Edward Humby and Philip Robinson

SPECIAL ENGLISH

PETER STREVENS
General Editor

Computers

SPECIAL ENGLISH

COMPUTERS

SPECIAL ENGLISH

COMPUTERS

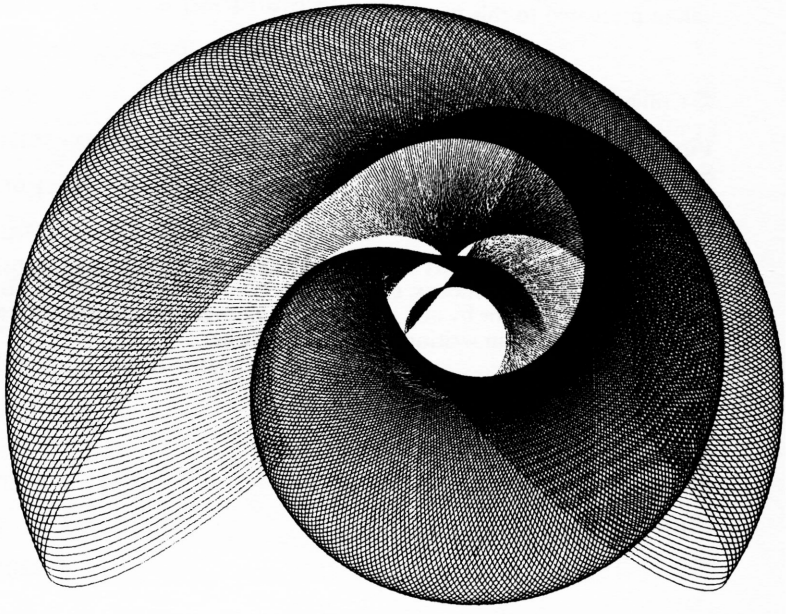
Other titles in this series:

Accounting
Advertising
Air Travel
British Banking
British Banking Overseas
Computer Programming
The Department Store
Import/Export
The Jet Engine
Legal Problems
Marketing Petroleum Products
The Motor Car: Books 1 & 2
Nursing
Office Practice: Books 1 & 2
Physical Education
Seafaring
Television

General Editor: PETER STREVENS

Edward Humby and Philip Robinson

Computers



Collier Macmillan International Inc.
New York
Cassell & Collier Macmillan Publishers Ltd.
London

Grateful acknowledgement for permission to use the photographs reproduced in this book is made to the following (numbers refer to the pages on which the pictures appear):

The British Aircraft Corporation (74); Draper's Record (83); IBM (U.K.) Limited (1, 5, 6, 10, 20, 23, 27, 36, 56, 69, 80, 88); International Computers Limited (6, 13, 24, 25, 32, 34, 37, 38, 41, 53, 58, 66, 71, 74, 78, 87); Midland Bank (viii, 44); Swiss Bank Corporation (65). The pictures on pages 17, 28, 32 and 38 were supplied by The National Giro Centre.

Cover photograph: International Computers Limited.

The pattern on the title page was produced by a computer on a graph plotter. The computer was instructed how to produce patterns by a FORTRAN program, and different patterns were made by altering the values presented to this program.

First Printing 1971
Second Printing 1972
Third Printing 1975
Fourth Printing 1975

Collier Macmillan International, Inc.
866 Third Avenue, New York, N.Y. 10022

Cassell & Collier Macmillan Publishers Ltd.,
35 Red Lion Square, London WC1R 4SG

ISBN 0 02 974870 4

CONTENTS

PREFACE

The Special English series from Collier Macmillan International introduces titles on a wide range of technical subjects that will be of interest to students of English as a second language. Each volume illustrates the special English of a particular trade or profession in both its spoken and written forms. It is not possible, of course, for books of this size to cover the subject matter exhaustively, so the authors have concentrated on those topics and activities that should have the widest appeal. The conversations which are the basis of each chapter or unit are deliberately written in the colloquial and idiomatic speech used by technicians and specialists as they go about their everyday activities.

It must be emphasized that these books are *not* primarily intended to teach the subject matter itself, although the technical content is accurate in every respect. Nor are they intended to teach the introductory stages of English. It is assumed that the reader is already familiar in his own language with the subject matter of the book, and has a good grounding in the basic grammatical patterns and vocabulary of English. He will use these books to improve his knowledge of English within the framework of a technical vocabulary that is of interest to him either privately or professionally.

The authors in this series each have their individual approach, but all the volumes are organized in the same general way. Typically, each book is based on a series of situational dialogues, followed by narrative passages for reading comprehension. Exercises give the student practice in handling some of the useful and more difficult patterns, as well as lexical items, that occur in each unit. Tape recordings, of the dialogues and selected exercises, may be used either in the language laboratory or for private study. Each volume is provided with a glossary of technical terms, with i.p.a. equivalents as used in the Daniel Jones Pronouncing Dictionary.

PETER STREVENS
General Editor

INTRODUCTION

Special English: Computers is intended for students of English as a second language who wish to become familiar with the vocabulary of computers and computing. The book is intended for classroom work with a teacher, but it will also be found suitable for private study.

Each unit contains a dialogue, in colloquial conversational style, based on activities within the field of computing. The dialogue is followed by a group of exercises which give the student practice with the structures and new vocabulary items that have been introduced. Although these exercises may be written, they are primarily for oral practice, with the emphasis on repetition until the student is entirely fluent and can make the appropriate changes and substitutions quickly and accurately.

The exercises are followed by a reading passage with comprehension questions. Keys to the exercises are provided at the back of the book, together with a glossary of the technical terms connected with computing. These are marked in the text with an asterisk. The International Phonetic Alphabet is used as a guide to pronunciation, and colloquial and idiomatic expressions are footnoted.

The tape recording that accompanies the book may be used by the teacher in the classroom or the language laboratory. For the student working alone, it will provide a model for pronunciation as well as a means of taking dictation for practice in spelling. The exercises have pauses for student response, but there are no pauses in the dialogues. This has been done on purpose to provide the maximum amount of recorded material. Most tape recorders are now equipped with a pause button, which will enable the listener to stop the tape after each sentence and repeat it aloud before proceeding to the next one. If pauses are required for language laboratory work, a copy may be made and the pauses inserted of a length to suit the requirements of the students. A demonstration tape, free of charge, is available from the publisher upon request.

UNIT 1

WHAT IS A COMPUTER?

Dialogue

John Codewell is discussing a career in computing with the Personnel Officer of a large firm.

John: A computer's a machine that thinks, isn't it?

Personnel Officer: Well, yes. But it has to be told what to think; like any other machine it's only a slave. I'll give you a more exact definition. It's an electronic machine that processes *data under the control of a *stored *program.

John: What's the difference between a computer and a *computer installation?

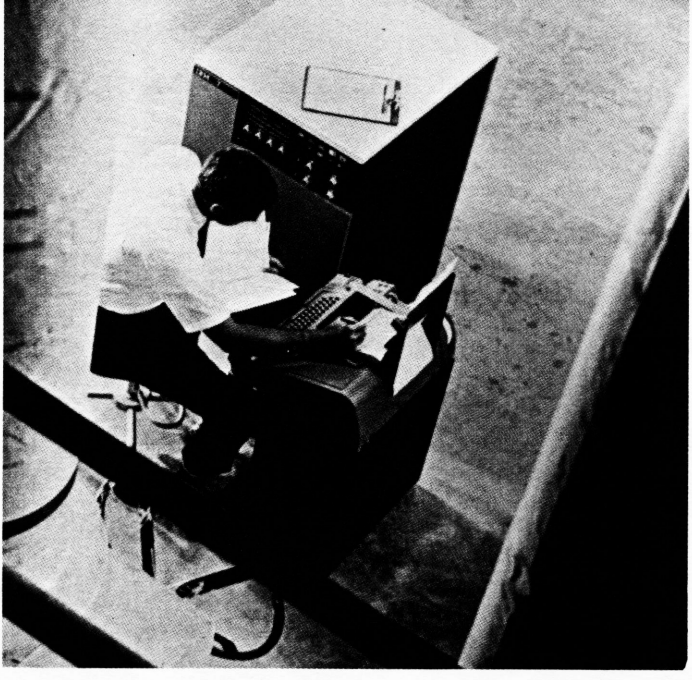
Personnel Officer: A computer installation really consists of three things: the *hardware, or machinery; the *software, computer programs telling the machine what to do; and the staff who work with the installation, sometimes called the *"liveware". Take the hardware first. This comprises a number of separate machines linked together in a system.

John: Isn't the main one the *central processing unit?

Personnel Officer: Yes, and the C.P.U. might be called the computer proper, because it contains all the *memory banks and logic and arithmetic units which actually do the calculations. But by itself it's as useless as a brain without a body. The brain needs sense impressions to work on, and the C.P.U. needs data to process. Data simply means information. It's fed into the machine via *input units, and the C.P.U. processes it and communicates results to the user via *output units.

John: What a lot of jargon![1] What are the input and output units called?

[1] jargon: words used in a specialized, often technical sense



A one-man computer

Personnel Officer: They're called "peripherals". There are slow peripherals—*card readers, *paper tape readers and *console typewriters for inputting data; and printers, card punches and paper tape punches for outputting results.

John: Why do you call them "slow"?

Personnel Officer: Because they're mechanical devices.

John: So I suppose magnetic devices—tape decks and so on—are called "fast" peripherals?

Personnel Officer: That's right. Tape decks work in much the same way as ordinary tape recorders, but there are even faster magnetic peripherals—*drums and *discs, for instance. All these can be used both for input and output. Then there are *visual display units.

John: Yes, I see. I think I've got the idea of the hardware. But what about the software? What exactly is a "computer program"?

Personnel Officer: It's a set of instructions telling the computer how to solve a problem, whether the problem is calculating a square root or working out Mr. Jones' pay for the week.

John: Surely the machine isn't big enough to hold programs to solve every possible problem?

Personnel Officer: No, not all at once. And here you've hit on one of the most important features of the modern computer. A program can be stored in its memory just as long as it's needed, and then removed, or, as we say, "deleted from store". When it's not needed it's held in a library, usually on one of the magnetic devices.

John: So the software tells the hardware what to do and computer personnel write the software?

Personnel Officer: That's what computer *programmers do, but there are many other categories of computer personnel. There are *operators to run the machines, electronic engineers to service them, *systems analysts to define the problems, not to mention[2] the data preparation staff who punch up the cards and paper tape, and the people in administration, like myself.

[2] not to mention: also, as well as

EXERCISE 1: STRUCTURAL PRACTICE

Notice this structure from the conversation:

What about the software?

Use this structure to respond to the following statements:

Example: This is the hardware.
Prompt: *software*
Response: What about the software?

Now, you do it.

1. This is the hardware.	*software*
2. These are the fast peripherals.	*slow peripherals*
3. These are the input units.	*output units*
4. This is the programmer.	*operator*
5. These are the mechanical devices.	*magnetic devices*
6. This is the card reader.	*card punch*
7. This is the paper tape reader.	*paper tape punch*
8. These are the drums.	*discs*

EXERCISE 2: PROGRESSIVE SUBSTITUTION DRILL

Statement: A computer installation consists of three things.
Prompt: *The*
Response: *The* computer installation consists of three things.

Now, you do it.

Statement: A computer installation consists of three things.
Prompts:
1. The
2. different
3. various
4. Any
5. parts
6. is made up
7. C.P.U.
8. contains

EXERCISE 3: FURTHER STRUCTURAL PRACTICE

Notice this structure:

I suppose magnetic devices *are called* "fast" peripherals?

Use this structure to make complete sentences:

1. mechanical devices / "slow" peripherals
2. the whole thing / a computer installation
3. card readers / input units
4. programs / the software
5. the C.P.U. / the computer proper
6. the peripherals / input/output devices
7. circuits such as those / arithmetic units
8. drums and discs / magnetic devices

Reading and Comprehension

A computer is an electronic calculating machine, but by itself it is of no more use than a car engine without a chassis, or a control panel without machines to monitor and control. It needs data to process and input units to feed it this data. It also needs output units to which it can send its results. Thus it is more meaningful to talk about a computer installation, consisting of a Central Processor, which is the computer proper, and a number of input/output units, or peripherals. And just as there are people of widely different skills and knowledge in a ship's crew—engineers, deckhands, navigators, stewards, pursers and radio operators, for example—so too the personnel in a computer installation are specialists in many different fields. There will be systems analysts to analyse and define the problems the computer has to solve; programmers to write the instructions the machine must obey to solve the problems; data preparation staff to *punch the programs on to cards or paper tape, and operators to run the machine and monitor its progress. There will also be a messenger service and a magnetic library staff, together with a superstructure of managers and supervisors. All are necessary for the smooth functioning of a computer installation.

A computer room

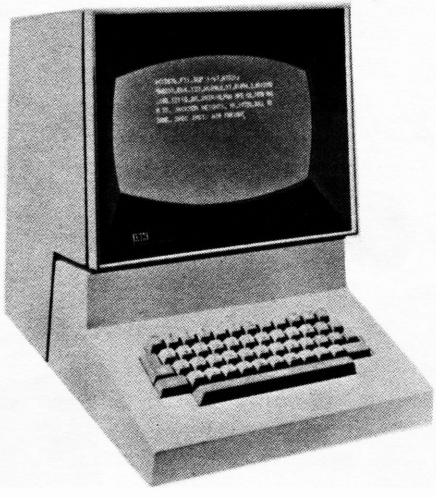
EXERCISE 4: COMPREHENSION QUESTIONS

1. What does a computer process?
2. What is another name for input/output units?
3. What are the people called who analyse and define problems for the computer to solve?
4. Who writes the instructions the machine is to obey?
5. What do data preparation staff do?
6. What do operators do?
7. Name two other categories of computer personnel.
8. Which part of a computer installation is the computer proper?

EXERCISE 5

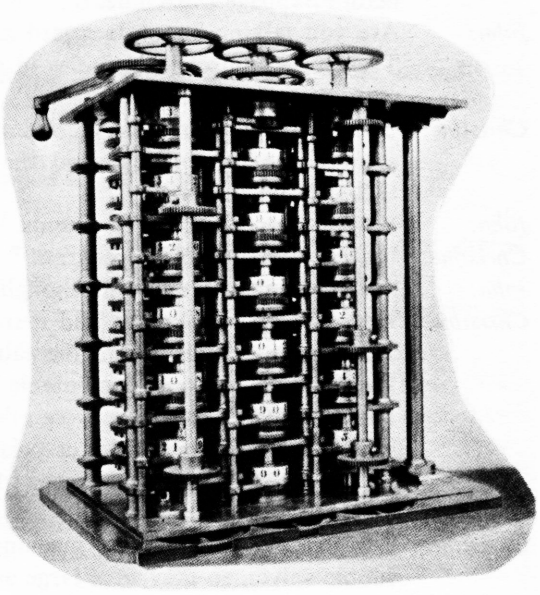
Use the following words and phrases in sentences of your own to show that you understand their meaning and use:

1. data
2. input units
3. useless
4. slow peripherals
5. software
6. deleted from store
7. define
8. magnetic



A visual display unit

Babbage's differential engine



UNIT 2

THE HISTORY OF THE COMPUTER

Dialogue

John is looking round a science museum with his cousin Christine, who is taking a university computer course.

John: So this is where it all started—with Babbage's Differential Engine?

Christine: Yes, he showed that a machine could do calculations by itself, using levers and gear wheels, provided you supplied it with a list of operations to follow.

John: It was a mechanical device, then? It must have been very slow compared with today's electronic machines!

Christine: Well, he never actually completed it. It isn't only the development of electronics that has made modern computers possible, though: at least two other things had to be thought

up first. One of them was the representation of data as holes punched in cards. I think that was discovered in France before Babbage's machine.

John: Are you talking about Jacquard and his weaving looms? They were controlled by holes in cards telling them what patterns to produce, weren't they?

Christine: Oh, yes, that's right. And Hermann Hollerith was the first man to use this principle—in the 1890's, to process the U.S. census data.

John: You mentioned two developments. What was the other one?

Christine: It's the idea of a stored program.

John: I've heard about that—who thought it up?

Christine: Well, Babbage may have had it in mind, but it was only described in detail by mathematicians in this century. There was a paper on "Computable Numbers" by Dr. A. M. Turing in 1936. And there were other important papers by John von Neumann and his associates in the 1940's.

John: As usual, though, I suppose the ideas of theorists had to wait for technological inventions to put them into practice?

Christine: Yes, that's right. The early machines were based on *thermionic valves, so they were large and clumsy and got very hot. It was only when *transistors and *printed circuits and *magnetic cores were invented that computers became really useful.

John: By the way, what's the difference between *analogue and *digital computers?

Christine: Digital computers depend on electronic devices which at any moment are in one of two states: on or off. So they can only answer "yes" or "no" to a question. But the analogue machine gives answers in degrees, according to the strength of an electric current. As the name suggests, they're *simulators. The *abacus is a digital-type device, and a slide-rule is an analogue.

John: It sounds as if analogue machines are closer to the way our brain works than digital machines.

Christine: Yes, I suppose so. But in practice they're only used by scientists and engineers. Almost all commercial machines are digital computers.

John: Well, thanks very much. You certainly know your stuff; you could always get a job as a guide here, if you're ever out of work!

EXERCISE 1: STRUCTURAL PRACTICE

Notice this structure from the conversation:

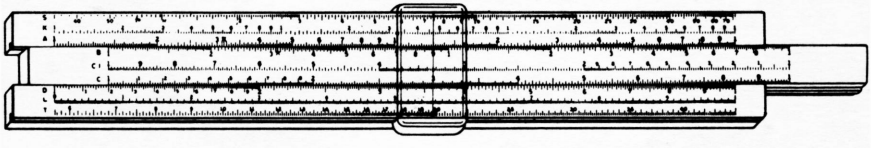
> *What's the difference between* analogue computers *and* digital computers?

Use this structure to respond to the following statements:

Example: These are analogue computers, and those are digital computers.
Response: What's the difference between analogue computers and digital computers?

Now, you do it.

1. These are analogue computers and those are digital computers.
2. This is an abacus and that's a slide rule.
3. This is the paper on "Computable Numbers" and those are von Neumann's papers.
4. This is Babbage's machine and these are other early machines.
5. This is the "on" state and this is the "off" state.
6. These are input units and those are output units.
7. This is the C.P.U. and these are the peripherals.
8. These are thermionic valves and those are transistors.



The slide rule—an analogue-type device

EXERCISE 2: PROGRESSIVE SUBSTITUTION DRILL

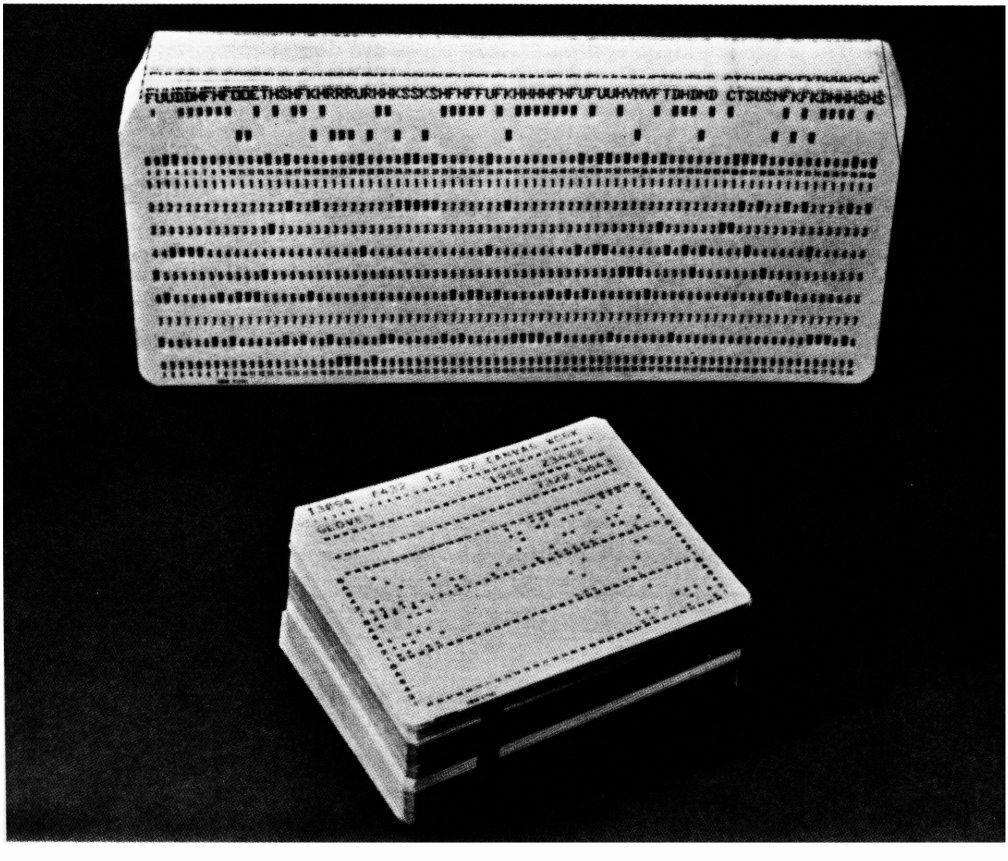
Statement: In practice they're only used by scientists.
Prompt: *engineers*
Response: In practice they're only used by *engineers.*

Now, you do it.

Statement: In practice they're only used by scientists.
Prompts:

1. engineers
2. theory
3. useful to
4. Theoretically
5. operated by
6. it should only be
7. must always
8. serviced

Punched cards



EXERCISE 3: FURTHER STRUCTURAL PRACTICE

Notice this structure:

It isn't only the development of electronics *that has made* modern computers *possible.*

Use this structure to make complete sentences:

1. analogue machines / scientific advances
2. transistors / fast circuits
3. stored programs / automatic processing
4. this invention / computing
5. thermionic valves / arithmetic circuits
6. punched cards / data processing
7. electronic devices / this development
8. magnetic tapes / bulk storage

Reading and Comprehension

Major inventions are usually the result of technology catching up with the ideas of theorists. Charles Babbage (1792–1871) showed that it was possible to make a machine perform arithmetical calculations according to a predetermined series of instructions. But this discovery was of little value until thermionic valves and, later, *bistable electronic circuits had been invented. At the same time, with the growth of industry and commerce in the nineteenth century, it became necessary to find a way of storing vast quantities of data in a form in which they could be easily retrieved and processed. Hermann Hollerith (1860–1929) used data stored in the form of holes punched in cards, an invention of the French weaver Joseph Marie Jacquard (1752–1834).

Theorists evolved the idea of the stored program; in the 1940's these three lines of research came together and the earliest true computers, such as ENIAC and EDSAC, were built. Since then things have developed so fast that we now talk about "third-" and "fourth-generation" computers. Technology has caught up with theory and every year there are new discoveries and inventions; *integrated circuits, *thin film, *tunnel diodes and super-cooled memory banks are some of the latest.

EXERCISE 4: COMPREHENSION QUESTIONS

1. What did Babbage's machine do?
2. What had to be invented before Babbage's discovery became useful?
3. What was Jacquard's invention?
4. When were the earliest true computers built?
5. What have replaced thermionic valves?
6. What became necessary in the nineteenth century, with the growth of industry and commerce?
7. Which generation of computers do we now talk about?
8. Name some of the latest discoveries and inventions in computers.

EXERCISE 5

Use the following words and phrases in sentences of your own to show that you understand their meaning and use.

1. commerce
2. clumsy
3. depend on
4. in practice
5. bistable
6. theorists
7. developments
8. major
9. thought it up
10. machine time

UNIT 3

PEOPLE WHO USE THE COMPUTER

Dialogue

John is being interviewed by a computer consultant.

Consultant: Our clients are companies that want to computerize, to switch over their information processing to machines. We study their business, and then tell them the best computer installation to buy for their purposes. Sometimes we tell them they don't need a computer at all, only punched card equipment, or we tell them that they should simply hire time at a *computer bureau.

John: So a lot of your work is systems analysis? You find out the way information is passed from one department to another, and then work out how the same system can be computerized?

Consultant: Right!

John: Then you're not the same as a service bureau?

Consultant: No, we don't have any machines of our own. A service bureau has its own machines, and sells machine time to other users at so much per hour.[1]

John: Does a service bureau have any programmers?

Consultant: Only if it also does program writing on a contract basis. Some companies—software houses—specialize in that sort of work.

John: It sounds to me as if these companies—and yours—only want people with previous experience. Do you think I'd do best to start with a manufacturer or a computer user after all?

Consultant: I'd suggest a manufacturer, John. For one thing, a computer manufacturer will give you all the basic training you need, free. Most of our staff began with manufacturers.

[1] so much per hour: a certain amount per hour

Computer users are another possibility, but they often try to recruit programmers from among their own staff.

John: What's the difference between a programmer's work with a manufacturer and a computer user?

Help for the designer

Consultant: A computer manufacturer doesn't only sell a collection of machines, he also sells a number of basic computer programs with them; things like *compilers, *utility routines and *applications packages. For instance, he might sell a package to handle the general run of pension fund calculations.[2] But a computer user, whether it's a bank or an engineering company, may find that its own pension fund is completely different from the general run; then it will want to have its own program written, rather than use the manufacturer's package.

So if you're working with a user, you'll be programming for his particular problems, but if you're working for a computer manufacturer, you'll be writing programs that can be more widely used.

[2] the general run of pension fund calculations: the form of pension fund calculations in general use

EXERCISE 1: STRUCTURAL PRACTICE

Notice this structure from the conversation:

Then you're *not the same as* a service bureau?

Use this structure to respond to the following statements:

Example: We're a computer consultancy.
Prompt: *service bureau*
Response: Then you're not the same as a service bureau?

Now, you do it.

1. We're a computer consultancy.	*service bureau*
2. It's a generalized package.	*this scheme*
3. He is the operator.	*the programmer*
4. Those are slow peripherals.	*fast peripherals*
5. Here is a console typewriter.	*an electric typewriter*
6. It's an applications package.	*a utility routine*
7. These are his needs.	*mine*
8. They are service bureaus.	*software houses*

EXERCISE 2: PROGRESSIVE SUBSTITUTION DRILL

Example: Some companies specialize in that sort of work.
Prompt: *this*
Response: Some companies specialize in *this* sort of work.

Now, you do it.

Statement: Some companies specialize in that sort of work.
Prompts:
1. this
2. Those
3. We
4. kind
5. program writing
6. Software houses
7. do
8. concentrate on

EXERCISE 3: FURTHER STRUCTURAL PRACTICE

Notice this structure from the conversation:

A computer manufacturer *doesn't only* sell machines, he *also* sells programs.

Use this structure to respond to the following questions:

Example: Does a computer manufacturer only sell machines?
Prompt: *programs*
Response: He doesn't only sell machines, he also sells programs.

Now, you do it.

1. Does a computer manufacturer only sell machines?	*programs*
2. Do they only want the manufacturer's package?	*this special program*
3. Do you only supply utility routines?	*applications packages*
4. Does the client only need a program?	*computer time*
5. Do data preparation staff only punch up the cards?	*the paper tape*
6. Do these users only have their own machines?	*their own programs*

Reading and Comprehension

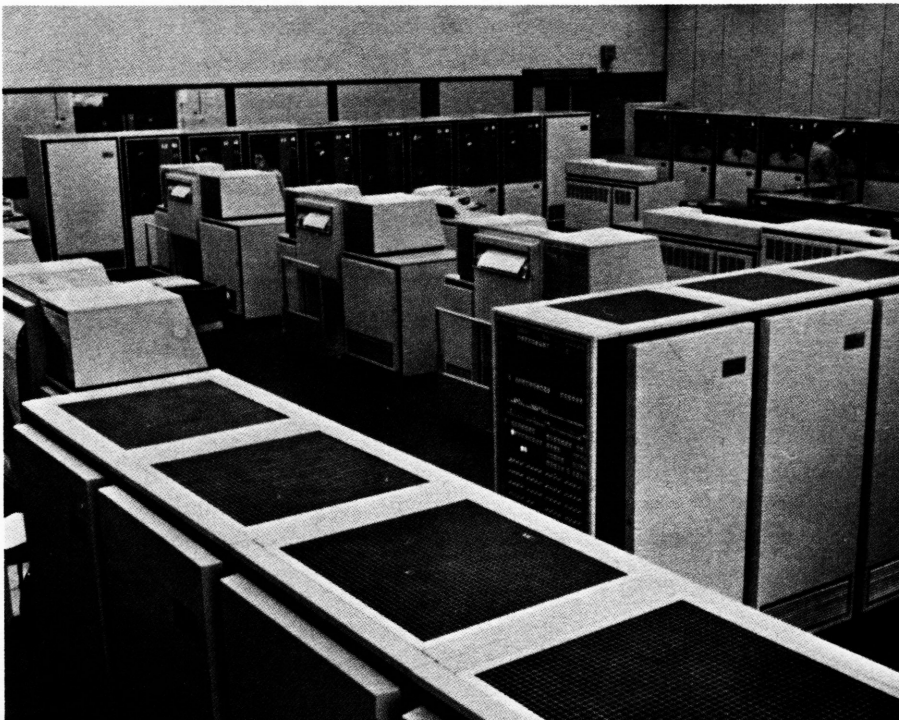
Every type of problem whose elements can be *quantified is capable of being solved by a computer. In factories and offices computer installations are taking over much of the manual and clerical work; in universities they are used not merely by chemists and physicists, biologists and economists, but also by members of the arts faculties to solve such problems as the authorship of ancient texts, or to hasten the translation of dead languages. The largest installations cost several million pounds, and even the smallest cost many thousands, so not all users can afford to buy their own machines. Instead they rent time from computer service bureaus, and if they cannot maintain their own programming staff, they get their programs written by software houses.

This is one of the trends of the future: large, central installations, perhaps with their own specialist programming staff, linked to remote users who can communicate their problems and get their answers over *transmission lines.

Already there are a great number of computer manufacturers, particularly in the United States, all offering machines with different performance figures and adapted to different needs. It is to help customers make the best choice among these machines that consultancy companies have been formed; for the computer may well be the most expensive piece of equipment the customer will ever buy.

EXERCISE 4: COMPREHENSION QUESTIONS

1. What sort of problems can be solved by a computer?
2. How much do the largest installations cost?
3. Where can users rent computer time?
4. What do software houses do?
5. What do consultancy houses do?
6. What sort of problems can computers solve in arts faculties?
7. How are computer installations linked to remote users?
8. Name some people in universities who use computers.



EXERCISE 5

Complete the following sentences, using the appropriate words from the list below:

a. is capable
b. hasten
c. cost
d. can afford
e. rent
f. get
g. help
h. are taking over

1. The largest ——— several million pounds, and even the smallest ——— many thousands.
2. They ——— time from computer service bureaus.
3. Consultancy companies ——— customers make the best choice.
4. Every problem whose elements can be quantified ——— of being solved by a computer.
5. They are used by members of arts faculties to ——— the translation of dead languages.
6. They ——— their programs written by software houses.
7. Computer installations ——— a lot of manual and clerical work.
8. Not all users ——— to buy their own machines.

UNIT 4

HARDWARE: THE CENTRAL PROCESSOR

Dialogue

John has taken a job with a computer manufacturer. The engineer is showing him round the machine room.

John: Just what is a memory bank?

Engineer: It's made up of thousands of tiny rings linked by a *matrix of wires, on which they're threaded. Each ring is a ferrite *magnetic core—it's made of ceramic coated with ferric oxide—which holds one *"bit" or *binary *digit of information. It can be in one of two states: either "on" or "off".

John: What does that mean exactly?

Engineer: It means it's magnetized either in one direction or the other. The signals sent by the crossed wires determine which way it's magnetized.

John: But why is each core threaded by several wires?

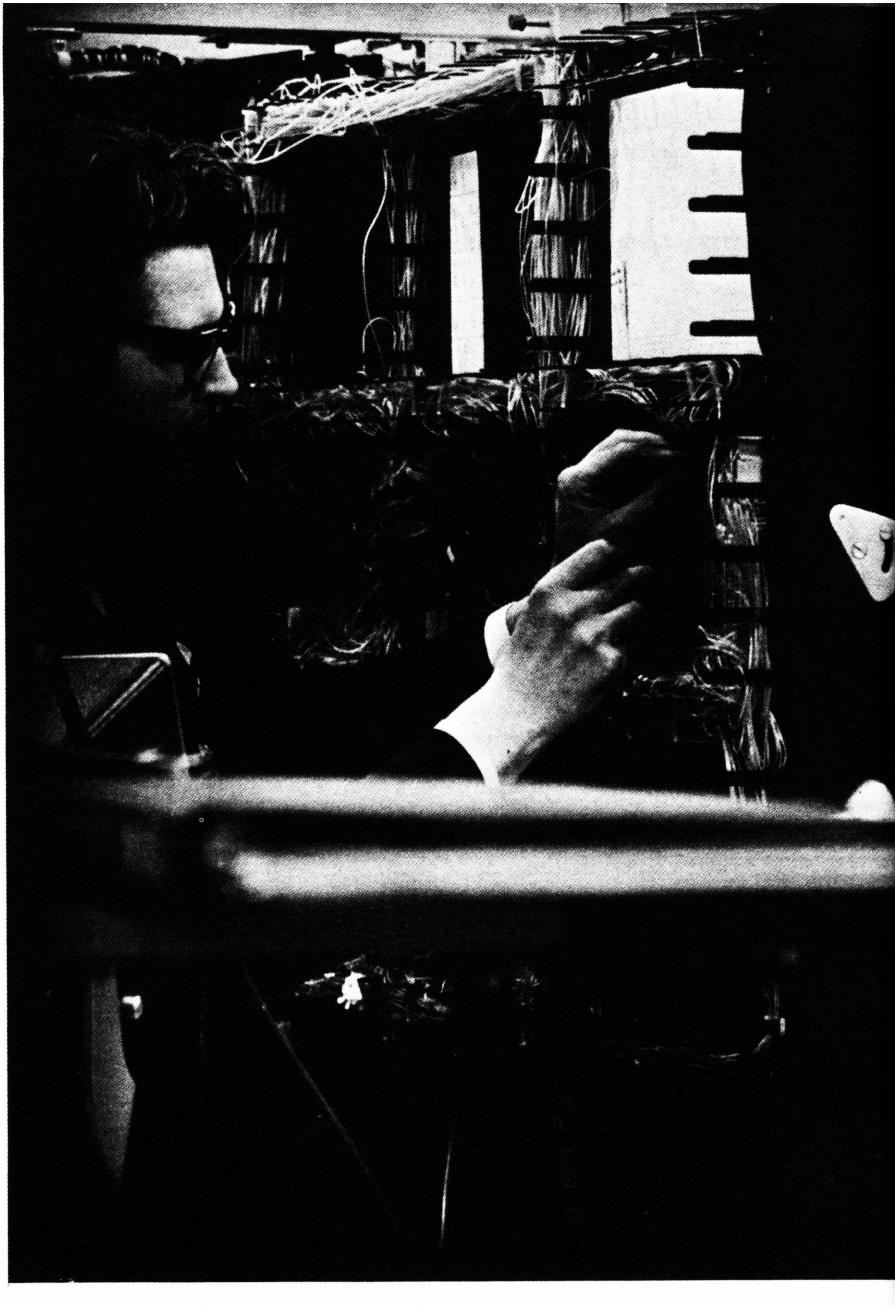
Engineer: Some of them are "read" wires—they simply transmit the present state of the core. Others alter the state. Others transmit the present state, and then regenerate it. Of course, you often don't have cores in modern machines, you have circuits printed on thin film.

John: And a program can test whether a core is "on" or "off"?

Engineer: Yes. Combine a set of cores into groups—six of them, say—and you can represent a character like a letter or a number.

John: What about the *logic and arithmetic units in the C.P.U.? Are they different?

Engineer: Yes, they're special circuits. Some of them are called *"gates". They're the means of implementing *Boolean logic. For instance, an AND gate means that if two input circuits both convey a signal, the single output circuit will also convey a signal. You've also got OR gates, *NAND, which means NOT AND gates, and several others. As for



Testing a circuit in the central processor

the arithmetic circuits, all decimal arithmetic can be converted to binary arithmetic, using a *radix of two instead of ten. Once you've done that, you can do any arithmetic operation by means of cores and circuits.

John: And what about the third part of the C.P.U., the *control circuits? What are they for?

Engineer: Well, they control the order in which instructions are obeyed, and synchronize them with a *clock pulse generated inside the C.P.U. They also control the peripheral units and perform the Read and Write instructions issued by a program.

John: So the C.P.U. is the real brain of a computer installation, and the peripherals are only the limbs?

Engineer: In general, yes. You *can* have logic and memory and control circuits in the peripherals—in fact, that's what you had in *tabulators, before computers were used. But the trend with computers, as with human beings, has been towards specialization, so we separate the thinking part of the system from the active part.

EXERCISE 1: STRUCTURAL PRACTICE

Notice this structure from the conversation:

Why is each core threaded by several wires?

Use this structure to respond to the following statements:

Example: Each core is threaded by several wires.
Prompt: *why*
Response: Why is each core threaded by several wires?

Now, you do it.

1.	Each core is threaded by several wires.	*why*
2.	A radix of two is used.	*when*
3.	The present state is altered by these wires.	*how often*
4.	Some circuits are called gates.	*why*
5.	The circuit is printed.	*what...on*
6.	The cores are magnetized.	*how*
7.	The clock pulse will be generated.	*where*
8.	The decimal value is converted.	*how*

EXERCISE 2: PROGRESSIVE SUBSTITUTION DRILL

Statement: Once you've done that you can do any arithmetic operation.
Prompt: *Now that*
Response: *Now that* you've done that you can do any arithmetic operation.

Now, you do it.

Statement: Once you've done that you can do any arithmetic operation.
Prompts:
1. Now that
2. calculation
3. mastered
4. learnt
5. When
6. a difficult
7. a complex
8. will be able to

EXERCISE 3: FURTHER STRUCTURAL PRACTICE

Notice this structure:

You often don't have cores, *you have* circuits printed on film.

Use this structure to make complete sentences

1. tape decks / magnetic discs
2. punched cards / paper tape
3. decimal arithmetic / binary arithmetic
4. typewriters / display units
5. computers / tabulators
6. keys to press / switches
7. these units / special circuits
8. valves / transistors

Direct access storage facilities

Reading and Comprehension

The central processor is the heart, or rather brain, of a computer installation. It controls the input/output peripherals, stores data in its memory banks, and obeys the instructions of a stored program according to a generated clock pulse. Since transistors, magnetic cores and printed circuits replaced the thermionic valves and complex wiring of the early machines, modern processors have become smaller; miniaturization of components makes desk computers and even wristwatch computers possible. This trend has been accelerated by the weight requirements of space rockets. The power of a processor, however, is partly a function of the number of binary bits of information it can hold; that is, the number of memory cells it has.

Although modern machines are becoming smaller, they are also becoming more and more powerful. Ten years ago the average commercial machine might have only ten thousand or so memory cells; today's machines may have several million. And because they are smaller, the electro-magnetic signals have a shorter distance to travel, which means that modern processors are also much faster. So third-generation computers, for this and other reasons, are often twenty times as fast as their second-generation predecessors.

EXERCISE 4: COMPREHENSION QUESTIONS

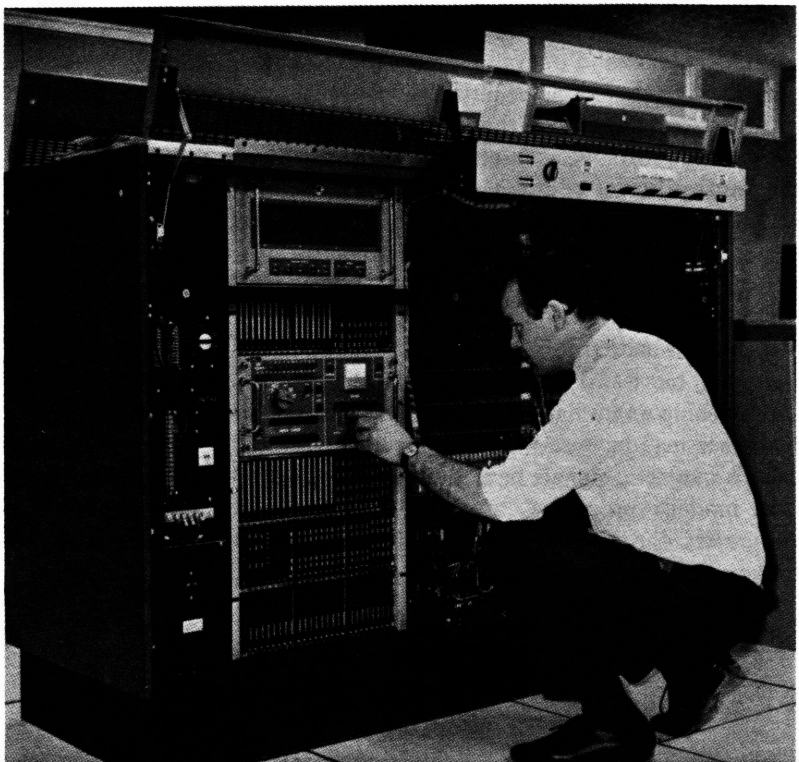
1. What is the heart of a computer installation?
2. Where is data stored?
3. What makes desk computers possible?
4. How many memory cells might a modern machine have?
5. Why does their small size make modern processors faster?
6. How much faster are third-generation computers than second-generation computers?
7. How has the development of space rockets affected the size of computers?
8. What have replaced thermionic valves and complex wiring?

EXERCISE 5

Use the following words and phrases in sentences of your own to show that you understand their meaning and use:

1. memory bank
2. trend
3. binary digit
4. predecessors
5. gates
6. miniaturization
7. combine
8. binary arithmetic

Inside the central processor



UNIT 5

HARDWARE: THE PERIPHERALS

Dialogue

Engineer: Here's the printer. The paper on the printer is called continuous stationery, because it's one long sheet which can be separated into pages by tearing along perforated lines. The printed line has a maximum of one hundred and twenty characters in it.

John: Those tall cabinets with windows must be magnetic tape decks. And this is a paper tape reader, isn't it?

Engineer: Yes, but in fact it's also a tape punch. With paper tape you often get the reader and punch in one unit, used both for input and output. As for cards, we haven't got any readers

or punches in this installation, though they have in the one you'll be using. Usually you only need one type of slow peripheral.

John: Is there any difference between cards and paper tape from a user's point of view?

Engineer: Well, for one thing, paper tape is cheaper. Then if you drop a spool, all you have to do is to rewind it; if you drop a pack of cards you may have to sort them into sequence order again. On the other hand, it's much easier to amend a file of punched cards; you just insert or replace them by hand. You've got to cut paper tape and splice in the new pieces. So there are advantages on both sides.

John: Those units over there, something like gramophone record players—are they the magnetic disc transports?

Engineer: That's right. These disc packs, or cartridges, can be replaced, so they're called exchangeable discs. But there are much bigger discs on horizontal spindles. They're called fixed discs because they can't be removed and replaced.

John: What's this long unit?

Engineer: It's a document reader.

John: You mean it can actually read hand-written documents—you don't have to punch your data in a special code?

Engineer: That's the idea;[1] but at present you still have to write your characters in a stylized way.

John: Does it work by *optical character recognition?

Engineer: Yes—O.C.R. It senses characters photo-electrically. An *M.I.C.R. or magnetic ink character recognition device works—well, its name tells you how. There's one other peripheral in this installation.

John: The console typewriter?

Engineer: That's right. It's much the same as an ordinary electric typewriter, except that it has a bigger character set.[2] And it can be used for output as well as input; the operators type in instructions for transmission to the central processor, and the processor types back replies. You can also have similar typewriters, called *interrogating typewriters, at locations remote from the machine room and linked to the C.P.U. by transmission lines. They're used for *realtime work.

[1] that's the idea: that's right
[2] character set: set of characters, i.e., type

EXERCISE 1: STRUCTURAL PRACTICE

Notice this structure from the conversation:

All you do is rewind *it.*

Use this structure to respond to the following statements:

Example: What do I do if I drop a spool?
Prompt: *rewind*
Response: All you do is rewind it.

Now, you do it.

1.	What do I do if I drop a spool?	*rewind*
2.	What do I do if I want individual sheets?	*separate*
3.	What do I do if I tear the paper tape?	*splice*
4.	What do I do if I need the engineer?	*call*
5.	What do I do if I drop the cards?	*sort*
6.	What do I do if I remove a disc pack?	*replace*
7.	What do I do if I need time at a computer bureau?	*hire*
8.	What do I do if I want different data discs?	*exchange*

Magnetic tape drive

Exchangeable disc store

EXERCISE 2: PROGRESSIVE SUBSTITUTION DRILL

Statement: Don't you have to punch the data in a special code?
Prompt: *way*
Response: Don't you have to punch the data in a special *way*?

Now, you do it.

Statement: Don't you have to punch the data in a special code?
Prompts:
1. way
2. write
3. characters
4. stylized
5. Doesn't he
6. like this
7. the programmer
8. Does

EXERCISE 3: FURTHER STRUCTURAL PRACTICE

Notice this structure:

It's not very *different from* an ordinary typewriter.

Use this structure to complete the following sentences:

1. Decimal arithmetic / rather / binary arithmetic
2. Logic units / entirely / control circuits
3. These stylized characters / altogether / handwriting
4. That magnetic tape deck / probably / this one
5. A computer consultant / very / a service bureau
6. Thermionic valves / completely / transistors
7. Punched cards / not so / paper tape
8. AND gates / a little / OR gates

Reading and Comprehension

Devices similar to today's input/output units were invented and used many years before computers. *Data processing (D.P.) installations based on punched cards used to have *verifiers, interpreters, *collators and other machines capable of sorting, reading or punching. They also had tabulators—machines that could read a pack of punched cards, tally certain columns as it went along, and print out totals at the end. To do this it needed a program in the form of a replaceable *plugboard that was specially wired up for each job, and to this extent a tabulator was a primitive computer.

Today's computers still make use of slow, or basic, peripherals such as card and paper tape units, and printers, but they also have a wide range of fast magnetic units such as tape transports, discs and drums. These magnetic devices are called *"backing-store" since they supplement the processor's core store. Other new devices include the *graphical or *visual display units, based on cathode ray tubes; document readers and two-way typewriters. And in the new field of remote computer usage even such standard telecommunications devices as the telephone, telegraph and Telex lines can now be used for the transmission of data to a central processor.

EXERCISE 4: COMPREHENSION QUESTIONS

1. Name some machines in data processing installations using punched cards.
2. In what form does a tabulator hold a program?
3. Name some basic peripherals.
4. Name some fast peripherals.
5. Why are magnetic devices called "backing-store"?
6. What new devices are based on cathode ray tubes?
7. What standard telecommunication devices can be used for data transmission?
8. What machine might be called a primitive computer?

EXERCISE 5

Complete the following sentences, using the appropriate words or phrases from the list below:

a. could read
b. make use
c. called
d. needed
e. have
f. invented
g. devices
h. wired up

1. They also ——— a wide range of fast, magnetic units.
2. These magnetic devices are ——— "backing-store".
3. To print out totals it ——— a program.
4. Today's machines still ——— of basic peripherals.
5. Other new ——— include visual display units.
6. The plugboard was specially ——— for each job.
7. A tabulator was a machine that ——— a pack of punched cards.
8. Similar devices were ——— many years before computers.

UNIT 6

THE COMPUTER OPERATOR

Dialogue

John has been introduced to the operating staff.

Shift leader: Hallo, John. You've been punching the cards for Tom's program, haven't you? How did it go?

John: It wouldn't have got on the machine at all but for the senior operator. He noticed I'd forgotten to supply some *parameter cards, so he punched them up himself and then ran the job.

Shift leader: We don't just hit keys on a console typewriter, you know! We're always correcting things like that. But I'd say the operator's main job is to maximize *throughput, to see that as many jobs as possible are run successfully on the machine during his shift.

John: What about machine faults—when there's a fault in the central processor, for instance? Can the operator do anything about those?

Shift leader: Oh, yes. Of course, faults in the hardware can only be cured by the engineer, but it's up to the operator to report them as soon as they occur and to switch over to alternative devices if possible: for example, a program that's failed on one tape deck could be re-run on another. Then sometimes the card reader will go wrong and chew up a few cards because they're bent, or worn—that's called a *card wreck.

John: Yes, that's what happened to me. And sometimes the operator has to repunch the mutilated cards and refeed them, doesn't he?

Loading the tape deck

Shift leader: Yes, but at other times that isn't possible, so you have to reverse the last card that was read to tell the programmer which one it was. The cards have notches in them, so you can tell when they are reversed. The operator's kept busy—just some of his duties are to reload the printer when *"Paper Low" is signalled, load and unload peripherals as messages are received from the machine, and watch to see if the peripherals are activated at the right time.

John: What about *multi-programming—running several programs at the same time? Doesn't that increase the operator's problems?

Shift leader: It certainly does. For one thing, it means he's kept busier than ever attending to the peripherals. For another, he has to try and achieve a good *"job mix". For instance, you have to try and ensure that you don't run two programs at the same time which both use a card reader when you have only got one card reader available.

John: I've heard that the problems of operating big, multi-programming machines are so complex that some people are writing programs called *"Operating Systems", to solve them. Will they make operators obsolete?

Shift leader: They'll certainly take away a lot of our functions—but I expect there'll be new ones!

EXERCISE 1: STRUCTURAL PRACTICE

Notice this structure from the conversation:

It's up to the operator *to* report *them.*

Use this structure to respond to the following statements:

Example: Who reports faults in the hardware?
Prompt: *operator*
Response: It's up to the operator to report them.

Now, you do it.

1.	Who reports faults in the hardware?	*the operator*
2.	Who investigates problems in the company?	*the systems analyst*
3.	Who alters mistakes in the punched cards?	*the data preparation staff*
4.	Who corrects errors in the software?	*the programmer*
5.	Who refeeds cards in the card reader?	*him*
6.	Who reports failures in the tape decks?	*us*
7.	Who detects faults in the console typewriter?	*the person using it*
8.	Who repairs breakdowns in the new machine?	*the manufacturer*

EXERCISE 2: PROGRESSIVE SUBSTITUTION DRILL

Statement: We don't just hit keys on a console typewriter.
Prompt: *They*
Response: *They* don't just hit keys on a console typewriter.

Now, you do it.

Statement: We don't just hit keys on a console typewriter.
Prompts:

1. They
2. type instructions
3. only
4. remote terminal
5. send data
6. always
7. You
8. by means of


ICL

EXERCISE 3: FURTHER STRUCTURAL PRACTICE

Notice this structure:

He's kept busier than ever attend*ing* to the peripherals.

Use this structure to make complete sentences:

1. (look after) the console
2. (load) the input/output units
3. (run) programs
4. (switch) tape decks
5. (service) the peripherals
6. (obey) the programmers' instructions
7. (take care of) the installation
8. (watch for) program events

Reading and Comprehension

In the days of the early machines programmers used to do their own operating; they both wrote the software and operated the hardware. Soon the two functions became so complex that they had to be separated, and nowadays programmers may rarely see the machines on which their programs are run.

A computer operator's job is both manual and intellectual. He needs to work fast and efficiently at manual jobs—changing magnetic tapes and discs, for instance, but he also needs to know a lot about the programs he is asked to run so that he can interpret the programmer's instructions correctly. Program events are automatically *logged by the machine, but the operator has to report on the result of the run, and the reason for any failure. He has to recognize as soon as possible a malfunction in the machine and take appropriate action to *"fail-soft". In a multi-programming environment he has to work out optimum job-mixes. In fact, in large installations his job has become so complex that special programs—operating systems—have been written to do much of it for him. Nevertheless, the profitability of an installation depends, in the first place, on the computer operators—even though most of them do not stay at this job long, but go on to do other types of work in the computing field.

EXERCISE 4: COMPREHENSION QUESTIONS

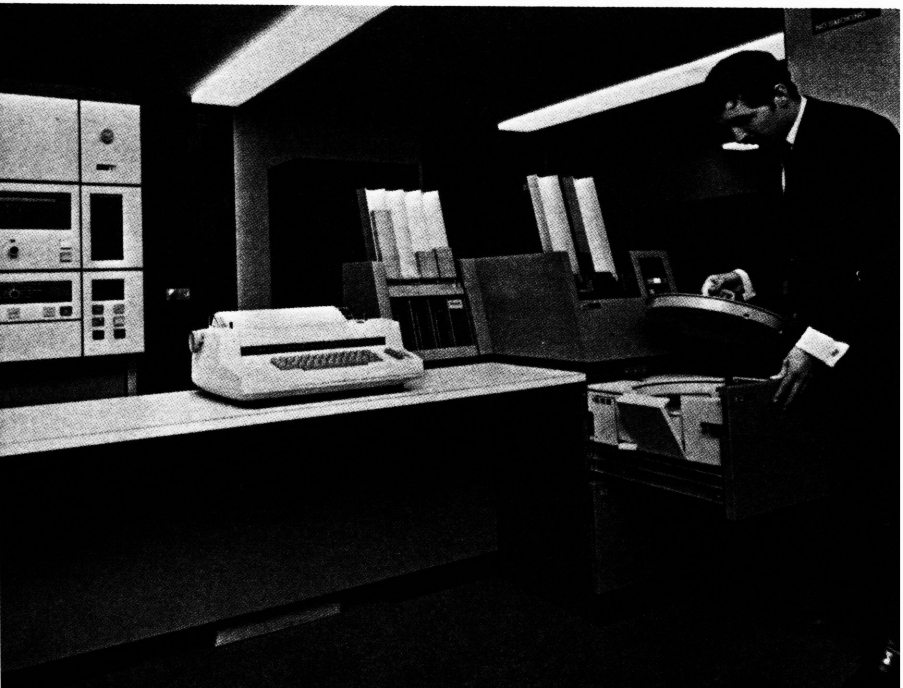
1. When did programmers do their own operating?
2. Name a manual part of the operator's job.
3. Why does an operator need to know a lot about the programs he has to run?
4. What sort of events are logged by the machine?
5. On what does the operator have to report?
6. What does he do when he recognizes a malfunction?
7. In what circumstances does an operator have to work out an optimum job-mix?
8. What are the special programs called that do much of an operator's job for him?

EXERCISE 5

Use the following words and phrases in sentences of your own to show that you understand their meaning and use:

1. signalled
2. manual
3. in fact
4. during
5. reload
6. mutilated
7. appropriate
8. environment
9. job-mixes
10. work out

Loading a disc



ICL

1900 Series
Program operating instruction sheet
LOAD and GO

Section A

Program control letter: A

Programmer	A. K. BANSAL	Program type	LOAD and GO
Estimated running time	5 minutes	Program name	PACH
Maximum running time	10 minutes	Job code	0DD/AAB/

Section B **Special instructions**

Load 'MT. 262007' as 1st 'SCRATCH TAPE' on-line.

Section C **Peripheral requirements**

								others	
Card reader	1	Paper tape reader		Cassette stations		Magnetic tape units	3		
Card punch		Paper tape punch		E.D.S. Transports		Line printer	1		

Section D **Running requirements**

Program priority	Core	Drum	Program library	No. of scratch tapes
50	12864			

		M.S.N.	w.p.	M.S.N.	w.p.	M.S.N.	w.p.	M.S.N.	w.p.
Magnetic media		262420	NO	262163	NO	262007	YES		

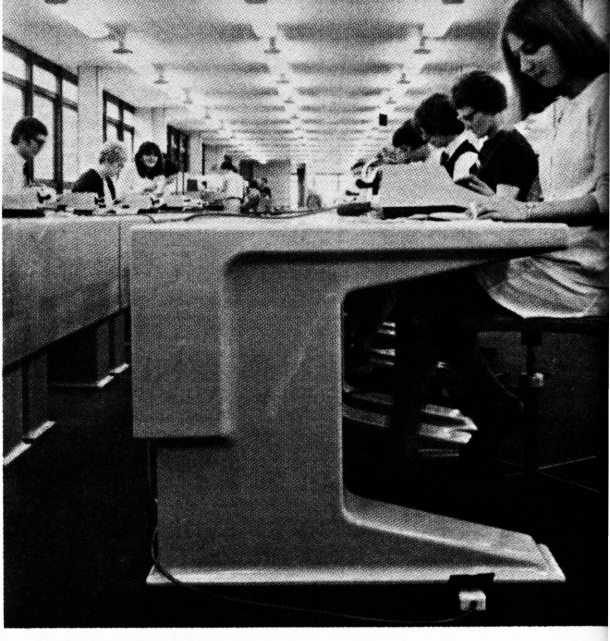
Section E **Operating instructions**

Step	Action	Event
1.	FI # PACH # CCBU from MT. 262420	HALTED :- LD
2.	Load cards 'PACH' in CR.	
3	ON # PACH 1	
4.	GO # PACH 21	HALTED :- OK

Error action	Event	Action
	HALTED :- C1	GO # PACH

FORM 14/95(8.69)

A completed instruction sheet—this tells the operator how to run the program



UNIT 7

DATA LOGICAL

Dialogue

John calls at the office of the Data Processing Manager.

D.P.M.: Ah, you must be John Codewell, from software production. You've transferred to us for the week prior to your programming course?

John: Yes, it's so that I can get to know what "data" really is.

D.P.M.: It's simply another word for the information, or an extract of information, which is regarded as the raw material for some transformation or updating process. "Data" is also used for the product of that process. It's like this: I could get a good picture of you as an employee in ordinary conversation. But if I am to deal in several ways with several hundred employees, I need information in a condensed and standard form; so your personnel card, which I have here, is drafted that way.

John: And that's data on it?

D.P.M.: Yes; your name, address, age, qualifications, staff number, date of joining, salary, etc.

John: I know about the binary nature of the computer's electronics and that the input is always a long string of binary digits. But how would it know from my record which group represents, say, my age?

D.P.M.: It's all a question of the organization of the data, its *format on the input media and an understanding by the programmer of that format, and of the implied structuring of the data.

But let's start at the beginning. We can get the computer to accept its bits in groups of six.

John: Let me see, that would give 2^6, that's—um—64 possible groups?

D.P.M.: That's right. Now the elements used in communication by human beings are letters and digits, from which we construct words and numbers. Our set of sixty-four, therefore, will cover the twenty-six letters, the ten digits and twenty-eight special *characters—like the hyphen, the percent sign and the decimal point.

John: Yes; looking at my personnel card again, I can see that there's a group of such characters in each of the printed boxes.

D.P.M.: Yes, and each is a unit of information. We call them "fields". A field will get added, compared, moved about and listed by the computer program; that's to say, it's the basic unit to which program instructions refer. But a section of program operates upon a larger grouping of data. We consider that groups of fields make up a *"record".

John: Do you mean that we would write a program for processing a record, and then use it on any number of records of the same kind?

D.P.M.: That's right. Such a collection is called a *"file". I shall now put your personnel record into its proper place in my file of personnel record cards.

John: Now let me just check. A file is a collection of records; each record is made up of fields; a field consists of characters, and a character is represented to the computer as a group of bits—is that right?

D.P.M.: That's the logical arrangement. You'll also have to learn about its physical preparation. There are forty girls out there in the data preparation department, all busy punching data like this into cards for the computer, and eight more busy packing printed reports off to customers. They're very important to us. After your course the writing of programs will seem all-important to you, but our customer's requirement is to get his data processed quickly and cheaply. He sees the program and the computer just as a means to an end.[1]

John: That's a sobering thought.

[1] a means to an end: a way of achieving an important aim

EXERCISE 1: STRUCTURAL PRACTICE

Notice this structure from the conversation:

I know about the nature *of* the electronics.

Use this structure to respond to the following questions:

Example: Do you know about the electronics?
Prompt: *nature*
Response: Yes, I know about the nature of the electronics.

Now, you do it.

1.	Do you know about the electronics?	*nature*
2.	Have you been told about the characters?	*stylization*
3.	Have you been informed about the data?	*organization*
4.	Have you learnt about that machine?	*working*
5.	Have you got the idea of the console?	*operation*
6.	Do you understand the memory bank?	*construction*
7.	Have you read about the disc packs?	*replacement*
8.	Have you found out about the paper tape?	*preparation*

EXERCISE 2: PROGRESSIVE SUBSTITUTION DRILL

Statement: We could write a program for processing a record.
Prompt: *They*
Response: *They* could write a program for processing a record.

Now, you do it.

Statement: We could write a program for processing a record.
Prompts:

1.	They	5.	to process
2.	design	6.	update
3.	use this information	7.	should
4.	The programmer	8.	might

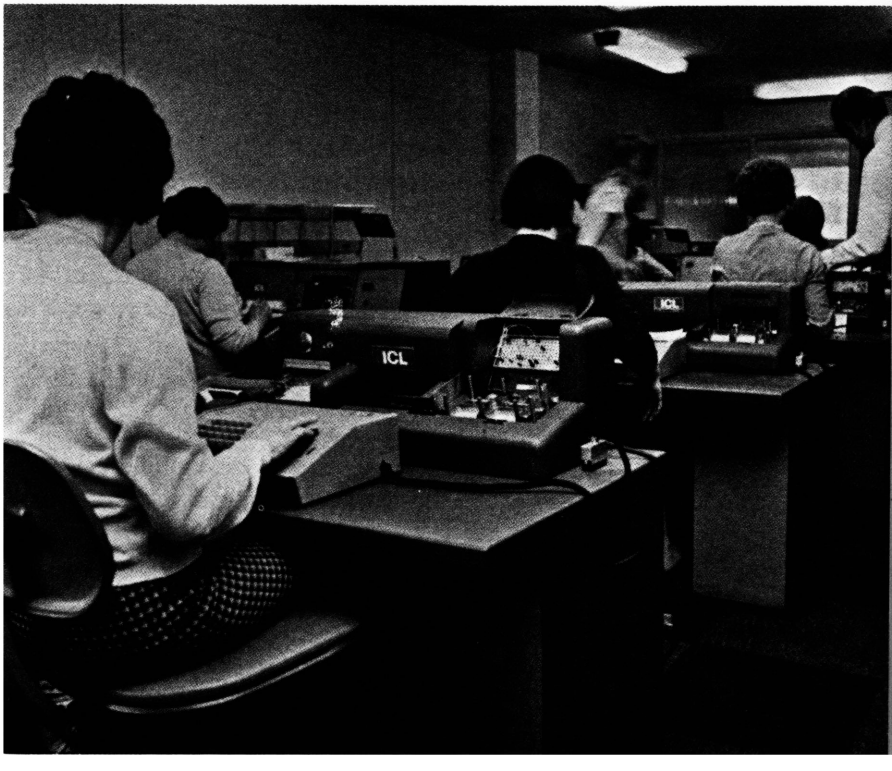
EXERCISE 3: FURTHER STRUCTURAL PRACTICE

Notice this structure:

Looking at this card *I can see* characters *in the* boxes.

Use this structure to make complete sentences:

1. this printout / numbers / columns
2. the output device / the holes / paper tape
3. the reader / the notches / cards
4. your layout chart / fields / records
5. the printer / paper / stacker
6. his listing / errors / format
7. that console log / gaps / schedule
8. the manual / sixty-four symbols / character set



Reading and Comprehension

Computers in general are machines for converting information (data) from one form into another more useful form. Efficiency demands that data for the computer must be organized and structured.

The elements which we use in general communication are letters and digits, which we group into words and numbers. For convenience of handling on the machine, pieces of information relating to one transaction will themselves be grouped into a comprehensive document or record; the parts of this record are known as "fields". For example, an electricity invoice record will contain fields like: date, name, address, meter readings, rates, total charge. When prepared for the computer, such records will be batched in collections called "files".

The *registers which hold numbers within the arithmetic unit of the computer are composed of electronic circuits and are the most expensive way in which to hold data. Even within the computer the data (and programs) not in use at any instant are normally relegated to a memory of cheaper construction; revolving magnetic discs, for example, which are cheap but slower in access than networks of magnetic cores. Input/output devices are really equipment connecting the computer to even larger and cheaper stores. Data not required from one day to the next can be transferred out of internal memory on to reels of magnetic tape and the cost of the transfer in time and equipment is worth it, because the computer is freed entirely for other work. If the volumes of data involved are low, it is even cheaper to store it on punched cards or punched paper tape.

Another consideration governs choice of input and output media, namely that the computer has to communicate with man. The line-printer is the common device by which the computer makes data visible in bulk to human beings, but, for smaller quantities the console typewriter can be used. For data which is constantly changing or is in graphical rather than character form cathode ray tube displays are used.

The following table (probably outdated even by the time this page is printed) gives some typical characteristics of the various data media.

	Capacity: characters per square inch	Cost in £ per 1000 characters	Access to characters per second	Serial or direct access
Printer	60	·00026	2,400	Serial
Punched cards	3	·0078	1,200	Serial
Magnetic tape	2000	·0003	80,000	Serial
Magnetic disc	2000	·18	200,000	Direct
Magnetic cores	625	750·	2,000,000	Direct

EXERCISE 4: COMPREHENSION QUESTIONS

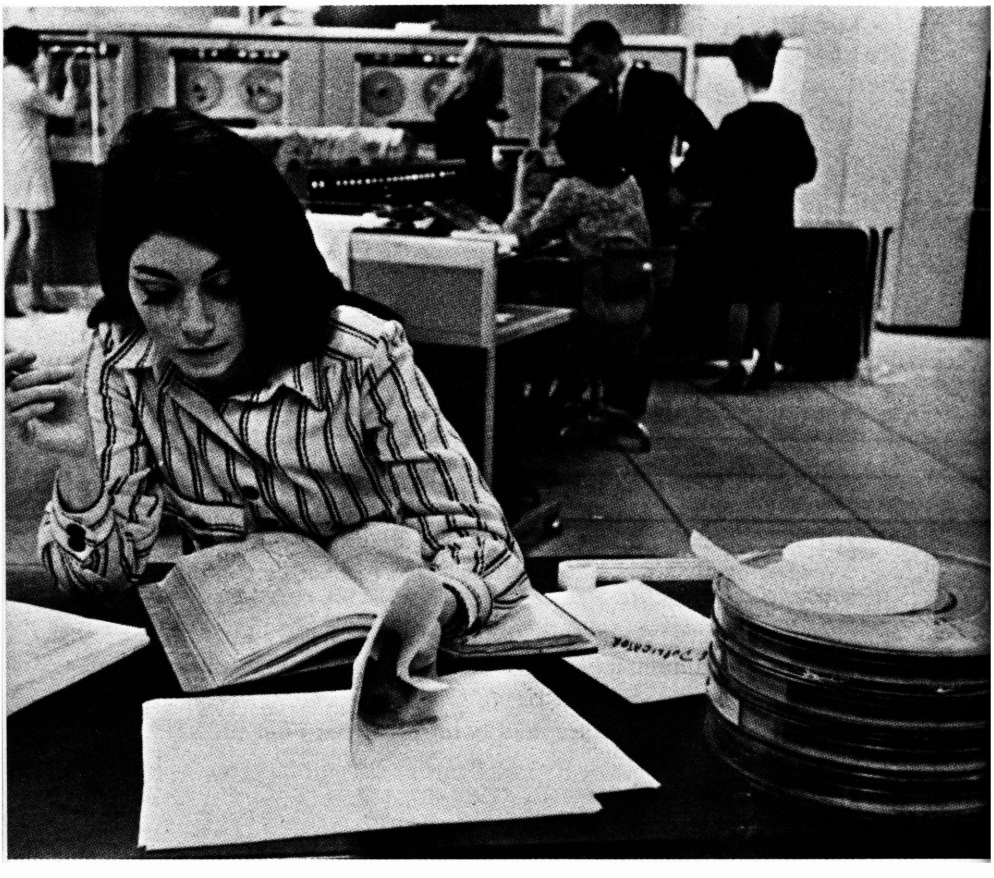
1. What elements do human beings use in general written communication?
2. What is the name for a collection of records?
3. What is the most expensive way of holding data in the computer?
4. Do magnetic discs give faster access to data than magnetic cores?
5. To what media could data be transferred to free the computer for other work?
6. By what common device does the computer make large quantities of data visible to human beings?
7. What device is used for output in graphical form?
8. Can magnetic tapes be accessed both serially and directly?

EXERCISE 5

Use the following words in sentences of your own to show that you understand their meaning and use.

1. record
2. converting
3. relegated
4. access
5. communicate
6. visible
7. characteristics
8. registers

Checking a program



UNIT 8

ORDER CODES

Dialogue

John is on a programming course. It is lunchtime.

John: Can I drink my coffee with you, Hal? I do think you're lucky to have programmed other computers before this one.

Hal: Yes, it does mean I only have to familiarize myself with another code; you're having to learn the art of programming as well.

John: Are *order codes very different for the various makes of computer?

Hal: They're similar in that[1] they take one *instruction at a time from the stored program, obey it and then pass to the next. But the codes do vary considerably from one type of machine to another in format, especially the number of addresses in each instruction.

John: Ours has a *"function", one "store address" and the number of a special *register.

Hal: Yes, you could call it a one-and-a-half address instruction format. Designers will try to utilize an instruction length which corresponds to the word length which is most economical for storing data; then the same physical storage can handle either data or program.

John: Do all the sets include the same kind of functions, like "add" and "read"?

Hal: They vary considerably in the number of functions—from a

[1] in that: in the way that

dozen up to a hundred or more. In the large sets, though, many functions are just variations on a basic set. They will all have a set of arithmetic instructions: "add", "subtract", "multiply" and "divide".

John: They couldn't do without some kind of "read" and "write" instructions for getting new data in and putting results out, could they?

Hal: That's true, and then there's a very important *subset for making comparisons, like checking if an item of data equals zero or if two items of data are equal to each other. Most important of all are the instructions for changing the route through the program according to the result of such a test. And now we've mentioned the instructions that are common to all computers.

John: It's these *branching instructions that make programming so interesting. The computer's ability to skip and *loop back on instructions, and choose alternate paths, is what makes it so much more versatile than earlier mechanical means of processing data.

EXERCISE 1: STRUCTURAL PRACTICE

Notice this structure from the conversation:

They couldn't do without "read" and "write" instructions, *could they*?

Use this structure to respond to the following statements:

Example: Sets are given "read" and "write" instructions.
Response: They couldn't do without "read" and "write" instructions, could they?

Now, you do it.

1. Sets are given "read" and "write" instructions.
2. Programmers use this code.
3. The machines need regular servicing.
4. New staff are given special training.
5. Modern machines are equipped with transistors.
6. The manager wants this information.
7. The operator can use alternative devices.
8. This client wants a special program.

EXERCISE 2: PROGRESSIVE SUBSTITUTION DRILL

Statement: The codes do vary considerably.
Prompt: *These*
Response: *These* codes do vary considerably.

Now, you do it.

Statement: The codes do vary considerably.
Prompts:
1. These
2. a lot
3. functions
4. The set's
5. Its
6. may
7. might
8. differ

EXERCISE 3: FURTHER STRUCTURAL PRACTICE

Notice this structure:

I do think you're lucky to have programmed *other* computers.

Use this structure to make similar sentences with the following words and phrases:

1. written / programs
2. learnt / machine codes
3. worked for / employers
4. used / computers
5. studied / instruction sets
6. practised with / systems
7. familiarized yourself with / order codes
8. mastered / functions

```
0112                LDN   1   1
0113                ADS   1   PAGE
0114                LDX   1   PAGE
0115                STO   0   LINK
0116                STO   1   BIN
0117                CALL  0   BINDEC
0118                LDX   1   DEC+1
0119                STO   1   HEAD+29
0120                LDCT  1   7
0121                LDN   2   0
0122                SUSBY     LPO
0123                LDX   0   LPCONA+1
```

Assembly code

Reading and Comprehension

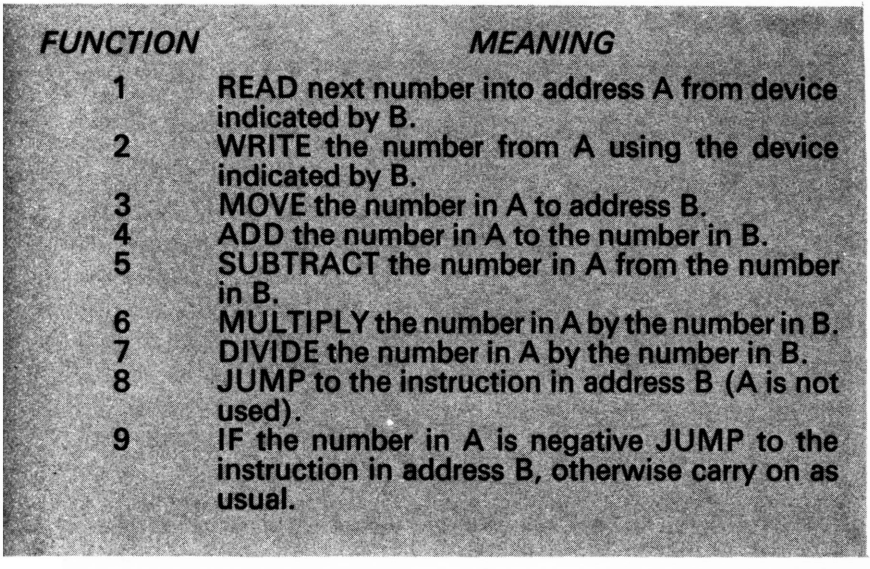
The store in many computers is organized as a collection of "words" each composed of binary bits. A word may take on the representation either of an instruction or a piece of data. Each word is referred to by its "address". As an example we may have a store in an imaginary computer set up with the following values in the given addresses.

Figure 1

ADDRESS	*CONTENTS*
01	11071
02	31011
03	51211
04	91107
05	51311
06	91108
07	31410
08	21072
09	80001
10	00015
11	00015
12	00021
13	00040
14	99999

This does not look very meaningful but it is the circuitry of the control unit in the computer which gives these numbers meaning. In our imaginary computer the five digits in each instruction are regarded as follows: digit 1 indicates the "function", digits 2 and 3 a first address (A), digits 4 and 5 the second address (B). The following table shows the possible functions and, for each, the use that is made of the two addresses.

Figure 2

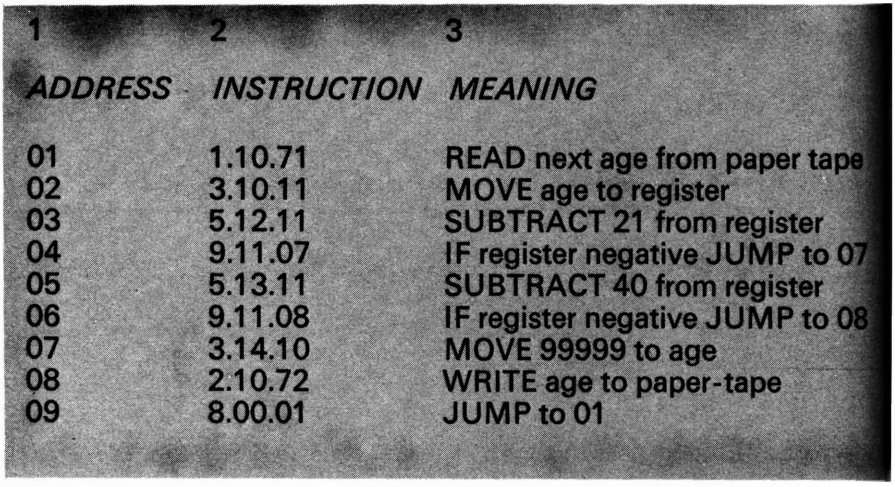


FUNCTION	MEANING
1	READ next number into address A from device indicated by B.
2	WRITE the number from A using the device indicated by B.
3	MOVE the number in A to address B.
4	ADD the number in A to the number in B.
5	SUBTRACT the number in A from the number in B.
6	MULTIPLY the number in A by the number in B.
7	DIVIDE the number in A by the number in B.
8	JUMP to the instruction in address B (A is not used).
9	IF the number in A is negative JUMP to the instruction in address B, otherwise carry on as usual.

Now consider Figure 1. The instruction in address 03 is 51211. 5 is the function code, 12 is address-1 and 11 is address-2. Figure 2 tells us that the computer will interpret this as "Subtract the value in word 12 from whatever value is in word 11. In the state shown in Figure 1 this subtracts 00021 from 00015 leaving −00006 in word 11. Our computer is designed to obey the instructions sequentially from word 01 onwards unless it meets a function 8 or 9 which can change the sequence.

The contents of words 01–09 in Figure 1 is, in fact, a program of instructions capable of reading in a string of numbers indicating people's ages and outputting a corresponding string in which any age outside the range 21 to 60 is replaced by the value 99999. The words 12–14 are used for storing useful constants. The contents of words 10 and 11 change as the program progresses. If the reader has the patience he can follow the behaviour of our imaginary computer on this particular program; Figure 3 will help him. Column 4 shows how the significant values change as the program operates on age 00015. Work out what happens when other ages are read in. Columns 5, 6 and 7 indicate which instructions in the loop are obeyed, for different age groups.

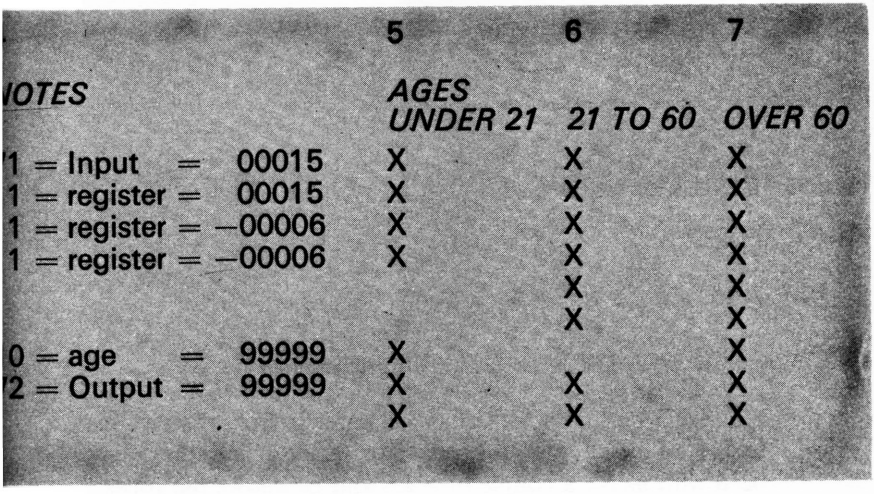
Figure 3



1	2	3
ADDRESS	*INSTRUCTION*	*MEANING*
01	1.10.71	READ next age from paper tape
02	3.10.11	MOVE age to register
03	5.12.11	SUBTRACT 21 from register
04	9.11.07	IF register negative JUMP to 07
05	5.13.11	SUBTRACT 40 from register
06	9.11.08	IF register negative JUMP to 08
07	3.14.10	MOVE 99999 to age
08	2.10.72	WRITE age to paper-tape
09	8.00.01	JUMP to 01

EXERCISE 4: COMPREHENSION QUESTIONS

1. What representations can the words of computer store take on?
2. What are the three elements of the instruction format in our imaginary computer?
3. Name the usual arithmetic functions.
4. In the given order code, which instructions can interrupt the natural sequence of instructions?
5. What is the name for the number which references a word in storage?
6. What gives meaning to the numbers making up the computer instruction?
7. Are the nine instructions of our imaginary code adequate for writing programs?
8. Which of the instructions in the given set would be used to put several copies of a piece of data in several different storage words?



	5	6	7
IOTES	*AGES*		
	UNDER 21	*21 TO 60*	*OVER 60*
1 = Input = 00015	X	X	X
1 = register = 00015	X	X	X
1 = register = −00006	X	X	X
1 = register = −00006	X	X	X
		X	X
		X	X
0 = age = 99999	X		X
2 = Output = 99999	X	X	X
	X	X	X

EXERCISE 5

Use the following words and phrases in sentences of your own to show that you understand their meaning and use.

1. jump instruction
2. address
3. format
4. function
5. storage word
6. order-code
7. basic
8. representation

UNIT 9

THE PROGRAMMER

Dialogue

Anna, a team leader, is talking to a trainee programmer.

Anna: Charlie, this *specification has just come in from the systems analyst. I'd like you and Roger to tackle it together; he's been with us for a couple of years and he knows the ropes.[1]

Charlie: Good. I see it contains all the file descriptions and printer layouts.

Anna: Yes, but you'll have to work out your own *algorithm and design intermediate tables. I believe in plenty of time, so I'm allowing you and Roger six weeks to plan the job; you should have the general and the detailed *block diagrams complete at the end of that time. The better you organize the job, the less time you'll spend later on getting the program to work.

Charlie: Have you set targets for all the stages?

Anna: Yes, I've represented it all here in this *PERT network. You see, I expect you to start testing after one week of coding, although it will be three weeks in all before you have the coding completed.

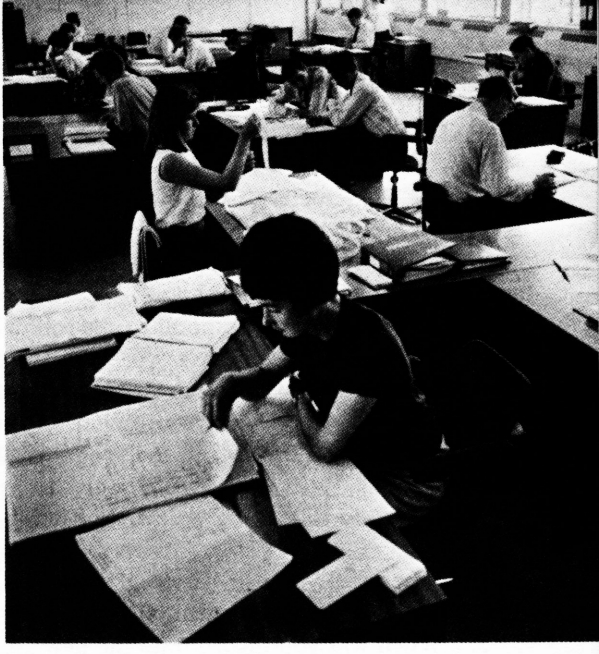
Charlie: Why not finish all the coding before we test?

Anna: Well, on the priority for this job you'll only get one test run on the computer each evening. It won't take a whole day to check each test run and make the corrections, so you'll have to overlap testing with coding. Besides, tackling the program in *modules will make it easier to understand.

Charlie: When do you think we'll be ready for a full system test?

Anna: I should imagine after about five weeks of testing. I hope

[1] knows the ropes: knows what to do



Programming

you'll be prepared for some overtime at the weekend then, when the computer's free, so that we can check out the system in one further week.

Charlie: And then we've finished?

Anna: Not by a long way;[2] every program must be thoroughly documented. You must write it up so that any programmer can pick up your *documentation, and determine from it how your program works; then he'll be in a position to mend it if it goes wrong, or extend it when new facilities have to be incorporated.

Charlie: What happens to our program meanwhile?

Anna: Well, we'll do some test runs for one week, and if that shows it's O.K., then we'll run in *parallel for six weeks.

Charlie: That means processing the real data each week both on the existing punched card system and in the computer, doesn't it?

Anna: Yes. This will not only give a thorough proof of your program, but it will also show if your operating instructions are comprehensible. We'll only run live on your program[3] when it's completely satisfactory.

Charlie: Fine, and since it's my first job, I'd like to give the report at your weekly progress meeting.

[2] Not by a long way: not nearly
[3] run live: use for real processing unsupported by any other system

EXERCISE 1: STRUCTURAL PRACTICE

Notice this structure from the conversation:

We'll only run the program *when* it's satisfactory.

Use this structure to respond to the following statements:

Example: When will you run the program?
Prompt: *satisfactory*
Response: We'll only run the program when it's satisfactory.

Now, you do it.

1.	When will you run the program?	*satisfactory*
2.	When will you test the system?	*assembled*
3.	When will you incorporate the new facilities?	*prepared*
4.	When will you start the project?	*scheduled*
5.	When will you examine the layouts?	*designed*
6.	When will you use the computer?	*free*
7.	When will you process the data?	*organized*
8.	When will you check out the system?	*completed*

EXERCISE 2: PROGRESSIVE SUBSTITUTION DRILL

Statement: The better you organize the job, the less time you'll spend coding.
Prompt: *project*
Response: The better you organize the *project*, the less time you'll spend coding.

Now, you do it.

Statement: The better you organize the job, the less time you'll spend coding.
Prompts:

1. project
2. work
3. more thoroughly
4. carefully
5. we'll take
6. we
7. checking
8. testing

EXERCISE 3: FURTHER STRUCTURAL PRACTICE

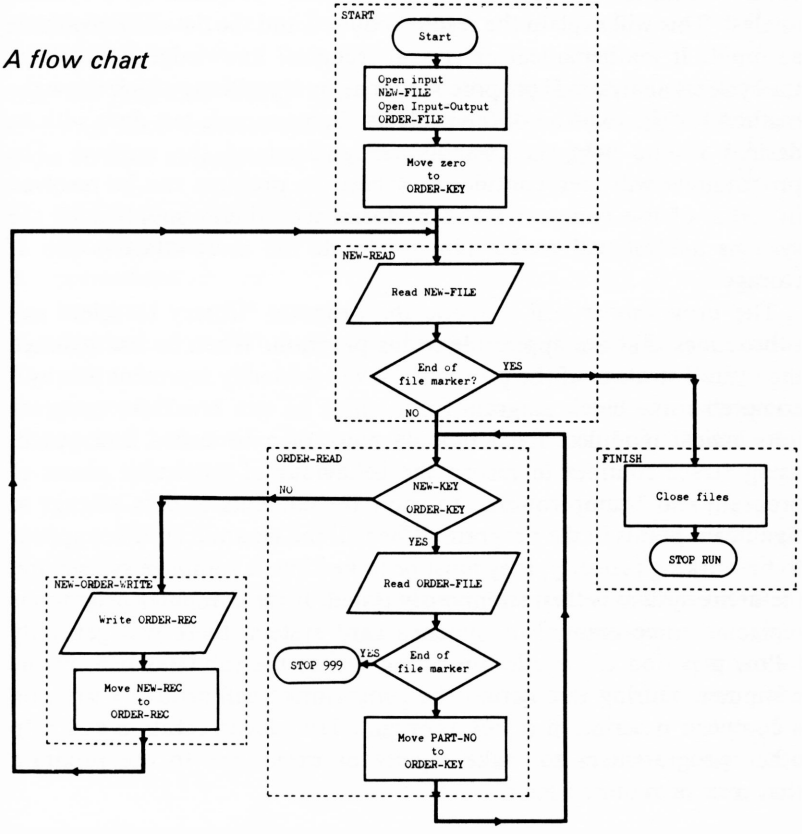
Notice this structure:

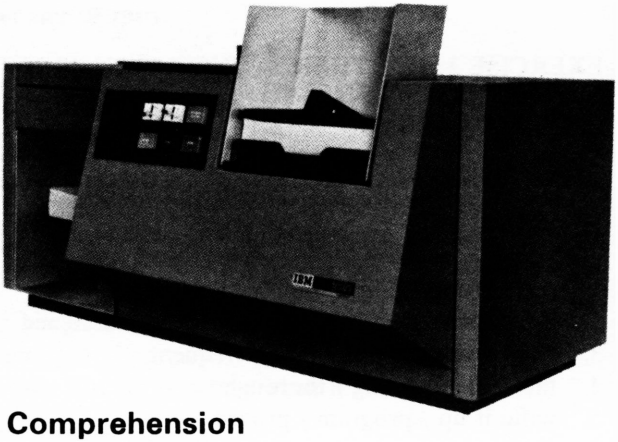
I expect you to start testing *although your* coding *will not be* complete.

Use this structure to make complete sentences:

1. start documenting / testing / finished
2. describe an algorithm / print layouts / designed
3. work overtime / day runs / frequent
4. tackle it / training / thorough
5. write it up / program / proved
6. repair errors / documentation / perfect
7. run live / new facilities / incorporated
8. begin planning / targets / fully set

A flow chart





A card reader

Reading and Comprehension

The scope of the programmer's job will vary with the size of the organization he works for, but generally programming requirements will come to him in the form of a "specification" prepared by a systems analyst. This will explain the results required and the data files available as input. If mathematical or special technical knowledge is involved, the systems analyst will propose formulae or algorithms which show the method for the solution of the problem. Some sample test data with its desired results help the programmer understand the method. The programmer will then consider how best the problem can be resolved in terms of the computer; and performance criteria supplied by the systems analyst will enable him to decide the most effective use of storage.

The programmer will examine the program *library to select any subroutines that are applicable to his program. When he has planned the overall strategy of the program, he will probably represent this by a comprehensive block diagram from which he can break the program into logical modules. Each module must then be coded and tested, using *trace routines to record the behaviour of particular pieces of program and *dump routines to show the contents of data storage at significant points in the program. When all the separate modules appear to be working properly, they must be linked into a complete system and the entire system is then strenuously tested. If the computer program is replacing some manual or punched card system, there will generally follow a period of parallel operation before the job runs live on the computer. During this period the programmer will probably write up a complete description of the program. This documentation will help other programmers to make repairs or extensions to the program that may be required later.

EXERCISE 4: COMPREHENSION QUESTIONS

1. What makes the scope of the programmer's job vary?
2. How will the programming requirement come to him?
3. Might the systems analyst propose formulae and algorithms?
4. What enables the programmer to decide the most effective use of storage?
5. What will the programmer select from the program library?
6. From what will the programmer break the program into logical modules?
7. What does he use to show the contents of data storage at significant points?
8. What will the programmer probably do during the period of parallel operation?

EXERCISE 5

Use the following words in sentences of your own to indicate their meaning. Each sentence should include two of the words and each word must be used at least once among the eight sentences.

1. specification
2. documentation
3. corrections
4. layouts
5. checked
6. planned
7. processed
8. proved

UNIT 10

COMPUTER LANGUAGES

Dialogue

John is at a Computer Society Symposium on computer languages.

John: Hallo, Prem. I was talking to a chap this morning who actually used to punch the instructions in binary for the computer when he first learnt programming.

Prem: We're so used to *autocodes and their automatic translation into *machine code that we often forget the tedious work they save us. Imagine remembering the numeric code for every function!

John: He said he used to have to lay his data out in a map of store and look up the address every time he referred to any data item.

Prem: And think what a game it must have been without *floating labels! Every time you moved some program about in store you

would have to alter the address in any instruction which branched into that piece of program.

John: Even a low level autocode must reduce program writing time to about a quarter of what it could be.

Prem: And, of course, the possible number of errors, and therefore testing time, is considerably reduced as well.

John: High level languages are even better. We do a great deal of commercial data processing at our installation, so we use COBOL extensively. It not only has time-saving advantages, but it simplifies documentation. It's very much easier to understand another programmer's routines when they are written in a language so close to English.

Prem: At our research establishment we're concerned with calculation rather than with data handling, so we have standardized on FORTRAN. Personally, I prefer ALGOL for this work, but I expect that is because we used it exclusively when I was at university in Madras.

John: I was amazed at this morning's talk to hear that there are so many different languages.

Prem: Well, if you're prepared to write the necessary translation program in the first place, a language can be made up for any particular application. There's a group of languages for formal algebraic manipulation, another set for simulation problems, another for machine tool control, another for processing information banks, and so on.

John: I guess you would say these are all problem-oriented languages. COBOL and FORTRAN are very much more general purpose and most computer manufacturers offer compilers to produce their own machine code from them. It's surprising that someone hasn't come up with a universal computer language.

Prem: There have been difficulties. One problem is that a language comprehensive enough to cope with all possible applications would need such a huge compiler.

John: What's more, it's difficult to see how its object code could be made absolutely efficient for every machine and every application.

Prem: And from the arguing that went on this morning, I imagine there's a good deal of prejudice. I mean, in the field of natural languages no one's managed to find a language that the whole world can use; why, they can't even find a common language for all my countrymen!

EXERCISE 1: STRUCTURAL PRACTICE

Notice this structure from the conversation:

We're concerned with calculation *rather than* data handling.

Use this structure to respond to the following questions:

Example: Is your firm concerned with data handling?
Prompt: *calculation*
Response: We're concerned with calculation rather than data handling.

Now, you do it.

1.	Is your firm concerned with data handling?	*calculation*
2.	Is that machine used by the programmer?	*the operator*
3.	Is the operator affected by card wrecks?	*machine faults*
4.	Are the staff used to COBOL?	*FORTRAN*
5.	Is the equipment useful to computer bureaus?	*computer consultants*
6.	Is he prepared for simulation problems?	*algebraic manipulation*
7.	Is the programmer slowed down by autocodes?	*numeric codes*
8.	Are they interested in magnetic discs?	*magnetic cores*

EXERCISE 2: PROGRESSIVE SUBSTITUTION DRILL

Statement: The possible number of errors is considerably reduced.
Prompt: *faults*
Response: The possible number of *faults* is considerably reduced.

Now, you do it.

Statement: The possible number of errors is considerably reduced.
Prompts:

1. faults
2. mistakes
3. usual
4. very much
5. greatly
6. increased
7. Our
8. was
9. probable
10. amount

Part of a COBOL program sheet, and the first statement of the program punched up as a card

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 NEW-FILE
                  OPEN I-O ORDER-FILE.
                  MOVE ZERO TO ORDER-KEY.
        NEW-READ.
                  READ NEW-FILE AT END GO TO FINISH.
        ORDER-READ.
                  IF NEW-KEY IS NOT GREATER THAN
                  ORDER-KEY GO TO NEW-ORDER-WRITE.
                  READ ORDER-FILE AT END STOP 999.
                  MOVE PART-NO TO ORDER-KEY
                  GO TO ORDER-READ.
        NEW-ORDER-WRITE.
                  MOVE NEW-REC TO ORDER-REC
                  WRITE ORDER-REC
                  GO TO NEW-READ.
        FINISH.
                  CLOSE NEW-FILE ORDER-FILE
                  STOP RUN.
```

FORM 14/41(7.68)

PROCEDURE DIVISION.

PAGE | SERIAL | CONT | A | B | COBOL STATEMENT | IDENTIFICATION

Gestetner Limited — A member of the ICL group of companies — 07-09289-01? — COBOL PROGRAM CARD — Printed in the U.K.

EXERCISE 3: FURTHER STRUCTURAL PRACTICE

Notice this structure:

> *A* language comprehensive *enough to* cope with all applications *would need* a huge compiler.

Use this structure to make complete sentences:

1. A memory extensive / hold all the records / several million words
2. A processor fast / cater for fifty remote consoles / integrated circuitry
3. A computer robust / be used for this work / solid state components
4. An installation versatile / handle all design work / these devices
5. A program comprehensive / produce the weekly payroll / a very thorough testing
6. A system complex / drive thirty peripherals / several operators
7. A machine powerful / offer multiprogramming / an operating system
8. A printer general / deal with several alphabets / a large character set

Reading and Comprehension

Let us call the natural language of a particular real computer code M (for machine code) and let us invent an imaginary machine A with code A. If a special translator program is written so that it can take a string of instructions in code A and turn them into a string of equivalent instructions in code M, then thereafter all programs can be written as though for machine A and, upon translation, they will work on machine M. This technique enables programmers to write in a code that is convenient to them and to the problem, rather than in the clumsy notation of the real computer (provided they have the appropriate translator program). Thus, some of the more tedious aspects of the programmer's task are passed over to the computer. The simpler *pseudo-codes will allow data names instead of numeric addresses, meaningful function names instead of code numbers, and *name tags for branch points. At this level they are called "autocodes" and their translators are called *"assemblers".

With more sophisticated translators, computer languages can be developed in which the statements appear almost like English sentences, or in which calculations are expressed in formulae familiar to mathematicians and engineers. Not only do they enable programs to be written more quickly and with fewer errors, but their purpose is more apparent. Such high level languages no longer reflect the nature of a particular machine, but can be oriented to particular problems. Some languages (e.g. ALGOL and FORTRAN) are aimed at general scientific problems and some (e.g. COBOL) at general commercial problems. These languages are quite independent of particular computers. Translation programs (or compilers) are provided for many different computers. The desire to create languages which can make programs operate on various machines has led to language standardization, and there is wide acceptance of these standards on a national and international level.

EXERCISE 4: COMPREHENSION QUESTIONS

1. What does the translation technique described enable programmers to do?
2. What are the tedious aspects of the programmer's job passed to?
3. What are the translators of autocodes called?
4. In what form can statements appear in languages developed with more sophisticated translators?
5. Give three advantages of high level languages.
6. Name two languages aimed at general scientific problems.
7. At what problems is COBOL aimed?
8. Are language standards only accepted at national level?

EXERCISE 5

Use the following words and phrases in sentences of your own to show that you understand their meaning and use:

1. autocodes
2. labels
3. documentation
4. commercial languages
5. translation
6. problem-oriented
7. universal
8. standardized

UNIT 11

OPERATING SYSTEMS

Dialogue

John is talking to Hal, who now works on operating systems.

John: I've heard about multi-programming and *time-sharing, but what's their connection with operating systems?

Hal: First of all, do you know what *"simultaneous working" means?

John: Isn't that the way a computer can carry out peripheral transfers and obey arithmetic or logical instructions at the same time?

Hal: That's right. The peripherals are much slower than the C.P.U., and it was realized early on that for most of a program run the processor is doing nothing but wait for a peripheral transfer to be completed. So simultaneous working was introduced. And soon you could have simultaneity between peripherals as well as between a peripheral and the C.P.U.; you could read a magnetic tape at the same time as you were punching a card, for instance. Now you can even run several programs at the same time—that's called "multi-programming".

John: But you'd still have the processor waiting most of the time, wouldn't you? What else can be done?

Hal: You can *offline some of your peripheral activity.

John: You mean, read and write data on machines not connected to the central processor?

Hal: No, John, that was what "offline" originally meant. Here it means that when a program says "read a data card", it doesn't matter if the record originally on the card has been transferred to a magnetic device. The operating system will trap all such instructions and the record will be read from the file held on a particular magnetic device.

John: So you'll transfer all the slow peripheral files to magnetic tape, or a disc or drum, before you run the program?

Hal: Yes, but you can only offline your input and output when you've got a lot of backing store in the form of magnetic media.

This is called the "file store" because it holds all the files which were originally held on slow media.

John: Do you get the same sort of process with the printed output?

Hal: Yes, you do. When a program says "print a line of results" the operating system changes it to "send the line to a *monitoring file on *backing-store".

John: And can a separate program then print the line from the monitoring file when the peripherals are under-used?

Hal: That's it.

John: What else does a modern operating system do?

Hal: It cuts down typewriter*"chatter"—that's what we call all those messages you see being typed at the console. Most of them are routine.

John: But what happens if things go wrong—if it can't find a file in the library, for instance?

Hal: The operating system takes care of that too. The programmer supplies parameters to tell it what to do in any circumstances; in other words, he gives his instructions to the operating system instead of to an operator. Time-sharing is another function of the operating system—working out an equitable sharing of *mill-time between background jobs and real-time users. And yet another function is helping the installation to "fail-soft" when there's a hardware failure, or when a program goes *illegal.

EXERCISE 1: STRUCTURAL PRACTICE

Notice this structure from the conversation:

The peripherals *are much* slower *than* the C.P.U.

Use this structure to respond to the following questions:

Example: Are the peripherals slow?
Prompt: *the C.P.U.*
Response: They're much slower than the C.P.U.

Now, you do it.

1.	Are the peripherals slow?	*the C.P.U.*
2.	Are transistors small?	*thermionic valves*
3.	Are tabulators simple?	*computer installations*
4.	Is this language useful?	*that language*
5.	Is this program complicated?	*the manufacturer's package*
6.	Are fixed discs big?	*exchangeable discs*
7.	Are magnetic peripherals fast?	*mechanical ones*
8.	Is this operator busy?	*that one*

EXERCISE 2: PROGRESSIVE SUBSTITUTION DRILL

Statement: What happens if it can't find a program in the library?
Prompt: *in store*
Response: What happens if it can't find a program *in store?*

Now, you do it.

Statement: What happens if it can't find a program in the library?
Prompts:
1. store
2. in any programs
3. locate
4. the system
5. when
6. the operator
7. he
8. there

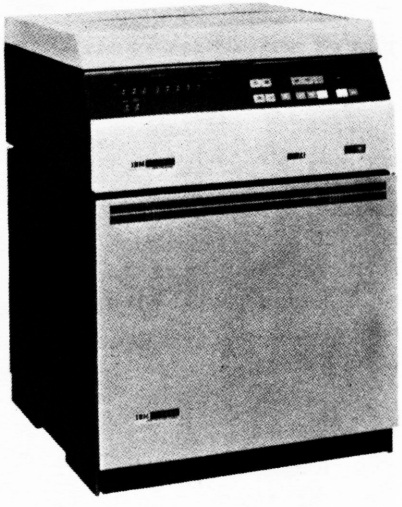
EXERCISE 3: FURTHER STRUCTURAL PRACTICE

Notice this structure from the conversation:

You can only offline your input *when you've got* backing store.

Use this structure to complete the following sentences:

1. transfer the slow files / magnetic tape
2. use a computer / the necessary software
3. control the peripherals / a control circuit
4. convert to binary arithmetic / a radix of two
5. make corrections like this / paper tape
6. make up a computer language / a translation program
7. work out a specialized program / all the details
8. represent a character / a group of sets of cores



Paper tape reader and control

Reading and Comprehension

Computer installations are steadily getting bigger and more complex. Multi-programming, realtime working by remote users, the increasing number of peripherals in simultaneous use: these and other factors are making it almost impossible for a human operator to solve the problems that arise during a computer run. So another level of programming has been devised: operating systems, which are programs permanently held in the machine to control the running of other programs. But operating systems are so big that only part of them can be held in *core store at any given time. The rest is held in backing-store and called into core store in segments, as and when required.

Operating systems also make the running of a computer installation more efficient in other ways. They schedule the order in which jobs are performed, according to priorities; they reduce typewriter "chatter"; they help the system to fail-soft by monitoring malfunctions, and they enable much of the input and output on slow peripherals to be performed offline. They reduce human error by doing many of the jobs that human operators used to do, and they increase the profitability of an installation by "maximizing the throughput", or enabling more programming jobs to be run in a given time.

EXERCISE 4: COMPREHENSION QUESTIONS

1. Are computer installations getting smaller?
2. Can a complete operating system be held in core store?
3. Name three functions performed by operating systems.
4. What is the meaning of "maximizing the throughput"?
5. How do operating systems reduce human error?
6. What can be performed offline by using operating systems?
7. Name two factors that make it almost impossible for a human operator to solve the problems that arise during a computer run.
8. What governs the order in which jobs are performed?

EXERCISE 5

Use the following words and phrases in sentences of your own to show that you understand their meaning and use:

1. simultaneous working
2. offlining
3. backing-store
4. typewriter chatter
5. fail-soft
6. segments
7. monitoring file
8. sharing of mill-time

UNIT 12

APPLICATIONS

Dialogue

John meets Bill, who joined the company with him.

John: What are you doing nowadays, Bill?

Bill: I'm in applications branch; I've been working on a payroll job. Things like pay, pension fund calculations, rating assessments, stock control and invoicing—anything that used to be done by accounts clerks—are the first jobs that new users hand over to the computer. It's only later that they think of computerizing other things—like the *numerical control of machine tools.

John: What's the advantage of using the computer for that?

Bill: Well, you can write a program telling a particular machine how to make a thrust bearing, for example. The parameters given to the program define the dimensions of the bearing, where the holes are to be cut, the angles and so on. The program can be punched on paper tape and fed into a tape reader with a control unit which relays the instructions to the machine. The machine can then manufacture thrust bearings virtually without human intervention.

John: What about *process control? Is that another industrial application?

Bill: Yes, it refers to programs which control the manufacture of chemicals, for instance, or high speed steel alloys. Here you've got the principles of *cybernetics, particularly *"feedback".

John: I see. A program stored in the computer receives information from the chemical plant and tells it to increase this or that ingredient, raise or lower the temperature, and so on.

Bill: That's it. Then you've got *linear programming—that's finding optimum values for a set of variables. You might have a fleet of lorries making regular deliveries to different locations; a linear program will tell you the best route for each lorry to take.

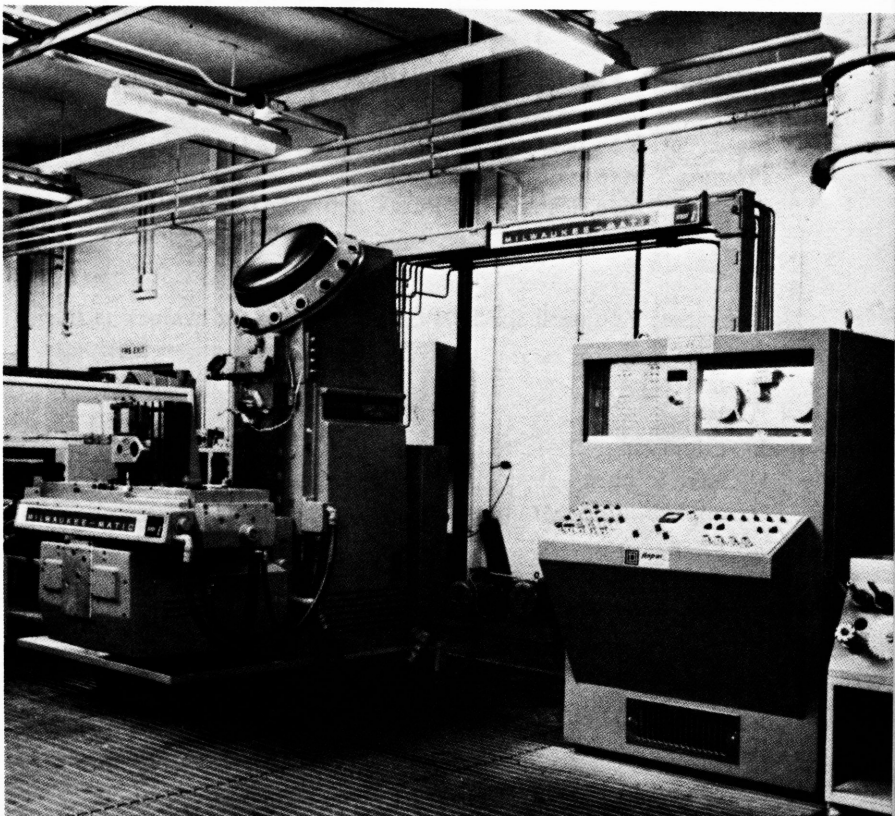
John: What's a *PERT network, by the way?

Bill: It's a kind of *flowchart on which you state all the activities and events that will occur in a large project. You show which activities can only begin when others have been completed, and you give time estimates for each activity; finally you arrive at the *"critical path", the route through the flowchart giving the shortest time in which the whole project can be completed. For big projects the flowchart is so complicated you need special PERT programs to evaluate them.

John: So PERT is a management tool? It gives managers the information they need in the form in which they need it?

Bill: Yes, work's already being done on things like *data banks and *integrated management information systems. Eventually computers will control all the operations of a company, not just accounting processes and individual machines.

Computer control of machine tools



EXERCISE 1: STRUCTURAL PRACTICE

Notice this structure from the conversation:

The flowchart *is so* complicated *that you need* special programs.

Use this structure to respond to the following questions:

Example: Is the flowchart complicated?
Prompt: *special programs*
Response: Yes, it's so complicated that you need special programs.

Now, you do it.

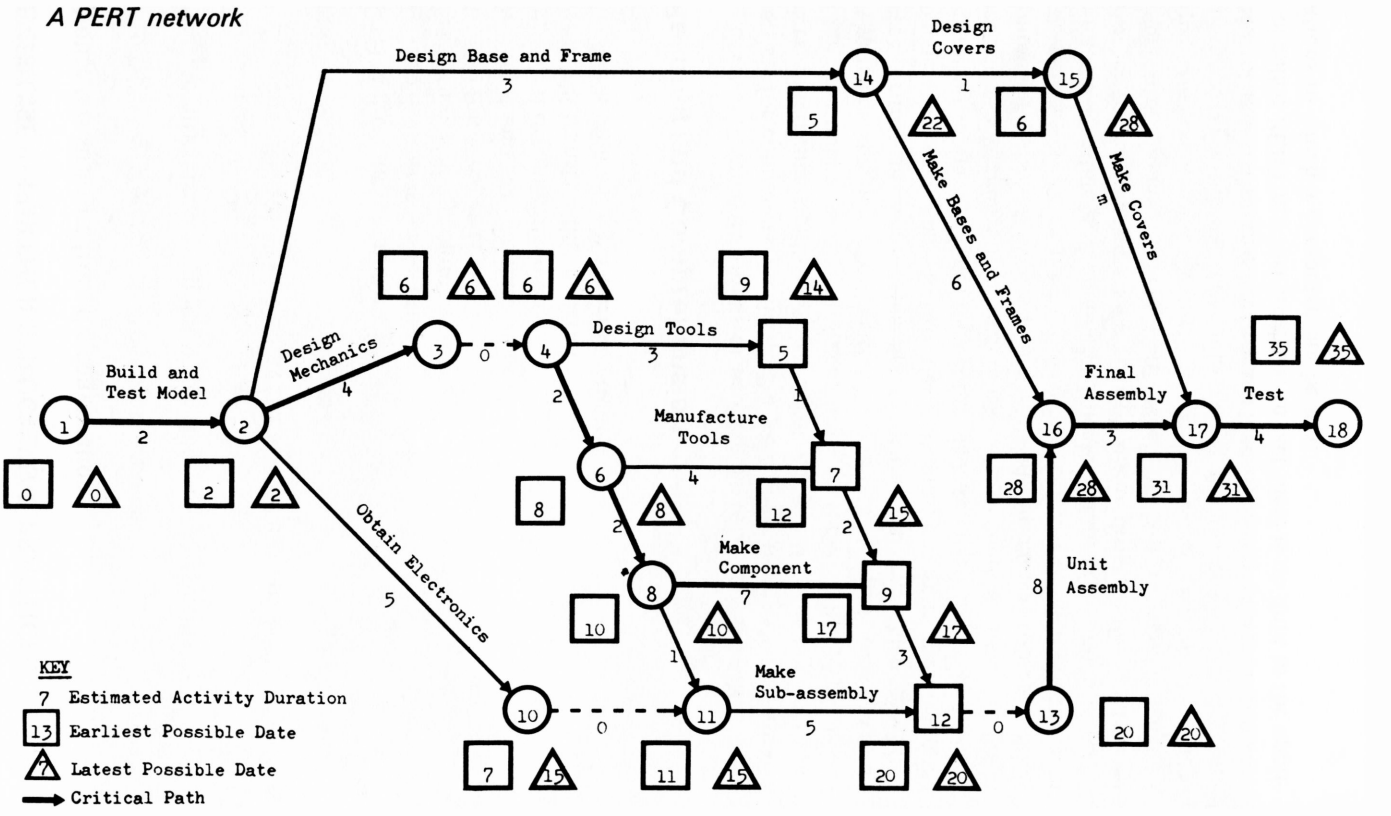
1. Is the flowchart complicated?	*special programs*
2. Is the machine different?	*this code*
3. Is the installation large?	*extra staff*
4. Is the payroll job complex?	*more instructions*
5. Is the data extensive?	*an experienced programmer*
6. Is the language specialized?	*these compilers*
7. Are the peripherals numerous?	*a different level of programming*
8. Is the specification technical?	*special algorithms*

EXERCISE 2: PROGRESSIVE SUBSTITUTION DRILL

Statement: You need special PERT programs for evaluating them.
Prompt: *networks*
Response: You need special PERT *networks* for evaluating them.

Now, you do it.

Statement: You need special PERT programs for evaluating them.
Prompts:
1. networks
2. We
3. calculating
4. have to have
5. application programs
6. these
7. it
8. will need


A PERT network
Design Base and Frame
3
Design Covers
1
Make Bases and Frames
6
Make Covers
m
Design Mechanics
4
Design Tools
3
Manufacture Tools
4
Make Component
7
Obtain Electronics
5
Make Sub-assembly
5
Unit Assembly
8
Final Assembly
3
Test
4
Build and Test Model
2
KEY
7 Estimated Activity Duration
13 Earliest Possible Date
7 Latest Possible Date
Critical Path

EXERCISE 3: FURTHER STRUCTURAL PRACTICE

Notice this structure from the conversation:

What's the advantage of us*ing* the computer?

Use this structure to respond to the following statements:

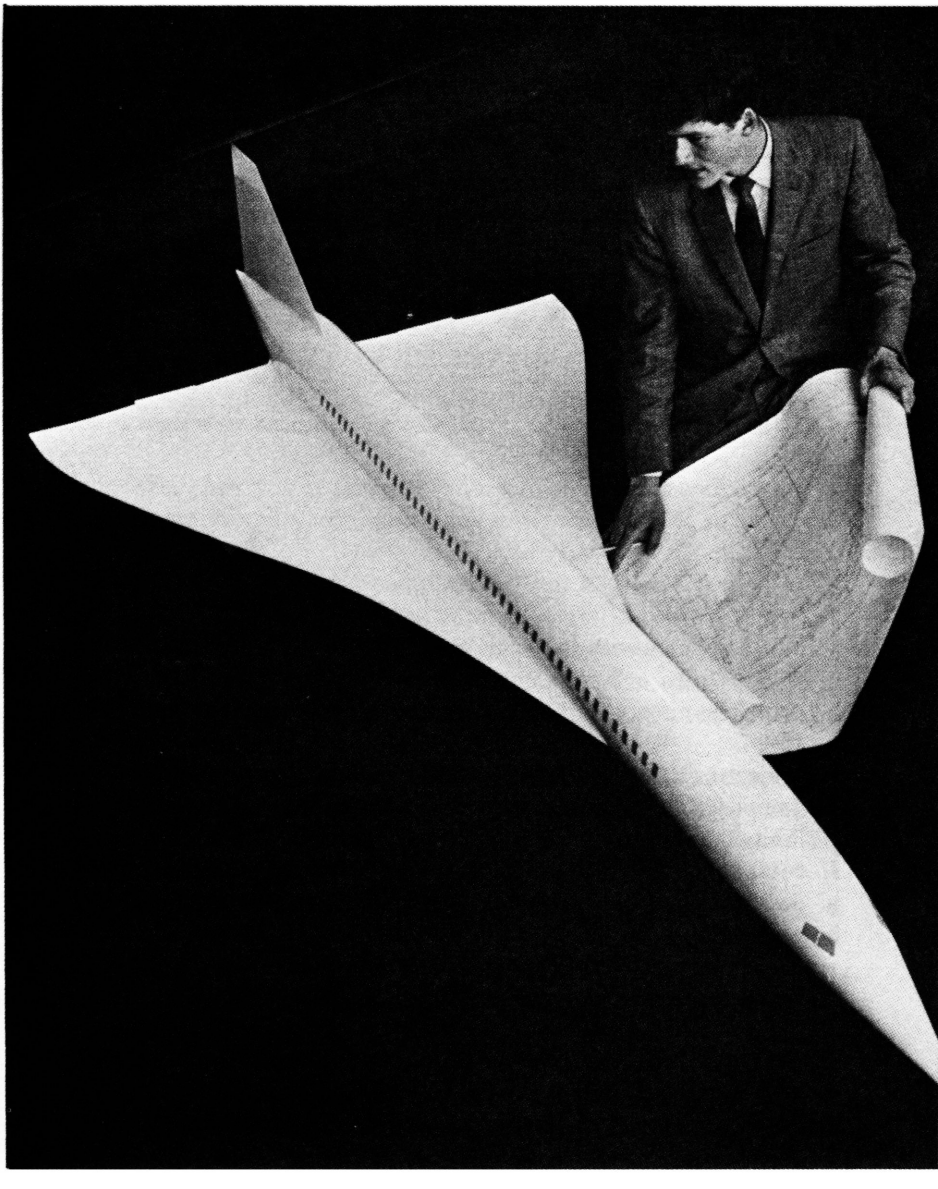
Example: We'll use the computer.
Response: What's the advantage of using the computer?

Now, you do it.

1. We'll use the computer.
2. We'll transfer the records.
3. We'll test the program.
4. We'll use paper tape.
5. We'll find the optimum values.
6. We'll organize the data.
7. We'll translate this material.
8. We'll offline the peripheral activity.

Reading and Comprehension

Every day someone thinks of new commercial and industrial applications for computers. At first there were programs to do the work of clerks: invoicing, stock-keeping, payroll and pension fund calculations. Then computers were introduced into factories. The principles of cybernetics, particularly "feedback", were developed and programs were written to control machines such as lathes, which produced machine parts according to fixed designs. These programs monitored the operation of the machines they controlled, reported faults, and, when necessary, switched off the machines or gave them another design to follow. This sort of programming is called "numerical control". Soon computers were being used for even more sophisticated applications in both the scientific and commercial fields: for instance, flood control, weather forecasting, air traffic control, linear programming and ticket reservation. Some of these were based on new disciplines such as *queueing theory and *games theory. There have also been a large number of applications in the military and aerospace fields; the development of the first atomic bomb was aided by a very powerful software tool called PERT network analysis. In fact there is no limit to the applications to which a computer can be put.



EXERCISE 4: COMPREHENSION QUESTIONS

1. Name some types of clerical work taken over by computers.
2. What is "numerical control"?
3. What software tool aided the development of the atomic bomb?
4. Where, besides offices, have computers been introduced?
5. Name three more sophisticated applications of computers.
6. On what new disciplines are they based?
7. Which principle of cybernetics in particular is involved in numerical control?
8. In which non-commercial fields have there been a large number of computer applications?

EXERCISE 5

Fill in the blanks with the appropriate words:

a. principles
b. sort
c. project
d. completed
e. limit
f. company
g. designs
h. programs

1. ——— can do the work of clerks.
2. This ——— of program is called numerical control.
3. The ——— of cybernetics have been developed.
4. Eventually computers will control all the operations of a ———.
5. These lathes produce parts according to fixed ———.
6. There is no ——— to the applications of a computer.
7. These activities can only begin when others have been ———.
8. The whole ——— can be completed in this time.

UNIT 13

THE SYSTEMS ANALYST

Dialogue

Anna, assigned to a customer project, talks to the systems analyst.

Anna: Well, Jim, you've practically finished your assessment here, and I understand you're ready for my team to begin some of the programs.

Jim: Yes, the customer's computer arrives in twelve weeks' time. We've concentrated on the costing system, but that will involve computerization of both payroll and stores control.

Anna: I'll be concerned with the programs, but I've read your draft report on the reorganization so that I'd be completely in the picture.[1]

Jim: That's a good idea. Stores control was a straightforward change-over from a mechanized system, but in the payroll job we had to redesign lots of the forms so that card punching would be simplified.

Anna: The printouts seem well standardized.

Jim: Yes, but we had to deal with twice that number of reports previously. People create new forms, but they never bother to discard the old ones, even when their original purpose no longer exists!

Anna: Was your report well received by the customer's management?

Jim: Yes, we've introduced standard shop by shop costing in place of the previous unit costing. But the shop foreman wasn't happy until we recommended direct *data capture equipment.

Anna: I see you suggest we tackle the programs in a particular order.

[1] in the picture: informed

Designing a system

Jim: Yes, the changes are too drastic to put into effect overnight, and the system is too big to run in parallel. We have to change over in stages and ensure that each stage is working satisfactorily before going on with the next.

Anna: There is some *float on some stages, but I suppose it's essential that the introduction of the new costing system coincides with the financial year?

Jim: Yes, but since we helped the customer change over to decimalized currency in 1971, that's one headache less for you.

Anna: I'm glad your boys have used *decision tables extensively in their specifications. These make the situation much clearer. With our new *preprocessor translation will be almost direct in many cases.

Jim: And when you attend our weekly meetings, your programmers can sort out any points that aren't clear in our documentation.

EXERCISE 1: STRUCTURAL PRACTICE

Notice this structure from the conversation:

We had to redesign the forms *so that* punching *would be* simplified.

Use this structure to respond to the following questions:

Example: Why did you redesign the forms?
Prompt: *punching / simplified*
Response: We had to redesign the forms so that punching would be simplified.

Now, you do it.

1.	Why did you redesign the forms?	*punching / simplified*
2.	Why did you introduce standard costing?	*accounting / easier*
3.	Why did you run in parallel?	*the changeover / gradual*
4.	Why did you draw flowcharts?	*the specification / clearer*
5.	Why did you reorganize the job?	*the changeover / straightforward*
6.	Why did you hold weekly meetings?	*the programmers / in the picture*
7.	Why did you standardize the printouts?	*the paperwork / reduced*
8.	Why did you use decision tables?	*the translation / direct*

EXERCISE 2: PROGRESSIVE SUBSTITUTION DRILL

Statement: You're ready for my team to begin some of the programs.
Prompt: *start*
Response: You're ready for my team to *start* some of the programs.

Now, you do it.

Statement: You're ready for my team to begin some of the programs.
Prompts:

1. start
2. group
3. They're
4. work
5. waiting
6. us
7. the programmers
8. expecting

EXERCISE 3: FURTHER STRUCTURAL PRACTICE

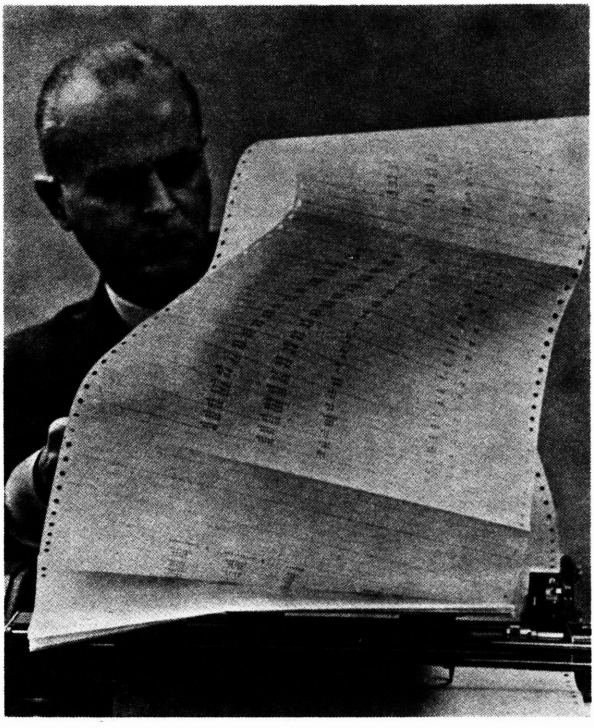
Notice this structure:

> The foreman *was not* happy *until we* recommended data capture equipment.

Use this structure to make similar sentences with the following words and phrases:

1. The customer / satisfied / ran in parallel
2. The programmer / advised / had written specifications
3. The payroll / mechanized / had consulted the unions
4. The equipment / installed / produced our report
5. The management / convinced / produced the results
6. The new system / introduced / decimalized the currency
7. The assessment / complete / had studied the organization
8. The report / well received / explained our plans

Checking the results



Reading and Comprehension

Wherever there is repetitive work, thought must be given to the design and maintenance of a system to perform that work most efficiently. Systems and analysts of systems existed, therefore, long before there were computers. However, it is the introduction of computers which creates the most dramatic changes to existing systems, and from which losses can accumulate most rapidly if the machine is improperly used. The systems analyst has therefore found himself in a key position since the introduction of computers.

He is essentially concerned with method. His job is to determine the objectives of the project under examination and to design the best method of reaching them. This generally means a close examination of the existing system, to discover its weaknesses and its redundancies. He must then consider if all the paperwork is necessary, if it conveys only the right information, by the most direct route, in the proper time, at the right frequency and with sufficient accuracy.

His redesign of the system must be appreciated by the people who are to operate it and to use its results. Scheduling the actual changeover to a new system is a skilled task, each step requiring careful preparation and planning. The truly successful system is one which is amenable to steady and controlled growth. This implies that the system must be well documented, but in particular there must be complete understanding between the systems analyst and the programmers who interpret his schemes on the computer.

The success or failure of computerization often depends on its architect, the systems analyst.

EXERCISE 4: COMPREHENSION QUESTIONS

1. What has created the most dramatic changes to existing systems?
2. Who has found himself in a key position since then?
3. The systems analyst's job is to determine the objectives of the project—and what else?
4. What must be taken into consideration by the systems analyst?
5. What is required when scheduling the changeover to a new system?
6. Which is the truly successful system?
7. What often depends on the systems analyst?
8. What is absolutely essential for a successful redesigning?

EXERCISE 5

Fill in the blanks with the noun equivalents of the verbs in italics:

Example: Does the machine *represent* the design on the screen?
Response: Yes, it's an accurate *representation*.

1. Does the machine *represent* the design on the screen?
   Yes, it's an accurate ———.
2. Can this program *calculate* wages?
   Yes, it saves us doing the ———.
3. Are we going to *computerize* the costing system?
   Yes, and it'll mean the ——— of the payroll, too.
4. Will it take long to *correct* the coding?
   No, you'll be able to make the ——— today.
5. When should we *introduce* the new costing system?
   The ——— should coincide with the new financial year.
6. Do we need to *translate* this material?
   Yes, you'll have to write a ——— program.

	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	Rule 7	Rule 8
Article in stock?	Y	Y	Y	Y	N	N	N	N
Article on order?	Y	Y	N	N	Y	Y	N	N
Permission to issue article obtained?	Y	N	Y	N	Y	N	Y	N
Apply for permission to issue article		X		X		X		
Issue article	X		X					
Reserve article		X		X	X	X		
Reject request							X	X

A decision table

UNIT 14

REALTIME

Dialogue

John is lunching in the canteen with his cousin Christine.

John: What's the difference between realtime working and ordinary commercial programming?

Christine: Well, take a standard program for calculating the weekly payroll. Every day a factory worker clocks in and clocks out, and the next day all the details are punched on to cards. On one day of the week all these cards are collected in to a *batch and fed into the machine with a program to calculate wages. So some of the data was collected days ago—it's "cold".

John: And realtime programs process "hot" data?

Christine: Right. Think of somebody who's booking a seat on a flight to Teheran on 21st August. When the booking clerk rings the central office, he wants to know straight away whether there's a seat available—and that means that the central office have got to know all the reservations that have been made at all the branch offices up to that very moment.

John: So you'd process every booking as soon as it came in, instead of waiting to batch up the reservations. Is that the kind of program you're writing?

Christine: No. I'm using realtime for another purpose. I'm working in *"conversational" or *"interactive" mode. I've got my own typewriter, just like an operator, only mine is *remote* from the machine room; it could even be in another country. I type in a segment and the *test data; then I get the machine to run the segment and send me the results on my console. If it goes wrong, I can ask the machine to give me *diagnostic information, make corrections to the program while it's still resident in the filestore, and try again. When there are a lot of users *online to the machine it's called *"multi-access".

John: Engineers use realtime working a lot, too, don't they?

Christine: Yes. Design engineers use *light pens and *graphical display units. They sketch a rough design on a cathode ray screen and give the machine the scale and other parameters. The machine displays an accurate representation of the design on the screen. They modify it, get another display, and so on.

John: It sounds to me as if realtime is the thing of the future.

Christine: It is! In no time at all everyone will be using computers, and most of them will want to work in realtime. Telephones, telex and telegraph equipment, for instance, will all be hooked in to central computer installations.

John: But isn't it difficult to link existing communications networks to computers? What about the *interface? Isn't there a lot more hardware needed?

Christine: Oh, yes. You have to have extra transmission lines which can be simplex, duplex or half-duplex, depending on the direction of signals they can carry. You need *uniplexors and *multiplexors to act as buffers, and you even need devices called Modems—modulator-demodulators—to convert the digital impulses from the computer to a form suitable for transmission along telephone lines, and vice versa.[1] Then . . . .

John: Just a minute! I'm not following you[2]. Let me fetch you another cup of coffee, then you can tell me some more.

[1] vice versa: (Latin) the other way round, the reverse
[2] I'm not following you: I can't understand what you are saying

EXERCISE 1: STRUCTURAL PRACTICE

Notice this structure from the conversation:

Is that the kind of program *you're writing*?

Use this structure to respond to the following statements:

Example: This is a program for making reservations.
Prompt: *write*
Response: Is that the kind of program you're writing?

Now, you do it.

1.	This is a program for making reservations.	*write*
2.	This installation used realtime.	*operate*
3.	This is "hot" data.	*type*
4.	This mode is more "conversational".	*test*
5.	This device is a light pen.	*use*
6.	This environment is multi-access.	*work in*
7.	This data is standardized.	*handle*
8.	This specification has just come in.	*code*

EXERCISE 2: PROGRESSIVE SUBSTITUTION DRILL

Statement: In no time at all everyone will be using computers.
Prompt: *remote terminals*
Response: In no time at all everyone will be using *remote terminals.*

Now, you do it.

Statement: In no time at all everyone will be using computers.
Prompts:
1. remote terminals
2. them
3. Very soon
4. everybody
5. we
6. may
7. working with
8. In the near future

EXERCISE 3: FURTHER STRUCTURAL PRACTICE

Notice this structure:

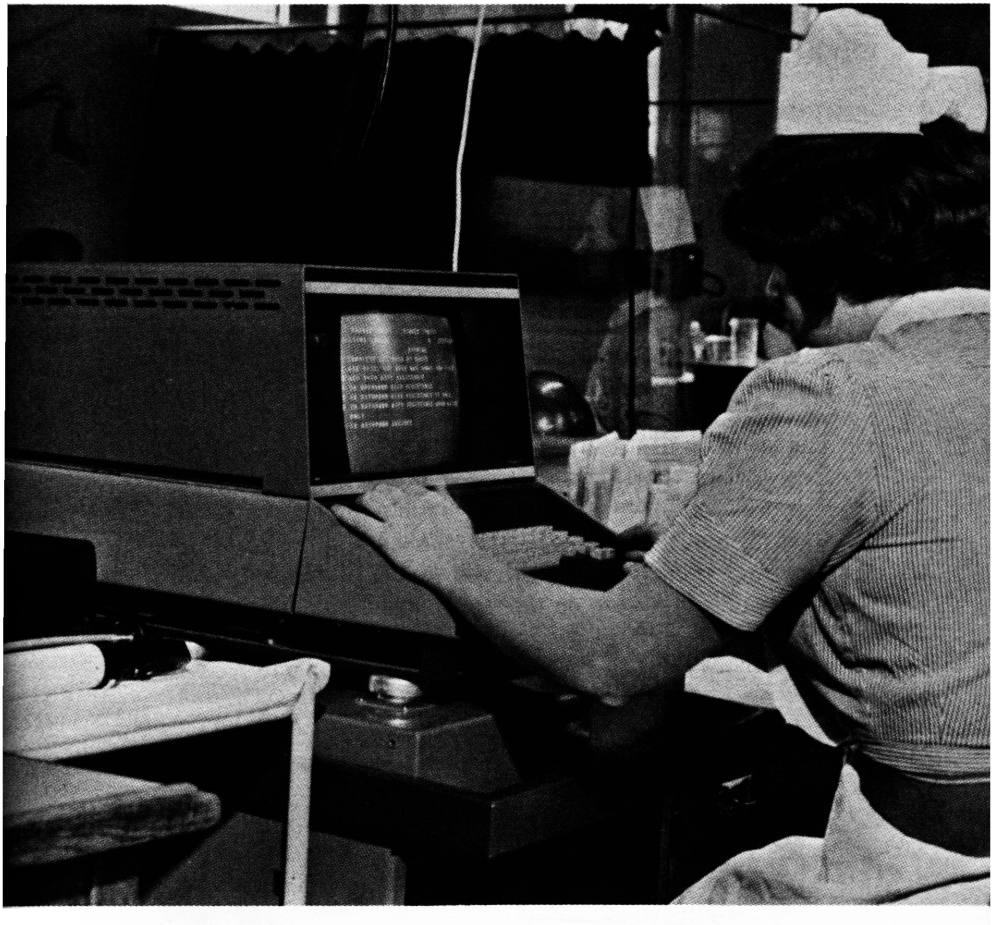
It sounds to me as if realtime *will be* the thing of the future.

Use this structure to make complete sentences:

1. multi-access facilities / available on most computers
2. computer networks / a likely development
3. the remote console / a general requirement
4. the interactive mode / more widely used
5. graphical displays / the natural device for designers
6. data banks / at everyone's disposal
7. programming / a career with prospects
8. computers / here to stay

Reading and Comprehension

Realtime working is one of the newest and most exciting developments in the computer field. It means that you ask the computer a question and get an answer within a matter of seconds; and the answer is based on the latest information in the machine's data banks. This is to be contrasted with the processing of "cold data" in calculating, for instance, an income tax assessment several months after the end of the tax year. Realtime working has two basic requirements. The computer system must be able to accept data as soon as it is created, and use it to update its files immediately by means of a program called in from backing-store; and there must be devices, such as remote typewriters, by means of which users can interrogate the system and receive answers almost immediately. Airline ticket reservation is one obvious example of realtime. Another is when a pilot coming in to land needs to know the position of all other aircraft in the vicinity, as well as their flight paths and estimated times of arrival. And realtime working is invaluable in the development of scientific and commercial programs, that is, while they are being tested. The programmer works with the system in "conversational" or "interactive" mode; he can test parts of his program, alter it in store, and ask the machine for intermediate results, all in realtime, instead of having to wait for the program to be run overnight.



Realtime working in a hospital

EXERCISE 4: COMPREHENSION QUESTIONS

1. By what means does a computer update files immediately?
2. Name one example of realtime working.
3. Name the mode in which a programmer is working while he is testing programs at the computer.
4. How quickly can you get an answer from the machine?
5. How do realtime users interrogate the system?
6. What does a pilot coming in to land need to know?
7. Does the calculation of an income tax assessment involve realtime?
8. Where is the program that the realtime user alters?

EXERCISE 5

Insert the following words in the sentences:

a. get
b. run
c. got
d. sketch
e. use
f. works
g. based
h. clocks

1. Engineers ——— realtime working a lot.
2. Every day a factory worker ——— in and ——— out.
3. It means you ——— an answer within a matter of seconds.
4. I get the machine to ——— the segment.
5. The programmer ——— with the system in interactive mode.
6. The answer is ——— on the latest, up-to-date information.
7. They ——— a rough design on a cathode ray tube.
8. I've ——— my own typewriter, just like an operator.

A printer/keyboard

APPENDIX

KEY TO EXERCISES

Unit 1

EXERCISE 1

1. What about the software?
2. What about the slow peripherals?
3. What about the output units?
4. What about the operator?
5. What about the magnetic devices?
6. What about the card punch?
7. What about the paper tape punch?
8. What about the discs?

EXERCISE 2

1. The computer installation consists of three things.
2. The computer installation consists of different things.
3. The computer installation consists of various things.
4. Any computer installation consists of various things.
5. Any computer installation consists of various parts.
6. Any computer installation is made up of various parts.
7. Any C.P.U. is made up of various parts.
8. Any C.P.U. contains various parts.

EXERCISE 3

1. I suppose mechanical devices are called "slow" peripherals?
2. I suppose the whole thing is called a computer installation?
3. I suppose card readers are called input units?
4. I suppose programs are called the software?
5. I suppose the C.P.U. is called the computer proper?
6. I suppose the peripherals are called input/output devices?
7. I suppose circuits such as those are called arithmetic units?
8. I suppose drums and discs are called magnetic devices?

EXERCISE 4: SUGGESTED ANSWERS

1. Data.
2. Peripherals.
3. Systems analysts.
4. Programmers.
5. They punch up programs and data on cards and paper tape.
6. They run the machine and monitor its progress.
7. A messenger service and magnetic library staff.
8. The C.P.U. or Central Processing Unit.

Unit 2

EXERCISE 1

1. What's the difference between analogue computers and digital computers?
2. What's the difference between an abacus and a slide rule?
3. What's the difference between the paper on "Computable Numbers" and von Neumann's papers?
4. What's the difference between Babbage's machine and other early machines?
5. What's the difference between the "on" state and the "off" state?
6. What's the difference between input units and output units?
7. What's the difference between the C.P.U. and the peripherals?
8. What's the difference between thermionic valves and transistors?

EXERCISE 2

1. In practice they're only used by engineers.
2. In theory they're only used by engineers.
3. In theory they're only useful to engineers.
4. Theoretically they're only useful to engineers.
5. Theoretically they're only operated by engineers.
6. Theoretically it should only be operated by engineers.
7. Theoretically it must always be operated by engineers.
8. Theoretically it must always be serviced by engineers.

EXERCISE 3

1. It isn't only the development of analogue machines that has made scientific advances possible.
2. It isn't only the development of transistors that has made fast circuits possible.
3. It isn't only the development of stored programs that has made automatic processing possible.
4. It isn't only the development of this invention that has made computing possible.
5. It isn't only the development of thermionic valves that has made arithmetic circuits possible.
6. It isn't only the development of punched cards that has made data processing possible.
7. It isn't only the development of electronic devices that has made this development possible.
8. It isn't only the development of magnetic tapes that has made bulk storage possible.

EXERCISE 4: SUGGESTED ANSWERS

1. It could perform arithmetic calculations by itself.
2. Thermionic valves and bistable electronic circuits.
3. A means of storing data in the form of holes punched into cards.
4. In the 1940's.
5. Transistors.
6. It became necessary to store vast quantities of accounting data in a form in which they could be easily retrieved.
7. "Third-" and "fourth-generation" computers.
8. Tunnel diodes; integrated circuits; super-cooled memory banks; thin film.

Unit 3

EXERCISE 1

1. Then you're not the same as a service bureau?
2. Then it's not the same as this scheme?
3. Then he's not the same as the programmer?
4. Then they're not the same as fast peripherals?
5. Then it's not the same as an electric typewriter?
6. Then it's not the same as a utility routine?
7. Then they're not the same as mine?
8. Then they're not the same as software houses?

EXERCISE 2

1. Some companies specialize in this sort of work.
2. Those companies specialize in this sort of work.
3. We specialize in this sort of work.
4. We specialize in this kind of work.
5. We specialize in this kind of program writing.
6. Software houses specialize in this kind of program writing.
7. Software houses do this kind of program writing.
8. Software houses concentrate on this kind of program writing.

EXERCISE 3

1. He doesn't only sell machines, he also sells programs.
2. They don't only want the manufacturer's package, they also want this special program.
3. We don't only supply utility routines, we also supply applications packages.
4. He doesn't only need a program, he also needs computer time.
5. They don't only punch up the cards, they also punch up the paper tape.
6. They don't only have their own machines, they also have their own programs.

EXERCISE 4: SUGGESTED ANSWERS

1. Any problem whose elements can be quantified.
2. Several million pounds.
3. From service bureaus.
4. They write programs for users who don't maintain their own programming staff.
5. They help users make a choice between different machines.
6. Problems such as the authorship of ancient texts, or the translation of dead languages.
7. They are linked by transmission lines.
8. Chemists and physicists, biologists and economists.

EXERCISE 5

1. c, c 2. e 3. g 4. a 5. b 6. f 7. h 8. d.

Unit 4

EXERCISE 1

1. Why is each core threaded by several wires?
2. When is a radix of two used?
3. How often is the present state altered by these wires?
4. Why are some circuits called gates?
5. What is the circuit printed on?
6. How are the cores magnetized?
7. Where will the clock pulse be generated?
8. How is the decimal value converted?

EXERCISE 2

1. Now that you've done that you can do any arithmetic operation.
2. Now that you've done that you can do any arithmetic calculation.
3. Now that you've mastered that you can do any arithmetic calculation.
4. Now that you've learnt that you can do any arithmetic calculation.
5. When you've learnt that you can do any arithmetic calculation.
6. When you've learnt that you can do a difficult calculation.
7. When you've learnt that you can do a complex calculation.
8. When you've learnt that you will be able to do a complex calculation.

EXERCISE 3

1. You often don't have tape decks, you have magnetic discs.
2. You often don't have punched cards, you have paper tape.
3. You often don't have decimal arithmetic, you have binary arithmetic.
4. You often don't have typewriters, you have display units.
5. You often don't have computers, you have tabulators.
6. You often don't have keys to press, you have switches.
7. You often don't have these units, you have special circuits.
8. You often don't have valves, you have transistors.

EXERCISE 4: SUGGESTED ANSWERS

1. The central processor.
2. In memory banks.
3. Desk computers are made possible by miniaturization.
4. Several million.
5. Because the electro-magnetic signals have a shorter distance to travel.
6. They are often twenty times as fast.
7. The weight requirements of space rockets have accelerated the trend towards the miniaturization of components.
8. Transistors, magnetic cores and printed circuits.

Unit 5

EXERCISE 1

1. All you do is rewind it.
2. All you do is separate them.
3. All you do is splice it.
4. All you do is call him.
5. All you do is sort them.
6. All you do is replace it.
7. All you do is hire it.
8. All you do is exchange them.

EXERCISE 2

1. Don't you have to punch the data in a special way?
2. Don't you have to write the data in a special way?
3. Don't you have to write the characters in a special way?
4. Don't you have to write the characters in a stylized way?
5. Doesn't he have to write the characters in a stylized way?
6. Doesn't he have to write the characters like this?
7. Doesn't the programmer have to write the characters like this?
8. Does the programmer have to write the characters like this?

EXERCISE 3

1. Decimal arithmetic is rather different from binary arithmetic.
2. Logic units are entirely different from control circuits.
3. These stylized characters are altogether different from handwriting.
4. That magnetic tape deck is probably different from this one.
5. A computer consultant is very different from a service bureau.
6. Thermionic valves are completely different from transistors.
7. Punched cards are not so different from paper tape.
8. AND gates are a little different from OR gates.

EXERCISE 4: SUGGESTED ANSWERS

1. Verifiers, interpreters and collators.
2. In the form of a replaceable plugboard.
3. Card and paper tape units, and printers.
4. Tape transports; discs; drums.
5. Because they supplement the processor's core store.
6. Graphical or visual display units.
7. The telephone, telegraph and Telex lines.
8. The tabulator.

EXERCISE 5

1. e 2. c 3. d 4. b 5. g 6. h 7. a 8. f

Unit 6

EXERCISE 1

1. It's up to the operator to report them.
2. It's up to the systems analyst to investigate them.
3. It's up to the data preparation staff to alter them.
4. It's up to the programmer to correct them.
5. It's up to him to refeed them.
6. It's up to us to report them.
7. It's up to the person using it to detect them.
8. It's up to the manufacturer to repair them.

EXERCISE 2

1. They don't just hit keys on a console typewriter.
2. They don't just type instructions on a console typewriter.
3. They don't only type instructions on a console typewriter.
4. They don't only type instructions on a remote terminal.
5. They don't only send data on a remote terminal.
6. They don't always send data on a remote terminal.
7. You don't always send data on a remote terminal.
8. You don't always send data by means of a remote terminal.

EXERCISE 3

1. He's kept busier than ever looking after the console.
2. He's kept busier than ever loading the input/output units.
3. He's kept busier than ever running programs.
4. He's kept busier than ever switching the tape decks.
5. He's kept busier than ever servicing the peripherals.
6. He's kept busier than ever obeying the programmers' instructions.
7. He's kept busier than ever taking care of the installation.
8. He's kept busier than ever watching for program events.

EXERCISE 4: SUGGESTED ANSWERS

1. In the days of the early machines.
2. Changing magnetic tapes and discs.
3. So that he can interpret the programmer's instructions correctly.
4. Program events.
5. On the result of a run, and the reason for any failure.
6. He has to take appropriate action to "fail-soft".
7. In a multi-programming environment.
8. They are called operating systems.

Unit 7

EXERCISE 1

1. Yes, I know about the nature of the electronics.
2. Yes, I've been told about the stylization of the characters.
3. Yes, I've been informed about the organization of the data.
4. Yes, I've learnt about the working of that machine.
5. Yes, I've got the idea of the operation of the console.
6. Yes, I understand the construction of the memory bank.
7. Yes, I've read about the replacement of the disc packs.
8. Yes, I've found out about the preparation of the paper tape.

EXERCISE 2

1. They could write a program for processing a record.
2. They could design a program for processing a record.
3. They could use this information for processing a record.
4. The programmer could use this information for processing a record.
5. The programmer could use this information to process a record.
6. The programmer could use this information to update a record.
7. The programmer should use this information to update a record.
8. The programmer might use this information to update a record.

EXERCISE 3

1. Looking at this printout I can see numbers in the columns.
2. Looking at the output device I can see the holes in the paper tape.
3. Looking at the reader I can see the notches in the cards.
4. Looking at your layout chart I can see fields in the records.
5. Looking at the printer I can see paper in the stacker.
6. Looking at his listing I can see errors in the format.
7. Looking at that console log I can see gaps in the schedule.
8. Looking at the manual I can see sixty-four symbols in the character set.

EXERCISE 4: SUGGESTED ANSWERS

1. Letters and digits.
2. A file.
3. In registers composed of electronic circuits.
4. No, slower.
5. Magnetic tape, cards or paper tape.
6. The line-printer.
7. The cathode ray tube display.
8. No, only serially.

Unit 8

EXERCISE 1

1. They couldn't do without "read" and "write" instructions, could they?
2. They couldn't do without this code, could they?
3. They couldn't do without regular servicing, could they?
4. They couldn't do without special training, could they?
5. They couldn't do without transistors, could they?
6. He couldn't do without this information, could he?
7. He couldn't do without alternative devices, could he?
8. He couldn't do without a special program, could he?

EXERCISE 2

1. These codes do vary considerably.
2. These codes do vary a lot.
3. These functions do vary a lot.
4. The set's functions do vary a lot.
5. Its functions do vary a lot.
6. Its functions may vary a lot.
7. Its functions might vary a lot.
8. Its functions might differ a lot.

EXERCISE 3

1. I do think you're lucky to have written other programs.
2. I do think you're lucky to have learnt other machine codes.
3. I do think you're lucky to have worked for other employers.
4. I do think you're lucky to have used other computers.
5. I do think you're lucky to have studied other instruction sets.
6. I do think you're lucky to have practised with other systems.
7. I do think you're lucky to have familiarized yourself with other order codes.
8. I do think you're lucky to have mastered other functions.

EXERCISE 4: SUGGESTED ANSWERS

1. Either instructions or data.
2. A function, a first address, a second address.
3. ADD, SUBTRACT, MULTIPLY, DIVIDE.
4. 8, an unconditional JUMP and 9, a conditional JUMP.
5. A word is referenced by its "address".
6. The circuitry of the control unit gives the instructions meaning.
7. Yes, in fact a sample program is given.
8. The MOVE instruction used repeatedly.

Unit 9

EXERCISE 1

1. We'll only run the program when it's satisfactory.
2. We'll only test the system when it's assembled.
3. We'll only incorporate the new facilities when they're prepared.
4. We'll only start the project when it's scheduled.
5. We'll only examine the layouts when they're designed.
6. We'll only use the computer when it's free.
7. We'll only process the data when it's organized.
8. We'll only check out the system when it's completed.

EXERCISE 2

1. The better you organize the project, the less time you'll spend coding.
2. The better you organize the work, the less time you'll spend coding.
3. The more thoroughly you organize the work, the less time you'll spend coding.
4. The more carefully you organize the work, the less time you'll spend coding.
5. The more carefully you organize the work, the less time we'll take coding.
6. The more carefully we organize the work, the less time we'll take coding.
7. The more carefully we organize the work, the less time we'll take checking.
8. The more carefully we organize the work, the less time we'll take testing.

EXERCISE 3

1. I expect you to start documenting although your testing will not be finished.
2. I expect you to describe an algorithm although your print layouts will not be designed.
3. I expect you to work overtime although your day runs will not be frequent.
4. I expect you to tackle it although your training will not be thorough.
5. I expect you to write it up although your program will not be proved.
6. I expect you to repair errors although your documentation will not be perfect.
7. I expect you to run live although your new facilities will not be incorporated.
8. I expect you to begin planning although your targets will not be fully set.

EXERCISE 4: SUGGESTED ANSWERS

1. The size of the organization he works for.
2. In the form of a specification prepared by a systems analyst.
3. Yes, if mathematical or special technical knowledge is involved.
4. Performance criteria supplied by the systems analyst.
5. Subroutines applicable to his program.
6. From the block diagram.
7. Dump routines.
8. Write up a complete description of the program.

Unit 10

EXERCISE 1

1. We're concerned with calculation rather than data handling.
2. It's used by the operator rather than the programmer.
3. He's affected by machine faults rather than card wrecks.
4. They're used to FORTRAN rather than COBOL.
5. It's useful to computer consultants rather than computer bureaus.
6. He's prepared for algebraic manipulation rather than simulation problems.
7. He's slowed down by numeric codes rather than autocodes.
8. They're interested in magnetic cores rather than magnetic discs.

EXERCISE 2

1. The possible number of faults is considerably reduced.
2. The possible number of mistakes is considerably reduced.
3. The usual number of mistakes is considerably reduced.
4. The usual number of mistakes is very much reduced.
5. The usual number of errors is greatly reduced.
6. The usual number of errors is greatly increased.
7. Our usual number of errors is greatly increased.
8. Our usual number of errors was greatly increased.
9. Our probable number of errors was greatly increased.
10. Our probable amount of errors was greatly increased.

EXERCISE 3

1. A memory extensive enough to hold all the records would need several million words.
2. A processor fast enough to cater for fifty remote consoles would need integrated circuitry.
3. A computer robust enough to be used for this work would need solid state components.
4. An installation versatile enough to handle all design work would need these devices.
5. A program comprehensive enough to produce the weekly payroll would need a very thorough testing.
6. A system complex enough to drive thirty peripherals would need several operators.
7. A machine powerful enough to offer multiprogramming would need an operating system.
8. A printer general enough to deal with several alphabets would need a large character set.

EXERCISE 4: SUGGESTED ANSWERS

1. To write in a code convenient to them and to the problem.
2. To the computer.
3. Assemblers.
4. Like English sentences or as formulae.
5. i. Programs are written more quickly.
   ii. They have fewer errors.
   iii. Their purpose is more apparent.
6. ALGOL, FORTRAN.
7. General commercial problems.
8. No, they are also accepted at international level.

Unit 11

EXERCISE 1

1. They're much slower than the C.P.U.
2. They're much smaller than thermionic valves.
3. They're much simpler than computer installations.
4. It's much more useful than that language.
5. It's much more complicated than the manufacturer's package.
6. They're much bigger than exchangeable discs.
7. They're much faster than mechanical ones.
8. He's much busier than that one.

EXERCISE 2

1. What happens if it can't find a program in store?
2. What happens if it can't find any programs in store?
3. What happens if it can't locate any programs in store?
4. What happens if it can't locate any programs in the system?
5. What happens when it can't locate any programs in the system?
6. What happens when the operator can't locate any programs in the system?
7. What happens when he can't locate any programs in the system?
8. What happens when he can't locate any programs in there?

EXERCISE 3

1. You can only transfer the slow files when you've got magnetic tape.
2. You can only use a computer when you've got the necessary software.
3. You can only control the peripherals when you've got a control circuit.
4. You can only convert to binary arithmetic when you've got a radix of two.
5. You can only make corrections like this when you've got paper tape.
6. You can only make up a computer language when you've got a translation program.
7. You can only work out a specialized program when you've got all the details.
8. You can only represent a character when you've got a group of sets of cores.

EXERCISE 4: SUGGESTED ANSWERS

1. No, they are not.
2. No, it cannot.
3. They schedule the order in which jobs are performed, reduce typewriter chatter, help the system to fail-soft and perform slow peripheral input and output offline.
4. Enabling more programming jobs to be run in a given time.
5. By doing many of the jobs that a human operator used to do.
6. The input and output on slow peripherals.
7. Multi-programming; real time working by remote users; the increasing number of peripherals in simultaneous use.
8. Priorities.

Unit 12

EXERCISE 1

1. Yes, it's so complicated that you need special programs.
2. Yes, it's so different that you need this code.
3. Yes, it's so large that you need extra staff.
4. Yes, it's so complex that you need more instructions.
5. Yes, it's so extensive that you need an experienced programmer.
6. Yes, it's so specialized that you need these compilers.
7. Yes, they're so numerous that you need a different level of programming.
8. Yes, it's so technical that you need special algorithms.

EXERCISE 2

1. You need special PERT networks for evaluating them.
2. We need special PERT networks for evaluating them.
3. We need special PERT networks for calculating them.
4. We have to have special PERT networks for calculating them.
5. We have to have special application programs for calculating them.
6. We have to have these application programs for calculating them.
7. We have to have these application programs for calculating it.
8. We will need these application programs for calculating it.

EXERCISE 3

1. What's the advantage of using the computer?
2. What's the advantage of transferring the records?
3. What's the advantage of testing the program?
4. What's the advantage of using paper tape?
5. What's the advantage of finding the optimum values?
6. What's the advantage of organizing the data?
7. What's the advantage of translating this material?
8. What's the advantage of offlining the peripheral activity?

EXERCISE 4

1. Invoicing, stock-keeping, payroll and pension fund calculations.
2. A type of programming to control machines such as lathes.
3. PERT network analysis.
4. They have been introduced into factories.
5. Flood control; weather forecasting; air traffic control; linear programming; ticket reservation.
6. Disciplines such as queueing theory and games theory.
7. Feedback.
8. The military and aerospace fields.

EXERCISE 5

1. h 2. b 3. a 4. f 5. g 6. e 7. d 8. c

Unit 13

EXERCISE 1

1. We had to redesign the forms so that punching would be simplified.
2. We had to introduce standard costing so that accounting would be easier.
3. We had to run in parallel so that the changeover would be gradual.
4. We had to draw flowcharts so that the specification would be clearer.
5. We had to reorganize the job so that the changeover would be straightforward.
6. We had to hold weekly meetings so that the programmers would be in the picture.
7. We had to standardize the printouts so that the paperwork would be reduced.
8. We had to use decision tables so that the translation would be direct.

EXERCISE 2

1. You're ready for my team to start some of the programs.
2. You're ready for my group to start some of the programs.
3. They're ready for my group to start some of the programs.
4. They're ready for my group to start some of the work.
5. They're waiting for my group to start some of the work.
6. They're waiting for us to start some of the work.
7. They're waiting for the programmers to start some of the work.
8. They're expecting the programmers to start some of the work.

EXERCISE 3

1. The customer was not satisfied until we ran in parallel.
2. The programmer was not advised until we had written specifications.
3. The payroll was not mechanized until we had consulted the unions.
4. The equipment was not installed until we produced our report.
5. The management was not convinced until we produced the results.
6. The new system was not introduced until we decimalized the currency.
7. The assessment was not complete until we had studied the organization.
8. The report was not well received until we explained our plans.

EXERCISE 4: SUGGESTED ANSWERS

1. The introduction of computers.
2. The systems analyst.
3. To design the best method of reaching them.
4. The existing system with its weaknesses and redundancies.
5. Careful preparation and planning.
6. The one which is amenable to steady and controlled growth.
7. The success or failure of computerization.
8. Complete understanding between the analyst and the programmers.

EXERCISE 5

1. representation
2. calculation
3. computerization
4. correction
5. introduction
6. translation

Unit 14

EXERCISE 1

1. Is that the kind of program you're writing?
2. Is that the kind of installation you're operating?
3. Is that the kind of data you're typing?
4. Is that the kind of mode you're testing?
5. Is that the kind of device you're using?
6. Is that the kind of environment you're working in?
7. Is that the kind of data you're handling?
8. Is that the kind of specification you're coding?

EXERCISE 2

1. In no time at all everyone will be using remote terminals.
2. In no time at all everyone will be using them.
3. Very soon everyone will be using them.
4. Very soon everybody will be using them.
5. Very soon we will be using them.
6. Very soon we may be using them.
7. Very soon we may be working with them.
8. In the near future we may be working with them.

EXERCISE 3

1. It sounds to me as if multi-access facilities will be available on most computers.
2. It sounds to me as if computer networks will be a likely development.
3. It sounds to me as if the remote console will be a general requirement.
4. It sounds to me as if the interactive mode will be more widely used.
5. It sounds to me as if graphical displays will be the natural device for designers.
6. It sounds to me as if data banks will be at everyone's disposal.
7. It sounds to me as if programming will be a career with prospects.
8. It sounds to me as if computers will be here to stay.

EXERCISE 4: SUGGESTED ANSWERS

1. By means of a program called in from backing-store.
2. Airline ticket reservation; giving information to a pilot when landing; the development of scientific and commercial programs.
3. He is working in conversational or interactive mode.
4. Within a matter of seconds.
5. By remote typewriters, for example.
6. He needs to know the position of other aircraft in the vicinity and their flight paths and estimated times of arrival.
7. The calculation of an income tax assessment does not involve realtime.
8. The programmer alters his program in store.

EXERCISE 5

1. e 2. h, h 3. a 4. b 5. f 6. g 7. d 8. c

GLOSSARY

NOTE: *Words included here have a special meaning related to computers and their use. Words not included may be found in a standard dictionary. Heavy type within the entries refers the reader to a separately glossed item.*

abacus ['æbəkəs] A primitive calculating device, comprising sliding balls on a frame of wires.

address [ə'dres] The identification of a core, or other elementary unit, in a computer memory. An instruction in a computer program is said to "address" a particular location in memory.

algorithm ['ælgəriðəm] A process represented by a sequence of instructions, similar to a mathematical procedure, designed to solve a particular problem. A program may be an algorithm, or may consist of a number of algorithms.

analogue computer ['ænəlɔg kəm,pju:tə] A machine in which variables are represented by physical analogues or correspondencies, e.g. by the sizes of gearwheels, the strengths of electrical currents, or volumes of gas or liquid.

applications packages [æpli'keiʃənz ,pækidʒiz] Sets of computer programs designed to handle the application of the computer to specific areas of human activity, e.g. the composing of printed documents, the control of water flow in rivers and reservoirs, weather forecasting, share registration, etc.

arithmetic unit [ə'riθmətik ,ju:nit] The circuitry in the central processor which performs arithmetic operations.

assembler [ə'semblə] A program which translates instructions in **assembly code** into **machine code**.

assembly code [ə'sembli ,koud] Assembly code is a language in which the numerical form of a **machine code** instruction is represented by a mnemonic, e.g. "ADX" for the numerical code meaning "add variable to accumulator".

autocodes ['ɔ:toukoudz] Computer languages which may be similar to a natural language such as English, and are **"problem-oriented"** rather than "machine-oriented". It is easier for a programmer to define a problem in such a language, but they need a **compiler** to translate them into **machine code**.

background job ['bækgraund ,dʒɔb] A job which is run on the computer without the intervention of the programmer, and at a time not specified by the programmer, simultaneously with the jobs of **realtime** users.

backing-store ['bækiŋ ,stɔ:] A type of store which is subsidiary to, but slower in access than, the computer's main **immediate access store**, though in many **software** situations it can be regarded as part of the main store. The name is applied to magnetic devices such as **discs**, drums and tapes.

batch [bætʃ] A set of jobs with similar features which will all be processed in a single computer run by the same computer program. Also, in the term batching of data, refers to the accumulation of data until there is a sufficient quantity to justify a computer run to process it (contrast with **realtime** working).

binary digit ['bainəri 'didʒit] A unit in the binary scale (see **radix**), with the value 0 or 1. Often abbreviated to **bit**.

bi-stable electronic circuits ['bai'steibl ilek'trɔnik 'sə:kits] Circuits with two possible stable states.

bit [bit] See **binary digit**.

block diagram ['blɔk 'daiəgræm] In programming, another word for the commoner **flowchart**. In general it means a visual representation of a process in which each element of the process is represented by a box containing a description of the element; the boxes are linked by lines.

Boolean logic ['bu:liən 'lɔdʒik] A deductive system which is concerned with classes, propositions and two-state circuit elements, represented by symbols and terms such as AND, OR, NAND, etc. Invented by the English mathematician George Boole (1815–1864).

branch [brɑ:ntʃ] In computer programming, a jump to another instruction which is not the instruction immediately following the branch instruction in the program as written. Branches may be conditional, depending on the result of a test, or unconditional.

branch points ['brɑ:ntʃ ˌpɔints] Places in a computer program where a choice is made between obeying different branches of the program, the choice usually depending on the input **data**.

buffers ['bʌfəz] Staging-posts, intermediate states or locations where information transmitted from one device waits until it can be accepted or processed by another. Both a **hardware** and a **software** term.

card wreck ['kɑ:d ˌrek] The result of the card-reader misreading a punched card, either because of a **hardware** malfunction, or because the card is old, or non-standard. A wreck often means that one or more cards in the pack are mutilated.

cathode ray screen ['kæθoud 'rei ˌskri:n] The anode of a cathode ray tube: the screen on which an image is created by an electron beam.

central processing unit (CPU) ['sentrəl 'prousesiŋ ˌju:nit] The heart or brain of a **computer installation**, consisting of main memory, logic and arithmetic and control units, embodied in electronic circuitry.

character ['kæriktə] A unit in a language, such as the numerals 0 to 9, the letters A to Z, and various conventional symbols such as %, £, +, represented in the computer by a group of **binary digits**. The number of bits depends on the machine.

chatter ['tʃætə] The messages typed on a console typewriter by the machine and by the operator in the course of a computer run.

circuitry ['sə:kitri] **Hardware** elements forming electric circuits.

clock pulse ['klɔk ˌpʌls] Electronic pulses emitted within the computer to control the timing of all activities of the machine.

collator [kə'leitə] A data processing device, invented before computers, which performed certain operations on packs of punched cards such as merging them into a single, ordered sequence.

compiler [kəm'pailə] A computer program that translates other programs written in **high-level languages** into **machine code**.

computer installation [kəm'pjutə instəˌleiʃən] A collection of machines, comprising a **central processing unit** and various **peripherals**.

computerization [kəm'pju:tərai,zeiʃən] The transferring of the control of various activities (e.g. accounting) to a computer. Applies to all aspects of the installation of a computer.

computer program [kəm'pju:tə ˌprougræm] A set of instructions, which may be written in one of a large number of computer languages, designed to solve a particular data processing problem. If the program is in **machine code** it may be obeyed directly by the machine when punched, e.g. in cards or paper tape. If it is in **assembly code** or a **high-level language** it will have to be translated into machine code by an **assembler** or **compiler** before it can be obeyed by the machine.

console ['konsoul] The **interface** between the machine and the human operator, by means of which the operator gives his instructions, and the machine informs the operator of the progress of a run. Nowadays the console is usually a typewriter; in the past it often was a display of lights and switches.

conversational mode [kɔnvə'seiʃənəl ˌmoud] A way of using computers in which the user communicates with the machine on a question-and-answer basis, for example **debugging** a program in **realtime**. Equivalent to **interactive mode**.

core store ['kɔ: ˌstɔ:] The memory store of a **central processing unit**, consisting in most present-day machines of ferrite **magnetic cores**.

critical path ['kritikəl 'pɑ:θ] The path through a **PERT** network which shows the least time in which a project can be completed; if any activity is delayed on the critical path, the completion of the whole project will be delayed.

cybernetics [ˌsaibə'netiks] The science or study of the way in which a system, whether organic or artificial, is controlled by a central headquarters.

data ['deitə] Information. Specifically, the values of variables on which a computer program is to operate on a particular occasion.

data bank ['deitə ˌbæŋk] A large number of files of data, usually held on a magnetic device and available to many different users, often working in **realtime**.

data capture ['deitə ˌkæptʃə] The instantaneous recording of new data.

data processing ['deitə 'prousesiŋ] The analysis of data to arrive at required statistical results, as well as results for individual cases.

debugging ['di:'bʌgiŋ] This inelegant term is in constant use to describe the testing of programs under development by locating and removing the "bugs" or errors in them.

decision table [di'siʒən ˌteibl] The representation in tabular form of a series of tests to be applied to **data**. Decision tables are not only a convenient substitute for **flowcharts** in many cases, but can also be converted into programming language statements by special **preprocessor** programs.

diagnostic information [daiəg'nɔstik infə'meiʃən] Information, such as the contents of store locations, supplied at the end of a program run to aid in the **debugging** of a program under development. The information is usually printed.

digit ['didʒit] A numerical element. In the scale of ten the digits are 0 to 9, in the binary scale there are only two digits, 0 and 1.

digital computer ['didʒitəl kəm'pju:tə] A machine in which **data** is represented, not by physical analogues, but by discrete hardware elements such as **magnetic cores**.

direct access [di'rekt 'ækses] A type of magnetic storage in which the access time of any given items of **data** does not depend on their relative positions in the store, but is approximately the same whatever their position. Discs and drums are direct access devices; magnetic tape is not.

disc [disk] A magnetic storage device on which data is recorded on a number of concentric, circular tracks. A disc unit contains one or more discs. The discs may be either exchangeable, when they can be removed and replaced, or fixed.

documentation [ˌdɔkju:men'teiʃən] The record of a problem and its solution on paper. Typical documentation might consist of: the analysis of the problem by a **systems analyst**; **flowcharts** of the program written to solve it; written descriptions of the program; and printed computer listings.

dump [dʌmp] Both a verb and a noun. A dump is the periodical transferring of information from the computer memory to a peripheral medium, or from one peripheral medium to another. Dumps from **core store** preserve the current state of a program and its **data** in case there should be a hardware failure before the next dump. Dumps from, e.g. a **disc** to a magnetic tape are to preserve the contents of the disc on another device in case it should become corrupted. Dumping is therefore of cardinal importance in file protection and preservation, and is also a way of saving expensive machine time in case the machine fails in the course of a long computer run.

duplex ['dju:pleks] A term used of a **transmission line** which allows information to pass in both directions, simultaneously.

exchangeable disc [iks'tʃeindʒəbl ˌdisk] See **disc.**

fail-soft ['feil ˌsɔft] The ability of a computer installation or a software system to continue working as well as possible, or with minimum degradation of performance, even when there has been a major hardware or software failure.

feedback ['fi:dbæk] The transmission of the results of an operation back to its initiator, as soon as they become available. The term was originally used in **cybernetics** and referred, for example, to a chemical process monitored and controlled by a central system which issued instructions, from moment to moment, based on the results of its previous instructions, which were fed back to it. Feedback is an important concept in describing the behaviour of both human beings and machines.

field [fi:ld] An element in, or part of a **record.**

file [fail] A collection of records. A personnel file, for example, would contain information about all the individuals in an establishment. The information about each individual would constitute his record; such information might include his name, age, marital status, salary, grade and so on, and each of these items would constitute a **field** in his record.

fixed disc ['fikst 'disk] See **disc.**

float [flout] A term used in **PERT** network analysis to describe the spare time available for activities which are not on the **critical path.**

floating labels ['floutiŋ 'leiblz] **Labels** used in **autocodes** may be termed "floating" because they may be attached to different groups of instructions on different occasions.

flowchart ['flouʃɑ:t] Both a verb and a noun. A flowchart is the representation of the solution of a problem by means of a diagram. The elements are linked by lines to show the sequence in which they are to be performed, and there will be special means of indicating conditional branches (see **branch**).

format ['fɔ:mæt] The layout of a data record specified by a **systems analyst** or **programmer**: for example, the composition of a print line, or the character positions occupied by each field in the record when it is held on a punched card or in a magnetic tape block.

function ['fʌŋkʃən] The imperative part of an order in a machine's **order code** e.g. "Add", "Read", "Branch if—".

games theory ['geimz ˌθiəri] A field of mathematics concerned with the determination of optimum strategy in a conflict between two or more contestants.

gate [geit] A hardware term referring to a circuit in which a single output signal is derived from one or more input signals. Used in the performance of **Boolean logic** instructions.

graphical display unit ['græfikl dis'plei ˌju:nit] Equivalent to **visual display unit.**

half-duplex ['hɑ:f ˌdju:pleks] A term used of a **transmission line** which allows information to pass in both directions, but not simultaneously.

hardware ['hɑ:dwɛə] The physical machinery in a computer installation.

high-level languages ['hai 'levl 'læŋgwidʒiz] Computer languages which resemble natural languages such as English, or mathematical expressions. COBOL, FORTRAN and ALGOL are examples. Low-level languages (such as assembly languages) bear a close resemblance to **machine code**.

illegal [i'li:gəl] A program "goes illegal" during a run when, through an error in the program, the computer is asked to perform an instruction which does not exist in its order code, or cannot be performed in its context.

immediate access store [i'mi:diət 'ækses ˌstɔ:] The computer memory, in which all locations can be reached equally quickly; the access time is one-thousandth of the time required to reach locations on backing-stores such as discs and drums.

information bank [infə'meiʃən ˌbæŋk] Synonymous with **data bank**.

input ['input] The presentation of data or programs to a **central processor** via an input peripheral. *Also used as a verb.*

instruction [in'strʌkʃən] An order in a machine's **order code**.

integrated circuit ['intigreitid 'sə:kit] A circuit produced by chemical means on a single physical element, such as a silicon chip.

integrated management information system (IMIS) ['intigreitid 'mænidʒmənt infə'meiʃən ˌsistəm] A term which expresses an aim rather than an existing reality: the aim of designing a computer system which will not only handle all the data processing of a firm, but will issue periodic reports on which management decisions at all levels can be made.

interactive mode [intər'æktiv 'moud] Synonymous with **conversational mode**.

interface ['intəfeis] The connection between two hardware units: the meeting-place where the **output** from one unit is converted to a form acceptable as the **input** to another unit. The term is also used in **software**.

intermediate tables [intə'mi:diət ˌteiblz] Information built up in tabular form by a computer program to help in solving a particular problem. They are called "intermediate" because they do not form part of the final results.

interpreter [in'tə:pritə] A **data processing** device that prints the English equivalent of a punched code on the card on which it is punched: *or* a computer program which performs, in terms of the order code of the machine on which it is run, instructions written in the **order code** or **assembly code** of another machine. Differs from a translator program, which translates one language into another, and so requires an extra machine run before the original program can be obeyed. But whereas a program only needs to be translated once, it has to be interpreted each time it is run; in addition, the running of an interpreted program is usually much slower than the running of a translated program.

interrogating typewriter [in'terəgeitiŋ 'taipraitə] An input/output device by means of which a user, often remote from the computer installation, can monitor the progress of his program run, modify it, and receive reports from the machine.

job mix ['dʒɔb miks] A selection of programs to be run simultaneously. The selection is made by the operating staff on the basis of certain criteria: e.g. that there should be a mixture of "peripheral-bound" programs with "mill-bound" programs or that no programs should be run together if they require the same, unique peripherals.

label ['leibl] Has several different meanings in computer programming, e.g. the identifying name given to a group of instructions, or to a file of information. There are also end-of-tape labels, containing control information relating to the file. *Also used as a verb.*

layout ['leiaut] Equivalent to **format**.

library ['laibrəri] Usually refers to a collection of **data**, programs or subroutines required in many different applications, and so needing to be available to every user. Such libraries are usually held on magnetic devices.

light pens ['lait ˌpenz] An input/output device by means of which images on a **cathode ray screen** can be detected, or modified.

linear programming ['liniə 'prougræmiŋ] A field of mathematics whose aim is the best solution of a problem with a given set of variables and subject to a given set of constraints. Used, e.g., to calculate the quickest route for a lorry to take when making a number of deliveries.

liveware ['laivwɛə] Computer personnel.

log [lɔg] The record of a computer run, often in the form of a sequence of messages output by a console typewriter. *Also used as a verb.*

logic unit ['lɔdʒik ˌju:nit] The circuits in the **central processor** which perform logical instructions such as AND and OR (see **Boolean logic**).

loop [lu:p] A sequence of instructions in a program which are performed over and over again until a certain condition is fulfilled. *Also used as a verb.*

machine code [mə'ʃi:n ˌkoud] The basic language of a computer, represented in printed form as groups of numbers and consisting of a set of **orders**. An order can be punched as a pattern of holes on a card, or as a group of magnetic polarities on a tape. When read by the machine this is converted into electronic impulses switching the magnetic cores either "on" or "off". Other computer languages such as **assembly code** and **autocodes** must be translated into **machine code** before they can be obeyed by the computer.

magnetic core [mæg'netik 'kɔ:] A minute ring made of ceramic coated with ferric oxide, and threaded by fine wires. By passing electric currents through these wires, the core can be magnetized either in one direction or the other. Most present-day computer memories consist of many thousands of these rings, arranged in matrices (see **matrix**).

magnetic ink character recognition (MICR) [mæg'netik 'iŋk 'kæriktə rekəgˌniʃən] The encoding and reading of information recorded on paper by means of magnetized ink.

maintenance ['meintənəns] The day-to-day servicing of **hardware** by operators and electronic engineers. Also, when applied to **software**, the updating and amending of computer programs.

matrix ['meitriks] In **hardware**, a frame, e.g. of **magnetic cores**, and in **software**, often applied to tables in general; in mathematics, an array of numbers.

medium ['mi:diəm] The physical form in which **data** is held. **Discs**, drums and magnetic tapes are fast media; punched cards and paper tape are slow media.

memory bank ['meməri ˌbæŋk] That part of the central processor in which **data** and programs are stored, and from which they are retrieved, as opposed to, e.g., the logic and arithmetic units where **functions** are performed.

microsecond ['maikrou ˌsekənd] One-millionth of a second.

mill [mil] An old term for the **central processing unit**.

millisecond ['milisekənd] One-thousandth of a second.

mill-time ['mil taim] That part of a computer run during which the central processor is active, as opposed to peripheral time, i.e. the time consumed during a run by peripheral activity.

module ['mɔdju:l] A convenient unit, either of **hardware** or **software**, which can be combined in different ways with other modules to form larger wholes.

monitor ['mɔnitə] To record and comment on the progress of an operation (usually a program run).

monitoring file ['mɔnitəriŋ ˌfail] The file in which the progress of an operation (usually a program run) is recorded. The file is usually stored on a magnetic device, from which it can be printed out.

multi-access ['mʌlti 'ækses] The ability of a computer installation to allow many users to use the computer, often from remote locations, effectively at the same time.

multiplexor ['mʌltipleksə] A hardware device linking a central processor to a large number of communications devices (such as remote consoles), which it services in turn.

multi-programming ['mʌlti 'prougræmiŋ] The running of more than one program at the same time.

name tags ['neim tægz] **Labels** attached to branch points.

numerical control [nju:'merikl kən'troul] The control of machinery by means of computer programs, i.e. lists of instructions in a numerical code. The programs may be punched on to paper tape and read by a **peripheral** linked to a machine tool, together with a small processor which translates the program instructions into electrical impulses, which in turn control the tool.

offline ['ɔflain] Originally meant "not connected to the **central processor**". Data processing units such as **collators** and **interpreters** were offline devices. Has since come to refer to an input or output instruction which is not performed as written at the time the program is run. *Also used as a verb.*

online ['ɔnlain] The reverse of **offline** in both its meanings. If a program prints a line when it is run, the printing is said to be onlined; if, when the program is run, the print instruction is trapped by the operating system and causes a record to be written on e.g. a magnetic tape, from which it is printed out on another occasion, the printing is said to be offlined.

operand ['ɔpərænd] The variable on which a **function** operates.

operating instructions ['ɔpəreitiŋ in'strʌkʃənz] The written instructions given by a programmer to an operator on e.g. how a program is to be run, what to do in the case of various program events, or in the case of a failure.

optical character recognition (OCR) ['ɔptikəl 'kæriktə rekəg,niʃən] The ability of a machine to read characters written in the natural alphabets, instead of being encoded into holes punched on cards, magnetic polarities, etc.

optimize ['ɔptimaiz] To make something work in the best possible way in a given set of circumstances. Can apply to a program, an algorithm, a system, etc.

order code ['ɔ:də ˌkoud] The list of **functions** or instructions which the **hardware** of a computer is designed to obey.

output ['autput] The transmission of results by a computer to a physical medium, such as punched cards, magnetic tape, printed stationery, etc. *Also used as a verb.*

paper low ['peipə ˌlou] The condition of a printer, signalled to an operator on his **console**, when the printer is about to run out of stationery.

parallel running ['pærəlel 'rʌniŋ] Running an old system and the new one designed to replace it at the same time, using the same data, to make sure the new system produces the same results as the old. Particularly applies to **computerization**; until a computer system is proved to work it is wise to continue to run the old, clerical system in parallel with the new.

parameter [pə'ræmitə] A variable whose particular value on a given occasion is an item of data: e.g. in a personnel record, age would be a parameter; if Mr. Jones is 45, 45 is an item of data being the particular value of the parameter "age" for Mr. Jones' record.

peripheral [pə'rifərəl] An input or output device.

PERT [pə:t] The acronym for Program Evaluation and Review Technique. A method of controlling and monitoring the progress of a large project by breaking up the project into activities and events, and assigning estimates of cost, time and manpower to each activity in order to arrive at estimates for the total project.

plugboard ['plʌgbɔ:d] Physical frames containing sockets into which plugs could be inserted—a means of programming tabulators.

preprocessor ['pri:'prousesə] A computer program performing a specific function, such as the conversion of data from one form into another, before the data is processed by the main program.

printed circuit ['printid 'sə:kit] An electrical circuit printed or etched by chemical means instead of wired.

priority [prai'ɔriti] The relative importance of something in a group, e.g. of one program in a **batch**. Priority is usually assigned in the form of a numerical value.

problem-oriented ['prɔbləm ˌɔ:ri:entid] Usually applied to a computer language. A problem-oriented language takes into account the problems whose solutions, in the form of programs, are to be written in that language. The term is opposed to "machine-oriented"; a machine-oriented language, such as assembly code, is designed to accommodate itself to the hardware of a particular machine.

process control ['prouses kən,troul] The control of manufacturing processes by computer programs. Similar to numerical control, except that it refers to processes (such as the manufacturing of chemicals) rather than machines.

processor ['prousesə] See **central processing unit.**

program ['prougræm] A list of instructions written in a programming language.

programmer ['prougræmə] One who translates a problem into a sequence of instructions written in a computer language. To do this he often uses **flowcharts** and other software tools.

pseudo-code ['sju:dou ˌkoud] A programming language, whether **assembly code** or an **autocode**, which must be translated into machine code before a program written in it can be obeyed by a computer.

punch [pʌnʃ] A hardware device for punching holes in cards (a card-punch), or paper tape (a papertape punch). *Also used as a verb.*

quantify ['kwɔntifai] To give numerical values to the elements of a problem.

queueing theory ['kju:iŋ ˌθiəri] A form of probability theory used in the study of problems in which the demand for a service is irregular and which therefore requires that waiting space and possibly priorities be determined.

radix ['rædiks] The base or scale of a numerical system. The radix of the decimal system is ten; the radix of the binary system is two, and of the octal system is eight.

realtime ['riəl'taim] The processing of an item of **data** by using it to update a file almost as soon as it is generated. Opposed to batch processing, in which **data** is collected over a period of time and is only processed when a sufficient quantity has been collected to justify a computer run.

records ['rekɔ:dz] The units which compose a **file**.

register ['redʒistə] A location in the computer memory which is designed to perform a specific function, or to temporarily hold the result of performing a function.

report [ri'pɔ:t] Any set of printed results. *Also used as a verb.*

run [rʌn] A computer run or program run is the obeying of a program. *Also used as a verb.*

running live ['rʌniŋ 'laiv] Putting a computer system into effect, using real data, as opposed to merely testing the system using artificial data.

scheduling ['ʃedju:liŋ] As applied to operating systems, means the resolution of priorities among a set of programs which are to be run simultaneously, or **multi-programmed**.

simplex ['simpleks] A term used of a **transmission line** which allows information to pass in one direction only.

simulation [simju:'leiʃən] Use of a computer program to model the significant characteristics of some real situation, where live experimentation might be expensive or dangerous.

simultaneous working [ˌsiməl'teiniəs 'wə:kiŋ] A means of using the **central processor** and a **peripheral**, or one peripheral and another, at the same time. This can be achieved by the way a program is written, by an operating system or the way a machine is designed, or by any of these in combination.

slide-rule ['slaid ru:l] A simple calculating device which works on an analogue principle; numerical quantities are represented by lengths on a ruler, and arithmetic operations are performed on them by moving slides on the ruler.

software ['sɔftwɛə] Computer programs, as opposed to **hardware**, the physical

machinery. Nowadays the term software is often restricted to the set of standard programs offered by a manufacturer together with his machine.

specification [ˌspesifi'keiʃən] The analysis of a problem, defining what it is and how it is to be solved.

standardization [ˌstændədai'zeiʃən] The imposing of standards, e.g. on data formats, programming, systems, etc.

store address ['stɔ: əˌdres] A number referring to a particular location in **core store**.

stored program ['stɔ:d 'prougræm] A program which is read into the computer store for the duration of a run, and deleted from store at the end of the run.

subroutine ['sʌbru:ti:n] A set of instructions designed to perform a specific task, and capable of being obeyed over and over again by a program, or by more than one program. It is therefore ideally written as a self-contained **module**.

subset ['sʌbset] A set which is part of another, greater set. The letters H to M are a subset of the alphabet.

systems analyst ['sistəmz ˌænəlist] The individual who analyses a data processing problem and writes a **specification** which a computer programmer is to follow in the program he writes.

tabulator ['tæbju:leitə] A data processing machine which antedated the computer. Programs could be stored in it in the form of **plugboards**.

tag [tæg] Has several different meanings in computer programming. A tag can be the identification of a magnetic tape record, or a word attached to a group of **direct access** records giving the address of other records which follow them in sequence but are held in different locations. Tag can also be equivalent to **label**. *Also used as a verb.*

tally ['tæli] *Noun and verb.* A count, or to keep count of.

tape decks ['teip deks] The hardware units on which magnetic tapes are mounted for reading or writing.

test data ['test deitə] **Data,** often artificially created, to test a program under development.

thermionic valve ['θə:miɔnik 'vælv] An electronic device used both in computers and radios, e.g. for modifying or amplifying electromagnetic signals.

thin film ['θin 'film] A form of computer storage alternative to **magnetic cores,** with a very fast access time.

throughput ['θru:put] The volume of work done, or programs run, by a computer in a given period of time.

time-sharing ['taim ˌʃeəriŋ] The simultaneous use of a computer installation by a number of realtime users, often working in parallel with **background jobs**.

trace [treis] *Noun and verb.* A trace is a means of charting the progress of a program run by, e.g. printing out the contents of specified core locations at certain points. An aid to **debugging** which, though valuable, is a last resort, because it is more costly in machine time than using human intelligence and intuition.

transistor [trænˌzistə] A solid-state semi-conductor which in most applications, e.g. in computers and radios, has replaced the **thermionic valve**. It is smaller, more robust and requires less power.

transmission lines [trænz'miʃən ˌlainz] The communication channels between a computer and a remote input/output device (e.g. telephone and telegraph lines).

tunnel diodes ['tʌnəl 'daioudz] A diode is a two-terminal switching device; a tunnel diode has special properties first discovered by the Japanese scientist Esaki.

typewriters ['taipraitəz] All computer typewriters are "two-way" or "interrogating", because not only can the user type his messages to the computer on them, or input data, but the computer can type back messages or results to the user. A "console typewriter" is one used by the computer operator and thus forming part of the computer installation.

uniplexor ['ju:nipleksə] A hardware device linking the central processor to a communications unit such as a remote console.

update ['ʌp'deit] To amend a file by incorporating fresh **data**.

utility routine [ju:'tiliti ru:ˌti:n] A standard library program of frequent use in all computer systems, such as a program for printing out the contents of a magnetic tape.

verifier ['verifaiə] A data processing machine which checks the punching of **data** on cards.

visual display unit (VDU) ['viʒuəl dis'plei ˌju:nit] A cathode ray tube used for displaying **data** either in character or diagrammatic form.

word [wə:d] In computer usage, a basic conceptual unit in a central processor consisting of a group of **binary digits**. The number of digits varies from machine to machine. Some machines have a fixed word length, which means that every instruction in its **order code** must be of that length; others have a variable word length.

American terms

There is little difference between the special computing terms used in England and in the USA. This is mainly because the use of computers began earlier and has become far more widespread in America than in Europe; and most literature on computers and computer techniques has been written in America. Therefore, Britons have been willing to adopt American terms; they even use the American spelling "program" when referring to computer programs, though the word "programme", which is used outside the computer field, already existed.

Of the words in this book the average American would use "CPU time" in place of our "mill time" and would speak of a "vacuum tube" where we talk of a "thermionic valve".

Each computer manufacturer tends to favour his own expressions, and one may therefore find a greater difference between the language used in one installation and another than between computing terms encountered on one side of the Atlantic and the other.

COLLIER MACMILLAN ENGLISH PROGRAMME

Edward Humby, A.C.W.A., F.B.C.S., is the manager of a software department at International Computers Limited. Philip Robinson is a technical officer with the same firm. Both have contributed to computer journals and books.

PETER STREVENS, formerly Professor of Applied Linguistics and Director of the Language Centre at the University of Essex, is co-author of ENGLISH 901. He is General Editor of the Special English series.

SPECIAL ENGLISH is a series providing second language instruction for students with specialized interests, within a programme that includes basic courses, supplementary readers, practice, reference and remedial material, tape recordings and gramophone records.

COLLIER MACMILLAN INTERNATIONAL INC.
866 Third Avenue, New York, N.Y. 10022

CASSELL & COLLIER MACMILLAN PUBLISHERS LTD.
35 Red Lion Square, London WC1R 4SG

ISBN 0 02 974870 4