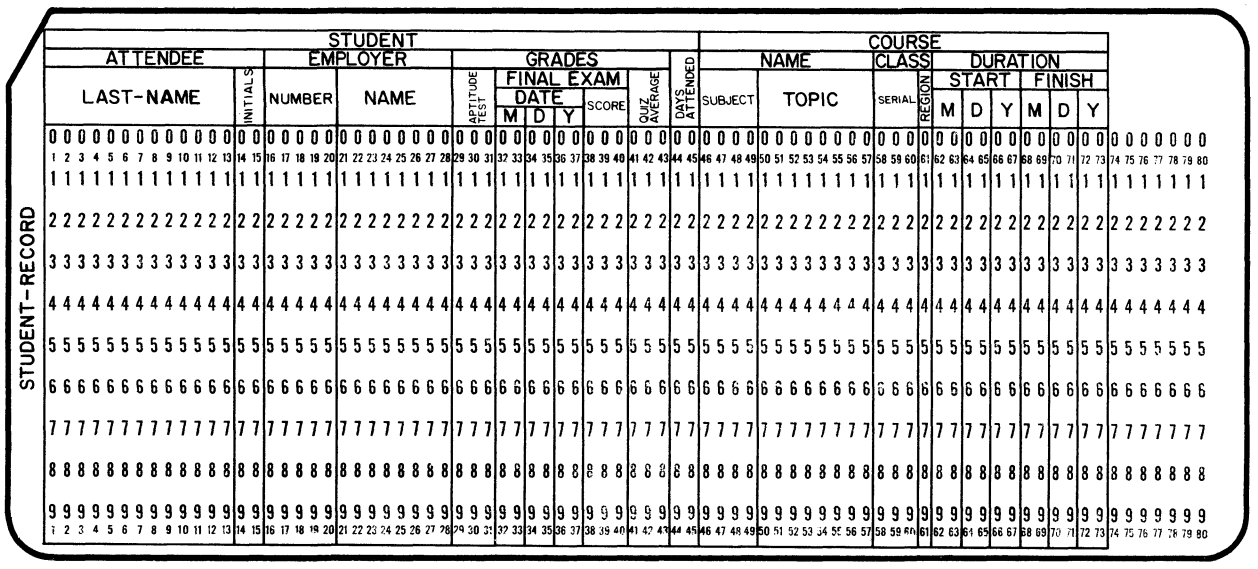
```
    02  AMOUNT
```

Now that we have entered all items at all levels, the record description is complete.

●●●

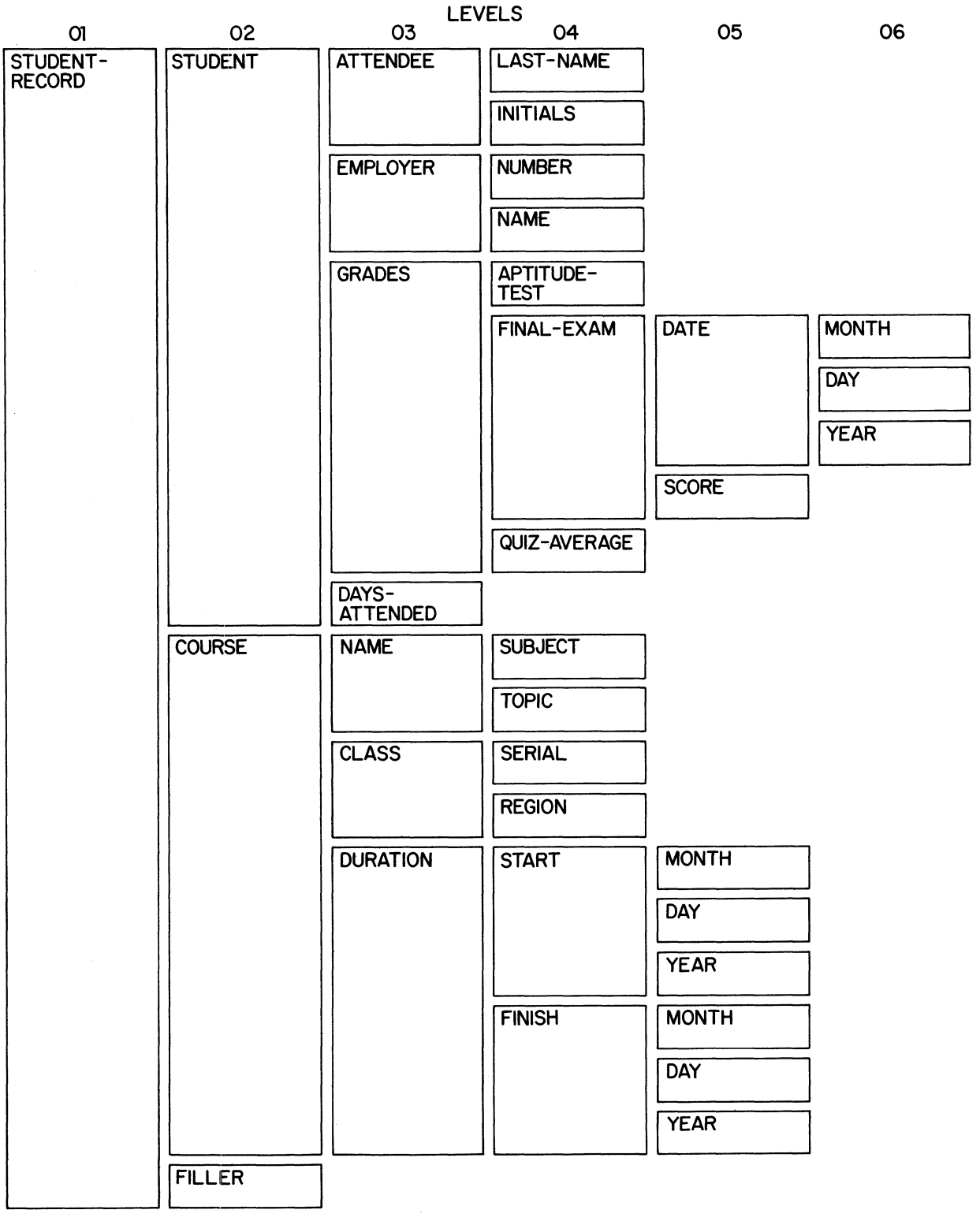
95 *The card record below offers a chance to draw up a record description with several levels of items. As in the last exercise, we are not concerned with the usage and pictures of these items; only with their levels and names. While our objective is to prepare a record description, we generally find it easier to draw a schematic diagram of the record first. By doing this, we can arrange items into their proper levels and sequence before making entries on a program sheet.*

On a piece of scratch paper, draw a schematic diagram of the record below. Notice that the last portion of the record is unused; you must consider these unused positions as an item in the record -- you must assign the appropriate level number to it, and call it FILLER.



●●●

The correct solution for this frame is printed on the next page.



Check carefully to make certain that your diagram has all of the items arranged into the same levels and sequence as shown above. If you made any errors, correct your diagram.

95 Next, enter the level numbers and names for the record you just diagrammed, on program sheets. Once again, these are not complete entries; our objective right now is just to make partial entries in the correct sequence.

●●●

```
01  STUDENT-RECORD
    02  STUDENT
        03  ATTENDEE
            04  LAST-NAME
            04  INITIALS
        03  EMPLOYER
            04  NUMBER
            04  NAME
        03  GRADES
            04  APTITUDE-TEST
            04  FINAL-EXAM
                05  DATE
                    06  MONTH
                    06  DAY
                    06  YEAR
                05  SCORE
            04  QUIZ-AVERAGE
        03  DAYS-ATTENDED
    02  COURSE
        03  NAME
            04  SUBJECT
            04  TOPIC
        03  CLASS
            04  SERIAL
            04  REGION
        03  DURATION
            04  START
                05  MONTH
                05  DAY
                05  YEAR
            04  FINISH
                05  MONTH
                05  DAY
                05  YEAR
    02  FILLER
```

97 *If you have used programming languages in which every data item must have a different name, you are probably wondering about some of the data names in the record with which we have been working. The names MONTH, DAY, and YEAR are used three times, and NAME is used twice.*

Duplicate names of this sort are permitted in COBOL. The philosophy is that you should be able to use the most reasonable names for your data items. If five of the items in a record are addresses, for instance, then by all means call each one of them ADDRESS. (The main restrictions, you will recall, are that a name cannot exceed 30 characters and that no name can be spelled the same as a reserved word.)

But when it comes to processing one of these data items, in the Procedure division, there must be a method for specifying which particular ADDRESS you want to process. In COBOL, the method is called "qualification" of names.

Reading assignment: DATA NAMES
Two ways of naming data items
Qualification of names

●●●

98 Qualification is really quite simple. In our STUDENT-RECORD, we could make unique reference to each of the NAMEs by calling one NAME OF EMPLOYER, and calling the other NAME OF _____.

●●●

COURSE

99 Names are qualified in the [Data division] [Procedure division].

●●●

ONLY in the Procedure division

Of course, when we write the Data division, we must take care to make it possible to qualify all duplicate names that we use.

100 When the two data items have the same name, {only one / both} must be qualified.

●●●

both

101 There are two main requirements that qualifiers must meet:

1. A qualifier must be the name of a group item that contains the item whose name is to be qualified.

2. A qualifier of one item's name must be different from any possible qualifier of another item with the same name.

Let's apply these requirements in the case of STUDENT-RECORD. Close to the end of that record are two items named DAY. The first is the day on which the class started; the second is the day on which the class finished. Suppose that we qualify the first DAY by referring to it as DAY OF START.

Now the second DAY must be qualified. Four ways of qualifying it are listed below, only one of which meets the requirements. Pick out the correct way, and explain why each of the other three ways is not correct.

```
DAY OF COURSE
DAY OF CLASS
DAY OF FINISH
DAY OF DURATION
```

●●●

Only DAY OF FINISH is correct. COURSE and DURATION are also possible qualifiers of the other DAY, so there would still be doubt as to which one was being referred to. CLASS does not contain DAY, so it is not a possible qualifier at all.

102 The third day in STUDENT-RECORD can be qualified by referring to it as DAY OF [DATE] [GRADES] [FINAL-EXAM] [STUDENT-RECORD] [STUDENT].

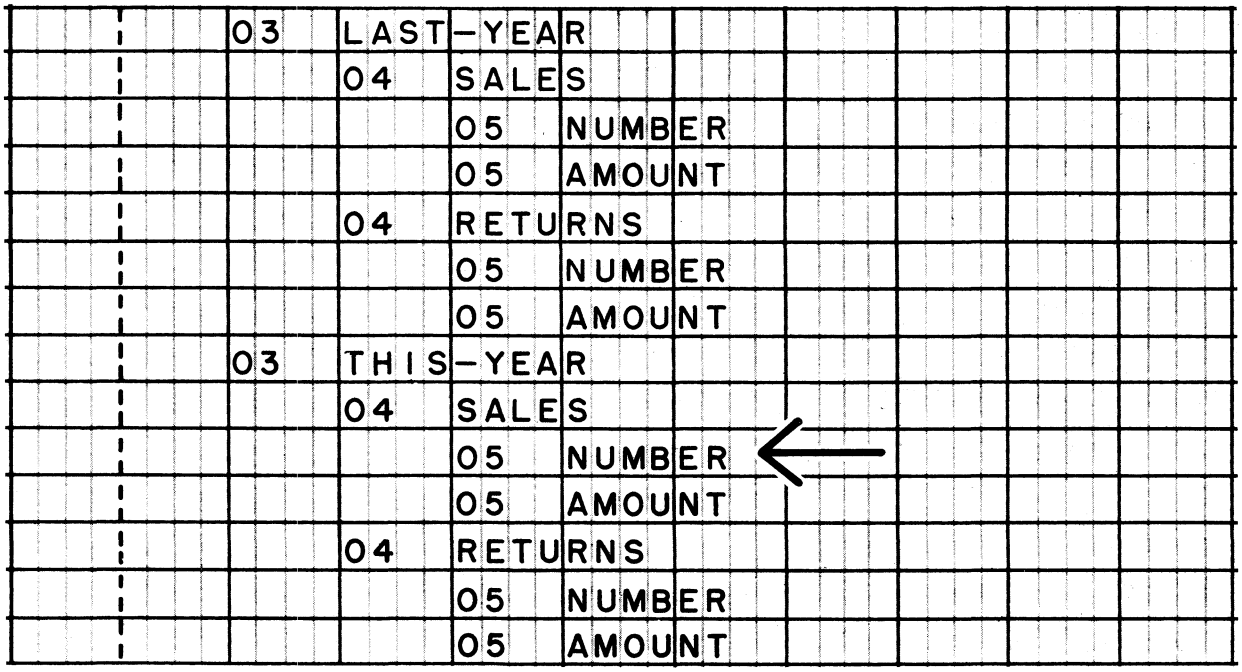
●●●

All of these, except STUDENT-RECORD, are acceptable qualifiers. *Some are better than others, though, from the standpoint of "making sense". DAY OF GRADES or DAY OF STUDENT don't make as much sense as DAY OF DATE or DAY OF FINAL-EXAM. More than one qualifier may also be used; for instance, DAY OF DATE OF FINAL-EXAM.*

103 Sometimes it is absolutely necessary to use two or more qualifiers for a name. Examine the excerpt from a record description, below.

We want to refer to the item marked by an arrow. Obviously, we cannot refer to it merely as NUMBER, as there are four NUMBERs; and we cannot refer to it as NUMBER OF SALES, as there are two of those.

We must, therefore, go one step further, and refer to it as _____.



●●●

NUMBER OF SALES IN THIS-YEAR

The words IN and OF can be used interchangeably. Use whichever word sounds "right" to you.

104 Since THIS-YEAR was the "unique" qualifier in the example above, could we have referred to the item simply as NUMBER IN THIS-YEAR?

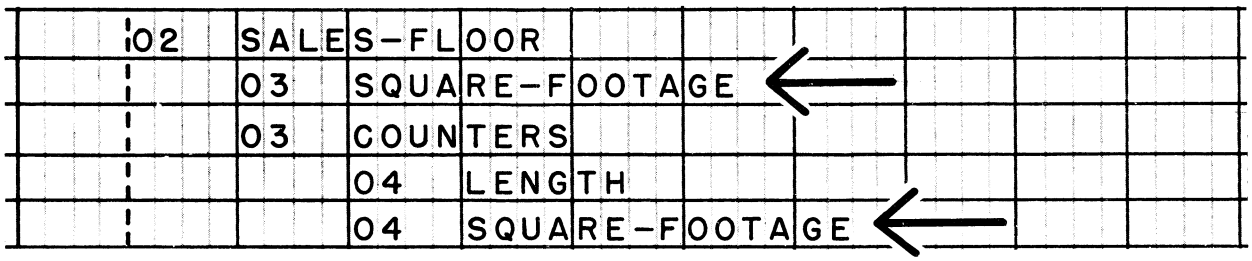
●●●

No

Make sure that you see this point. There are two NUMBERs in THIS-YEAR. Both qualifiers are definitely needed.

105 *Keep in mind that the programmer makes qualification of a name possible by the way he uses that name in the Data division. He must be careful to avoid situations in which a name cannot be uniquely qualified.*

Examine this excerpt from a record description, paying particular attention to the items marked by arrows. These items have the same name, and because of their places in the structure of this record, the name cannot be uniquely qualified.



The only possible qualifier for the first SQUARE-FOOTAGE is SALES-FLOOR. But there are two SQUARE-FOOTAGEs in SALES-FLOOR, so it is not a unique qualifier. (Note that it is incorrect to name the items as shown, even though the second SQUARE-FOOTAGE can be uniquely qualified.)

Can you think of an easy way to correct the error in this example?

●●●

Simply change one of the names; for instance, the first SQUARE-FOOTAGE might be renamed FLOOR-AREA. When this is done, no qualification is needed for any of the names in this example.

106 *To sum up, COBOL permits you to use duplicate names for data items -- provided that each name can be qualified to make it different from every other name.*

This certainly applies to the name of records, as well as smaller data items. The reference handbook states that it is permissible for two (or more) records to have the same name, but that the names of files must be unique.

It should now be clear to you that records with the same name

{must be in different files / may be in the same file} .

●●●

must be in different files

LESSON 4

107 *So far, we have been dealing with the level numbers and names of data items. In this lesson, we will add the PICTURE, VALUE, and USAGE clauses, in order to construct complete item description entries. These three clauses are the only ones needed for most items, and our emphasis will be on learning to use them correctly. (To avoid the confusion of too many facts and rules at one time, we will omit the REDEFINES and OCCURS clauses for the time being.)*

The reference handbook for the previous course explains the meanings of the more common picture characters (X, A, 9, S, V, and P), and the significance of the usage words (DISPLAY, COMPUTATIONAL, COMPUTATIONAL-1, COMPUTATIONAL-2, and COMPUTATIONAL-3). If you have forgotten what these characters and words mean, return to that handbook now, and refresh your memory before you continue with this lesson; otherwise, go on to study the format of item description entries.

Reading assignment: ITEM DESCRIPTION

●●●

108 An item description entry must contain at least a _____ and a _____, plus a period.

●●●

level number; data name (or FILLER)

109 The reserved word FILLER is used for items that

[contain no information, or blank spaces]
[contain information that will not be processed].

●●●

EITHER contain no information, or blank spaces OR contain information that will not be processed.

110 The picture of a data item may indicate several things about the item. However, a picture will never indicate

[the location of an assumed decimal point]
[the presence of an operational sign in an item]
[the data code in which the item will be stored]
[the editing that is to be done to form the item]
[the initial value of the item]
[the class of the item: numeric, alphabetic, or alphanumeric].

●●●

the data code in which the item will be stored AND the initial value of the item

The data code is given in the USAGE clause; the initial value is specified in the VALUE clause.

111 Descriptions of elementary items { must have / may have / must not have } PICTURE clauses.

●●●

must have

112 Descriptions of group items { must have / may have / must not have } PICTURE clauses.

●●●

must not have

113 VALUE clauses are allowed in item description entries only in the _____ section of the Data division.

●●●

Working-Storage

114 Descriptions of elementary items in the Working-Storage section

{must have / need not have / must not have} VALUE clauses.

●●●

need not have

115 Descriptions of group items in the Working-Storage section

{must have / need not have / must not have} VALUE clauses.

●●●

must not have

116 USAGE clauses are allowed in [elementary items] [group items] in the [File section] [Working-Storage section].

●●●

BOTH elementary items AND group items; BOTH File section AND Working-Storage section

117 The USAGE clause may be omitted if the item's usage is _____.

●●●

DISPLAY

118 The USAGE clause may also be omitted for an item in a record description, if usage has already been specified for _____.

●●●

a group item that contains this item

119 Let's apply these rules to a few sample data items, beginning with independent items. You will recall that independent items are entered in the [File section] [Working-Storage section], and that they are [elementary items] [group items].

●●●

Working-Storage section ONLY; elementary items ONLY

120 The level number of an independent item is required to be _____.

●●●

77

121 Since an independent item is an elementary item, its description must contain a _____ clause.

●●●

PICTURE

122 Suppose that we need a work area large enough to hold 20 alphanumeric characters. On a COBOL program sheet, write the item description entry that is needed to set up such an area. Name the item WORK-AREA.

●●●

```
77  WORK-AREA, PICTURE X(20).
```

The USAGE clause is omitted, because the usage of an alphanumeric item must be display; however, you might have included the clause: USAGE IS DISPLAY.

123 On the same program sheet, write the entry to define another independent item, named DIFFERENCE. This item is to contain a sign and five digits, stored in "packed decimal" (internal decimal) form.

Note: The picture character S, which you will use to indicate that this item contains an operational sign, must be written as the left-most character of the picture -- even though the sign will actually be stored in the right-most byte of the item in storage.

●●●

```
77  DIFFERENCE, PICTURE S9(5),
    COMPUTATIONAL-3.
```

Points to check: (1) The picture could also have been written S99999; however, the abbreviated form -- with a number enclosed in parentheses -- is generally used whenever a picture character is to be repeated more than four times in succession. (2) The usage word, COMPUTATIONAL-3, is mandatory in this case, to indicate that the data is packed-decimal. (3) You might also have used the words USAGE IS before COMPUTATIONAL-3.

124 *COBOL students often wonder whether to include or omit optional words in an entry -- for instance, the words USAGE IS in the entry you just wrote. The best rule is to include these words when they help the reader to understand an entry; otherwise, to omit them.*

Consider the USAGE clause. If the reader doesn't know what COMPUTATIONAL-3 means, he will not know what USAGE IS COMPUTATIONAL-3 means either. And, if he does know the meaning of COMPUTATIONAL-3, the words USAGE IS are unnecessary. Here the optional words don't clarify the meaning, so they might as well be omitted.

Take another case: the "greater than" relation test, in which it is permissible to use just the word GREATER, in place of IS GREATER THAN. But the procedural statement IF BALANCE IS GREATER THAN MINIMUM-LEVEL... is easier to read and understand than its abbreviated equivalent, IF BALANCE GREATER MINIMUM-LEVEL ... Here the optional words ought to be included.

●●●

125 *For most items, you will not be concerned about initial values. For example, when you describe the items that make up an input record, you simply want to reserve an area in storage to receive the record; specific values are going to be put into those items when the data file is read.*

In some instances, though, you may want to specify the initial value of an item. For this, of course, you would use the VALUE clause in the item description. One such instance is when you want to set up a constant; another is when it is important for a work area to have a certain value at the outset of program execution.

Keep this fact in mind: Unless you specify a value, there is no way of knowing what the initial contents of an item will be. Storage is not cleared before your object program is loaded, so you must not assume that items contain blanks or zeros at the start of a run.

On your program sheet, write the entry to define a constant whose value is 500. This number is to be stored in binary form, and named LIMIT.

Note: Binary data is always signed, so don't forget the S in the picture.

●●●

```
77  LIMIT, PICTURE S999, VALUE 500,
      COMPUTATIONAL.
```

126 *A non-numeric literal is used in the VALUE clause for alphabetic and alphanumeric items. Accordingly, you will want to enter a non-numeric literal in the next practice item.*

Write the item description entry for an alphabetic constant which is to serve as the title of a report. The contents of the item are to be DEPRECIATION SCHEDULE and the item is to be named TITLE.

●●●

```
77  TITLE, PICTURE A(21),
      VALUE 'DEPRECIATION SCHEDULE'.
```

The usage of alphabetic data must be display (BCD), so the USAGE clause may be omitted, as shown. It would also have been correct to write DISPLAY or USAGE IS DISPLAY in the entry.

127 Independent items, by definition, cannot be subdivided, nor can they be combined into groups. If items are to be combined or subdivided, they must be described as records.

The main reason for combining items into records is to make it possible to refer to a group of items by one name, and therefore, to process the entire group at one time. To illustrate, suppose that we had defined three independent items, as follows:

```
77  DEPARTMENT, PICTURE 999.
77  EMPLOYEE-NUMBER, PICTURE 9(6).
77  SHIFT, PICTURE 9.
```

Now, if the data in these three items were to be moved to some other location, say to an output record, three MOVE statements would have to be used. By combining the three items into a record, as shown below, we could move the data in the complete group item with just one MOVE statement.

```
01  EMPLOYEE-IDENTIFICATION.
    02  DEPARTMENT, PICTURE 999.
    02  EMPLOYEE-NUMBER, PICTURE 9(6).
    02  SHIFT, PICTURE 9
```

On your program sheet, write the entries that describe a record named ADDRESS, composed of STREET (20 alphanumeric characters), CITY (20 alphabetic characters), STATE (5 alphabetic characters), and ZIP-CODE (5 digits stored in external decimal --- BCD --- form, with no sign).

●●●

```
01  ADDRESS.
    02  STREET, PICTURE X(20).
    02  CITY, PICTURE A(20).
    02  STATE, PICTURE A(5).
    02  ZIP-CODE, PICTURE 9(5).
```

Points to check: (1) ADDRESS must have level number 01. (2) ADDRESS must not have a picture, since it is a group item. All of the other items must have pictures. (3) Do you have a period at the end of every entry?

128 Combining two or more items to form a record does not affect your ability to process each of the items separately; it simply adds the capability of processing the whole group as a combined item.

Also, by combining items, you can sometimes take advantage of the fact that the usage of a group item applies to all items in the group. For example, if the usage of a group item is specified as being COMPUTATIONAL, then every elementary item in that group is taken to be binary -- and there is no need to repeat the usage word. Apply this labor-saving fact to the following problem.

Write the entries needed to define a record named TOTALS, made up of MINOR-TOTAL (5 digits), MAJOR-TOTAL (7-digits), and FINAL-TOTAL (9 digits). Each elementary item is in packed decimal form, and contains a sign.

●●●

```
01  TOTALS, COMPUTATIONAL-3.
    02  MINOR-TOTAL, PICTURE S9(5).
    02  MAJOR-TOTAL, PICTURE S9(7).
    02  FINAL-TOTAL, PICTURE S9(9).
```

The idea was to write COMPUTATIONAL-3 in the description of the group item only; however, you might have written COMPUTATIONAL-3 for each elementary item in the group, in which case you could have omitted it at the 01 level.

129 Let's suppose that the record description you just wrote appears in the Working-Storage section, and you want to specify an initial value of zero for each of the three totals. Is the following an acceptable way of setting the initial values? If not, why not?

```
01  TOTALS, COMPUTATIONAL-3, VALUE ZERO.
    02  MINOR-TOTAL, PICTURE S9(5).
    02  MAJOR-TOTAL, PICTURE S9(7).
    02  FINAL-TOTAL, PICTURE S9(9).
```

●●●

No, because a VALUE clause is permitted for elementary items only. *In this case, a VALUE clause would have to be written in the description of each level-02 item.*

130 Records in the Working-Storage section are always group items, but records in the File section can be elementary items. This means that it is possible for a record description in the File section to consist of only a level-01 entry; in that case, the level-01 entry would contain a level number, a name, and

{a PICTURE clause
a VALUE clause
no descriptive clauses}

●●●

a PICTURE clause *(Every elementary item description must contain a PICTURE clause.)*

131 *Perhaps you will not have many opportunities to define a record as an elementary item -- but you should be aware that it can be done, in the File section. (The general practice, as you have learned, is to define all of the items that make up a record.) One fairly common situation in which you will probably want to define a record as an elementary item is a job in which transaction records are processed against master records, and updated master records are written out; the items that make up the output master record are the same as those that make up the input master record.*

In this situation, an input master file must be described, followed by the input master record description, in which the complete record structure would be shown. An output master file must also be described, followed by the output master record description -- but no purpose would be served by duplicating the descriptions of the items that constitute a record; so the record will be described as an elementary item whose size is equal to the total length of the input record.

Items would then be processed -- updated -- in the input area; and the completely updated record moved to the output area by a procedural statement such as MOVE MASTER-RECORD OF INPUT-FILE TO MASTER-RECORD OF OUTPUT-FILE.

Write the record description for the output MASTER-RECORD in the situation discussed above. The total length of the record is 250 alphanumeric characters.

●●●

```
01  MASTER-RECORD, PICTURE X(250).
```

132 *The final Data division entry format is the "condition name" entry. This entry closely resembles an item description entry, but instead of describing an item, its function is to assign a name to a particular value of an item. You can think of this entry as being a supplement to an item description entry; the item description entry says, in effect, "An item exists, and these are its size, usage, and other characteristics"; the condition name entry supplements this information by saying, "Here is one specific value that might be found in the item, and a name by which we can refer to the condition that exists when the item contains that value".*

Reading assignment: CONDITION-NAME

●●●

133 **A condition name is useful only if you know which item it is associated with. If you saw the following series of entries in a program, you would know that MALFUNCTION is a condition name that is associated with [SERVICE-HISTORY] [MACHINE-NUMBER] [TYPE-OF-CALL] [PREVENTIVE-MAINTENANCE] [DOWN-TIME].**

```
01  SERVICE-HISTORY.
    02  MACHINE-NUMBER, PICTURE 9(8).
    02  TYPE-OF-CALL, PICTURE 9.
        88  PREVENTIVE-MAINTENANCE, VALUE 7.
        88  MALFUNCTION, VALUE 4.
    02  DOWN-TIME, PICTURE 999V9.
```

●●●

TYPE-OF-CALL

Condition name entries are required to <u>follow</u> the elementary item with which they are associated. In this example, two condition names are associated with TYPE-OF-CALL.

134 **Again referring to the example in the previous frame, we would say that PREVENTIVE-MAINTENANCE is the name of the condition that will exist during the execution of the program, whenever the value of _____ is _____.**

●●●

TYPE-OF-CALL; 7

135 Condition names are used in IF sentences, and if they are used properly, they make the sentences a lot more meaningful to the reader.

Having defined a condition name as shown below, the programmer can write a sentence such as: IF HIGHEST-PRIORITY, GO TO FILL-ORDER-AT-ONCE. Without the condition name, the programmer would have had to write: IF PRIORITY-CODE = 'G', GO TO FILL-ORDER-AT-ONCE.

```
    03  PRIORITY-CODE, PICTURE X.
        88  HIGHEST-PRIORITY, VALUE 'G'.
```

Condition names are defined in the _____ division, and used in the _____ division.

●●●

Data; Procedure

136 *As he writes the Data division, the programmer must try to anticipate the need for condition names in the Procedure division. As a general rule, condition name entries should be written for any item that will assume a limited set of predictable values -- for instance, a record identification code.*

Write the item description entry and condition name entries for the following item: a level-02 item named MARITAL-STATUS containing a single digit stored in BCD form (usage is DISPLAY); if the digit is 0, it signifies that the person is SINGLE; 1 signifies MARRIED; 2 means DIVORCED; and 3 means WIDOWED.

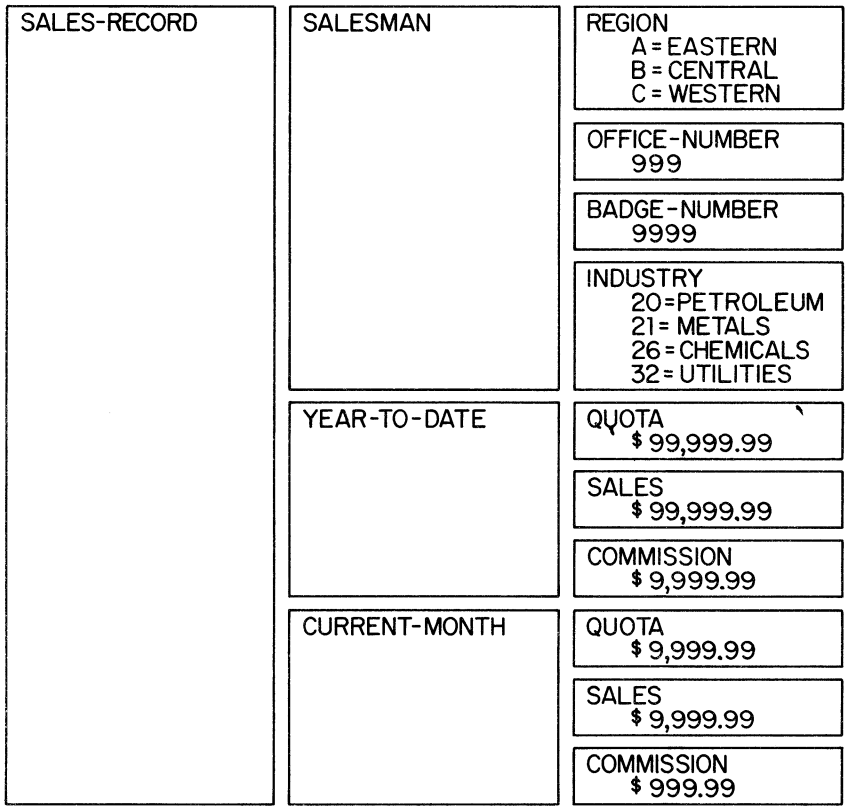
●●●

```
    02  MARITAL-STATUS, PICTURE 9.
        88  SINGLE, VALUE 0.
        88  MARRIED, VALUE 1.
        88  DIVORCED, VALUE 2.
        88  WIDOWED, VALUE 3.
```

Points to check: (1) Make sure that you have five separate entries, each terminated by a period. (2) Every condition name entry must begin with level number 88. (3) You could have used the figurative constant ZERO in place of the literal 0.

137 *This lesson has given you some practice in writing entries for individual items and small groups of items. To conclude the lesson, take a new program sheet and write the complete record description for the record diagrammed below.*

All items in the SALESMAN group are in BCD (DISPLAY) mode; all other items are packed decimal (COMPUTATIONAL-3). Actual values are shown for REGION and INDUSTRY -- write condition name entries for these values; maximum values are shown for the remaining items. Note that the dollar values are not actually punctuated with symbols; for these items, the values let you know where the assumed decimal points are located. All packed decimal items have operational signs.



●●●

The correct solution for this frame is printed on the next page.

```
01  SALES-RECORD.
    02  SALESMAN.
        03  REGION, PICTURE A.
            88  EASTERN, VALUE 'A'.
            88  CENTRAL, VALUE 'B'.
            88  WESTERN, VALUE 'C'.
        03  OFFICE-NUMBER, PICTURE 999.
        03  BADGE-NUMBER, PICTURE 9999.
        03  INDUSTRY, PICTURE 99.
            88  PETROLEUM, VALUE 20.
            88  METALS, VALUE 21.
            88  CHEMICALS, VALUE 26.
            88  UTILITIES, VALUE 32.
    02  YEAR-TO-DATE, COMPUTATIONAL-3.
        03  QUOTA, PICTURE S9(5)V99.
        03  SALES, PICTURE S9(5)V99.
        03  COMMISSION, PICTURE S999V99.
    02  CURRENT-MONTH, COMPUTATIONAL-3.
        03  QUOTA, PICTURE S9999V99.
        03  SALES, PICTURE S9999V99.
        03  COMMISSION, PICTURE S999V99.
```

Points to check: (1) DISPLAY usage could have been specified for the SALESMAN group; however, DISPLAY is assumed when no usage is specified. (2) The picture of REGION can be either A or X. (3) The literals in the condition name entries for REGION must be non-numeric -- enclosed in quotation marks. (4) If you did not write COMPUTATIONAL-3 at the 02 level for YEAR-TO-DATE and CURRENT-MONTH, it would have been necessary to write COMPUTATIONAL-3 for every 03 item in those groups; note that it is not sufficient to write COMPUTATIONAL-3 for YEAR-TO-DATE only, since that applies only to the items in that group -- so COMPUTATIONAL-3 must be written for CURRENT-MONTH also. (5) Did you remember to use the picture character V to show the location of assumed decimal points? (6) Because QUOTA, SALES, and COMMISSION are duplicated names, they will have to be qualified when they are used in the Procedure division; for instance, SALES IN YEAR-TO-DATE and SALES IN CURRENT-MONTH.

LESSON 5

138 *The Procedure division is the most loosely structured of the four divisions. The division header is the only fixed entry that you are required to use in every program -- and from that point on, you're on your own to construct whatever paragraphs you need and to arrange them in any sequence that does the job.*

We will begin this lesson with a quick look at the overall format of the division.

Reading assignment: PROCEDURE DIVISION

●●●

139 The Procedure division is required to contain at least

{one paragraph / two paragraphs} .

●●●

one paragraph

140 The names of paragraphs (procedure names) are

{reserved words / programmer-supplied names} .

●●●

programmer-supplied names

141 Suppose that we have a very simple program that contains only one paragraph in the Procedure division. Must that paragraph have a name?

●●●

Yes

142 How many sentences must each paragraph contain?

●●●

One or more

143 What is the difference between a sentence and a statement?

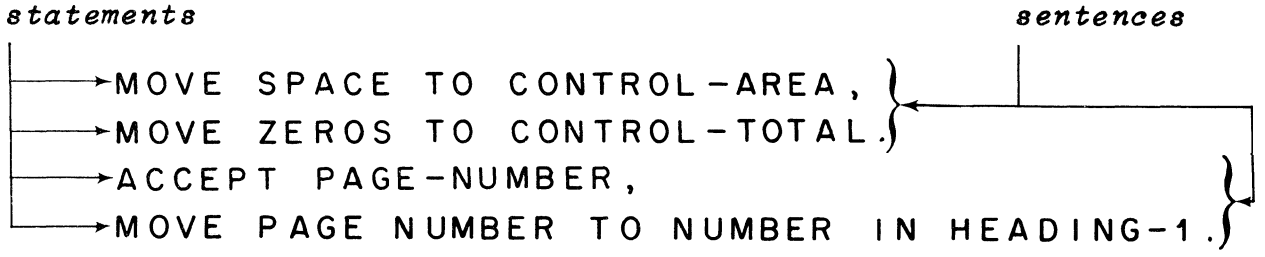
●●●

A sentence is an entry; therefore, it must be terminated by a period. A statement specifies an action to be taken, and is found within a sentence. Each sentence contains at least one statement, but it may contain more than one.

144 In the sample paragraph below, pick out the sentences and the statements.

```
INITIALIZE.
    MOVE SPACES TO CONTROL-AREA,
    MOVE ZEROS TO CONTROL-TOTAL.
    ACCEPT PAGE-NUMBER,
    MOVE PAGE-NUMBER TO NUMBER IN HEADING-1.
```

●●●



In most cases, it does not matter how many statements are written in a sentence. All four of the statements above might have been combined into one sentence, since that would have satisfied the requirement for at least one sentence per paragraph. (Another way of stating this rule is that every paragraph must end in a period.) On the other hand, the statements might have been written as four separate sentences -- a method preferred by many programmers because it simplifies the insertion and deletion of statements.

145 *Each type of statement has a specific format. The remaining pages of the reference handbook give the formats of the most commonly used statements. In this lesson, and in the lessons that follow, you will study all of the formats given in the reference handbook.*

In the handbook, the various verbs are arranged in alphabetical order. Some verbs have more than one format, and are presented on more than one page; for instance, ADD (1) and ADD (2). The reading assignment will specify exactly which format you are to study.

You will not study the statements in alphabetical order. Instead, as you work on a programming problem, you will be assigned to study only the statements that apply to the problem. And your work will involve all four divisions, rather than just the Procedure division -- so that you will get a good idea of the relationship of procedural statements to the descriptions of data items and files, assignments of input-output devices, and so on.

None of the procedural words should come as surprises. You were introduced to them all in the previous course. But don't make the mistake of thinking that this time you must memorize all of the details about each entry -- just try to pick up only as much information as you need to write the required entries for the problem that you are working on.

The amount of practice that you will get with each verb will be quite limited. There are literally hundreds of ways in which most statements can be used; if we explored every one of them, this course would never end! So, you will get practice in using the statements in one or two typical ways, in simple problems.

●●●

146 *Let's start with a really simple problem, involving three statements. This problem doesn't represent a practical computer application; on the contrary, it is just about the simplest procedure imaginable.*

The problem is to print the data punched in a single card. The steps in the procedure are (1) to obtain the data from an input device, (2) to transfer the data to an output device, and (3) to stop the run. Because only one card is involved, we will treat it as low-volume input, not as an input file; likewise, only one line is to be printed, so we will treat it as low-volume output, rather than as an output file. The input-output verbs we will use, therefore, are ACCEPT and DISPLAY, rather than READ and WRITE.

●●●

147 *ACCEPT and DISPLAY -- and STOP, also -- have more than one format. You are to read about only the first format of each of these verbs at this time.*

Reading assignment: ACCEPT (1)
DISPLAY (1)
STOP (1)

●●●

148 An ACCEPT statement obtains up to 80 characters of data, and moves them into a working-storage area. Suppose that, for our problem, an 80-character item named CARD-DATA has already been defined. Which of the entries below is the correct way to get the data from our card into CARD-DATA?

```
ACCEPT DATA, MOVE INTO CARD-DATA.

ACCEPT; MOVE TO CARD-DATA.

ACCEPT CARD-DATA.
```

●●●

ACCEPT CARD-DATA.

149 Now that we have the card's data in working storage, we want to DISPLAY the data. In order to display the data,

{we must first move it from CARD-DATA to an output area
we need only to write DISPLAY CARD-DATA
we need to convert the data to DISPLAY usage} .

●●●

we need only to write DISPLAY CARD-DATA

150 Our last step is to terminate the execution of the program. The statement that we must write to accomplish this is _____.

●●●

STOP RUN

151 *Let's now write the complete COBOL program -- not just the Procedure division -- for this problem. This will give you an opportunity to tie together information that you studied in previous lessons. Feel free to refer to any part of the reference handbook.*

Take a new program sheet, and write the Identification division. Make only the required entries for the division. The name of our program is ONECARD.

●●●

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
    'ONECARD'.
```

152 Next, the Environment division. In this division, the Configuration section is always required, while the Input-Output section is optional.

For our problem, the Input-Output Section {must be included / must be omitted} because _____.

●●●

must be omitted, because there are no input or output files

153 Write the Environment division of our program. Enter just IBM-360 in the Source-Computer and Object-Computer paragraphs, omitting the model numbers.

●●●

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.
    IBM-360.
OBJECT-COMPUTER.
    IBM-360.
```

154 In the Data division, we must define the item we called CARD-DATA. To do this, we need [a File section] [a Working-Storage section].

●●●

ONLY a Working-Storage section

We must omit the File section because we have no input or output files.

155 We must describe CARD-DATA as an 80-character $\left\{\begin{matrix}\text{independent item}\\\text{record}\end{matrix}\right\}$.

●●●

independent item

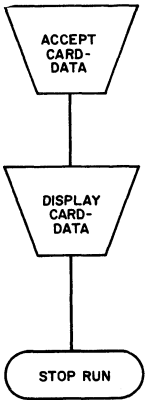
Items in working storage are defined as records only if they are subdivided into smaller items.

Write the Data division of the program. CARD-DATA is to accommodate 80 alphanumeric characters.

●●●

```
DATA DIVISION.
WORKING-STORAGE SECTION.
77  CARD-DATA, PICTURE X(80).
```

157 Finally, write the Procedure division. To refresh your memory, here is a procedure flowchart for our little program.

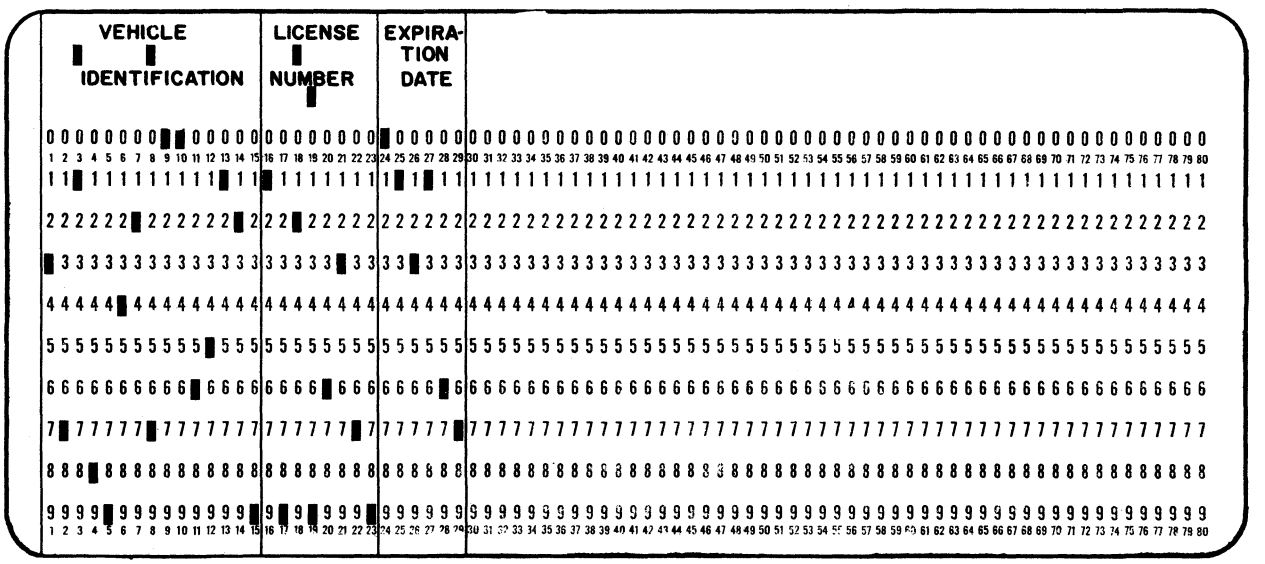


●●●

```
PROCEDURE DIVISION.
PROCESS-DATA.
    ACCEPT CARD-DATA.
    DISPLAY CARD-DATA.
    STOP RUN.
```

Did you remember that a paragraph name is required? If you forgot, correct your work. The name does not have to be PROCESS-DATA, of course.

158 *Our second problem is a slight modification of the first one. Instead of merely printing out the data in a card, in this problem we will insert spaces between the fields on the output line. Let's suppose that the card looks like this:*



Our first program would have printed a solid line of characters:

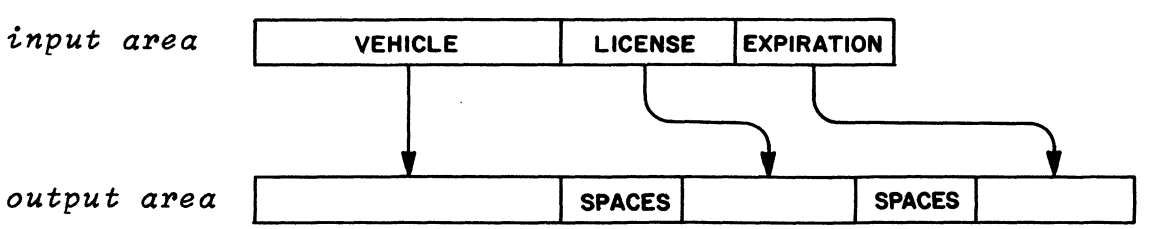
37A8942G0065129 19BR6379 013167

This time we will leave five spaces between fields:

37A8942G0065129 19BR6379 013167

●●●

159 *The logic of this job is significantly different from the logic of the first job. Before, a single area in working storage served for both input and output. Now, we will set up two areas, and move the data items from one area to the other.*



Before, we treated the data as a single (independent) item. Now, we will treat it as a record, subdivided into three smaller items.

●●●

160 *The Identification and Environment divisions that you wrote for the first problem are suitable for the second problem, but we will need different Data and Procedure divisions. We will define our data items first.*

On a fresh program sheet, write the division and section headers for the Data division. (As before, we will use only the Working-Storage section.) Then, write the record description for the input record. Name the record CARD-DATA. Look back at the preceding frames to see the card format.

Note that we will define only the first 29 positions of the card, and ignore the blank positions (30-80). We are able to do this because the ACCEPT statement lets us obtain only as many characters as we want, beginning with position 1 (up to a maximum of 80 characters).

●●●

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01  CARD-DATA.
    02  VEHICLE, PICTURE X(15).
    02  LICENSE, PICTURE X(8).
    02  EXPIRATION, PICTURE 9(6).
```

161 **Next, write the record description of the output record. Name the record OUTPUT-LINE. Name the three items that will receive data AREA1, AREA2, and AREA3. Be sure to define filler areas between the data areas, and to assign initial values to the filler (each filler area is to contain 5 spaces).**

●●●

```
01  OUTPUT-LINE.
    02  AREA1, PICTURE X(15).
    02  FILLER, PICTURE X(5), VALUE SPACES.
    02  AREA2, PICTURE X(8).
    02  FILLER, PICTURE X(5), VALUE SPACES.
    02  AREA3, PICTURE 9(6).
```

162 The logic of our procedure will be to accept the input record, then to move data from the input record to the output record, then to display the output record, and finally to stop the run. Which one of the ACCEPT statements below will properly handle the first step of the procedure?

```
    ACCEPT CARD-DATA.

    ACCEPT VEHICLE, LICENSE, EXPIRATION.

    ACCEPT CARD-DATA, VEHICLE, LICENSE,
        EXPIRATION.
```

●●●

ACCEPT CARD-DATA.

Only one data name can be written in an ACCEPT statement, so it must be the name of the record as a whole.

163 Which DISPLAY statement is the correct one for our problem?

```
    DISPLAY AREA1, AREA2, AREA3.

    DISPLAY AREA1, FILLER, AREA2, FILLER,
        AREA3.

    DISPLAY OUTPUT-LINE.
```

●●●

DISPLAY OUTPUT-LINE.

This statement will cause the entire output record to be displayed, including the spaces between data characters. The first choice, above, is a legitimate DISPLAY statement -- but it would cause the three data items to be displayed with no spaces between them; thereby undoing our careful insertion of filler items into the record. The second DISPLAY statement is illegal, because the word FILLER cannot be used in procedural statements.

164 *The ACCEPT and DISPLAY statements are just as simple for this problem as they were for the first problem. Before we display the output line, though, we will have to move the input data to the output area.*

Reading assignment: MOVE

●●●

165 Which sentence is correct for our problem?

```
    MOVE VEHICLE TO AREA1;
    MOVE LICENSE TO AREA2;
    MOVE EXPIRATION TO AREA3.
```

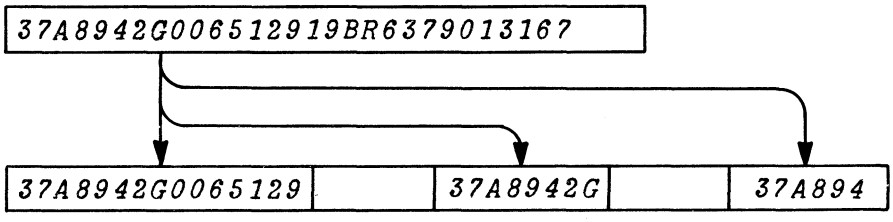
```
    MOVE CARD-DATA TO AREA1, AREA2, AREA3.
```

```
    MOVE CARD-DATA TO OUTPUT-LINE.
```

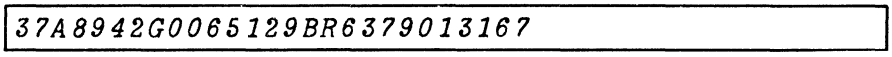
●●●

```
MOVE VEHICLE TO AREA1;
MOVE LICENSE TO AREA2;
MOVE EXPIRATION TO AREA3.
```

The second sentence is incorrect for a couple of reasons. For one, it tries to move data from a group item (CARD-DATA) to an elementary numeric item (AREA3), which is a violation of the rules; if you tried to do this in a program, the compiler would detect your error and issue a diagnostic message informing you of your goof. But suppose that the picture of AREA3 had been X(6) instead of 9(6); now the MOVE statement would be legal, but it would not give the desired result. It would cause the same data to be moved to each of the receiving items, as diagrammed here:



The third sentence specifies a group item as the receiving item, which is permissible, but which causes the boundaries of elementary items within the group to be ignored. The result would be:

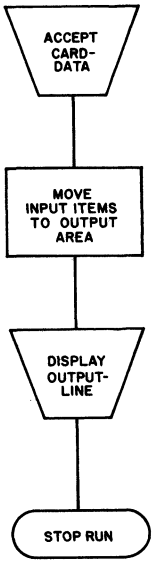


166 *It is often very desirable to write comments within a program to explain what the program is supposed to do. You have seen that a Remarks paragraph may be included in the Identification division, allowing you to explain the purpose of the program. No remarks are permitted in the Environment and Data division. In the Procedure division, you may make any number of NOTE entries, to explain the processing that is being done.*

Reading assignment: NOTE (1)
NOTE (2)

●●●

167 Using the same program sheet on which you wrote the Data division, now write the complete Procedure division for the second problem. Begin the division with a NOTE paragraph that briefly explains the procedure. Then write the procedural statements that correspond to the flow-chart at the right.



●●●

```
PROCEDURE DIVISION.
EXPLANATION.
    NOTE THIS PROCEDURE READS ONE CARD,
    MOVES THE CARD DATA TO AN OUTPUT AREA,
    PRINTS A SINGLE LINE, AND STOPS.
START.
    ACCEPT CARD-DATA.
    MOVE VEHICLE TO AREA1.
    MOVE LICENSE TO AREA2.
    MOVE EXPIRATION TO AREA3.
    DISPLAY OUTPUT-LINE.
    STOP RUN.
```

168 *The third problem of this lesson calls for you to write a short program on your own. The problem involves accepting, moving, and displaying a few characters of data. This time, though, the input will come from the console keyboard, rather than from a card; and the output is to be punched into a card, rather than printed. Before you start on the problem, study the slightly different formats of ACCEPT and DISPLAY which you will use in your program.*

Reading assignment: ACCEPT (2)
DISPLAY (3)

●●●

169 Here is the problem: A console operator will type 15 characters, which may be both letters and digits, on the console keyboard. The first 10 of those characters are to be punched into columns 25-34 of a card; the remaining 5 characters are to be punched into columns 1-5 of the same card. Only one card is to be punched; then the run is to be stopped.

Write the Data and Procedure divisions for this problem. (The Identification and Environment divisions required for this problem are the same as those you wrote for the first problem.) Invent your own names for data items and procedures.

●●●

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01  INPUT-DATA.
    02  FIELD-1, PICTURE X(10).
    02  FIELD-2, PICTURE X(5).
01  OUTPUT-DATA.
    02  FIELD-A, PICTURE X(5).
    02  FILLER, PICTURE X(19), VALUE SPACES.
    02  FIELD-B, PICTURE X(10).

PROCEDURE DIVISION.
BEGIN.
    ACCEPT INPUT-DATA FROM CONSOLE.
    MOVE FIELD-1 TO FIELD-B.
    MOVE FIELD-2 TO FIELD-A.
    DISPLAY OUTPUT-DATA UPON SYSPUNCH.
    STOP RUN.
```

LESSON 6

170 *Most computer problems involve more operations than the problems you worked in Lesson 5. To begin with, most problems involve iteration or "looping" -- in which a series of steps is executed over and over until the result is obtained or until the end of a file is reached. Logical decisions must be made to determine when to exit from the loop. Also, most problems involve at least a small amount of arithmetic.*

In this lesson, you will study the arithmetic verbs, as well as sequence control words - IF, GO TO, and PERFORM. As in the last lesson, you will read about these statements when they are needed for a problem. Needless to say, you may also be called upon to write statements which you studied in the last lesson.

●●●

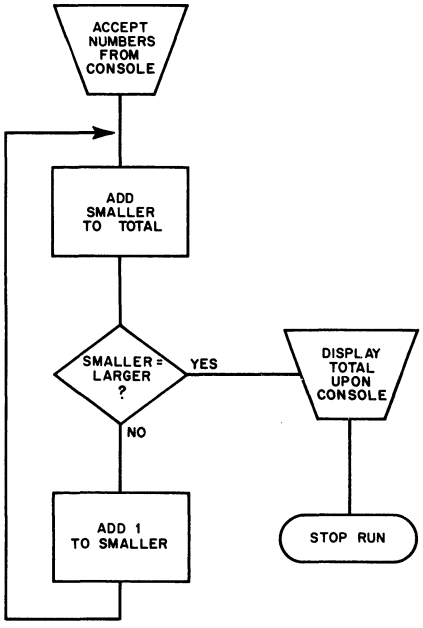
171 *Throughout this lesson, we will program just one problem, although we will twist it around a bit and add things to it as we go along. Here is the basic problem: Given two numbers, the first smaller than the second, we want to get the sum of all the whole numbers between the first number and the second number, inclusive. For example, if the numbers are 45 and 52, we want to know what 45 + 46 + 47 + 48 + 49 + 50 + 51 + 52 equals. To make the problem as straightforward as possible, we will say that both numbers must be unsigned (therefore regarded as positive). The numbers are to be entered by way of the console, and the sum is to be typed out on the console printer.*

●●●

172 *Let's develop a possible solution for this problem. We will get the two numbers into storage by using an ACCEPT FROM CONSOLE statement. Now we have to develop the sum. We can do that by defining a work area with an initial value of zero, and adding the smaller number into the area; after that, we can repeatedly increase the smaller number by 1, and repeatedly add it to the total, until the smaller number is equal to the larger number. At this point, we have the sum we wanted; we can display it upon the console, and stop the run.*

●●●

173 *This flowchart will help you to visualize the procedure.*



Of course, this is not the only possible solution to this problem, and a little later in this lesson, we will look at another, somewhat different, solution. For the time being, though, let's stick with this procedure, and write the COBOL entries for it.

Before you proceed to the next frame, take a minute to satisfy yourself that this procedure really will do the job. Take any pair of numbers, the simpler the better, say 2 and 5; then go through the procedure step by step, to see how the total of 2 + 3 + 4 + 5 is developed. Remember that the initial value of the total is zero.

●●●

174 *Some new procedural words and formats are involved in this problem.*

Reading assignment: ADD (1)
IF (1)
IF (2)
TEST CONDITIONS
GO TO
DISPLAY (2)

●●●

175 As with our previous problems, the first step is to define the data items in the Data division. On a program sheet, write the necessary division and section headers, and then write the descriptions of:

(1) the work area in which the sum will be developed, named TOTAL, 8 digits in length; initial value is zero.

(2) the input record, named NUMBERS; subdivided into SMALLER and LARGER, each 4 digits in length.

●●●

```
DATA DIVISION.
WORKING-STORAGE SECTION.
77  TOTAL, PICTURE 9(8), VALUE ZERO.
01  NUMBERS.
    02  SMALLER, PICTURE 9999.
    02  LARGER, PICTURE 9999.
```

176 Next, begin the Procedure division. Write the division header. Name the first paragraph OBTAIN-DATA, and in it write the entry required to accept the input numbers from the console.

The console operator will have been instructed to type two four-digit numbers, with no spaces, and with the smaller number first. So, if the input values are 45 and 52, the operator will type 00450052.

●●●

```
PROCEDURE DIVISION.
OBTAIN-DATA.
    ACCEPT NUMBERS FROM CONSOLE.
```

177 At this point, we must start a new paragraph, because we want to be able to branch to the following statement. (Refer to the flowchart of the solution in an earlier frame.)

Name the new paragraph COMPUTE-SUM; then write the first entry of the paragraph -- to add SMALLER to TOTAL.

●●●

```
COMPUTE-SUM.
    ADD SMALLER TO TOTAL.
```

178 Now write the IF sentence to display the total and stop the run, if SMALLER is equal to LARGER.

●●●

```
    IF SMALLER IS EQUAL TO LARGER,
        DISPLAY TOTAL UPON CONSOLE,
        STOP RUN.
```

179 Finally, write the statements to add 1 to SMALLER, and to branch back to the beginning of this paragraph. These will complete the program.

●●●

```
    ADD 1 TO SMALLER.
    GO TO COMPUTE-SUM.
```

180 Since the format of IF sentences is fairly flexible, the decision in this procedure might have been programmed in several different ways. Consider the three ways written below. For each one, figure out whether or not it would be correct for our problem.

It will probably help you to refer to the flowchart of the job (in an earlier frame). If you are not certain how to go about evaluating IF sentences, it might be helpful to re-read the section on FLOW OF CONTROL in the reference handbook for the previous course.

```
    IF SMALLER IS EQUAL TO LARGER,
        DISPLAY TOTAL UPON CONSOLE,
        STOP RUN;
    OTHERWISE,
        ADD 1 TO SMALLER,
        GO TO COMPUTE-SUM.
```

```
    IF SMALLER IS NOT EQUAL TO LARGER,
        ADD 1 TO SMALLER,
        GO TO COMPUTE-SUM;
    OTHERWISE,
        DISPLAY TOTAL UPON CONSOLE,
        STOP RUN.
```

```
    IF SMALLER IS NOT EQUAL TO LARGER,
        ADD 1 TO SMALLER,
        GO TO COMPUTE-SUM.
    DISPLAY TOTAL UPON CONSOLE.
    STOP RUN.
```

●●●

All three of these ways of using the IF statement would get our job done correctly.

181 *Our first solution to the "sum-of-all-integers-between-two-numbers" problem is not exactly the most efficient way of going about it. If the numbers happened to be 0000 and 9999, we would add to the total 10,000 times, add 1 to the smaller number 9,999 times, and test whether the numbers were equal 10,000 times!*

There is an easier way. If we knew (1) how many numbers are in the series, and (2) the average value of the numbers; we could simply multiply the number of numbers by the average, and that would give us the total! For instance, suppose that the numbers are 1 and 5; there are five numbers and their average value is 3. Multiplying 5 times 3 gives 15, which is equal to 1 + 2 + 3 + 4 + 5.

This method works for whatever numbers happen to be the input. If the input numbers are 45 and 52, we can find out the number of numbers in the series by subtracting the smaller number from the larger, and adding 1; 52 - 45 + 1 = 8. We can get the average value by adding the two numbers and dividing their sum by 2; (45 + 52) / 2 = 48.5. (Note that we have to allow for one decimal place in the average, in the event that the sum of the two numbers is odd.) Now we can get the sum of all numbers in the series; 8 x 48.5 = 388.

●●●

182 *A little later in this lesson, you will be writing the COBOL arithmetic statements to obtain the desired result by this method, so take a moment to study the outline below. Make certain that you understand what operations are to be performed, and what intermediate results are to be obtained.*

A. To calculate the number of numbers in the series --
1. Subtract the smaller number from the larger, to get the difference;
2. Add 1 to the difference, to get the number of numbers.

B. To calculate the average value of the numbers in the series --
1. Add the smaller and the larger numbers, to get a temporary sum;
2. Divide the temporary sum by 2, to get the average.

C. To calculate the total (the sum of all numbers between two numbers) --
1. Multiply the number of numbers by the average.

This program {will / will not} involve a "loop", as the previous program did.

●●●

will not

183 *The problem, as you can see, requires all four arithmetic operations -- addition, subtraction, multiplication, and division. Study the formats of the arithmetic verbs, and decide which formats you would use for the steps of calculating the total. Pay particular attention to the fact that there are two formats for each verb; in each case, the second format contains the word GIVING. Learn the difference between arithmetic statements with and without the GIVING clause.*

Reading assignment:
ADD (2)
DIVIDE (1)
DIVIDE (2)
MULTIPLY (1)
MULTIPLY (2)
SUBTRACT (1)
SUBTRACT (2)

●●●

184 Now take a COBOL program sheet, and write the entries needed to calculate the TOTAL by the method outlined in an earlier frame. Assume that the input area is called NUMBERS, and contains SMALLER and LARGER, each four digits long. The output item is named TOTAL. These items are already defined.

Take it from this point. Supply the statements that would fit between ACCEPT NUMBERS FROM CONSOLE and DISPLAY TOTAL UPON CONSOLE in the program. Define whatever working storage items you need to calculate intermediate results. One restraint: write your calculations in such a way that the original values of SMALLER and LARGER are not changed.

●●●

```
77  DIFFERENCE, PICTURE 9(4).
77  INTEGERS, PICTURE 9(5).
77  TEMP-SUM, PICTURE 9(5).
77  AVERAGE, PICTURE 9(4)V9.
```

```
    SUBTRACT SMALLER FROM LARGER, GIVING
        DIFFERENCE.
    ADD 1 TO DIFFERENCE, GIVING INTEGERS.
    ADD SMALLER, LARGER, GIVING TEMP-SUM.
    DIVIDE 2 INTO TEMP-SUM, GIVING AVERAGE.
    MULTIPLY INTEGERS BY AVERAGE, GIVING
        TOTAL.
```

185 *The statements for calculating the total may have been written in several different ways. Even though your work is not exactly like the answer printed in the preceding frame, the printed answer should help you in checking your own work. Also, make certain that you understand every entry in the printed answer.*

Points to check in your work: (1) Are your data items large enough? The item that will hold the number of numbers must be five digits long to allow for the extreme case in which the numbers are 0000 and 9999. The "temporary sum" item must also be five digits, to hold the sum of a pair of numbers such as 5000 and 6000. The average must be able to contain five digits, including one decimal. The items could be larger than this, but must not be smaller.

(2) It is permissible for items to do "double duty". For instance, the item named INTEGERS in the printed answer could have been used to hold both the difference between the input numbers, and the total number of numbers. If it were used in that way, the following procedural entries would be used:

```
SUBTRACT SMALLER FROM LARGER,
    GIVING INTEGERS.
ADD 1 TO INTEGERS.
```

You might also have used AVERAGE for both the temporary sum and the average value of the numbers, provided that its picture were 9(5)V9. These procedural entries could then be used:

```
ADD SMALLER, LARGER, GIVING AVERAGE.
DIVIDE 2 INTO AVERAGE, GIVING AVERAGE.
```

Statements like these are correct COBOL, but sometimes seem like double-talk to a reader.

(3) Above all, make sure that you have adhered closely to the formats given in the reference handbook. Beginners sometimes get the mistaken impression that they can make up any formats they please, just as long as they use reserved words, and come up with incorrect statements like DIVIDE A BY B (should be DIVIDE B INTO A), or MULTIPLY A TIMES B (should be MULTIPLY A BY B).

●●●

186 *The little series of statements which you have written to calculate the total can be thought of as a subroutine. I'll grant you that the title "subroutine" is a bit pretentious in this instance -- but the idea that you could treat these statements as a subroutine is an important one.*

The idea goes like this: A program can be thought of as a set of subroutines. Each subroutine (some people prefer to say "program module") is a series of statements that produce a certain result. Once a subroutine has been written, it can be "plugged into" other programs where the same result is desired.

In COBOL, it is especially easy to "plug in" subroutines. We use the verb PERFORM.

Reading assignment: PERFORM (1)
PERFORM (2)

●●●

187 What must we do in order to use our total-calculation entries as a subroutine? First, we must change our work from just "a series of entries" into a "procedure". We do that simply by adding a paragraph name at the start of our entries, like this:

```
TOTAL-CALCULATION.
    SUBTRACT SMALLER FROM LARGER, GIVING
        DIFFERENCE.
    ADD 1 TO DIFFERENCE, GIVING INTEGERS.
    ADD SMALLER LARGER GIVING TEMP
```

Second, we must insert our procedure into a program -- and here we have a choice of two ways. One way would simply be to copy the procedure at the appropriate point or points in the program. Another way is to add the procedure to the end of the program, and PERFORM the procedure at the appropriate point or points.

What statement would you write in order to PERFORM this procedure?

●●●

```
    PERFORM TOTAL-CALCULATION.
```

188 *Although you will find many uses for the basic arithmetic verbs (ADD, SUBTRACT, MULTIPLY, and DIVIDE) you may prefer to use the all-purpose arithmetic verb, COMPUTE, much of the time. Among the advantages of using COMPUTE are that you can specify more than one operation in a statement, and that you do not have to define intermediate work areas.*

Reading assignment: COMPUTE
ARITHMETIC EXPRESSIONS

●●●

189 Two COMPUTE statements can serve the same purpose as the five arithmetic statements you wrote earlier. First, write the COMPUTE statement to find the AVERAGE of the two input numbers, SMALLER and LARGER. (Your earlier data descriptions of these items are just as appropriate for COMPUTE as for the other arithmetic verbs.)

●●●

```
       COMPUTE AVERAGE = (SMALLER + LARGER) / 2.
```

Points to check: (1) Parentheses are necessary to cause the addition to be done before the division. If the parentheses were omitted in the solution above, the result would be equal to the SMALLER plus one-half of the LARGER. (2) Did you leave spaces before and after the equal sign, the plus sign, and the divided-by sign?

190 Next, write a statement that computes the TOTAL. This statement should subtract the SMALLER from the LARGER, and add 1, to get the number of numbers in the series, and then multiply that by the AVERAGE.

●●●

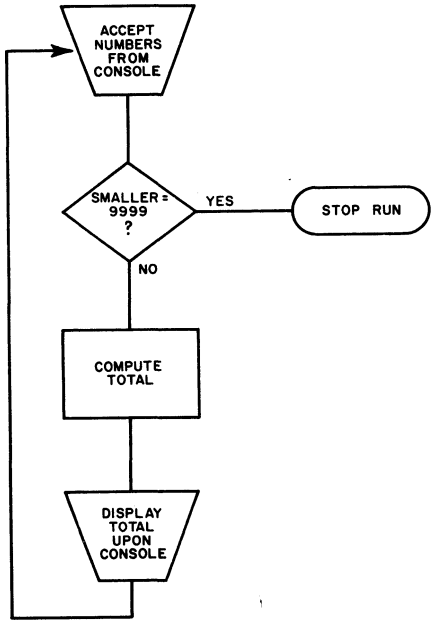
```
       COMPUTE TOTAL = (LARGER - SMALLER + 1)
           * AVERAGE.
```

As before, the parentheses are absolutely necessary, in order to control the order in which operations occur.

191 *From this small amount of practice with arithmetic statements, you can see that they are not at all difficult. Instead of spending more time with arithmetic, then, let's turn to some other topics that are related to our problem.*

Up to now, the input has consisted of just one pair of numbers. Now, we want to process more than one pair of numbers. This is simple enough - we will ACCEPT a pair of numbers from the console, COMPUTE the total, DISPLAY the total upon the console, and then GO TO the beginning of the procedure to accept another pair of numbers.

Needless to say, we must provide a way of getting out of this loop. To do this, we will tell the operator to type 99999999 when he is finished, and in our procedure we will test to see if the first number, SMALLER, is equal to 9999; if it is, we will stop the run. Our new procedure looks like this:



●●●

192 On a program sheet, write the statement to stop the run if SMALLER equals 9999.

●●●

```
    IF SMALLER = 9999, STOP RUN.
```

193 Another way of accomplishing the same end is by using a condition name. Suppose that the input area had been described in this way:

```
01  NUMBERS.
    02  SMALLER, PICTURE 9(4).
        88  FINISHED, VALUE 9999.
    02  LARGER, PICTURE 9(4).
```

This gives us a name, FINISHED, for the condition that exists when the value of SMALLER is 9999. On your program sheet, write an entry, using this condition name, which will have the same effect as the last previous entry you have written.

●●●

```
    IF FINISHED, STOP RUN.
```

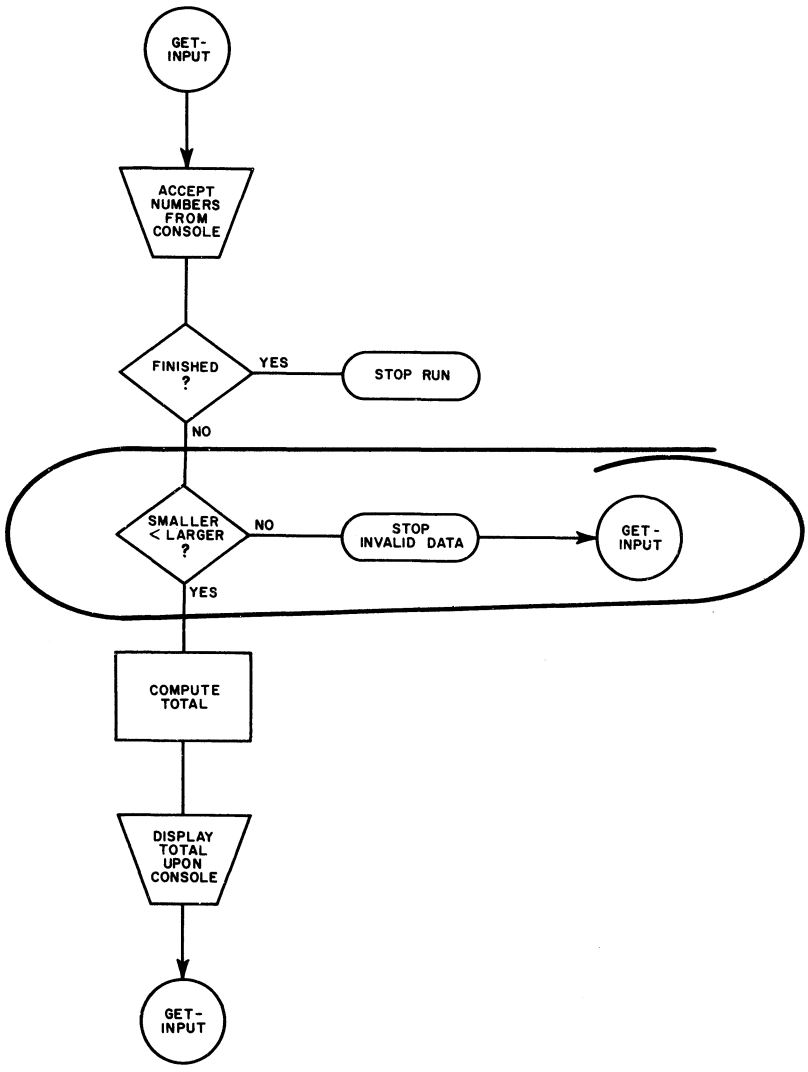
194 *Here is another aspect of our problem: Our solution assumes that the first number is the smaller one. If the operator were to make the mistake of typing the larger number first, we would get the wrong result. To keep this from happening, we should check the input to make sure that SMALLER is really smaller than LARGER.*

If SMALLER is not smaller, then what? Well, one possibility is to stop the execution of the program temporarily, and type a message to the operator telling him what is wrong.

Reading assignment: STOP (2)

●●●

195 *The check on the validity of the input, and a stop when the data was not valid, could be worked into the program like this:*



Write the entry that is represented by the blocks circled in the flowchart. The message we want to display before halting the program is:

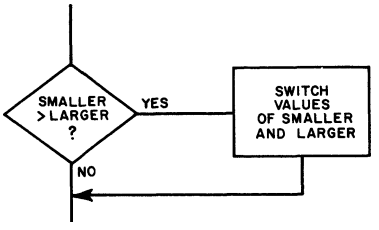
TYPE SMALLER NUMBER FIRST

The flowchart indicates that when execution of the program is resumed, we want to branch back to the beginning of the procedure, which is named GET-INPUT.

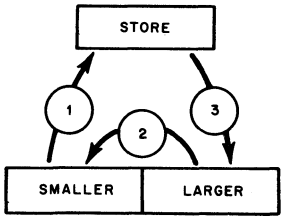
●●●

```
           IF SMALLER IS NOT LESS THAN LARGER,
               STOP 'TYPE SMALLER NUMBER FIRST',
               GO TO GET-INPUT.
```

196 *The "STOP-literal" way of handling the invalid data situation provided the occasion for introducing you to the second format of the STOP verb -- but unfortunately, it demonstrated a poor programming practice. It would not be wise to delay the execution of a program in any situation that could be handled by the computer itself. This is certainly a situation that the computer can be programmed to take care of; in the event that SMALLER is actually the larger number, we can move the value of SMALLER into LARGER, and move the value of LARGER into SMALLER.*



Switching the numbers requires not two, but three moves -- to avoid destroying one of the values in the process. And it requires a work area to hold one of the values temporarily. The three moves are sketched below.



On a program sheet, write (1) the Data division entry that describes the item called STORE (this item must hold four digits); and (2) the Procedure division entry that switches the values of SMALLER and LARGER, if the original value of SMALLER is greater than LARGER.

●●●

Data division entry:

```
77  STORE, PICTURE 9(4).
```

Procedure division entry:

```
    IF SMALLER IS GREATER THAN LARGER,
       MOVE SMALLER TO STORE,
       MOVE LARGER TO SMALLER,
       MOVE STORE TO LARGER.
```

197 Yet another approach to the difficulty of invalid input is to try to prevent it by displaying clear-cut instructions to the operator at the beginning of the run. The message might be:

TYPE TWO 4-DIGIT NUMBERS AS A STRING OF 8 DIGITS. NO SPACES.
SMALLER NUMBER MUST BE FIRST. TO STOP RUN, TYPE 99999999.

Which is correct:

- The DISPLAY verb cannot be used, because the message is too long.
- Alphanumeric messages such as this one cannot be displayed.
- Two DISPLAY statements can be used for the message, one for each line.
- The message is illegal because it contains reserved words.

●●●

Two DISPLAY statements can be used for the message, one for each line.

198 Write the two statements to display the message in the preceding frame. Treat each line of the message as a separate non-numeric literal. (You will find that both literals are too long to be written on a single line of a program sheet. If you have forgotten how to continue non-numeric literals, refer to the PROGRAM SHEET FORMAT AND RULES section of the reference handbook.)

●●●

```
SEQUENCE (PAGE) 1 3 (SERIAL) 4 6  CONT 7  A 8  B 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72
                                          DISPLAY 'TYPE TWO 4 DIGIT NUMBERS AS A STRING OF 8 DIGITS. NO
                                  -         ' SPACES.' UPON CONSOLE.
                                          DISPLAY 'SMALLER NUMBER MUST BE FIRST. TO STOP RUN, TYPE 9999
                                  -       '9999.' UPON CONSOLE.
```

199 *I suppose that we could go on forever, thinking up variations on this basic problem. But it has already served its purpose as a vehicle for learning to use several important COBOL statements. Obviously, you have not learned everything there is to know about every verb you have studied, but remember that our objective was to see how various statements are used in procedures -- not to study all of the "ifs, ands, and buts" of every statement.*

●●●

LESSON 7

200 *I have saved an important set of procedural verbs for last: the input-output verbs used for files of data -- OPEN, READ, WRITE, and CLOSE. Although these verbs are more simple to use than, say, the arithmetic verbs, the verbs themselves are only one part of the total requirement for processing files. The requirement involves:*

A. Environment division entries --

1. an Input-Output section, containing at least a File-Control paragraph, and sometimes an I-O-Control paragraph.

2. a SELECT entry for each file, to assign the file to an input-output device (SELECT entries are made in the File-Control paragraph).

B. Data division entries --

1. a File section.

2. a file description (FD) entry for each file.

3. a record description of each record in a file, following the FD entry for the file.

C. Procedure division entries --

1. an OPEN statement to make a file ready for reading or writing.

2. a READ statement to obtain a record for processing, or a WRITE statement to release an output record.

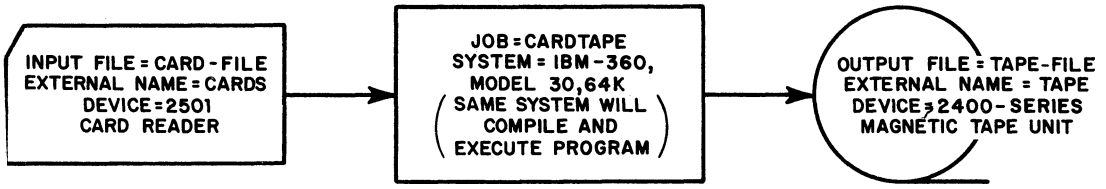
3. a CLOSE statement to terminate the processing of a file.

You have already written the Environment and Data division entries in earlier lessons; in this lesson, you will combine them with the Procedure division entries to produce two complete programs for processing files.

Both of the problems you will work on stress the input and output operations; for the sake of clarity and simplicity, no arithmetic operations and very few sequence control operations have been used. The first problem is a card-to-tape job -- a file of punched cards is to be written on tape. The second problem is a tape-to-print job -- the records in a tape file are to be listed on continuous-form paper.

●●●

201 This is a system flowchart of the card-to-tape job.



The flowchart provides the information that you need in order to complete the Identification and Environment divisions. Take a new COBOL program sheet, and write these two divisions.

For this problem, limit the Identification division to the required entries only. The program name is CARDTAPE. In the Environment division, write both the Configuration section and the Input-Output section; omit the I-O-Control paragraph of the Input-Output section.

You may, of course, refer to the sections in your reference handbook that deal with IDENTIFICATION DIVISION ENTRY FORMATS and ENVIRONMENT DIVISION ENTRY FORMATS.

●●●

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
    'CARDTAPE'.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.
    IBM-360, F30.
OBJECT-COMPUTER.
    IBM-360, F30.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT CARD-FILE, ASSIGN 'CARDS'
        UNIT-RECORD 2501 UNIT.
    SELECT TAPE-FILE, ASSIGN 'TAPE'
        UTILITY 2400 UNIT.
```

202 Here are some additional specifications about this job, which you need to write the Data division:

> There is only one type of record, named CARD-RECORD, in the input file; each CARD-RECORD contains 80 alphanumeric characters. The data from each card is to be moved to the output record, named TAPE-RECORD. Output records are to be written in the output file in blocks of 25 records. The output is to be recorded in mode F, with standard label records.

In this job, we don't intend to process any of the items within the records. Define the input record as an 80-character elementary item, and define the output record in the same way.

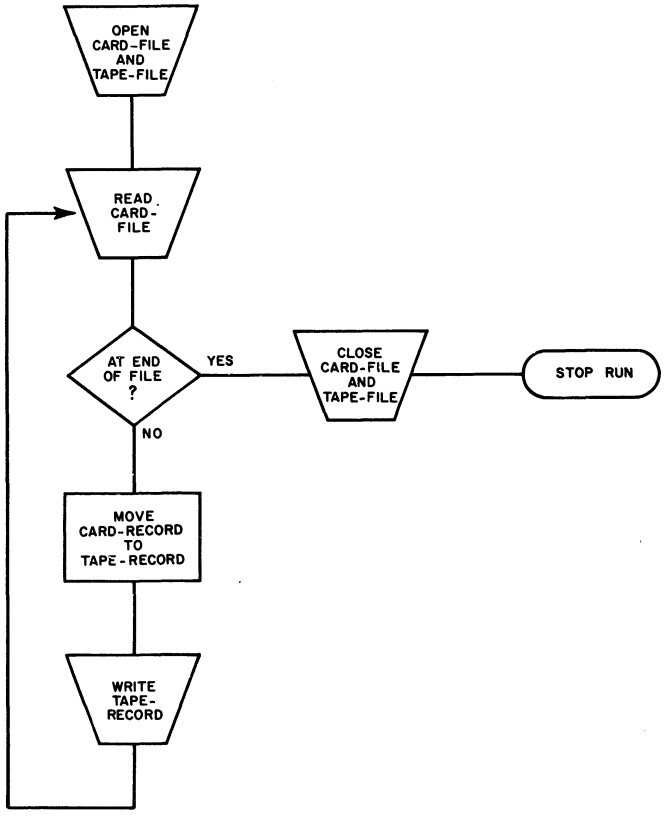
Take another program sheet to write the Data division. For entry formats, consult your reference handbook under DATA DIVISION ENTRY FORMATS.

●●●

```
DATA DIVISION.
FILE SECTION.
FD  CARD-FILE, RECORDING MODE F,
        LABEL RECORDS OMITTED,
        DATA RECORD IS CARD-RECORD.
01  CARD-RECORD, PICTURE X(80).
FD  TAPE-FILE, RECORDING MODE F,
        BLOCK CONTAINS 25 RECORDS,
        LABEL RECORDS STANDARD,
        DATA RECORD IS TAPE-RECORD.
01  TAPE-RECORD, PICTURE X(80).
```

Points to check: (1) For punched card files, the recording mode is always F, and label records are always omitted. (2) The description of a record (01-entry) must follow the FD entry for the file that contains that record.

203 *This procedure flowchart shows how the files will be processed.*



Notice, especially, that the files are opened at the beginning of the run, and closed at the end of the run.

Reading assignment:
```
OPEN (1)
OPEN (2)
OPEN (3)
CLOSE
```

●●●

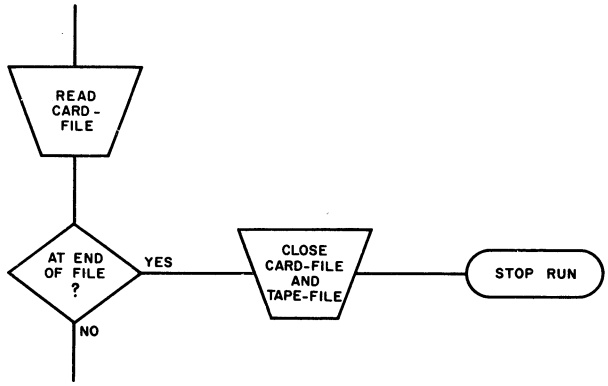
204 Begin the Procedure division on the program sheet on which you wrote the Data division. Name the first procedure OPEN-FILES, and in it write the entry or entries to open both files.

●●●

```
PROCEDURE DIVISION.
OPEN-FILES.
    OPEN INPUT CARD-FILE, OUTPUT TAPE-FILE.
```

It is also correct to write two OPEN statements.

205 *The READ entry does double-duty. It not only makes a record available from an input file, but also determines when the end of the file has been reached (that is, when all of the data records have been read); on an end-of-file condition, it causes other statements written in the entry to be acted on. The blocks printed below represent the operations that will be done by statements and clauses in the READ entry of our program.*



Reading assignment: READ

●●●

206 Now, start a new procedure in the Procedure division; name it READ-A-CARD. Write the READ entry that corresponds to the blocks printed in the preceding frame.

●●●

```
READ-A-CARD.
    READ CARD-FILE; AT END,
        CLOSE TAPE-FILE, CARD-FILE;
        STOP RUN.
```

Points to check: (1) Make sure that you wrote READ CARD-FILE, not READ CARD-RECORD. (2) The commas and semicolons printed in the answer above are not required, but the complete READ entry must be ended by a period -- the CLOSE and STOP statements must be part of the READ entry, not separate entries.

207 *We are now at the point in the procedure where the input data is to be moved into the output record and a record is to be written.*

Reading assignment: WRITE (1)

●●●

208 Complete the procedure named READ-A-CARD, by writing entries to:

1. move data into the output record.
2. write the output record.
3. branch back to the beginning of this procedure.

●●●

```
    MOVE CARD-RECORD TO TAPE-RECORD.
    WRITE TAPE-RECORD.
    GO TO READ-A-CARD.
```

Points to check: (1) The WRITE entry must specify TAPE-RECORD, not TAPE-FILE; in COBOL, we read a file, but we write a record. (2) The three entries above might have been written as three statements in one sentence; it is not necessary for the WRITE statement to be a separate entry.

209 *The card-to-tape program is now complete. Before we turn to the next problem, let's take a minute or two to look back over this program. Here are a couple of points that should be emphasized:*

First, it was necessary to split the Procedure division into two procedures (paragraphs). This made it possible to branch back to the READ statement. It would have been an error to branch to the beginning of the first procedure, since you cannot open a file that is already open. Note that the first procedure must be given a name, even though it is executed only at the start of the program, and we never branch back to it; there must be a header entry for every paragraph.

Second, note particularly that it was not necessary to "assemble" a block of 25 records prior to writing. This is because a WRITE entry does not cause data to be written on the tape; instead it releases records, which are then handled by an input-output control program (part of the operating system). The control program takes care of filling up a block. Each block is filled with the number of records you specified in the FD entry for the file. The actual transfer of data to an output device occurs when a block is full, or when the file is closed; the COBOL programmer doesn't have to worry about it.

●●●

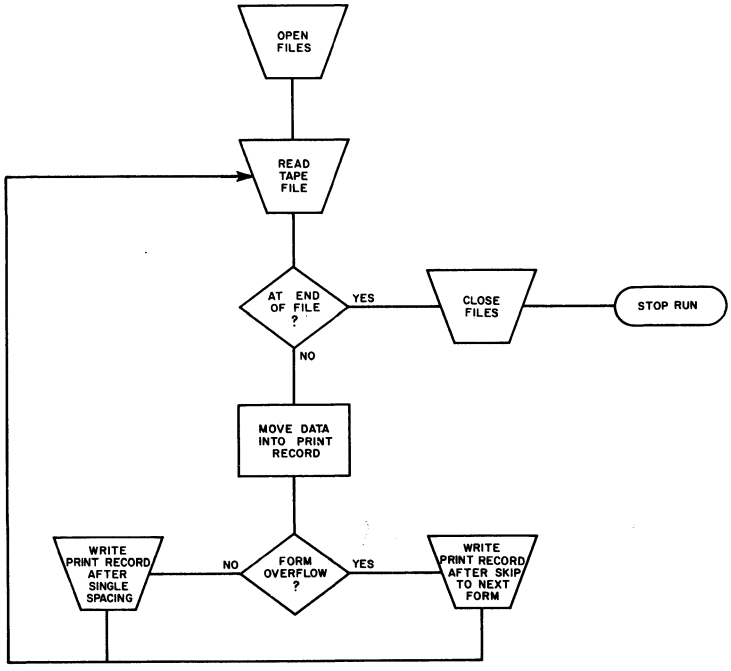
210 *Our second problem of this lesson is a tape-to-print job. The input is a file of records on magnetic tape, and the output is a listing of those records. One record is to be printed on each line of the output form, until the bottom margin or "overflow line" of the form has been reached; at that point, we want to skip to the first printing line of the next form, and go on printing line after line.*

Each line that is printed is a record, so the output consists of a series of related records. Hence, the output, like the input, is a data file. Perhaps you, like so many other people, are not accustomed to thinking of printed reports in these terms; however, when you use COBOL, you must learn to think of your input and output as being made up of records within files. (The only exception is the low-volume input and output which you accept and display.) For this problem, the complete printed report is a file, and each line of the report is a record.

●●●

211 *The logic of this program, then, is basically the same as that of the last program. There is one input file and one output file. These files must be opened at the beginning and closed at the end. Each time input is read, it is moved to form an output record, which is then written.*

Here is a procedure flowchart of the job. Observe the similarities between it and the flowchart of the previous job.



●●●

212 *The main differences between this and the previous job are that this time we must test for the form-overflow condition, and that along with writing a record we must control form spacing and skipping.*

Testing for overflow is done by an IF statement. To do this, a name must have been given to the overflow condition; this name is defined in the I-O-Control paragraph of the Environment division. Let's study this briefly, to get an idea of what we must do to define such a name. (You will find this reading assignment back in the reference handbook section titled ENVIRONMENT DIVISION ENTRY FORMATS.)

Reading assignment: APPLY

●●●

213 *Controlling form spacing and skipping is the function of one of the formats of the WRITE verb. When you read about this format, you will also learn some facts about how data items must be set up when printing is done.*

Incidentally, there is a close resemblance between the format of WRITE for printing, and that for punching cards. You will probably find it useful for your future work to read about the format for punching, even though there is no problem in this course that involves a punched card output file.

Reading assignment: WRITE (3)
WRITE (2)

●●●

214 We might use this statement in our program:

```
    WRITE ACCOUNT AFTER SKIP-TO-NEXT-FORM
```

If this statement is actually to cause a skip to channel 1, the value of the item named SKIP-TO-NEXT-FORM must be _____.

●●●

1

215 The item named SKIP-TO-NEXT-FORM might be defined like this:

```
77  SKIP-TO-NEXT-FORM, PICTURE X, VALUE '1'.
```

You would write this entry in the _____ section of the _____ division.

●●●

Working-Storage; Data

216 The WRITE statement that appeared a couple of frames back was taken from this IF sentence:

```
    IF AT-BOTTOM-OF-FORM,
    WRITE ACCOUNT AFTER SKIP-TO-NEXT-FORM.
```

This overflow test will work, provided that the name AT-BOTTOM-OF-FORM has been defined in an _____ entry, in the _____ section of the _____ division.

●●●

APPLY; I-O-Control; Environment

217 Let's modify the IF statement above, so that it meets the entire writing and form-controlling needs of our problem:

```
    IF AT-BOTTOM-OF-FORM,
    WRITE ACCOUNT AFTER SKIP-TO-NEXT-FORM;
    ELSE, WRITE ACCOUNT AFTER SINGLE-SPACE.
```

The value of the item named SINGLE-SPACE must be _____.

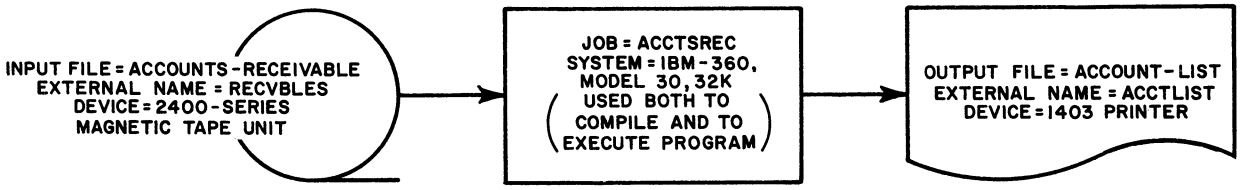
●●●

space (blank)

218 *This has been a fast look at the entry that we may use in the Procedure division to print a line and to control paper forms. It shows us that we must anticipate the needs of this procedural entry when we write the Environment and Data division of the program.*

●●●

219 This system flowchart gives most of the information you need to write the Identification and Environment divisions.



Take a new program sheet, and write these divisions. As before, write just a "bare-bones" Identification division. (There is simply no point in writing Author and Date-Written paragraphs, and so on, for this exercise. However, those will undoubtedly be required for the work you do on-the-job.)

While it is not mentioned in the flowchart above, remember that we want to define an overflow name in the Environment division of this program. Let's make that name AT-BOTTOM-OF-FORM, since that was the name used in the procedural statements you looked at earlier.

●●●

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
    'ACCTSREC'.
```

The Environment division is printed at the top of the next page.

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.
    IBM-360, E30.
OBJECT-COMPUTER.
    IBM-360, E30.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT ACCOUNTS-RECEIVABLE, ASSIGN
        'RECVBLES' UTILITY 2400 UNIT.
    SELECT ACCOUNT-LIST, ASSIGN 'ACCTLIST'
        UNIT-RECORD 1403 UNIT.
I-O-CONTROL.
    APPLY AT-BOTTOM-OF-FORM TO
        FORM-OVERFLOW ON ACCOUNT-LIST.
```

220 *To prepare the Data division, you will need detailed information about the files and records.*

First, the input file, ACCOUNTS-RECEIVABLE:

(1) recorded in mode F
(2) all data is BCD (external decimal code)
(3) each record is 44 characters long
(4) 50 records per block
(5) no label records

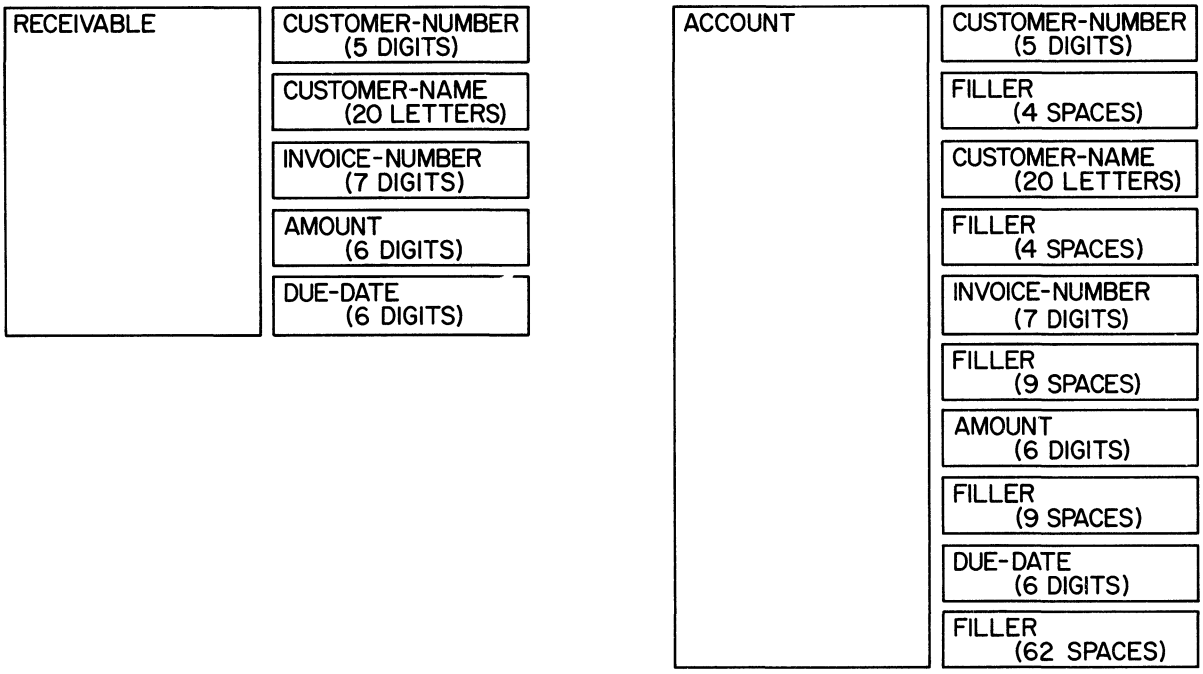
Next, the output file, ACCOUNT-LIST:

(1) each record is 132 characters long (the capacity of the printer)
(2) all data is BCD
(3) recording mode is F
(4) one record per block
(5) no label records

These characteristics apply to all files assigned to unit-record devices (card files and printed files).

●●●

221 *Each file contains just one type of record. The input record is named RECEIVABLE, and the output record is named ACCOUNT. These diagrams show the structures of the records:*



●●●

222 From the information given in the last two frames, write the File section of the Data division. It is not necessary to use the same names for corresponding items in the two records; to do so would require that each name be qualified each time it is used. Try this method: write a "prefix" letter before each name, a different prefix for input than for output. You might write I- before input items, O- before output items -- for example, I-DUE-DATE and O-DUE-DATE.

This is very important: You must add an additional item as the first item of the ACCOUNT record -- for form-control purposes. This item, technically, is not part of the 132-character output record, so it is not shown in the record structure above; however, this extra position must be accounted for in your record description. (Remember that the extra position is added at the beginning of only those records that are in files to be printed or punched.)

●●●

The solution for this frame is printed at the top of the next page.

```
DATA DIVISION.
FILE SECTION.
FD  ACCOUNTS-RECEIVABLE, RECORDING MODE F,
    BLOCK CONTAINS 50 RECORDS, LABEL RECORDS
    OMITTED, DATA RECORD IS RECEIVABLE.
01  RECEIVABLE.
    02  I-CUSTOMER-NUMBER, PICTURE 9(5).
    02  I-CUSTOMER-NAME, PICTURE A(20).
    02  I-INVOICE-NUMBER, PICTURE 9(7).
    02  I-AMOUNT, PICTURE 9(6).
    02  I-DUE-DATE, PICTURE 9(6).
FD  ACCOUNT-LIST, RECORDING MODE F, LABEL
    RECORDS OMITTED, DATA RECORD IS ACCOUNT.
01  ACCOUNT.
    02  FILLER, PICTURE X.
    02  O-CUSTOMER-NUMBER, PICTURE 9(5).
    02  FILLER, PICTURE X(4).
    02  O-CUSTOMER-NAME, PICTURE A(20).
    02  FILLER, PICTURE X(4).
    02  O-INVOICE-NUMBER, PICTURE 9(7).
    02  FILLER, PICTURE X(9).
    02  O-AMOUNT, PICTURE 9(6).
    02  FILLER, PICTURE X(9).
    02  O-DUE-DATE, PICTURE 9(6).
    02  FILLER, PICTURE X(62).
```

223 **Finish the Data division by writing a short Working-Storage section. In it, describe two independent items: SKIP-TO-NEXT-FORM, with a value of 1, and SINGLE-SPACE, with a value of space. These are the items that contain form-control codes, and we will use their names in WRITE statements.**

●●●

```
WORKING-STORAGE SECTION.
77  SKIP-TO-NEXT-FORM, PICTURE X, VALUE '1'.
77  SINGLE-SPACE, PICTURE X, VALUE SPACE.
```

224 Write the Procedure division. Invent your own names for paragraphs (procedure names). You may turn back to take another look at the procedure flowchart, printed in an earlier frame. You may also re-read the frames in which we discussed the way to program the WRITE statement to handle form-overflow.

Here's something to consider when you write the statements to move data to the output record: Do not assume that the FILLER items contain spaces. (We could not give these items an initial value of spaces, because the VALUE clause is forbidden in item description entries in the File section. And, as we pointed out once before, the word FILLER cannot be used in procedural statements -- so you must not write MOVE SPACES TO FILLER.) This problem can be solved simply by moving spaces into the entire record, as a whole, before moving any of the data items.

●●●

```
PROCEDURE DIVISION.
BEGIN-RUN.
    OPEN  INPUT ACCOUNTS-RECEIVABLE,
          OUTPUT ACCOUNT-LIST.
PROCESS-DATA.
    READ  ACCOUNTS-RECEIVABLE; AT END,
          CLOSE ACCOUNTS-RECEIVABLE,
                ACCOUNT-LIST;
          STOP  RUN.
    MOVE   SPACES TO ACCOUNT.
    MOVE  I-CUSTOMER-NUMBER TO
          O-CUSTOMER-NUMBER.
    MOVE  I-CUSTOMER-NAME TO O-CUSTOMER-NAME.
    MOVE  I-INVOICE-NUMBER TO
          O-INVOICE-NUMBER.
    MOVE  I-AMOUNT TO O-AMOUNT.
    MOVE  I-DUE-DATE TO O-DUE-DATE.
    IF AT-BOTTOM-OF-FORM,
          WRITE ACCOUNT AFTER
                SKIP-TO-NEXT-FORM;
    ELSE, WRITE ACCOUNT AFTER SINGLE-SPACE.
    GO TO PROCESS-DATA.
```

225 *Just as with any problem you can name, this problem might have been solved in various ways. The solution printed in this book is certainly not the only correct solution. Let me point out a couple of things that might have been done differently.*

(1) To control skipping and spacing, we have used the "data-name" option of the AFTER clause in the WRITE statement. We might have used the "integer" option, and written this entry:

```
    IF AT-BOTTOM-OF-FORM, WRITE ACCOUNT
    AFTER ADVANCING 0; OTHERWISE   WRITE
    ACCOUNT AFTER ADVANCING 1.
```

With this entry, the items we defined in Working-Storage would have been omitted. And the entry could have been abbreviated even more by dropping the word ADVANCING, which is optional. My opinion is that the "data-name" option is more easily understood by someone reading your program -- provided that you create sensible data names. Only another COBOL programmer would know that the integer 0 in the above entry designates a skip to channel 1. (Then again, you could write a NOTE entry to explain it.)

(2) Some people object to the practice of blanking out the output record each time, before moving new data into it. One way to avoid this is to assemble the output in a work area, and then move the complete record to the output area. What you could do is define a record in the Working-Storage section, and in it describe every item you want in the output record; the advantage is that in this section, you can assign initial values of spaces to all filler items. However, it would not be legal to WRITE a record that is in working storage. Therefore, you would have to define a separate output record, such as:

```
01  ACCOUNT, PICTURE X(133).
```

And you would have to transfer data from working storage to the output record, with an entry like MOVE WORK-RECORD TO ACCOUNT. So that, in the end, you wind up moving 133 characters after all. Which is not very different from filling the output area with spaces at the outset.

The idea, as I stated at the beginning, is that both ways are equally correct solutions. Use whichever method seems best, or easiest, to you. Possibly when you become proficient at COBOL, you will begin to consider which solution requires the least storage space when the object program is compiled, or which solution gives the shortest run time. For now, take the approach that any method is good, as long as it produces the desired results!

●●●

LESSON 8

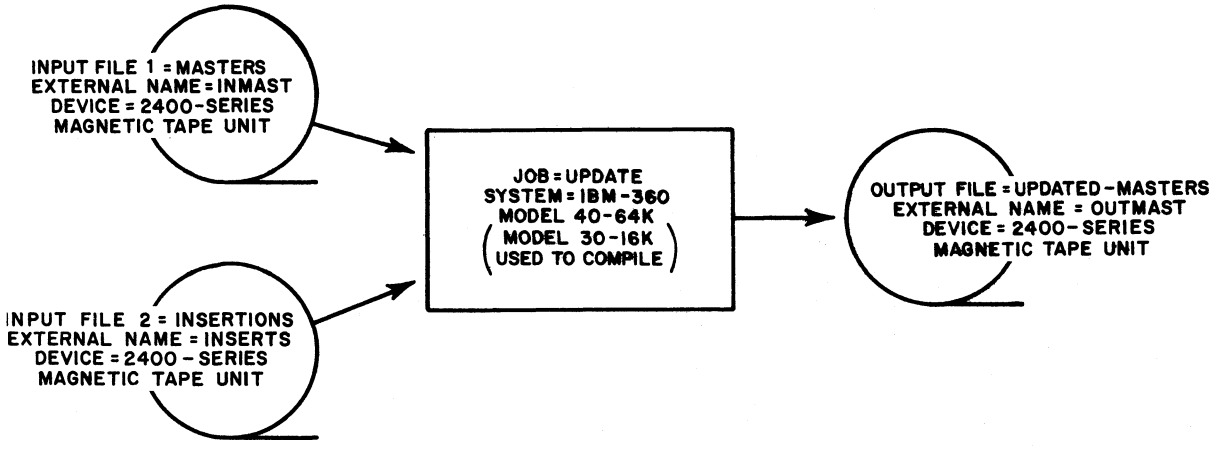
226 *In this final lesson, you will not be given any reading assignments. If you wish, you may consider this lesson to be a test of how well you can apply what you have learned in previous lessons. However, don't think of it as a "recall" test -- you may refer to your reference handbook as often as you like; so this is, in part, a test of how well you can figure out what information you need and look up that information when you need it.*

The lesson covers just one problem. The problem puts greater emphasis on input and output operations than the other problems you have programmed. As a result, the Procedure division for this problem will be somewhat longer. On the other hand, the entries required in the Data division are held to a minimum -- you have already had sufficient practice in making lengthy data description entries.

●●●

227 *The problem is to update a file of master records by inserting the records for new accounts into their proper places in the file. Both the master file and the insertion file are on magnetic tape, and the updated file is also to be written on magnetic tape.*

We have, then, two input files and one output file. The system flowchart for the job looks like this:



●●●

228 Based on the information given in the system flowchart (preceding frame), write the Identification and Environment divisions for the UPDATE job.

●●●

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
    'UPDATE'.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.
    IBM-360, D30.
OBJECT-COMPUTER.
    IBM-360, F40.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT MASTERS, ASSIGN 'INMAST'
        UTILITY 2400 UNIT.
    SELECT INSERTIONS, ASSIGN 'INSERTS'
        UTILITY 2400 UNIT.
    SELECT UPDATED-MASTERS, ASSIGN 'OUTMAST'
        UTILITY 2400 UNIT.
```

229 All three files are recorded with record-length control fields (mode V); there are ten records per block, and standard labels. Each file contains one type of record, with characters represented by BCD (external decimal code) throughout. The records in the three files all have exactly the same format: the first 15 digits constitute an item called NUMBER, which is the identifying number of the record; the remaining 135 characters make up various data items, none of which are processed in this program.

Using the above information, write the Data divison. Name the input master record MASTER; the insertion record INSERT; and the output master record OUT-RECORD. When you write record descriptions for MASTER and INSERT, follow the description given above; however,describe OUT-RECORD merely as a 150-character elementary alphanumeric item.

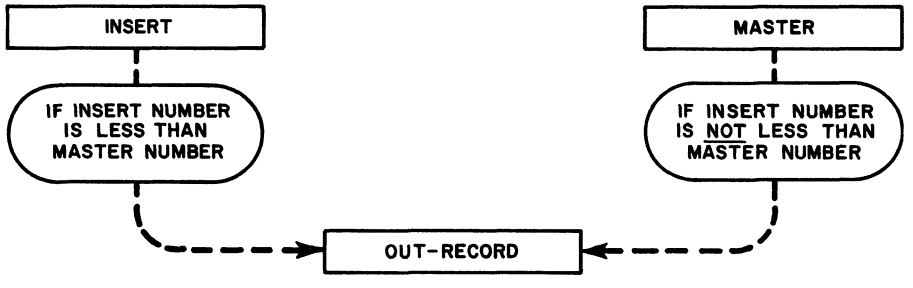
●●●

The solution for this frame is printed on the next page.

```
DATA DIVISION.
FILE SECTION.
FD   MASTERS, BLOCK CONTAINS 10 RECORDS,
     LABEL RECORDS ARE STANDARD, DATA RECORD
     IS MASTER.
01   MASTER.
     02  NUMBER, PICTURE 9(15).
     02  FILLER, PICTURE X(135).
FD   INSERTIONS, BLOCK CONTAINS 10 RECORDS,
     LABEL RECORDS ARE STANDARD, DATA RECORD
     IS INSERT.
01   INSERT.
     02  NUMBER, PICTURE 9(15).
     02  FILLER, PICTURE X(135).
FD   UPDATED-MASTERS, BLOCK CONTAINS 10
     RECORDS, LABEL RECORDS ARE STANDARD,
     DATA RECORD IS OUT-RECORD.
01   OUT-RECORD, PICTURE X(150).
```

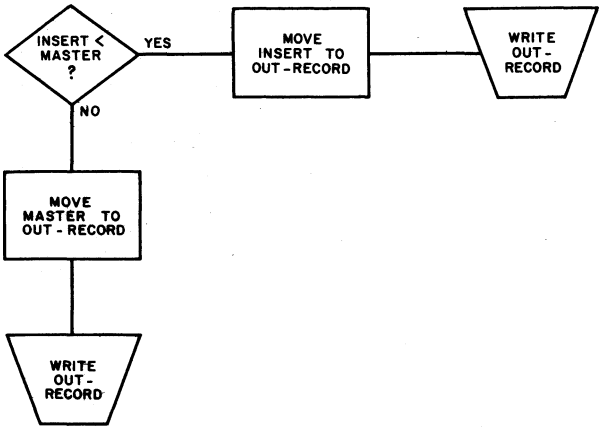
230 *In the next few frames, we will discuss the processing logic of this job, and develop the procedure flowchart from which you will write the Procedure division.*

The logic of the job boils down to a pair of conditional moves, as shown in the diagram below. We want to compare the identifying numbers of two records -- one from each of the input files. If the number of INSERT is lower, the INSERT record is to be moved to the output area; otherwise, the MASTER record is to be moved to the output area. An output record can then be written.



●●●

231 *The selection of the proper output data is the crucial decision to be made, the "heart" of the processing to be done -- which is always a good place to start developing a flowchart. We will work backward and forward from here.*



●●●

232 How can we get to the point where we can make this decision about the first record from each input file? Clearly, we must read a record from [the MASTERS file] [the INSERTIONS file].

●●●

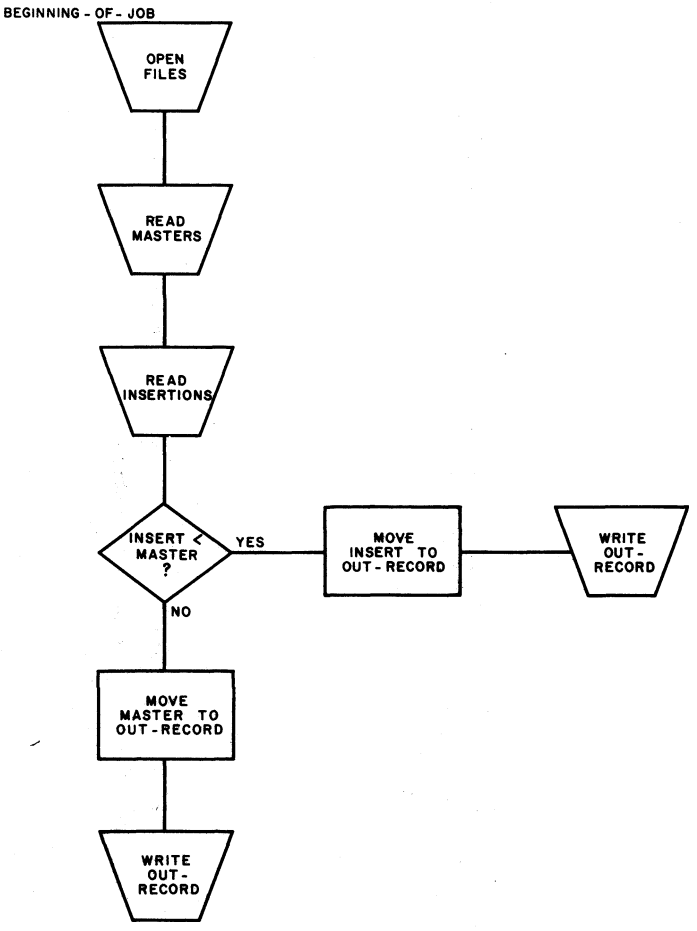
BOTH the MASTERS and the INSERTIONS file

233 But before we can read records from the files, we must _____ the files.

●●●

OPEN

234 *In considering how to process the first record of each input file, we have worked our way back to the beginning of the job. This is what our flowchart looks like at this point. (I have chosen the name BEGINNING-OF-JOB to be the name of the first COBOL procedure.)*



●●●

235 At the first step drawn on the flowchart above, we will open

{the two input files
the input and output master files
both the two input files and the output file} .

●●●

both the two input files and the output file

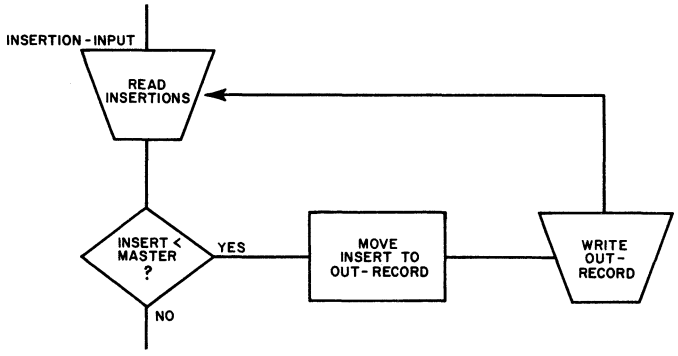
236 Let's say that the first comparison showed that the number of the insertion record was lower, so the INSERT record was moved to OUT-RECORD, which was then written. In this event, the next logical step would be to

{ move MASTER to OUT-RECORD, since it must be the next output record.
go back and read both files again, to prepare for the next comparison.
read only the INSERTIONS file again, and make another comparison.
close the INSERTIONS file, and branch back to the beginning of the job. }

●●●

read only the INSERTIONS file again, and make another comparison

237 *This flowchart segment shows the branch to read the INSERTIONS file again.*



●●●

238 If the comparison of the first records had shown that the INSERT record was not less than the MASTER, the other leg would have been taken at the decision block -- that is, the MASTER record would have been written out. And we would now have to obtain another MASTER.

In order to get the next MASTER, we can

[write another READ MASTERS step]
[branch back to the original READ MASTERS step].

●●●

ONLY write another READ MASTERS STEP

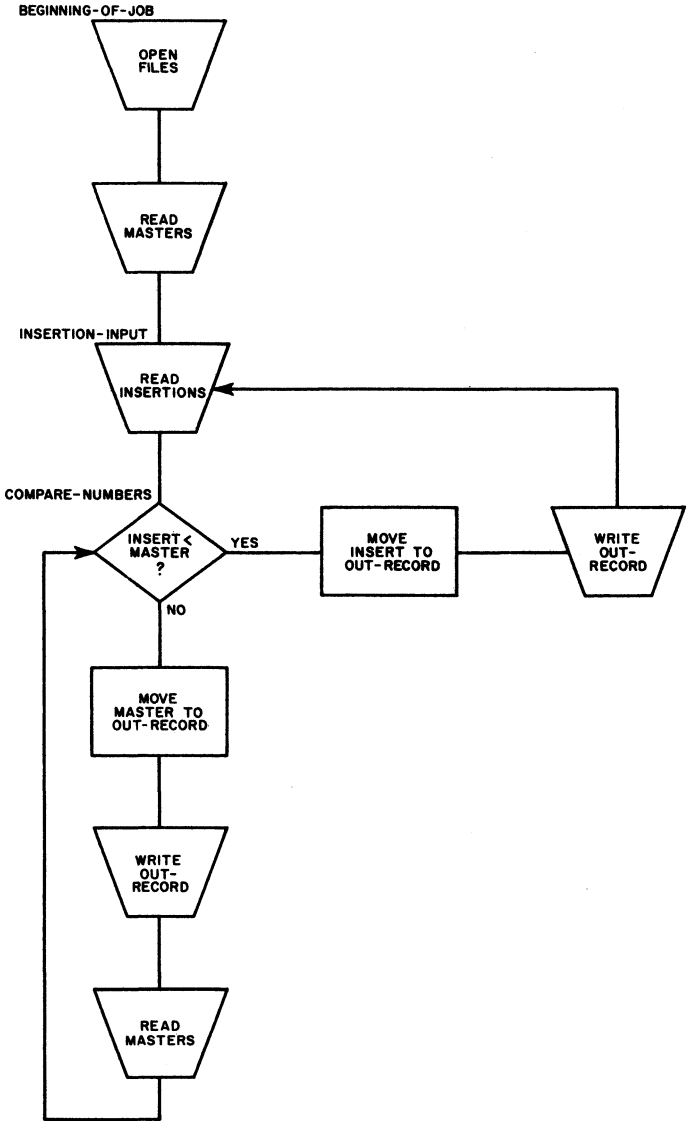
A simple branch back to the original READ MASTERS step would also cause control to flow through the READ INSERTIONS step, which would be an error.

239 We will insert a new READ MASTERS step into our flowchart. After another MASTER record is read, we must go to the step where _____.

●●●

the INSERT number and the MASTER number are compared

240 *We have now "closed the loops" of our process. The flowchart has developed to this point:*



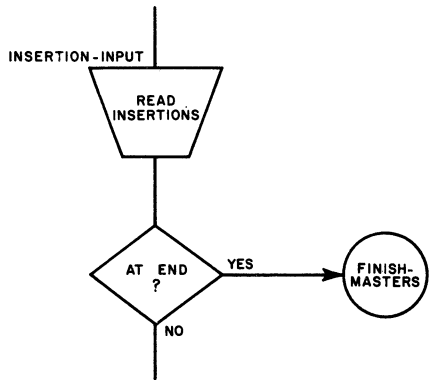
Notice that I have been supplying procedure names as we have gone along. The names INSERTION-INPUT and COMPARE-NUMBERS will be needed in GO TO statements.

●●●

241 *If the input files were infinitely long, and processing went on forever, our present flowchart would be adequate. The facts of life, though, are that data files -- like all good things -- must end. And we must test to find out when the end has come.*

Let's be more specific. Every READ statement is required to contain an AT END clause; we will show the AT END tests as decision blocks in our flowchart. Furthermore, we will have to consider that one file must necessarily run out before the other; thus, at the moment that we find that there are no more records in the INSERTIONS file, there must be at least one more MASTER record that has not yet been written out, and vice versa.

Here the AT END test is shown in the INSERTION-INPUT procedure:



At the end of the INSERTIONS file, we will branch to a procedure named FINISH-MASTERS. When this branch occurs,

[a previously-read MASTER is waiting to be processed]
[the MASTERS file is ready to be closed]
[there may be more MASTER records that have not yet been read].

●●●

a previously-read MASTER is waiting to be processed, AND there may be more MASTER records that have not yet been read

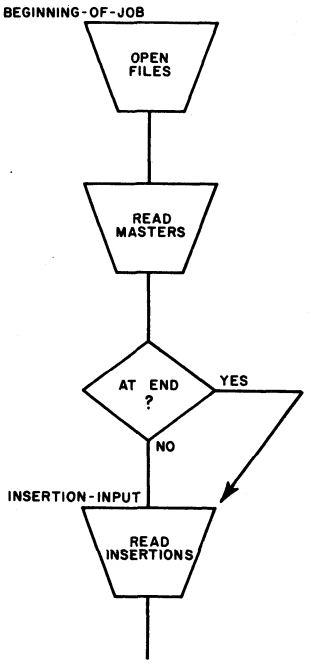
242 We will have to insert AT END decisions after both of the READ MASTERS steps. Examine the flowchart on the preceding page. Determine how often control will flow through the first READ MASTERS step, following OPEN FILES.

●●●

Only once

243 *Although this sort of duplication of steps is sometimes eliminated by writing "program switches", we find that it is often easier in COBOL to duplicate a step instead of fussing with a switch. So in this program, we have two READ MASTERS steps (so far) -- the first of which is executed only at the very outset of processing; an AT END branch cannot, under any circumstances, occur after the first READ MASTERS step.*

Nevertheless, the format of the READ statement insists that we provide the AT END test. We will satisfy this requirement by writing a branch to the next step, as diagrammed below.



●●●

244 *The second READ MASTERS step presents an entirely different situation. It is part of a loop that reads and writes MASTER records, and it will be executed repeatedly during the running of the program. If the MASTERS file runs out first, the end of file condition will be detected at this step; therefore, this AT END test is a real one.*

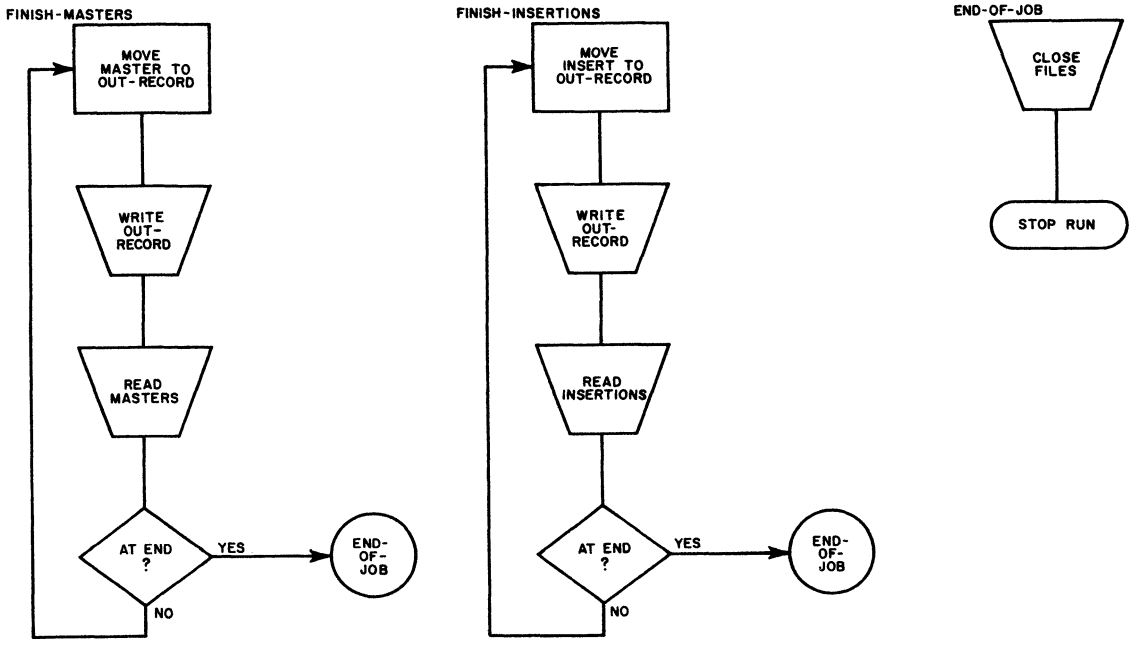
At the end of the MASTERS file, we will branch to a procedure named FINISH-INSERTIONS. If this branch is taken, we know that a previously-read INSERT is waiting to be processed, and there may be more INSERT records that have not yet been read.

●●●

245 *Here is how we will finish up the remaining file after we have reached the end of the other file: we know that one previously-read record from the remaining file is waiting in the input area, so we will move it to OUT-RECORD, and write it out. Then, because there may be more records in the remaining file, we will read the file and test for the AT END condition. When we are at the end of the remaining file, both files are finished, so we branch to the end of job procedure -- close files and stop the run.*

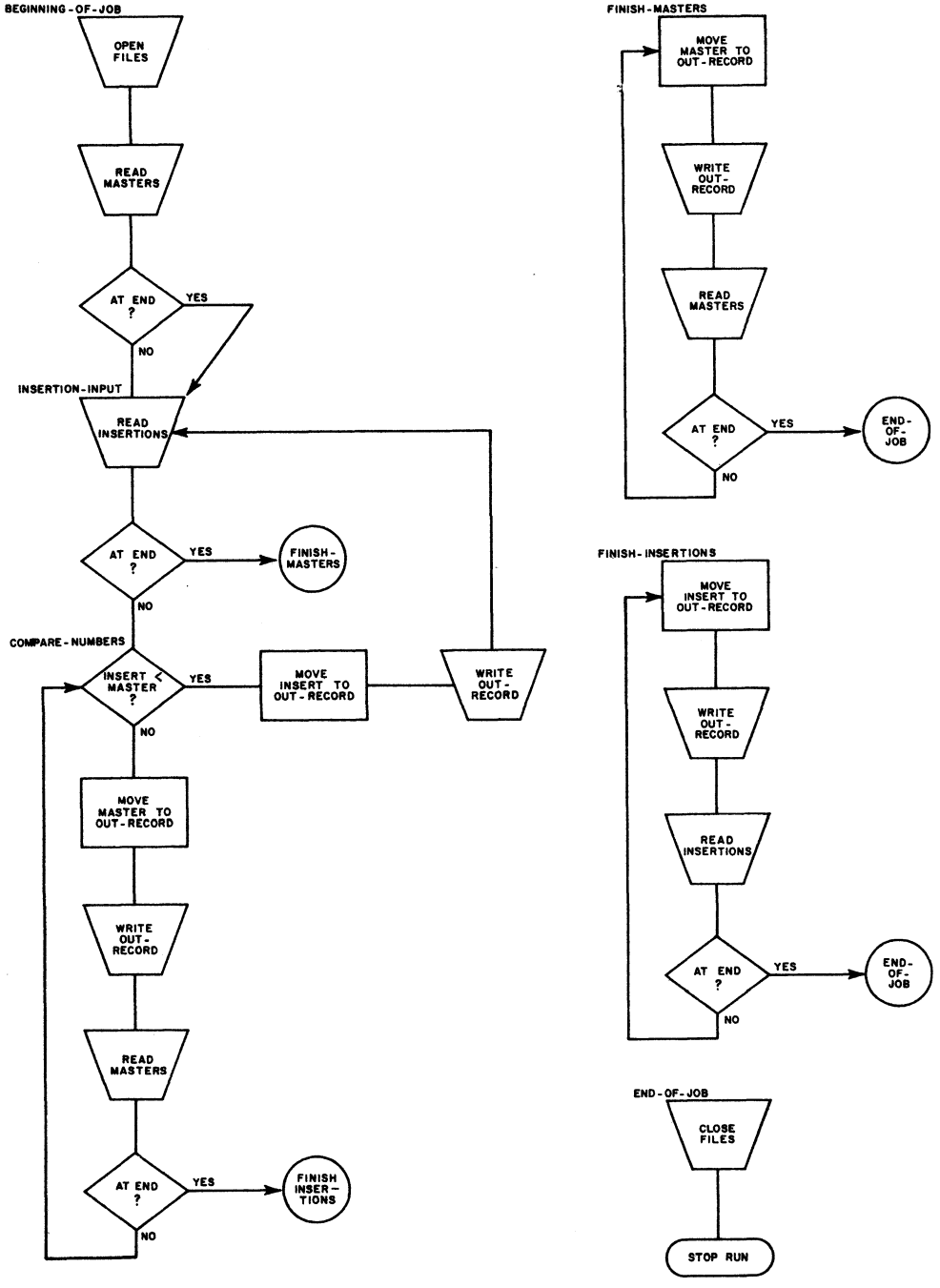
It would be pointless to go back to the main routine and compare the numbers of records. In fact, it would be an error, since there are no more records in the other file. Besides, we know that all of the records in the remaining file must be written out, so all we have to do is read-move-write, until we come to the end of the file.

Two "finishing-up" procedures have been diagrammed below -- one for each input file. Two procedures are needed because we have no way of knowing which file will run out first when they are processed by the main routine. If the INSERTIONS file runs out first, we will branch to FINISH-MASTERS; if the MASTERS file runs out first, we will branch to FINISH-INSERTIONS.



●●●

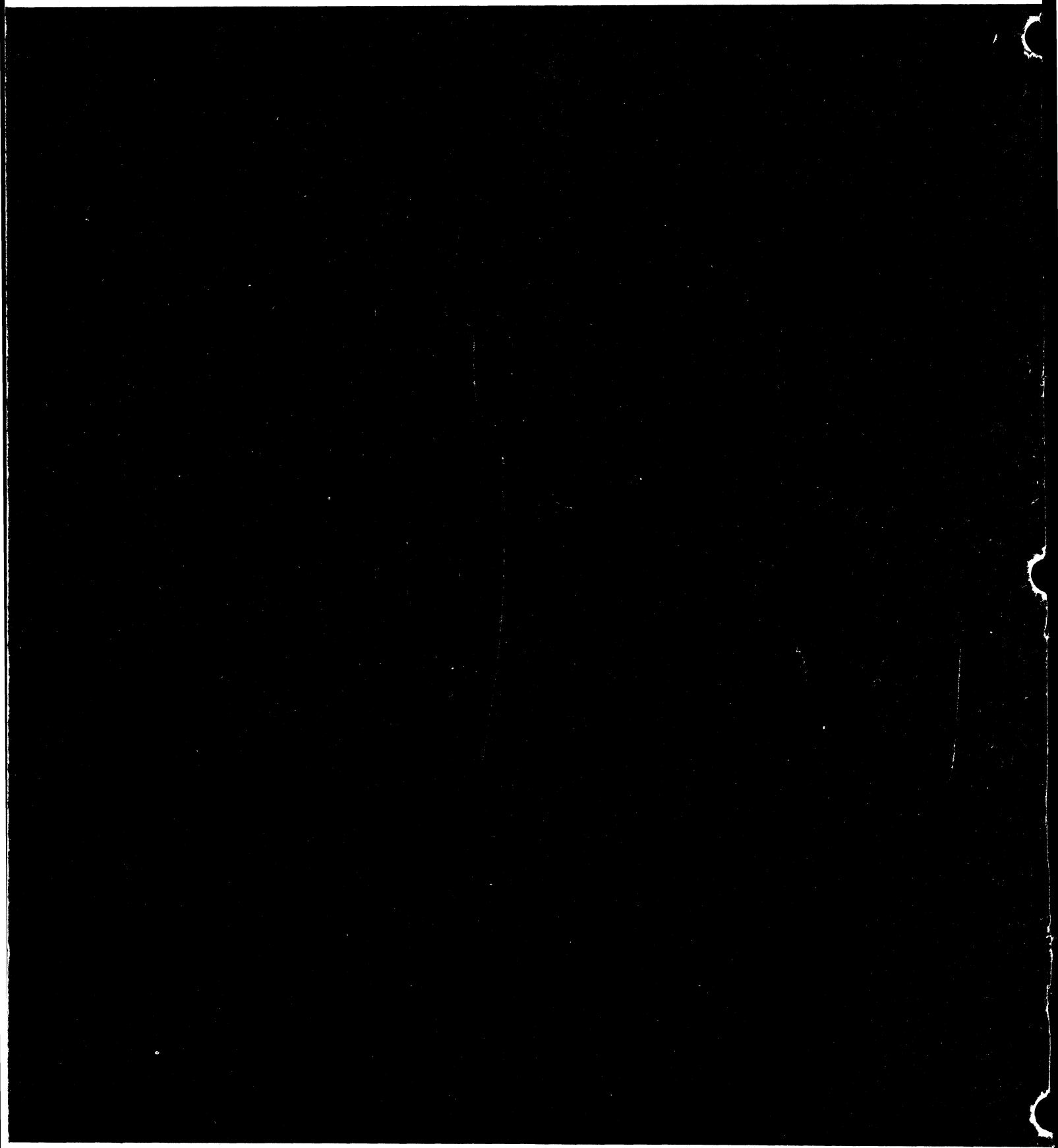
246 This is the completed flowchart for the UPDATE job. Write the Procedure division that corresponds to this flowchart. All of the procedure names are shown, and many of the COBOL statements can literally be copied from the flowchart. Be careful, though! Some of the flowchart notes are like COBOL, but not exactly like it -- for instance, OPEN FILES, INSERT<MASTER, and CLOSE FILES. Use your reference handbook to find the actual formats. (The correct solution for this frame is printed on the next page.)



●●●

```
PROCEDURE DIVISION.
BEGINNING-OF-JOB.
    OPEN  INPUT MASTERS, INSERTIONS;
          OUTPUT UPDATED-MASTERS.
    READ  MASTERS; AT END,
          GO TO INSERTION-INPUT.
INSERTION-INPUT.
    READ  INSERTIONS; AT END,
          GO TO FINISH-MASTERS.
COMPARE-NUMBERS.
    IF NUMBER OF INSERT < NUMBER OF MASTER,
          MOVE INSERT TO OUT-RECORD,
          WRITE OUT-RECORD,
          GO TO INSERTION-INPUT.
    MOVE MASTER TO OUT-RECORD.
    WRITE OUT-RECORD.
    READ  MASTERS; AT END,
          GO TO FINISH-INSERTIONS.
    GO TO COMPARE-NUMBERS.
```

```
FINISH-MASTERS.
    MOVE MASTER TO OUT-RECORD.
    WRITE OUT-RECORD.
    READ  MASTERS; AT END, GO TO END-OF-JOB.
    GO TO FINISH-MASTERS.
FINISH-INSERTIONS.
    MOVE  INSERT TO OUT-RECORD.
    WRITE OUT-RECORD.
    READ  INSERTIONS; AT END,
          GO TO END-OF-JOB.
    GO TO FINISH-INSERTIONS.
END-OF-JOB.
    CLOSE MASTERS, INSERTIONS,
          UPDATED-MASTERS.
    STOP  RUN.
```



International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, New York