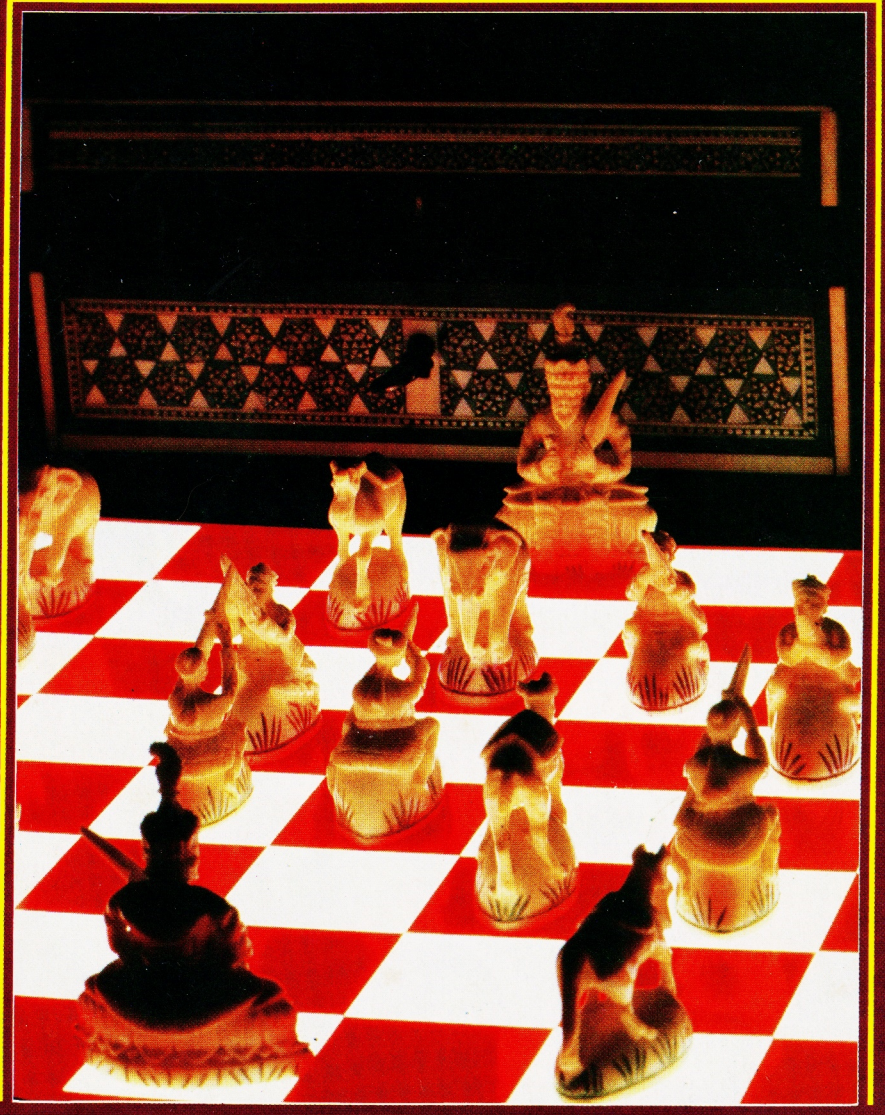
AN ARGUS SPECIALIST PUBLICATION

October 1985 £1.10

COMPUTING Today



WARGAMES
campaigns by computer

OMNI READER
cut price character recognition

HASHING
no more late data

FRAMEWORK
writing a graphics design program

LITTLE GIANT
ACS PX-1000 text processor reviewed

FROM THE CLASSROOM
teacher's eye view

fischertechnik®

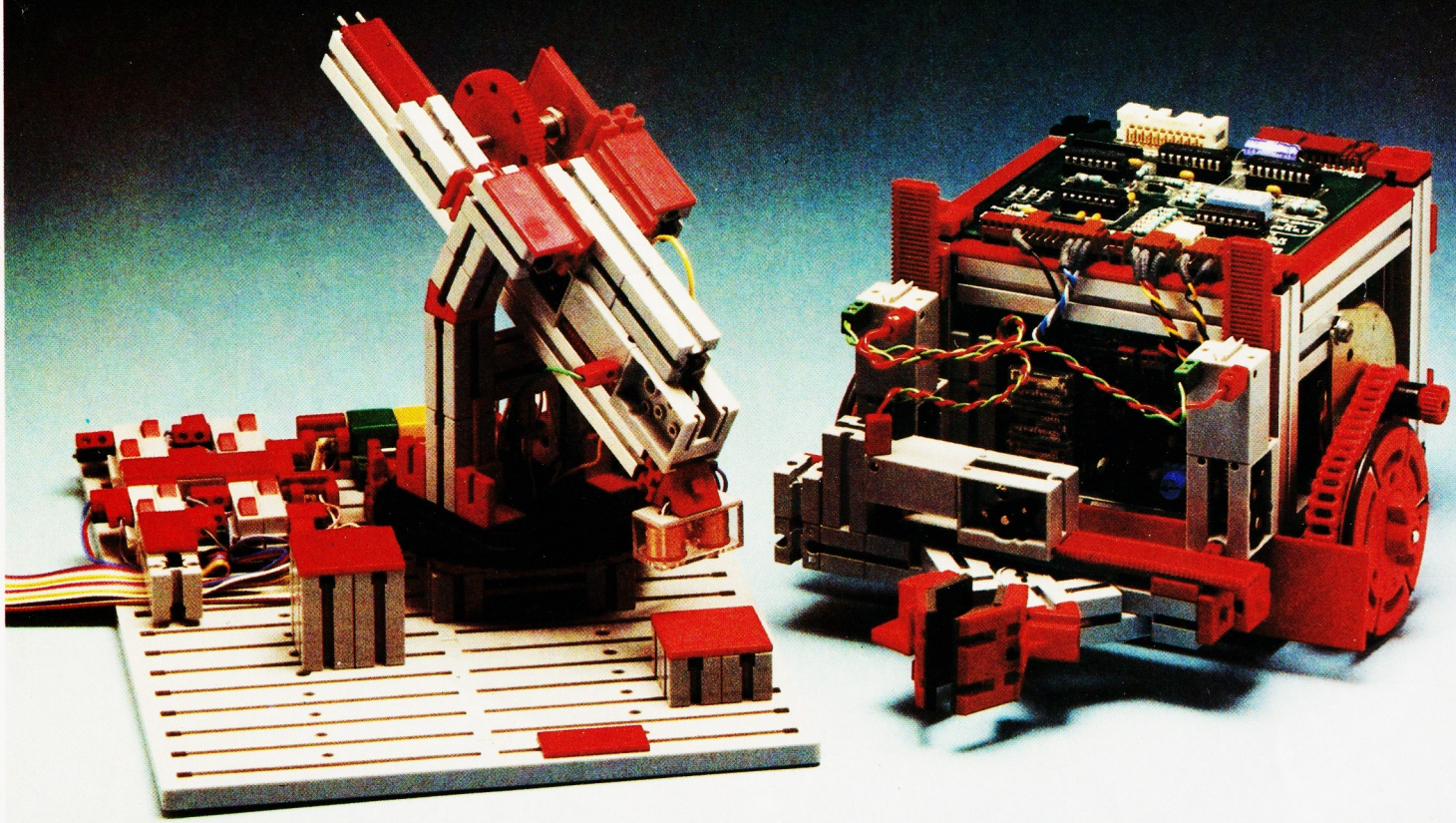
COMPUTER CONTROLLED ROBOTS

Fischertechnik Computing Kit

10 different models from one kit.
All suitable for computer control.

BBC Buggy

Computer controlled mobile robot.
Pen kit and Grab Arm now available.



Contact your regular stockist or main Fischertechnik distributors,
Economatics (Education) Ltd., Epic House, Orgreave Road, Handsworth, Sheffield S13 9LQ
Telephone Sheffield (0742) 690801



COVER

Ths month's cover features a chess set bought by the editor in the far east many years ago. Wargaming starts on p36. (PS: No prizes for spotting the mistake on thsi proof!)

Editor:
Don Thomasson
Assistant Editor:
Jamie Clary
Technical Illustrator:
Jerry Fowler
Additional Illustration:
Grant Robertson
Advertisement Manager:
Anthony Shelton
Classified Sales Executive:
Caroline Falkner
Advertisement Copy Control: Sue Couchman, Lynn Collis
Publishing Director:
Peter Welham
Origination and Design:
Design International
Cover Design:
Argus Design

ABC **Member of the Audit Bureau of Circulation**
ISSN 0142-7210

Computing Today is normally published on the second Friday in the month preceeding the cover date. Distributed by: Argus Press Sales & Distribution Ltd, 12-18 Paul Street, Lodnon EC2A 4JS. 01-247 8233. Printed by: Alabaster Passmore & Sons Ltd, Maidstone, Kent.

The contents of this publication including all articles, designs, plans, drawings and programs and all copyright and other intellectual property rights therein belong to Argus Specialist Publications Limited. All rights conferred by the Laws of Copyright and other intellectual property rights and by virtue of international copyright conventions are specifically reserved to Argus Specialist Publications Limited and any reproduction requires the prior written consent of the Company. © 1985 Argus Specialist Publications Limited.

Subscription notes; UK (£16.20) including postage. Airmail and other rates upon application to Computing Today Subscriptions Department, Infonet Ltd., Times House, 179 The Marlowes, Hemel Hempstead, Herts. HP1 1BB England (phone 0442 48432).

CONTENTS

VOL 7 NO 10 OCTOBER 1985

REGULARS

NEWS 4

TALKING SHOP 7
Some of the ills of the home computer industry considered.

FROM THE CLASSROOM 20
A new regular column for teachers and the taught in school computing.

ALGORITHM ANGLES 23
What do you do if you want to use a neat piece of programming in the main program elsewhere without the chore of re-writing or re-numbering?

BOOK PAGE 42
Computerised detective agencies and more, in some historical future-gazing.

SERIES

WARGAMES 36
It's better to pretend than to actually do it, and this series will look at how computers can help the pretence.

LESSONS OF HISTORY 40
We chart the development of computer languages, from assembler via BASIC to fossilisation.

GENERAL FEATURES

HASHING 19
Speed up searching of data files of all sorts using this simple method.

ADDING BASIC COMMANDS 26
Ever though fo extending your micro's BASIC yourself? Here's how one reader did it for himself.

BLOCKCODE 29
Subtitled 'Goodbye GOTO', this describes a proposed new language (or pseudo-language) which encourages good programming habits.

SPECIAL FEATURES

GRAPHICS DESIGN PROGRAM ... 12
Taking the labour out of creating computer graphics — a readily transposible utility, written originally for the BBC B.

GENIE ADJUSTMENTS 18
Converting MCBAS, VGBASI and SPRITE to a disc-based system.

REVIEW: PX1000 TEXT PROCESSOR 33
A portable WP that will fit in your pocket without tearing a hole — except metaphorically.

REVIEW: OMNI-READER 24
A text recognition system that might make the electronic office a little chapter — if not nearer.

Next Month's 'Computing Today' 22

We regret that due to lack of space, the third part of the series 'Learn Unix' has been held over to next month.

EDITORIAL & ADVERTISEMENT OFFICE: 1 Golden Square, London W1R 3AB. Telephone: 01-437 0626. Telex: 8811896.

NEWS

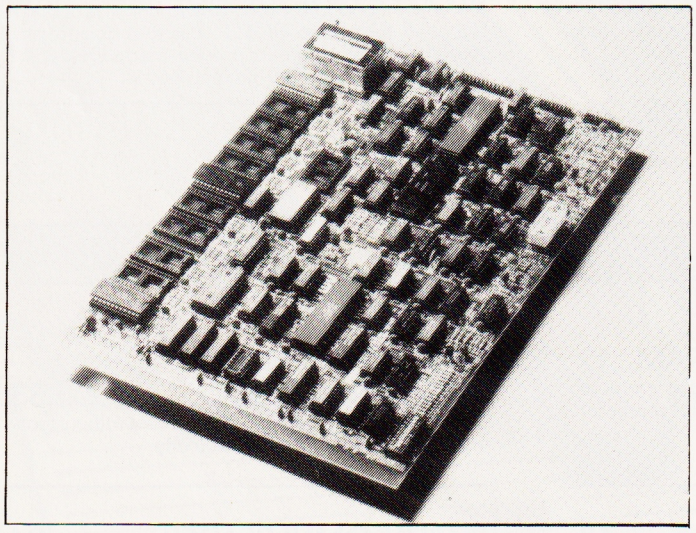
NASCOM REVIVAL

While the fortunes of the micro-computer business continue to diminish, Lucas Control Systems have re-launched the NASCOM 2 range of single board computers and ancillaries at a time when its sales are steadily increasing.

The NASCOM 2 is claimed to offer in a single board computer sufficient computing power and interface circuitry for the majority of industrial applications. It is available with very powerful software programming aids and is used, say Lucas, in applications as varied as automatic test equipment, data collection and analysis, label printing, lift support, and retail cash terminals. Further hardware extensions can be made via the NASBUS edge connector which is a format supported by Lucas and other suppliers. Lucas can offer additional memory, graphics controllers and hard floppy disc controllers hardware extensions, as well as a software and hardware design facility for custom specials.

The General Manager of Lucas Control Systems Peter Seddon remains optimistic about the growth of the NASCOM 2 business for industrial applications and says "We have not been as successful as we had hoped with our business machines, which also use the NASCOM 2 products". He also points out that "Lucas Control Systems are not just offering a standard product but a complete engineering hardware and software service, based upon nearly 5 years of applying NASCOM 2's to solve a wide range of problems".

For further information contact: Lucas Control Systems Ltd, Welton Road, Wedgnock Industrial Estate, Warwick CV34 5PZ. Telephone 0926 497733 Telex 312333.



● Epson are getting ready for a competitive market this autumn by introducing 'big savings' on two of their most popular printers, the RX-100+ and the FX-80+, plus some associated packages for word-processing. For example, the RX-100+ wide carriage 100 cps printer now has an RRP of £399, a reduction of £51.

● Cumana have introduced a new 'Super Value Disk Drive Starter Pack' for owners of BBC, Electron, Spectrum and Dragon micros who don't already have disc drives. However, the pack's are such good value that they couldn't — or wouldn't — tell us the prices! Try your local 'selected' retail outlet.

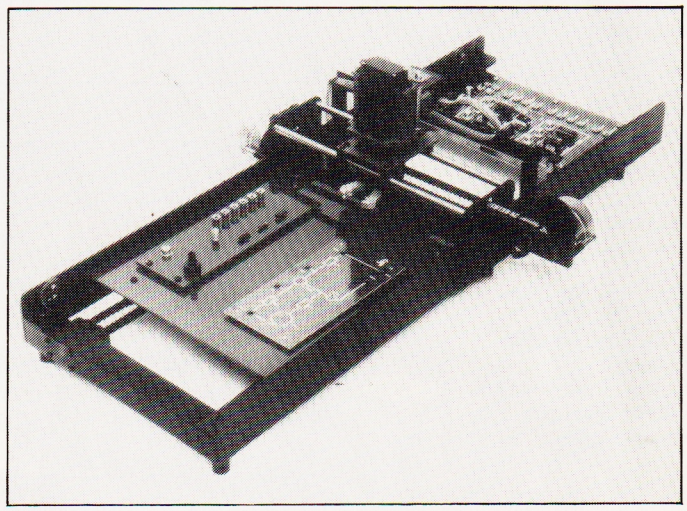
SINCLAIR COMPILER

Oxford Computer Systems announce that shipments of BLAST — the first fully compatible BASIC compiler to be released for any Sinclair computer — commenced this month.

BLAST can compile programs either directly into Z80 machine code or into a compact pseudo machine code (p-code). Users can expect a speed increase of up to 40 times that of BASIC for any program that will run under the interpreter. Any length of program can be compiled.

BLAST has been designed for maximum compatibility with Spectrum Basic — this extends to variable and program storage formats. This high level of compatibility with the interpreter enables machine code extensions to BASIC to be compiled successfully. BLAST provides a number of features which can be used to prevent unauthorised tampering with compiled programs.

BLAST also comes complete with a comprehensive tookit designed to aid program development. The BLAST compiler is available now and retails at a recommended price of £24.95. Oxford Computer Systems, Hensington Road, Woodstock, Oxfordshire, tel 0993 812700.



NEW ROBOTIC SYSTEM FROM LJ ELECTRONICS

The new LJ Electronics TRACER Robotic Teaching System has been specifically designed to provide a cost-effective introduction to the world of robotics.

Capable of being driven by any microcomputer, with a suitable TTL level I/O facility the TRACER features stepper motor drive on its X and Y axes and DC servo motor drive for the Z axis and gripper open/close. The unit is supplied with a pen carrier with three different coloured pens. This enables the user to perform numerous plotting tasks in up to three colours.

A PCB assembly task kit is also supplied with the unit. Using this kit students can write control routines to perform PCB assembly tasks, effectively simulating the industrial usages of XYZ robotic devices. For full details on the TRACER and other LJ Teaching Systems please contact: LJ Electronics Ltd, Francis Way, Bowthorpe Industrial Estate, Norwich NR5 9JA, tel: 0603 784001.

NEW COURSES IN INFORMATION TECHNOLOGY

The Business & Technician Education Council (BTEC) is launching new BTEC Higher National Diploma courses in Information Technology (IT) and Business Information Technology (BIT) this autumn.

These courses will initially be run on a pilot basis from September 1985 at all 11 polytechnics and colleges with the possiblity of three more centres being added which are still in the process of being approved.

The introduction of these courses comes in the wake of the findings of the IT Skills Shortages Committee under the Chairmanship of John Butcher MP, Junior Minister for Trade and Industry. His committee's second report, published earlier this year and focusing on the shortage of technician skills, concluded: '...it is clear that there is a need both for greater numbers of technicians who have particular IT skills and for a general level of IT skills in the technician workforce.'

The BTEC project which resulted in these new IT courses arose initially out of discussions in the winter of 1983/84 between the Council and representatives from the Department of Trade and Industry, the Engineering Council, the National Computer Centre, International Computers Ltd, the Manpower Services Commission and colleges and polytechnics. It was decided at that stage that BTEC should take the initiative in a three-stage project with the objective being the provision of technician level IT courses.

Stage one of the project, a series of detailed discussions with industry, was completed this spring. The second, the development of suitable curricula, is nearing completion while the third stage, the piloting of courses, is to begin with the autumn term.

Draft guidelines for the IT courses will be issued this autumn though the definitive documents will not be produced and distributed until full evaluation of the pilot schemes has taken place and any necessary modifications have been made. Two categories of courses have been identified, each with differing requirements in order to meet particular industrial requirements:

Category 1, BTEC HND in Information Technology, aims to provide technicians able to analyse, design and implement IT systems;

Category 2, BTEC HND in Business Information Technology, aims to educate practitioners who can promote, facilitate and integrate IT-based systems within commerce, industry and administration.

The new pilot courses operate in both these categories. In addition, the Council is encouraging the inclusion of IT applications in all courses for which it has responsibility. BTEC has already made progress in this direction by introducing IT modules to be incorporated where needed in studies in any of its nine board sectors.

The draft guidelines being prepared define the principal areas for category 1 and 2 courses, around which centres can develop their own proposals.

The 14 polys and colleges which are either definitely taking part in the piloting stage, or are hoping to do so are:

Brighton Polytechnic (0273) 693655

Buckinghamshire College of Higher Education (0494) 22141;

Dorset Institute of Higher Education (0202) 524111;

Farnborough College of Technology (0252) 515511;

Hatfield Polytechnic (0770) 268100;

Huddersfield Polytechnic (0484) 22288;

Lancashire Polytechnic (0772) 22141;

Leeds Polytechnic (0532) 462971;

Leicester Polytechnic (0533) 462971;

Manchester Polytechnic (061) 228 6171;

Newcastle Polytechnic (0632) 326002;

Slough College of Higher Education (0753) 34584;

Teeside Polytechnic (0642) 218121;

Thames Polytechnic (01) 845 2030.

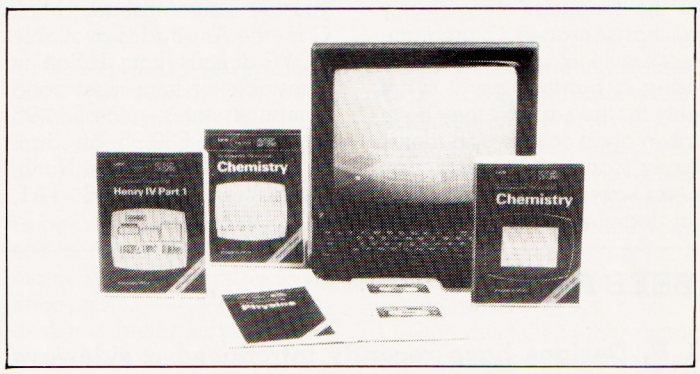
SEMINARS ON UNIX/ZENIX

Centre of Professional education are sponsoring authoritative seminars giving an evaluation of Unix and the Unix marketplace at the Sheraton Skyline, London, 7-8th October and repeated at Novotel Bonn-Hardberg, Bonn, 9-10th October.

This intensive 2-day seminar provides data processing professions with practical, no-nonsense answers to important questions such at 'What is UNIX and what are it's capabilities? What application software exists today?' These and other questions can be answered by Paul Para, specialist in UNIX training, support and software development. Having worked with UNIX since 1975, Mr Para has designed and implemented several software systems for UNIX including database management and office automation systems.

The course is highly suited to non-technical people needing to understand the benefits of UNIX as well as seasoned DP professionals wanting to know what the excitment is all about. Covering subjects like office automation software, communications, software development tools, unix and xenix on personal computer.

A detailed brochure containing a full agenda for the course can be obtained from Sue Brigham, Conference Manager, Media House, Weston Road, Slough, Berks. Slough (0753) 71011.



REVISION SOFTWARE

Charles Letts, market leaders in revision aid publishing, are launching Letts Keyfacts Revision Software.

The software, say Letts is not designed as an examination cramming-aid. It is planned and designed for use throughout the O-Level GCE, CSE and GCSE course.

Letts say they have utilised the expertise and knowledge gained through educational book publishing to produce eight sophisticated software packages that will greatly benefit the student. The software acts as a supplement and enhancement to book-based learning, rather than simply duplicating it.

The programs use a variety of approaches to maximise the students' interest, the aim being to encourage active participation by the student in their home studies. Keyfacts Revision Software claims extensive use of graphics presented in an imaginative, interactive way as a positive teaching aid.

The software runs on the BBC model B, the computer favoured by most schools, the Acorn Electron and Sinclair ZX Spectrum 48K. Later in the year programs will be available for the Commodore 64.

Letts Keyfacts Revision Software covers Biology, Chemistry, Computer Studies, English Literature (Henry IV and part I and the Merchant of Venice), Geography, Mathematics and Physics. Each package is presented in a plastic holder containing two cassette tapes. A booklet provides loading and running instructions together with details for the use of the program. Letts Keyfacts Revision Software is priced at £11.50 for each package. Charles Letts & Company Limited, Diary House, 77 Borough Road, London SE1 1DW.

C YOUR AMSTRAD GROW

HiSoft are very pleased to announce the availability of HiSoft C on the Amstrad computers; not real soon now but actually taking up space in stockroom.

The C programming language is becoming increasingly popular because of its combination of Pascal-like structured programming and easy machine-level interaction. This means that programs can take full advantage of the computer's environment (thus enabling fast and compact execution) while being easy to write and debug. Most of the leading software houses now write in C (e.g. Digital Research's GEM) and the C language is set to become the standard on the next generation of computers.

HiSoft say that their C on the Amstrad is a full specification compiler following Kernighan and Ritchie's definition of the language very closely so that seasoned C programmer's will feel very comfortable using it; but they say it is not only for the expert - they have taken great care to add many, many features to make HiSoft C an easy and interactive tool for learning C. There is an integrated editor which, they say, behaves just like Locomotive BASIC's editor, full support of the Amstrad's sound and graphics facilities, immediate execution allowing you to test out C features as you go along, full use of both tape and disc and an extensive manual.

A lot of effort has gone into producing the manual (now 150 pages in a ring binder) and it includes a complete guide to the C language, which, according to HiSoft, eliminates the need for any further books on the subject.

HiSoft think that, whether you are a newcomer to C or a seasoned expert, HiSoft C will permanently change the way you use your Amstrad. HiSoft C for the Amstrad is available NOW directly from HiSoft in Dunstable or from most good computer stores, price £34.95 on tape or £39.95 on disc. HiSoft, 180 High Street North, Dunstable, Beds LU6 1AT, (0582) 696421.

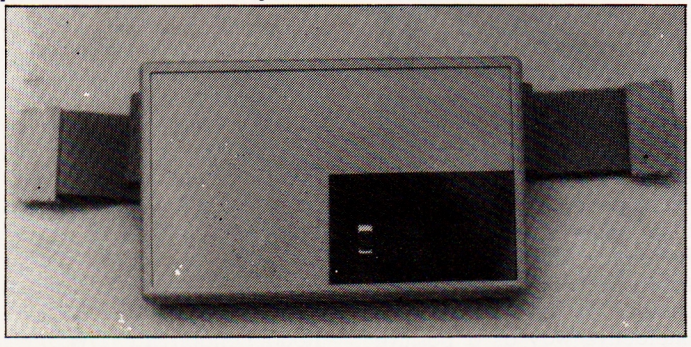
BEEB RAM

J. P. Designs have recently introduced a sideways RAM/emulator for use with the BBC Microcomputer, which allows the RAM to be shared by other external systems as well as providing the features normally found on sideways RAM units.

Supplied in a beige ABS box, to complement the BBC, this compact unit measures 150 x 95mm and is easily connected to the micro via three connectors which require no soldering.

The 16k static RAM is located between 8000-BFFF in the system memory map. The RAM can be written to directly, this feature allows machine code programs to be placed in the RAM by the assembler and can be used to develop software for small controller cards. A slide switch located on the top of the unit determines which system has access.

Retailing at £99.95 « VAT, BBC RAM comes complete with full fitting instrucitons and test routines. J.P. Designs. 37 Oyster Row, Cambridge, CB5 8LJ, Tel: Cambridge (0223) 322234.



COMPUTERAID SERVICES 1985 RECRUITMENT PROGRAMME

Computeraid Services, the independent maintenance operation within THORN EMI Information Technology specialising in IBM PCs and compatibles, is planning to expand its 88-strong staff to at least 108, over the remainder of 1985.

Of the twenty new recruits, a small proportion are intended to be trainees, including at least two graduates. Ten people are being sought immediately to enable Computeraid Services to cope with recently acquired maintenance contracts.

The positions to be filled will be mainly those of service engineers, but maintenance depot clerks and depot engineers are also being sought. Although Computeraid Services provides nationwide maintenance from its eight strategically placed service centres, the majority of the recruits will be based in the South of England.

Computeraid Services, only established since January 1984, is already one of the major third-party maintenance companies in teh UK. Commenting on the expansion, Maurice O'Brien, General Manager for Computeraid Services said, "Computeraid Services provides a competent and reliable service within an industry that is presently thriving. The demand for third party maintenance is constantly increasing, and we need to have the people in place in order to meet the resulting expansion." He added: "Up until now, Computeraid Services has limited its recruits to those with sufficient experience in the industry to enable them to start work almost immediately. Now we are pleased to have the spare resources to be able to take on trainees."

Conputeraid Services specialises in third-party maintenance of business PC and minicomputer systems and peripherals. It also supplies system handling and commissioning and quality control to manufacturers and importers. Based in Famborough, Computeraid Services has a nationwide chain of engineering depots and is prt of the Processing Services Division of THORN EMI Information Technology Limited. Further information from: Tony Carter, Computeraid Services, 21 Invincible Road, Famborough Hants GU14 7BR, tel: Famborough (0252) 548888.

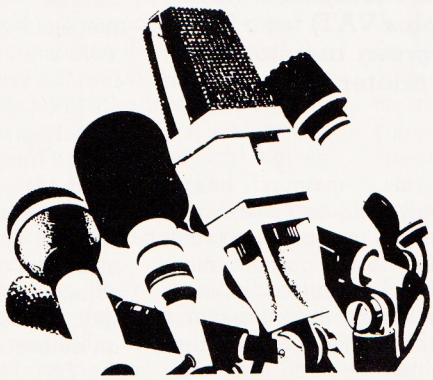
COMMODORES THAT THINK BBC

Micro Dealer U.K., the country's largest home computer distributor, has taken the Commodore BBC BASIC Emulator into stock, and been named exclusivel distributor by Aztec Software, the programming house involved in the development of the emulator, under the Shado label.

The Commodore BBC Emulator converts the Commodore 64 into a machine which can be programmed in exactly the same way as a BBC machine. The vast majority of commands available on the BBC are now available through the Commodore with a few exceptions; generally those caused by hardware differences between the two machines, such as the BBC Disk based random access commands, have not been implemented as the 1541 drive can not handle files in the same way as the BBC 8271 floppy disk controller.

The emulator provides the user with an 80 column mode, selectable by the MODE 0 command. In this mode, full hi-res graphics, upper and lower case characters and editing are possible. All error messages generated are the same as those given by the BBC computer, thus allowing the easier debugging of programs. The Commodore BBC Emulator comes complete with a self explanatory manual at a price of £14.95 and is available from all good retailers. For further information contact: Micro Dealer (UK), 29 Burrowfield, Welwyn Garden City, Hertfordshire AL7 4SS, Tel: (07073) 28181 (30 lines).

TALKING SHOP



Don Thomasson

During the past few months we have seen two major British computer manufacturers in difficulties, to the extent that they have needed outside financial help on a massive scale, but there is an even more ominous straw blowing in the wind. The wisp is difficult to observe, but glimpses of it appear now and then in an unmistakeable way. To see this clearly, we must first go back a few years.

In the mid nineteen-seventies the emergence of the microprocessor made microcomputers a viable proposition. The first system to achieve public recognition was the Pet, though the professional computer world was aware of a number of other systems that they might have considered more worthy of acclaim. The power of publicity, however, was more important than pure technical merit.

At the time the idea of a computer for home use had novelty value but little else to commend it. Users had to write most of the programs that they needed, off-the-shelf tapes not being widely available. This limited the potential market somewhat, and early sales were predominantly to buyers who had some prior knowledge of computers.

By 1980, a number of machines had appeared on the market, but none seemed to have gained a completely firm foothold on the ladder to success. To take an example which is especially familiar to me, the Sorcerer was given good marks for performance, but its price tag was a deterrent. When a cut-price batch came on to the market, it was snapped up with enthusiasm, however, and this led to the establishment of the European Sorcerer Club, a band of dedicated enthusiasts who did rather more than the official distributors to make the machine known. In Holland, the machine was adopted by the television service and later manufactured in that country, while it also sold well in Australia.

Despite the support of enthusiasts, which could scarcely compete with well-backed profesional support, the Sorcerer never really took off. Yet many are still in use, both in the homes and in professional establishments, and those who use them do not feel that there are many among more recent machines which are good enough to be considered as replacements.

In 1980, a new phenomenon appeared. It might have been called the 'nanocomputer', for it was cut down to the bare essentials, yet retained a useful degree of performance. The ZX80 was originally a kit of parts, which limited its market, but the ZX81, with its incredible chip-count of four, opened up wider fields. The keyboard was frustrating, and there were other disadvantages, but the price reduction carried the day, and the sales machine began to move into action.

The low price was soon seen to be a little deceptive. Cassette storage was cheap, abysmally slow and distressingly error-prone. Attempts to devise a better subsitute that would be cheaper than disc drives largely ended in failure, and as disc drive costs fell this kind of thing was abandoned — with one notable exception.

The need for an inexpensive printing device was met, after a fashion, but it was becoming clear that the reduction of the actual computer cost had not had a corresponding effect on the cost of a completely viable system.

So buyers of cheap systems learned the hard way that they needed to spend more than they had expected. The reactions we have heard, fall into four main groups;

- Those who have bought satisfactory equipment, at a cost of £1000-2000, and remain enthusiastic.
- Determined users who have made the most of what they have got, seeing the fight against limitations as a challenge.
- Those who have bought inadequate substitute devices, and have run into a maze of problems.
- Far too many who have given up in disgust, selling their computers or leaving them on a shelf to gather dust.

It is the fourth group who give cause for concern. Once discouraged, they are unlikely to try again, and their opinions may well discourage others. So what made them give up?

First, there are undoubtedly many who simply lacked the right mental approach, and who would have given up in any case, but others had more positive reasons. One complaint was that using the computer took up an excessive amount of time, and this is a valid objection. Long periods of sustained concentration are needed, and with cassette systems there may be long waits that are in themselves frustrating. Enthusiasm and dedication are both needed.

A more general objection was that the user manuals failed to provide enough information. Very few that we have seen could be called completely adequate and some have been really awful. Some machines have undoubtedly been wrecked because of the nature of their manuals, and others have suffered badly.

One difficulty is that almost every machine uses an idiosyncratic dialect of BASIC, so it is extremely difficult, if not impossible, to produce a universal tutorial that would cover all machines and therefore be economic to produce. Such a book, backed by a manual dealing with the special features of a given machine, would be a great help to novice users, and would almost certainly reduce the wastage of potential enthusiasts.

Whatever solution is adopted, it must be found soon if interest in home computing is to be sustained at the highest possible level. It is very much in the interest of the manufacturers that the potential market is maximised, because the industry needs to advance on as broad a front as possible.

We will do what we can to provide useful information and sustain interest, but the manufacturers — for their own good — must also play their part. Some have already got the message, and should benefit from that in the future.

IS IT A WP? IS IT A MICRO? NO, IT'S AN AMSTRAD.

Mid August saw one of the most extraordinary pieces of hype in the computing industry for a long time; it was to be 'the single most important computer launch of 1985', but what it proved to be was a fairly ordinary machine at a quite exceptional price. It was the Amstrad PCW8256 personal computer word-processor, and the price is £399 (plus VAT) for a 256K non-colour computer, including green monitor, disc drive and keyboard, plus a printer and word-processing (and other) software.

To some, the machine will be a bit of a disapointment, because it still uses an eight-bit processor (a Z80A), but according to Amstrad, this is perfectly adequate for virtualy everything the machine will be asked to do. When asked if he planned to produce a 16-bit machine, Amstrad's MD, Alan Sugar said "No one's explained to me why I should yet."

Are Amstrad following the right course? We believe they may be wiser than some pundits would have us think. Retaining the venerable Z80 allows compatability with the CP/M range of programs (OK, it's not the most up-to-date, but it's arguably the most extensive), and crafty bank-switching means that the usual limitation of memory size is avoided. As long ago as July 1983, we queried the merit of 16-bit processors in typical home and business applications, partly on the basis that the hardware is more expensive for a given level of performance (and this seems to be borne out by the price of the '8256).

To address 256K, and Z80 needs a bit of assistance, and this is provided by a custom IC which, amongst many other tasks, sits between the Z80 and the memory map. Amstrad attribute the extraordinarially low price of the machine to the customisation and the reduction in duplication between different sections of the circuitry. They also stress that the 8256 is a completely different machine from the others in their range, not having, for instance, any resident BASIC (it has to loaded from one of the discs provided with the system).

The other two items provide (besides the WP and BASIC) are CP/M+ operating system (also known as CP/M 3.0) and OSX, a graphics utility. Incidentally, the WP drives the machine directly, not via CP/M, but does produce CP/M compatible text files.

The claimed specs of the printer are 90CPS in draft mode, 20CPS in 'near letter quality' mode. The electronics in the printers are all Amstrad's, but the mechanism is made by Seikosha — let's hope they've done a better job than they did on the earlier Amstrad printer (see CT May 85, p55).

The disc supplied is of the same 3" format used on the 664 and the free-standing Amstrad drives. You can have an extra disc drive fitted for about £159, but in certain applications you already have an extra disc — as an area of RAM can be made to behave as a 112K RAM-disk.

So much for the main business, but a subsidiary piece of business that Amstrad were keen to just sneak in was the debacle over the CPC664. For the readers who have been a little out of touch, what happened was this: suddenly, with no prior announcement, the CPC6128 starts to appear in the shops, at a price of £399 inc VAT (for colour, £299 for green screen), with double the memory but an otherwise simmilar spec to the '664. Now the '664 was at this time, selling for rather more than this — result, confusion.

The upshot of this all is that Amstrad are ceasing to make the '664, which is now "well and truly" dead, according to Mr Sugar, and they're dropping the price of the '464 to £299 for colour (£199 for green screen). According to Alan Sugar, the '464 is "reported to be used by a well-known vacuum cleaner manufacturer to stock control his cars," but its future will be reviewed after Christmas. This is because it's seen as a games machine and the whole drift of Amstrad's marketing strategy is to develop the long-term market for serious uses for computers. "The low-end type Sinclair product has seen its day," Alan Sugar said earlier before the actual launch.

There is some clever marketing going on here, involving software compatibility. As readers may know, if you followed the rules in Amstrad's firmware manuals, then programs from the 464 will run on the 664 and the 6128. However, not so with the 8256. If you're very lucky, straight BASIC programs might run on the 8256, because the BASIC is very similar but not identical. However, the idea is to open up a clear differentiation between the 6128 and 8256, so that there will still be a market for the mono version of the '6128 (you'll have to buy the '6128 anyway if you want colour. . .).

There is some confusion as to exactly how much of the 128K of RAM is available to the user on the '6128. 41K is available in BASIC, in all screen modes, and 61K is available in the 'transient program area' under CP/M. But whether there are ways that the user can get at the full 128K (just imagine 128K of machine code to type in!) we won't know until we get our hands on the beastie. Other magazines have already published reviews, but as we've known some to have been published which just relied on the information in the manual, we'd like to see for ourselves.

The '8256 will be available on the high-streets solely through Dixon's, from around the third week in September. This exclusive arrangement with Dixon's will last until at least the end of December, although specialist outlet (as opposed to high-street multiples) will be able to get the '8256 through their normal wholesalers.

The products which are just yet twinkles in Mr Sugar's eye he's keeping to himself. However, we did discover that Locomotive Software had been working on the firm and software for the '8256 for a year, so he would have had his plans for this machine at an advanced stage when the CPC664 was launched back in May. And he must have known all about the '6128 — at the same time — so why the debacle over the '664? Could it be that it sold rather less well than expected? Or could it be that the fall in IC prices suddenly made the '6128 so much cheaper that he decided to get in ahead of the Commodores and Attaris? We'll probably never know.



HASHING

Paul Andreas Overra

No it isn't a misprint: hashing is a way of making easily searched data files, and it's also called key to address transformation.

For many applications it is necessary for a program to search for a data file for an item of specified value. Sometimes it is appropriate to keep a sorted 'index' of specified key fields so that techniques such as binary searching can be used to speed the search. With larger files one often avoids physical re-ordering of indexes by using binary tree structures. There are some situations where an alternative technique, called *hashing* or *key to address transformation*, is called for. When the technique is used for suitable cases, it is possible to achieve retrieval speeds that are second to none.

First let's illustrate the general characteristics of the problem of converting a key into a record number or location: suppose you were writing an address book program and wanted to include the ability to retrieve address details by supplying a list of details for all the people on file whose surname is Jones. You may, if you work for a large company, have many thousands of names in your computerised address book and you will certainly not want to wait for five minutes whilst the computer sifts through your file picking out the Jones from the Smiths, etc. You want 'instantaneous' retrieval, no matter how large the file is.

Hashing gets round this problem as follows: it takes some characteristic of the record and uses a pre-set rule (or *search key*) to point to the first location to be searched. Following the example of a computerised address book, let's suppose that we wish to make a file containing the names of all the people on the 'flannel panel' on this magazine. Suppose we use the number of letters in the surname to point to the first location to be searched. We immediately find that we are searching the same records for Thomasson and Robertson, for Fowler, Collis and Welham, and for Shelton and Falkner (this is known as a collision).

We could change the rule we use to point to the first location, but inevitably we cannot avoid the possiblity of having different names pointing to the same first record. So in hashing, we use a different rule to get to the second record to be searched from the first record searched; suppose we chose to use the number of the letters in the first names of the people. This will eliminate all but one of the possible collisions (the one between Jerry Fowler and Peter Welham), but this is still quite a saving because it means that to find the address of anyone on the 'flannel panel', you would have to look at a maximum of three locations.

So, hashing works as follows when searching for a record: use the pre-arranged rule to locate the first record to be searched; if this is not the desired record, use a different rule to shift the search from the first location and look again. If this record isn't the correct one, use the second rule again to move from the present search location to the next, and so on, until the correct record is found or until we are sure that it doesn't exist.

A couple of general points here: it must be possible to search every record in the file (this is because, assuming finite storage space but infinite possible combinations of stored data, we would want to be able to store at all possible locations). And secondly, we cannot use the simple rule of looking at the next record, because this would be the same for all records, resulting in very little saving of effort.

Addition of data has to use identical rules for the whole system to work, but here the computer should look for an empty record into which new data can be placed; ie, if the first record corresponding to the first calculated record number already contains data, then further attempts would be made until such time as an empty record was found and the data written here. Figure 1 shows how we might have written the address book discussed earlier; note that the collision between Jerry Fowler and Peter Welham doesn't occur as it might have done because of the order the records were written (if Lynn Collis had been written first, Jerry Fowler would have ended up in location 11 and Peter Welham in location 16).

Hash function mathematics can get quite complicated so we shall restrict ourselves to a specific class that is based on the concept of a *circular file*, Imagine that a file of n records are wrapped around a circle and

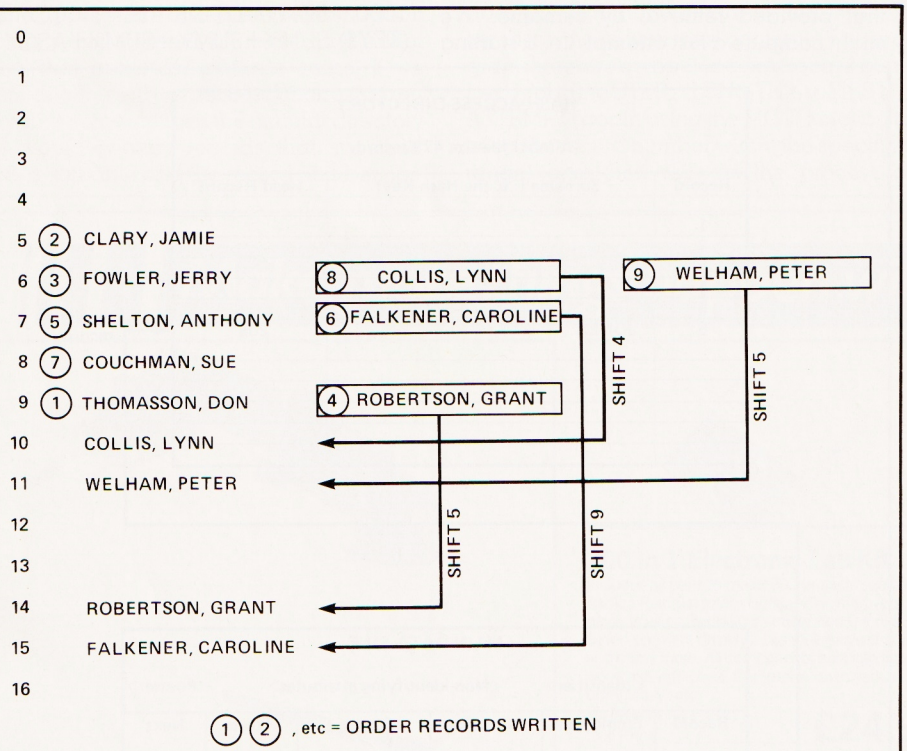
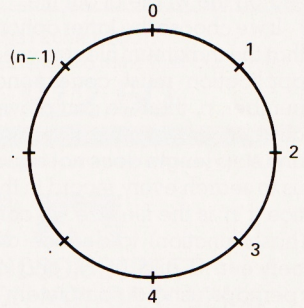


Fig. 1 (above) Writing the address book.

Fig. 2 (below) Representing a circular file.

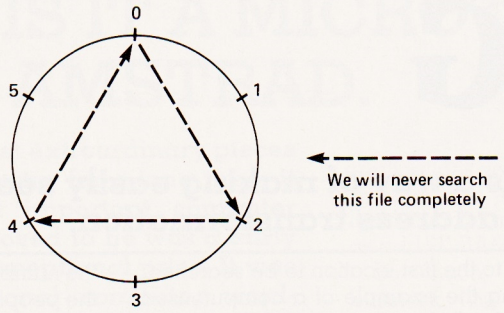
Record No.	Key	Other Fields
0		
1		
2		
.		
.		
(n-1)		

Table representation of a data file



Circular Form

Fig. 3 Stepping around a circular file.



that they have allotted 'record numbers' ranging from 0 to (n-1). Figure 2 shows the general ideas.

Our theoretical discussion concerns hash functions that operate on files which have a pre-defined constant maximum size. How can we be sure that in the worst possible case, our function would search every record in file? Try this experiment — draw a circular file with 6 records in it (records numbered from 0 to 5). Pick any record as a starting point and choose a fixed *step length* between 1 and 5. As you step around the file diagram, using your selected step length, mark each record that you visit. Try to find out which step lengths will allow every record in the circular file to be visited. Try the same experiment with a file with 5 records and step lengths from 1 to 4.

In the first case only step lengths of 1 or 5 will visit every record, but in the second case step lengths of 1, 2, 3, or 4 will work. Why the difference? It depends on whether the file size and the step size have any common factors. In the first case a file size of 6 has common factors with 2, 3 and 4,so only step lengths of 1 and 5 will search the file completely. In the second case neither 2, 3 not 4 have common factors with 5 so all of these step lengths (plus the step length of 1 which always works) will search the file completely. Figure 3 shows the result of stepping around a file with six records in it, using a step length of 2. It is quite obvious that, irrespective of where we decide to start, we will never look at every record.

If we create a data-file based on a circular file/fixed step length concept, we have a problem: how do we make sure that, given a constant step length, we can search the whole of a circular file irrespective of where we start? Either we make sure that our step length is not a factor of the maximum file size or we make sure that the maximum file size is a prime number. In such a case, because there are no common factors, we can guarantee that we can (in the worst possible case) search the whole of our file.

If we choose the latter option, i.e. stipulate that the maximum file size used in a hash file application must correspond to a prime number, n, then we can prove mathematically that: irrespective of the starting position, any step length does not equal 'n' will allow us to search every record in the file. In practice if 'n' is the file size we can use one rule (hash function) to deduce a starting point between 0 and (n-1). and a second rule (perhaps usng a completely different hash function) to deduce a step length between 1 and (n-1). If we are careful in our choice of functions we can use the differing step lengths to help reduce the probability of collisions.

For any given application the actual form that the hash function will take is going to vary. A good first-attempt function will try to spread the likely range of keys evenly around the file. A good step length function will make use of some facet of the key that is not used when calculating the starting location for the search.

The simple example of using the surname length is rather unsatisfactory, for a number of reasons, so let us look at two others we might choose to adopt for an address book that provided retrieval by surname. We might compute a first attempt, i.e. a starting point for the search, by converting each character of the surname to a number. One obvious possibility is to convert each character in a surname into its ASCII code then add them together. The sum must always be kept within the specified file size and the easiest way of doing this is to use the MOD function. If the file size is PRIME% then a typical Basic code might take the form:

```
1000 SUM%=0:FOR I%=1 TO LEN
(KEY$):SUM%+SUM%=SUM%+ASC(MID$
(KEY$,I%,1)):NEXT I%
1010 FIRST_ATTEMPT%=SUM% MOD
PRIME%
```

To compute a step length we could go back to using the length of the surname. This would ensure relative independance of the two search characteristics. It is unlikely, in the case of surnames, that the length of a name would approach the file size but it is common to incorporate the MOD restriction on the step length as well. A typical BASIC function that might be selected could be based on the formula STEP_LENGTH%= (LEN(KEY$) MOD (PRIME%−)) + 1, which produces a number between 1 and PRIME%−1 as required. If we include this calculation in the first Basic example code we end up with the following combination:

```
1000 SUM%=0:FOR I%=1 TO LEN
(KEY$):SUM%+SUM%=SUM%+ASC
```

HASH-ACCESS-DIRECTORY

Alloted file size 479 records

Record	Surname (i.e. the Hash Key)	Head Record
1		
2		
3		
4		
5		
6	JONES	208

example record

MAIN-DATA-FILE

	<Identifier>	<Non-identifying attributes>			<Pointer>
Record	Employee#	Initials	Street	Town	Next
1					
2					
.					
.					
.					
208	61440	P.G.	21 Cross Rd.	Enfield	521
.					
.					
521	69541	D.P.	9 The Grove	Feltham	0

Fig. 4 Typical file arrangement for a hash/multilist application.

```
(MID$(KEY$,I%,1)):NEXT I%
1010 FIRST_ATTEMPT%=SUM% MOD
PRIME%:STEP_LENGTH%=(LEN(KEY$)
MOD (PRIME%-1))+1
```

That's it. Given a surname, in the variable KEY$, the above two lines of code compute the place to start our search of the file and compute the step length that the search will use.

Functions such as these are used firstly to decide whereabouts to store data as it is being entered, and secondly to determine whereabouts to look for it when you need to find it again! If you wish (as you most probably would in any 'real application') to use a diskette file equivalent you would need to make allowances for the fact that your record numbers will most likely start from 1 and not from zero.

TYPICAL APPLICATION

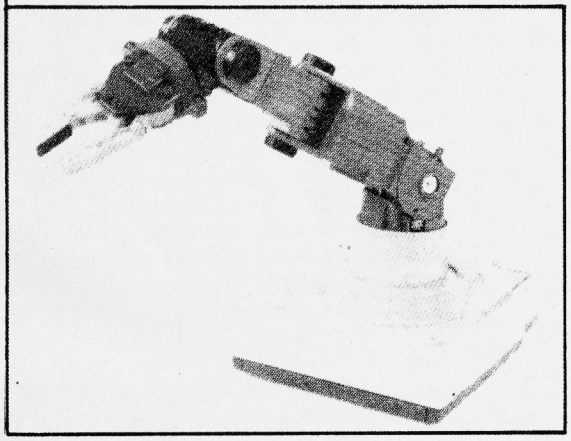
One area in which hashing is particularly well suited is the 'directory look up' procedure used in conjunction with 'multilist' chain structures. Hashing is used to access a circular file 'attribute-directory' containing the keys and their related 'pointers' giving the record number corresponding to the start of the appropriate linked list of 'chain'. Such pointers link together all items on file that have that particular attribute value. If we were dealing with an address book using retrieval by surname then the circular directory file would contain records that specified both a surname and the record number corresponding to the start of a chain of records of people with that same surname. Once the correct surname has been located no further searching is necessary — the program looks up the start of the chain and retrieves its information by following the chain pointers. A typical file structure is shown diagramtically in Figure 4: in practice one would no doubt see many other attributes stored, telephone number, postal codes etc. but the layout shown does give the general idea behind such schemes.

To access the directory the program takes the surname provided and turns it into two numbers, say FIRST_ATTEMPT% and STEP_LENGTH%. It then accesses the record whose record number is FIRST_ATTEMPT% and checks to see whether the surname stored in the directory record is the same. If it is, then the correct directory entry has been located. If the surname held in the directory entry does not match the surname that is being searched for, the search must continue. The program therefore computes a new record number by adding STEP_LENGTH% to FIRST_ATTEMPT% and checks to see whether the surname stored in the directory record is the same. If it is, then the correct directory entry has been located. If the surname held in the directory entry does not match the surname that is being serch for, the search must continue. The program therefore computes a new record number by adding STEP_LENGTH% to FIRST_ATTEMPT (again using the MOD function to keep the record number within the specified range) and then repeats the process of checking the surnames. This process continues until the correct direct entry is located or until a record is found that is 'empty' or, in the case of a directory that is full up, until the whole of the directory has been searched without success.

As explained earlier the same 'rules' are used to add surnames to the directory, to search the directory and to delete items from the directory. In practice it is rare for more than 2 or 3 attempts to be made before the right record is accessed. Often, if you take care selecting the hashing functions, you can achieve 'first time correct' access figures approaching 90% which means that you can locate most of your directory entries without any real searching at all.

Problem? Well, there are one or two points to watch out for. Collisions increase drastically as the directory becomes full so it is best to allow for some 20% more directory space than you really need. Deletion from the directory sometimes causes difficulties. Whatever scheme you use to delete an entry from the directory you must make sure that you can, or rather your program can, distinguish between a deleted record and an empty record. If you don't your program will confuse deleted records (which are saying "don't take any notice of this entry — but do make a further attempt") with empty records (which tell the program "do not bother searching any more"). Providing you are careful you will find 'key to address transformation techniques extremely useful — unless, that is, you really do make a complete and utter hash of it!

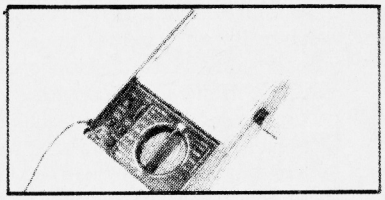
Make more things happen with Memoco.



Memoco Electron Robotic Arm

12 Axis of movement. Arm raise and lower. 270 degree rotation left or right. 90 degree Elbow movement left or right. 90 degree wrist movement either side of centre. 270 degree wrist rotation in either direction. Claw open and close Fitted with motor control circuit. Switched from 5 volt TTL. Controlled by computer. Separate motor driver power supply.

With position feedback ... £129.95
BBC B Interface card ... £49.00
Spectrum Interface card ... £79.00
Commodore 64 Interface card ... £49.00



200 in 1 Electronic Lab Kit

Includes all parts to make 200 projects such as Radio : Rain Detector : Burglar Alarm. Covers projects using Transistors : Integrated Circuits : Seven segment displays: Light Sensitive circuits and many more. All components built into fitted workcase with cover. Comprehensive manual. Completely safe.

Normal Price £34.00
Our Price ... £24.95

Range Doubler Multitester 43 Ranges

50,000 Ohms per volt DC. 10,000 Ohms per volt AC. 4.25" Colour coded mirrored display giving accurate reading without parellex error.

Normal Price £27.00
Our Price ... £15.45

Ni-Cad Battery Charger

A.A., C, D and PP3. Charges up to five batteries at a time.
Price ... £6.95

Battery Eliminators

Mains to DC regulated 300 mA 6v, 7.5v, 9v selectable. Suitable for most battery operated equipment.
Price ... £6.95

QuickShot™ II Joystick ... £9.50

1½–4½ v DC Motors Stall Current 300 mA
Dimensions: Length o/a: 45 mm; Dia: 24 mm; Shaft length: 5 mm; Shaft Dia: 2 mm
Price – £1.50 each or 10 for £12.50

ALL PRICES ARE INCLUSIVE OF VAT – FOR ORDERS UNDER £5 ADD £1.00 POSTAGE AND PACKAGING. FOR INFORMATION ONLY PLEASE SUPPLY S.A.E. TRADE ENQUIRIES WELCOME

MEMOCO ELECTRON

VISA

15 WINDSOR STREET
MELTON MOWBRAY, LEICS
TELEPHONE (0664) 63544
DEPT CT

GETTING GOOD GRAPHICS

James Tyler

If the prospect of writing a graphics design program has you reaching for the tranquilisers, fear not — there's help at hand!

In the past couple of years the capabilities of new micros appearing on the market have steadily become more and more impressive. Graphics especially, have become more advanced, leading to the possibility of producing very high quality images on most of the machines that are now commonly owned. Mind you, in order to display such images, there is of course the initial problem of the constructing the desired pieces of artwork!

Broadly speaking, there are two ways in which you can go about constructing an image that is to be displayed; put simply, either you get the computer to do the drawing or you do it yourself! Although the first method may sound like the beet, it might not be so. In practice, constructing images using the graphics commands provided on a machine can be very slow and labourious and in addition, the resulting program may be a very unwieldy list of DRAWs, MOVEs, PLOTs and colour changes.

The alternative way of constructing such pictures is to use some type of 'graphics design' utility which allows very sophisticated images to be formed with greater speed and accuracy. This then allows the user to concentrate just on the 'artistic' (?) side of the drawing, leaving the technicalities to the utillity. The resulting picture or image produced may then be saved in the form of the screen memory contents, or be hard-copied to a printer using an additional screen dump utiity.

In the circumstances where images are being designed for title pages and objects in video games, technical diagrams, company logos, advertisements or just 'doodling', the use of a graphics design utility is much more satisfactory.

A PRACTICAL EXAMPLE

The program given was written for the BBC Micro, and is a practical example of the type of Graphics Design Program described above and could be used for any of the applications mentioned. Despite the fact that it has been written for one particular machine, there is no reason why it, or something along the same lines, couldn't be written to run on any micro that supports high resolution colour graphics with all the typical plotting commands (including exclusive-OR plotting), PEEK and POKE (type) commands, and a way of testing which keys are pressed on the keyboard at any moment, such as INKEY$.

What follows is a description of the facilities that are offered by this program, together with explanations describing the techniques that I employed in the software to achieve the effects that each facility provides. Using this information, and by examining the listing of my program, you should be able to get a good idea of how to write a Graphics Design Program for your own micro — whether you merely use this article as a rough guideline and source of inspiration, or convert the listing given here directly.

FEATURES AVAILABLE

When pictures or diagrams are being drawn using this program, a number of different facilities may be selected at any time by the user. These allow dots, lines and curves to be drawn, areas of the screen to be filled with solid colour and one of five sizes of 'paint brush' to be applied. All of these drawing facilities may be used with any of the sixteen different colours or colour combinations that are available on the micro. Additionally, by adopting the technique of 'overlaying' patterns of different colours and textures, a variety of other effects may also be achieved — is this especially effective using the brush option (see later).

An additional facility provided by the program is one that enables specified small areas of the screen to be read or 'Memorized'. These may then be drawn or 'Recalled' to any other part of the screen either in their original form or upside-down, mirrored, or a combination of the latter two. This is especially useful and can greatly increase the speed and accuracy with which pictures may be constructed. Also, since these small areas of the screen or 'Images' may be saved to disk or tape, whole libraries of predefined objects, or 'Icons', can be built up by the user. These can later be selected as required when designing particular displays. An example might be the programmer who uses the Graphics Design Program to draw out his or her flowcharts when preparing documentation. In this case, the flowchart could be produced very quickly by merely selecting from an existing library the various images of decision, processing, or imput/ouput boxes etc which are the 'building blocks' of such flowcharts. Alternatively, the graphics characters or 'sprites' to be later incorporated in a video game could be saved in this way during and after development.

Other general but necessary functions are also provided by the Graphics Design Program to allow the user to change the current plotting colour, redefine or default the graphics colour palette, clear the screen and save/load whole screen designs. A summary of all the facilities available is given in the Summary Table.

PROGRAM STRUCTURE

The heart of the program is a single loop from which the procedures (Subroutines for the non-BBC minded!) for all the available options are called. This is located between lines 300 and 560 in the main listing (Listing 3), contained within PROCdis (display) — the 'core' routine of the program. The effect of this main loop is to repeatedly check and see if any of the relevant keys used in the operation of the program are being pressed at that moment by the user. If one is being pressed then the associated procedure is called (eg. the brush routine if 'B' is pressed) and then

any other remaining keys are similarly checked, upon the return to the loop. A few of the more basic facilities such as clearing the screen and updating the cursor position are actually present in the main loop since putting them in procedures would be wasteful, owing to their shortness.

By writing the Graphics Design Program with this structure I was able to make it fully interactive, and by keeping the associated procedures fast, a rapid response to key depressions was maintained. This may give the impression of more than one action being performed if several keys are pressed at once. (A similar effect is often achieved when playing various arcade-type games.) For example, if while the cursor is moved a brush is also applied, broad lines or continuous 'brush marks' will be drawn. Several of the available facilities may be simultaneously selected in this manner, often resulting in some interesting graphic effects.

As well as checking the keyboard on each repetition of the main loop, the cursor is also drawn at its current screen position — twice, using Exclusive-OR plotting. This way, the cursor is drawn and then immediately erased, still leaving any original graphics plotted in the same part of the screen intact. The cursor is therefore continuously flickering and may appear either as a crosshair or a single pixel (toggled by pressing the 'TAB' key). The cursor is moved about the screen by using the computer's four arrow keys and the speed of movement may be increased, when required, by additionally pressing the 'SHIFT' key. The current position of the cursor indicates where any points, lines, or brush marks etc will appear on the screen as well as where any FILLing will start from. The construction of a design therefore centres around the manupulation of the cursor by the user.

THE EVOLUTION OF A LISTING

Because the program has been written with the structure outlined above, futher options may easily be added (free memory permitting!) Usually this will involve adding a new procedure to the end of the program listing and inserting its associated conditional call into the main loop. Indeed, the bulk of the program was written this way. I started off with what was not much more than a multicoloured Etch-a-Sketch where dots could be drawn at the cursor position by pressing the Space Bar. I then added the various facilities as they were developed, testing them one after another. Occasionally, routines would be modified and improved when unforseen problems came to light. All in all, the Graphics Designer was certainly a program that *evolved*!

CURVES

My first enhancement to the original program was the addition of a curve drawing procedure. This is essential in order to produce displays with a softer, more rounded 'feel' about them. Two methods could be employed to produce curves; either the use of a plotting routine containing trigonometric functions, or the simpler way of allowing the user to construct curves merely by drawing a series of joined straight lines. I decided to opt for the second method for three reasons; the resulting routine would be shorter, faster, and of course easier to write! And so the Join facility was developed. To draw a line in the current colour, the user simply moves the cursor to the points on the screen where the ends of the line are going to be, pressing 'J' at each point. The points are then immediately joined to form a line. By carefully joining a series of such lines, each of a particular length and angle, very satisfactory curves and possibly even circles may be constructed with a bit of judgement. This is fairly well demonstrated by my efforts in Figure 3.

The algorithm for line drawing is as follows: Each time key 'J' is pressed the join procedure is called. This causes the value of a flag to be toggled to 0 or 1, indicating whether the first or second end of the line to be drawn is being specified by the user. If it is the first end, then the cursor's position is merely noted, otherwise it must be the second, in which case a line drawn in the current plotting colour from the cursor's present position to the one noted the last time 'J' was pressed. Simple but effective.

Also of course, if a line turns out not to be quite right, you can erase it just by drawing over it again in the background colour.

Summary of Keys Used

Key	Action
Grey arrow keys —	Move cursor
<TAB> —	Cursor ON/OFF
Space bar —	Draw point
Function keys (0-9) —	Colours 0 to9
Shifted fn keys (0-5)—	Colours 10 to 15
Key C —	Change palette
<CTRL> 'D' —	Default palette
Shifted fn key 9 —	Clear screen
Key F —	Fill with colour
Key B —	Apply brush
Keys 1 to 5 —	Alter brush size
Key J —	Join two points
Key M —	Memorize image
Key R —	Recall image
Key S —	Save screen
Key L —	Load design
Key P —	Dump to Printer
Key T —	Text entry
<CTRL> 'E' —	End program

Press <SPACE BAR> to load main program

FILLING WITH COLOUR

The next thing to do was add a fill facility that would enable large areas of the screen to be coloured in quickly. Rather than work out the code for an amazingly intelligent routine that would completely fill the most highly convoluted shapes. I chose instead to write a simple one that would fill a space from top to bottom until the first boundaries were reached. The position of the cursor is used as the starting point for filling. This works very well, and if while filling an area, a few nooks and crannies in the pattern get missed, all

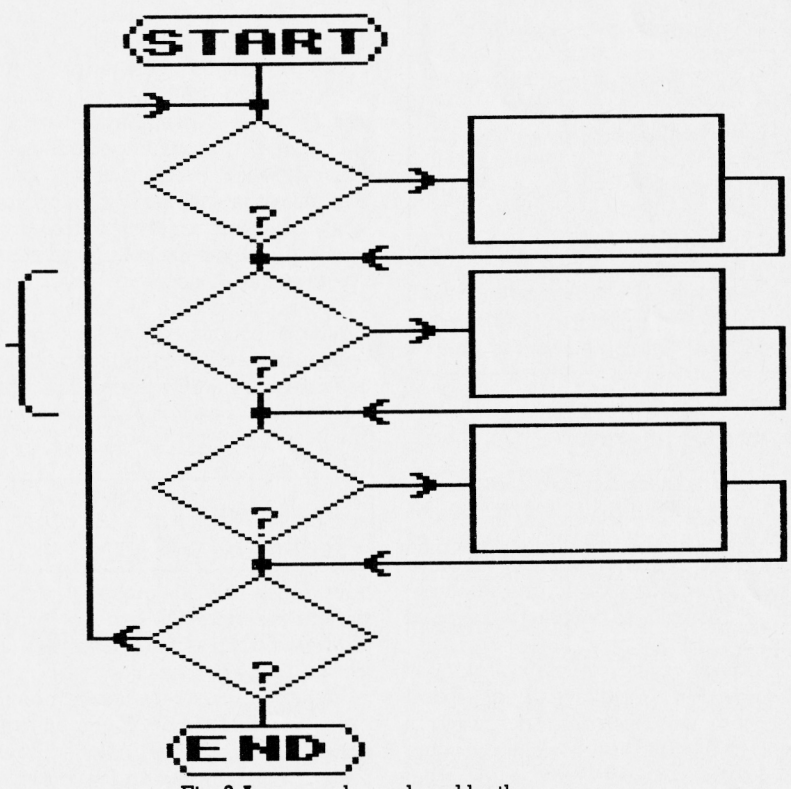


Fig. 2 An example produced by the program.

you have to do is move the cursor over to them and do another fill.

The fill procedure is called whenever the 'F' key is pressed. It then works in this manner: a check is first made that boundaries do indeed exist both directly above and below the cursor position. If they don't then fill is aborted. This safety check helps reduce the possibility of your current masterpiece being ruined by colour leaking our of the area to be filled, onto the surrounding screen — a bit like spilling a bottle of ink over a letter that you've half written! Assuming these boundaries *are* present the procedure performs it's task by filling the area, from the pixel row below the top boundary down to the row just above the bottom boundary, in the current plotting colour. Each pixel row, as it is encountered, is filled in a sideways direction up to the two side boundaries (or screen edges). On the BBC Micro, this single row fill is already catered for by the PLOT 77,X,Y command. On other micros however, it could easily simulated by advancing horizontally one pixel at a time in both directions, colouring each pixel until the left and right boundaries are met.

The fill facility is one which particularly lends itself to improvement. For example, by modifying the routine to only filling alternate pixels vertically and horizontally, a variety of shades or hues could be acheived.

Main Global Variables

PX - Horizontal Coordinate of Cursor

PY — Vertical Coordinate of Cursor

C —Current Plotting Colour

XF% — Cursor Appearance Flag

MF% — 'Image Memorized' Flag

d% — 'Join' Flag (see text)

H — Height of Plotting Area Used

W — Width of Plotting Area Used

m% — Starting Addr. of where Memorized Image is Stored

Array B () — Coordinates fo Corners of Memorized Image

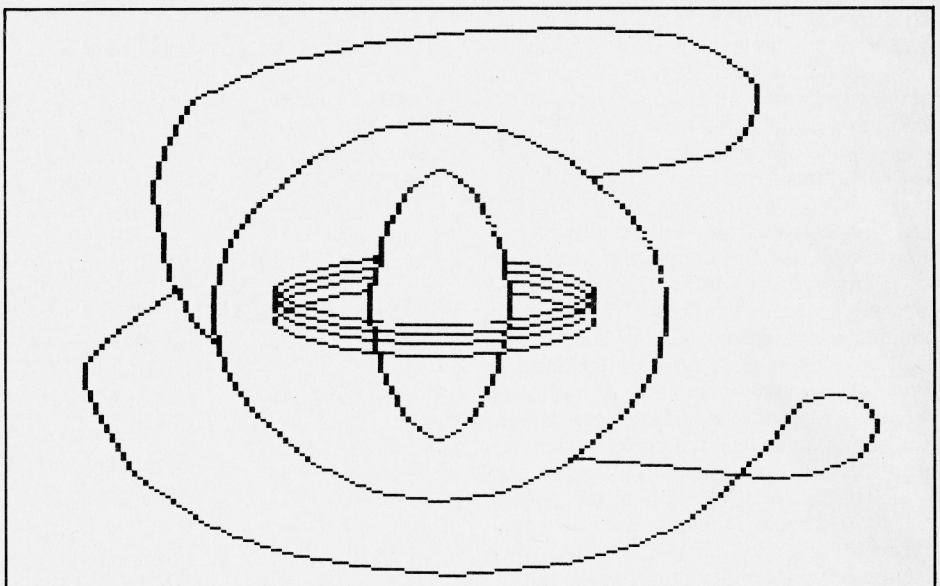


Fig. 3 Curves gnerated using the program.

PAINT BRUSHES

The difficulty with writing this facility was ensuring that it lived up to it's name. The problem was to be able to draw broad lines or smudges of determined width that also possessed a somewhat random character — a feature possible to obtain with paint brushes.

My first idea was to use some technique of plotting a group of randomly positioned dots in a specified area. This, however, turned out to be slow and ineffective. I then had a minor brainwave and opted for the following method of creating brushmarks; by painting user defined characters designed to look like 'smudges', as shown in Figure 4

There are five brush sizes available, brush 1 being the smallest. Each brush size is represented by four user defined characters. If you look closely at Figure 4 it can be seen that the characters for each brush size are in fact the same dot pattern but in a different 90 degree orientation. This is what gives the random element. When a brush of particular size is applied, the brush procedure chooses randomly one of the four predefined characters for that brush and prints it at the cursor position. A total of twenty different user defined characters are used (ASCII codes 224 to 243). If the smallest brush is selected (by pressing key '1') then one of the characters from CHR$224 to CHR$227 will be chosen for printing. Similarly, for brush Number 5, a character from CHR$240 to CHR$243 will be chosen. In the program, PROCbrsh does all this and is called each time key 'B' is pressed.

The brush size is changed within the main loop at line 480. Pressing any of the keys '1' to '5' causes the ASCII code of the first of the four characters for that brush size to be selected. (Read that last bit again if it sounds complicated!)

The brush routine is quite effective. By moving the cursor while also applying a brush it is possible to 'paint' lines of varying thickness depending upon the brush size used. Additionally, some interesting, multicoloured, mottled textures (mentioned earlier) may be obtained by painting over existing brush marks in another colour and perhaps a different brushsize too. This is possible because when a brush mark is printed, it is done with the machine's text and graphics cursors joined together. This allows characters to be overlayed on top of each other, any number of times.

An extra tip: Using a size 5 brush in black is a useful way of 'rubbing out' mistakes.

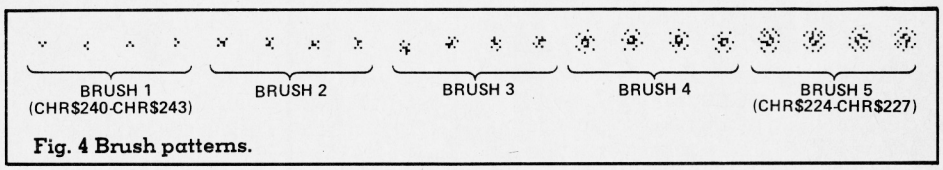


Fig. 4 Brush patterns.

COPYING SECTIONS OF THE SCREEN

There are undoubtedly, several ways in which this facility could have been implemented. The method described here works by firstly, copying the contents of the specified area of the screen. By using a temporary area of memory for storing images in this way, they may be recalled even after the screen has been cleared.

The two parts involved in the copying are performed by separate procedures, PROCmem (memorize) and PROCrecall — called when keys 'M' or 'R' respectively are pressed.

In order to memorize an image, it is necessary to firstly define the left, right, top and bottom boundaries of the rectangle of screen in which it lies. The position of these boundaries are defined by sliding four straight lines across the screen, (two horizontal lines and two vertical). Each line is moved to the correct possition using the two associated cursor keys and is then fixed by pressing <RETURN>. Once all four boundaries have been specified in this way, the program goes on to read the screen memory contents contained within them (in order of highest to lowest addresses). Control is then retumed to the main loop once the user has been given the chance to Save the stored image.

The stored image may be recalled and displayed at any time by pressing 'R'. When this is done, the new location of the image is specified ie. whether it is to appear in it's original form, or mirrored upside-down,

merged (overlayed) on the existing design, or even a combination of these. Finally, the image is POKEd onto the screen in the desired format, row by row.

A brief examination of PROCmem and PROCrecall in the listing may lead you to think that the code is complicated. This is not the case, however, and the facility is achieved in a relatively simple manner.

When the portion of the screen to be copied is initially specified, the position of the four boundary lines are used to calculate where the corners of the resulting rectangular image will be (in terms of screen coordinates). The example diagram below, showing sample boundaries helps to explain this. If the horizontal (X) co-ords of boundary lines A & B are 100 & 500 respectively, and the veritical (Y) co-ords of C & D are 200 & 800, then the XY positions of the rectangle's corners will be: top left - 100,800; top right - 500,800; bottom left - 100,200 and bottom right - 500,200.

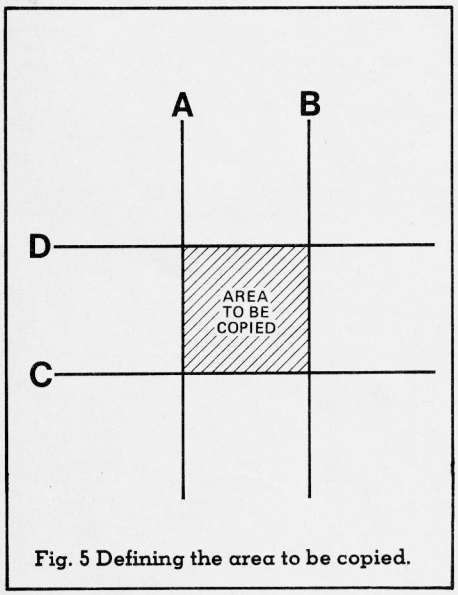


Fig. 5 Defining the area to be copied.

Following this, to copy the data stored in this area of the screen, a loop has to be set up in which the memory contents of each screen coordinate are read and stored, eg.

```
OFFSET = 0
FOR Y = 8000 TO 200 STEP -1
FOR X = 100 TO500
POKE (SPARE+OFFSET), (memory contents at co-ord X, Y)
OFFSET = OFFSET + 1
NEXT : NEXT
```

(Where SPARE is the starting address of free memory where the image is to be stored.)

As a point of interest the first 8 bytes of the image data hold the screen coordinates of the image's corners original location. Therefore, the image is always the size of the image plus 8. With the spare memory left in this program, images up to about 2.5K in size can be stored — about an eighth of the entire screen area.

As may be seen from the example loop above, some way of determining the actual memory address of a given screen coordinate is required. In the listing, this is done by FNadd at lines 1300 to 1350 (used, both when memorizing *and* recalling an image). The 'formula' used in this case is only specific to the screen memory addresses, their physical arrangement on the display and the range of the X and Y plotting coordinates used by the machine's BASIC. For example, the BBC uses 20K of RAM for the high resolution screen, which starts at address 12288 (or 3000 in Hex). The X coordinates are in the range 0 to 1279 (16 steps for 1 byte, horizontally) and the Y coordinates in the range 0 to 1023 (4 steps per vertical byte). The origin — point (0,0) — is at the bottom left corner of the display.

RECALLING IMAGES

Images are recalled using a loop with the same structure as the one described above. This time, though, the image data stored in the free memory area (from SPARE onwards) is POKEd onto the screen. In other words, in the example piece of 'pseudo code' given previously, the 4th line would be more along the lines of:

POKE (screen address at co-ord X, Y), (value of byte at address: SPARE+ OFFSET)

Reproduction of images in upside-down or mirrored form is simply a matter of altering the X or Y loops so that they increment or decrement respectively, ie. (again, using the previous example) for upside-down, change the 2nd line to:

FOR Y = 200 TO 800

The image is then POKED to the screen from top to bottom, instead.

For mirror – images, change the 3rd line to:

FOR X = 500 TO 100 STEP-1

The image is then POKEd back in a right-to-left direction, instead.

Additionally, each byte of the BBC's screen memory represents two adjacently horizontal pixels. Therefore, when mirroring images, the pixels in each byte must swap positions with each other before being POKEd to the screen. This is done by line 1950 in the listing, if mirroring is selected. Again, the formula used here is only specific to the two Acorn machines. To write a mirroring routine for another micro will firstly require a knowledge of how a pixel's colour and position is represented in a screen byte.

Merging, or overlaying, involves allowing any graphics existing on the screen already, to show through the gaps (black areas) in the new image. Obtaining this effect just means NOT POKEing any bytes of the image which are zero (black pixel pairs) back onto the screen when recalling.

In the listing, the heart of the recall routine which pokes the image back to the screen — inverted, mirrored, merged or whatever — is located at lines 1920 to 1970.

TEXT ENTRY

The text entry procedure in the program at lines 2670 to 2730, works in a similar way to the brush procedure, in that it works by firstly joining the text and graphics cursors together. Characters may therefore be placed at any position on the screen. To do this, for labelling or adding a title to a design, just position the cursor and press 'T' (to call the procedure). Then hit the relevant key.

LOADING AND SAVING

Most machines have the commands necessary to Save or Load the memory contents between specified addresses to/from tape or disk. Such BBC commands (*LOAD and *SAVE) were used here to enable either memorized images or complete screen displays to be saved for later alteration or use. This simply involves *SAVEing the free memory space used for image storage or the whole 20K of screen data, respectively. The data can then be *LOADed to where it originally resided in memory.

HARD-COPY

For many uses of a Graphics Design Program it is obviously often very desirable to make a hard-copy of a design. Although the program presented here has provision for a printer dump facility (press key 'P') the associated procedure, PROOCdmp (lines 2570 to 2650) does not contain the actual code for the dump itself. This is for two reasons; the description of such a routine would merit an entire article in itself and secondly; if you have a printer, you undoubtably already possess you're own favourite routine — whether in ROM or as a separate utility — to which a call could be placed in the relevant part of the listing (ie. line 2620). My own dump resides as machine code in an unused area of memory below the BASIC listing. It represents the different screen colours by printing various shades of grey.

CONCLUSION

There you have it; a complete rundown of how I went about creating a Graphics Design Utility which offers it's services in many varied fields. I hope the information here is sufficient to give you the inspiration and ideas for writing you're own version of the program. If you own a BBC of course, the only challenge is to type in the listings given here!

If you have a tape-based system, be sure to SAVE the listings on tape in this order: Introduction program, Brush definitions and Main program. Also, change any prompts in the program accordingly. If you use a disk system, make sure you save the brush definitions (as directed at the bottom of Listing 2) with the filename 'CHR$' and Listing 3 with the filename 'ART2'.

IMPROVEMENTS

No computer program is ever finished as this one may illustrate. Many additions could be made to enhance it. For example if you have *lots* of spare memory loafing about in your machine (64K RAM owners), you could do all sorts of things like keep two screen pictures in memory at once and merge bits from one to the other etc. — just one idea. Interfacing other peripherals could also prove interesting; a Track Ball, Mouse, Digitiser, Joystick, Light Pen . . . . . .

See what I mean?

Listing 1. Introduction.

```
 10REM"******************************
 15REM"*                            *
 20REM"* BBC Micro Graphics Design *
 30REM"*                            *
 40REM"*     James Tyler 1984       *
 50REM"*                            *
 60REM"*   Introduction Program     *
 70REM"*                            *
 80REM"******************************
 90
 95REM * Load predefined brush graphics *
100*L. CHR$ C00
110*TV255,1
120*FX220,0
130*FX4,1
140MODE7:VDU23;8202;0;0;0;
150PRINTCHR$141;CHR$131;CHR$157;CHR$129;SPC8;"But Is It Art ??"
160PRINTCHR$141;CHR$131;CHR$157;CHR$129;SPC8;"But Is It Art ??"
170PRINTSPC30"J.T 1984"
180PRINTSPC5"A Summary Of The Keys Used"
190PRINT"    Key"SPC20"Action  "
200PRINT"Grey arrow keys"TAB(20)"_  Move cursor"
210PRINT"     <TAB>"TAB(20)"_  Cursor ON/OFF"
220PRINT"Space bar"TAB(20)"_  Draw point"
230PRINT"Function keys(0-9)"TAB(20)"_  Colours 0 to 9"
240PRINT"Shifted fn keys(0-5)"TAB(20)"_  Colours 10 to 15"
250PRINTTAB(4);"Key C"TAB(20)"_  Change palette"
260PRINT"<CTRL> 'D'"TAB(20)"_  Default palette"
270PRINT"Shifted fn key 9"TAB(20)"_  Clear screen"
280PRINTTAB(4);"Key F"TAB(20)"_  Fill with colour"
290PRINTTAB(4);"Key B"TAB(20)"_  Apply brush"
300PRINT"Keys 1 to 5"TAB(20)"_  Alter brush size"
310PRINTTAB(4);"Key J"TAB(20)"_  Join two points"
320PRINTTAB(4);"Key M"TAB(20)"_  Memorize image"
330PRINTTAB(4);"Key R"TAB(20)"_  Recall image"
340PRINTTAB(4);"Key S"TAB(20)"_  Save screen"
350PRINTTAB(4);"Key L"TAB(20)"_  Load design"
360PRINTTAB(4);"Key P"TAB(20)"_  Dump to Printer"
370PRINTTAB(4);"Key T"TAB(20)"_  Text entry"
380PRINT"<CTRL> 'E'"TAB(20)"_  End program"
390PRINT"Press <SPACE BAR> to load main program"CHR$11
400REPEAT:UNTILGET=32
405REM *  Load and Run the main program  *
410*FX220,27
420PAGE=&1100:CHAIN"ART2"
```

Listing 2. Brush definitions.

```
L.
  10REM ************************************************
  15REM *                                              *
  20REM * Character definitions for all the brush sizes *
  30REM *                                              *
  40REM ************************************************
 100VDU23,224,0;&1400;8;0;
 110VDU23,225,0;&800;&810;0;
 120VDU23,226,0;&1000;&28;0;
 130VDU23,227,0;&810;&10;0;
 140
 150VDU23,228,0;&1824;&28;0;
 160VDU23,229,0;&814;&418;0;
 170VDU23,230,0;&1400;&2418;0;
 180VDU23,231,0;&1820;&2810;0;
 190
 200VDU23,232,0;&1000;&1824;&82C;0;
 210VDU23,233,0;&A14;&1438;0;
 220VDU23,234,0;&3410;&2418;8;
 230VDU23,235,0;&1C28;&2850;0;
 240
 250VDU23,236,&400B;&3A94;&9828;&1042;
 260VDU23,237,&4224;&AC18;&439;&48;
 270VDU23,238,&4208;&1419;&295C;&1002;
 280VDU23,239,&1200;&9C20;&1835;&2442;
 290
 300VDU23,240,&5224;&8C29;&1875;&2442;
 310VDU23,241,&5208;&3295;&992C;&1442;
 320VDU23,242,&4224;&AE18;&9431;&244A;
 330VDU23,243,&4228;&3499;&A94C;&104A;
 340
 350
 400REM  *  After checking this listing,
 410REM  *  RUN it, then save the character data
 420REM  *  with :       *SAVE"CHR$"0C00 0CFF
```

Listing 3. Main Program.

```
 10REM * ART2 - James Tyler 1984 *
 20
 30REM * Page Must Be =&1100 *
 40
 50DIM B(3),CL 30
 60*TV0,1
 70ENVELOPE1,1,0,0,0,0,0,0,127,-10,-50,-5,126,0
 80m%=&2680:d%=1:B%=223:MF%=0:XF%=1:OSCLI=&FFF7
 90PX=600:PY=500:H=830:W=1279
100MODE2
110HIMEM=m%-1
120ONERRORGOTO2790
130
140
150REPEAT
160  Li%=160:PROCdis
170  IFK=5 VDU26:END
180  Li%=180:S$="3000":L$="4FFF":PROCsave
190  UNTILFALSE
200
210
220DEFPROCdis
230VDU23;8202;0;0;0;
240VDU28,0,4,19,2
250PROCload
260*FX225,100
270*FX226,110
280*FX4,1
290C=7
300REPEAT
310  PROCpix(2,3,7)
320  IFXF% PROCcross
330  IFINKEY(-87)PROCload
340  IFINKEY(-70)PROCjoin
350  IFINKEY(-97)XF%=XF% EOR1:PROCwt(15)
360  IFINKEY(-56)PROCdump
370  IFINKEY(-83)PROCpalt
380  IFINKEY(-99)PROCpix(1,0,C)
390  IFINKEY(-36)PROCtext
400  IFINKEY(-1) HS%=32:VS%=16 ELSEHS%=8:VS%=4
410  IFINKEY(-58)PY=PY+VS%:IFPY>H PY=H
420  IFINKEY(-42)PY=PY-VS%:IFPY<0PY=0
430  IFINKEY(-122)PX=PX+HS%:IFPX>W PX=W
440  IFINKEY(-26)PX=PX-HS%:IFPX<0PX=0
450  IFINKEY(-68)PROCfill
460  IFINKEY(-101)PROCbrsh
470  K=INKEY(0)
480  IFK>48ANDK<54B%=223+4*(K-49):SOUND1,001,180,5
490  IFK=119CLG
500  IFK=82 ANDMF%=1 PROCrecall
510  IFK=77PROCmem
520  IFK=4VDU20,19,15,0;0;
530  IFK>99ANDK<116C=K-100:SOUND1,1,180,5
540  UNTILINKEY(-82) ORK=5
550*FX4,0
560ENDPROC
570
580DEFPROCmem
590LOCALC:PROCwt(50)
600REPEAT:RESTORE:GCOL3,7:C=0
610  REPEAT:X=0:READM$:CLS
620    PRINT'M$;" edge.."
630    REPEAT
640      MOVEX,0:DRAWX,H:MOVEX,0:DRAWX,H
650      IFINKEY(-1) HS%=32 ELSEHS%=8
```

```
660       IFINKEY(-26)X=X-HS%:IFX<0X=0
670       IFINKEY(-122)X=X+HS%:IFX>W X=W
680       *FX15,0
690       UNTILINKEY(-74)
700     SOUND1,1,180,5
710     MOVEX,0:DRAWX,H
720     B(C)=X:C=C+1
730     PROCwt(50)
740     UNTILC=2
750   REPEAT:Y=0:READM$:CLS
760     PRINT'M$;" edge.."
770     REPEAT
780       MOVE0,Y:DRAWW,Y:MOVE0,Y:DRAWW,Y
790       IFINKEY(-1) VS%=16 ELSEVS%=4
800       IFINKEY(-58)Y=Y+VS%:IFY>H Y=H
810       IFINKEY(-42)Y=Y-VS%:IFY<0Y=0
820       *FX15,0
830       UNTILINKEY(-74)
840     SOUND1,1,180,5
850     MOVE0,Y:DRAWW,Y:B(C)=Y:C=C+1
860     PROCwt(50)
870     UNTILC=4
880   *FX15,0
890   CLS
900   PRINT'"OK (Y/N)?"
910   G$=GET$
920   CLS
930   MOVEB(0),0:DRAWB(0),H
940   MOVEB(1),0:DRAWB(1),H
950   MOVE0,B(2):DRAWW,B(2)
960   MOVE0,B(3):DRAWW,B(3)
970   UNTILG$="Y"
980PROCcopy
990DATA RIGHT,LEFT,TOP,BOTTOM
1000ENDPROC
1010
1020DEFPROCpix(NT,G,COL)
1030GCOL6,COL:FORL=1TONT:PLOT69,PX,PY:NEXT
1040IFd%=0 GCOL3,C:DRAWDX,DY:DRAWPX,PY
1050ENDPROC
1060
1070DEFPROCcross
1080GCOL3,7
1090MOVEPX-50,PY:DRAWPX+50,PY:MOVEPX,PY-50:DRAWPX,PY+50
1100MOVEPX-50,PY:DRAWPX+50,PY:MOVEPX,PY-50:DRAWPX,PY+50
1110MOVEPX,PY
1120ENDPROC
1130
1140DEFPROCcopy
1150PRINT'"Reading screen...";
1160S%=&70:B%=0:N%=0:m%=&2680
1170A%=B(2):B%=B(3):C%=B(1):D%=B(0)
1180FORL%=0TO3
1190  !(m%+2*L%)=B(L%)
1200  NEXT
1210m%=m%+8
1220FORY%=A%TOB%STEP-4:FORX%=C%TOD%STEP16:LA%=S%:LB%=B%:S%=FNadd
(X%,Y%):B%=?S%: m%?N%=B%:?S%=&3F:?LA%=LB%:N%=N%+1
1225    NEXT:NEXT
1230?S%=B%
1240N%=N%+8:m%=&2680:MF%=1
1250CLS
1260PRINT"Save image (Y/N) ?":G$=GET$:CLS
1270IFG$="Y"L$=STR$~(N%+8):S$=STR$~(m%):PROCsave
1280CLS:ENDPROC
1290
1300DEFFNadd(x%,y%)
1310y%=1023-y%
1320x%=x%DIV8
1330y%=y%DIV4
1340M%=&3000+(y%DIV8)*640+y%MOD8+(x%DIV2)*8
1350=M%
1360
1370DEFPROCsave
1380PROCwt(20)
1390VDU23,0,10,103,0;0;0;
1400INPUT" Enter filename..."'F$
1410CLS:PRINT'"Insert disk,"'"then press <RETURN>":REPEAT:UNTILGET=13
1420$CL="SAVE "+F$+" "+S$+" +"+L$
1430X%=CL MOD256:Y%=CL DIV256
1440CALLOSCLI
1450PROCwt(200)
1460VDU23;8202;0;0;0;:CLS:PRINT" Saved."
1470G=INKEY(300)
1480ENDPROC
1490
1500DEFPROCload
1510PROCwt(10)
1520PRINT"Load picture (Y/N) ?":G$=GET$
1530CLS
1540IFG$<>"Y"ENDPROC
1550VDU23,0,10,103,0;0;0;
1560INPUT"Enter filename.. "'F$
1570VDU23;8202;0;0;0;
1580$CL="LOAD "+F$
1590X%=CLMOD256
1600Y%=CLDIV256
1610CLS
1620PRINT'"Insert disk,"'"then press <RETURN>":REPEAT:UNTILGET=13
1630?&3000=255
1640CALLOSCLI
1650IF?&3000=255:?&3000=0:MF%=1:PROCrecall
1660CLS
1670ENDPROC
1680
1690DEFPROCrecall
1700m%=&2680
1710B(0)=?m%+256*?(m%+1)
1720B(1)=?(m%+2)+256*?(m%+3)
1730B(2)=?(m%+4)+256*?(m%+5)
1740B(3)=?(m%+6)+256*?(m%+7)
1750PRINTCHR$12+"Enter position (Y/N)?"
1760G$=GET$
1770CLS
1780IFG$="Y" PROCpos
1790A%=B(2)
1800B%=B(3)
1810C%=B(1)
1820D%=B(0)
1830SY%=-4:SX%=16:PROCwt(10)
1840PRINT"Invert (Y/N)? ";:G$=GET$:PRINTG$
1850IFG$="Y" N%=A%:A%=B%:B%=N%:SY%=4
1860PROCwt(10)
1870PRINT"Mirror (Y/N)? ";:G$=GET$:PRINTG$
1880IFG$="Y" N%=C%:C%=D%:D%=N%:SX%=-16
1890PROCwt(10):CLS
1900PRINT"Merge (Y/N)? ";:G$=GET$:CLS
1910N%=8
1920FORY%=A%TOB%STEP SY%:FORX%=C%TOD%STEP SX%
1930    SA%=FNadd(X%,Y%)
1940    BY%=m%?N%
1950    IF SGN(SX%)=-1 BY%=(BY% AND&55)*2+(BY% AND&AA)/2
1960    IFG$="N" ORBY%>0 ?SA%=BY%
1970    N%=N%+1:NEXT:NEXT
1980m%=&2680
1990ENDPROC
2000
2010DEFPROCpos
2020GCOL3,7
2030WI=B(0)-B(1)+1:HE=B(2)-B(3)+1
2040X=B(1):Y=B(2)
2050REPEAT:MOVEX,Y
2060  PLOT1,WI,0:PLOT1,0,-HE:PLOT1,-WI,0:DRAWX,Y
2070  PLOT1,WI,0:PLOT1,0,-HE:PLOT1,-WI,0:DRAWX,Y
2080  IFINKEY(-1) HS%=32:VS%=16 ELSEHS%=8:VS%=4
2090  IFINKEY(-26)ANDX>7X=X-HS%
2100  IFINKEY(-122)ANDX<1100X=X+HS%
2110  IFINKEY(-58)ANDY<1019Y=Y+VS%
2120  IFINKEY(-42)ANDY>3Y=Y-VS%
2130  UNTILINKEY(-74)
2140B(1)=X
2150B(2)=Y
2160B(0)=B(1)+WI-1
2170B(3)=B(2)-HE
2180ENDPROC
2190
2200DEFPROCfill
2210X%=PX:Y%=PY
2220REPEAT:Y%=Y%-4:UNTILPOINT(X%,Y%) ORY%<1:IFY%<1SOUND1
,1,10,5:ENDPROC
2230Y%=PY:REPEAT:Y%=Y%+4:UNTILPOINT(X%,Y%) ORY%>H:IFY%>H
SOUND1,1,10,5:ENDPROC
2240GCOL0,C
2250REPEAT:Y%=Y%-4:PLOT77,X%,Y%:UNTILPOINT(X%,Y%-4)
2260ENDPROC
2270
2280DEFPROCjoin
2290GCOL0,C
2300d%=d%EOR1
2310IFd%=0 DX=PX:DY=PY:SOUND1,1,180,5 ELSEDRAWDX,DY
2320PROCwt(20)
2330ENDPROC
2340
2350DEFPROCbrsh
2360VDU5
2370GCOL0,C
2380PLOT0,-32,+16
2390PRINTCHR$(B%+RND(4))
2400VDU4
2410PROCwt(5)
2420ENDPROC
2430
2440DEFPROCpalt
2450PROCwt(10)
2460PRINT"Old colour ?"
2470LO%=GET-100
2480CLS
2490PROCwt(10)
2500PRINT"New ?"
2510AC%=GET-100
2520CLS
2530VDU19,LO%,AC%;0;
2540PROCwt(10)
2550ENDPROC
2560
2570DEFPROCdump
2580PRINT"Set up printer &     press <RETURN>."
2590REPEAT
2600  UNTILGET=13
2610CLS
2620REM  *  Your Printer Dump Call Here *
2630*PRINTER
2640VDU3
2650ENDPROC
2660
2670DEFPROCtext
2680SOUND1,1,180,5
2690PROCwt(5)
2700K=GET:IFK<31 ENDPROC
2710VDU5:GCOL0,C:PLOT0,-32,+16
2720PRINTCHR$K:VDU4:PROCwt(5)
2730ENDPROC
2740
2750DEFPROCwt(MS%)
2760TIME=0:REPEAT:UNTILTIME>MS%:*FX15,0
2770ENDPROC
2780
2790IFERR=17CLS:GOTO150
2800CLS
2810REPORT
2820G=GET:CLS:GOTOL%
```

GENIE ADJUSTMENTS

Andrew Howard

Converting the MCBAS, VGBAS1 and SPRITE programs of yesteryear to run on a disc-based system.

A year or so ago *Computing Today* published three of my programs for the Video Genie. These were MCBAS Machine Code to BASIC converter (Dec. '83), VGBASI BASIC extensions (April '84) and SPRITE software sprite simulation (August '84). All three of these programs were written for a cassette based Video Genie.

In June 1984 I was fortunate enough to acquire a double disk drive unit and Smal-LDOS. I wanted to convert these three programs so that they would run on a disk-based system. Unfortunately the details of conversion are not a simple relocation of the respective programs to a higher memory before saving it on disk. All three programs interact extensively with the interpreter through use of the DOS exits embedded in the code. DOS obviously uses these and adds its own routines to the standard interpreter to perform functions pertaining to a disk-based system. Each of the three programs must, therefore, be modified so that their use of the exits does not interfere with DOS, and vice-versa. In other words, both the DOS function and the respective program's function must be performed.

This article describes the changes which are necessary to the source code listing in each case in order for the program to operate properly with Smal-LDOS present.

CONVERSION OF MCBAS

This program converts each byte of a machine code program into a decimal number and places it in a DATA statement. The resulting data can then be POKED into memory.

The published listing was condensed slightly. Lines 10–20 were an ORG (41E2H) followed by a JP. This was the code to facilitate auto-run. Lines 30-40 were a message that loaded directly into 3C00H +, ie, a "Loading ... " message. The actual program began at 4300H. Line 50 contained the ORG statement for this, and lines 60–430 were all the text messages.

Obviously the disk version cannot load at 4300H. I chose to relocate it to the top of memory, but it can be placed anywhere as long as it is protected. It might be an idea to assemble two versions — the first loading in low memory so that high-memory programs can be converted, and the second loading in high memory. For top-of-memory, wait until you have made all the other modifications and then work out where the program must reside.

The loading message in lines 30 and 40 can be deleted to save some disk space. This may not be necessary.

MCBAS Version 1.1 (ie, disk version) requires a definite loading procedure described below. The loaded program must be initialised manually, and it can sometimes be annoying to have to remember the address of the program for initialisation via the SYSTEM command. So, change line 10 from ORG 41E2H to ORG 418EH. This is the address of command — thus this vector is free for use. The above modification will allow the use, when MCBAS has been loaded, to initialise it by typing the NAME command.

MCBAS version 1.0 relocated the BASIC program storage area to just above the end of the program. As MCBAS will no longer load below BASIC, as it were, the lines pertaining to this relocation are not necessary. Consequently a number of lines may be deleted, starting with 2440 and 2450. Lines 500–530 adjusted the start of BASIC pointer accordingly and disabled the auto-execute facility. These can be deleted — remember to move the INIT label to 540.

Throughout the program, the label 'BASIC' will now refer to the pointer holding the start of BASIC programs rather than the actual address of the start itself. Insert this line:

```
491   BASIC   DS   2
```

(or DEFS 2 if your assembler does not allow the abbreviation). This is the pointer. The following lines set the BASIC pointer on initialisation:

```
661             LD   HL,(40A4H)
662             LD   (BASIC),HL
```

The MCBAS OPEN command opens up previously closed programs, ie, it resets the pointer to BASIC programs to the original start. Obviously a change is necessary to line 830 to:

```
830             LD   HL,(BASIC)
```

Lines 570–660 set the vectors for the OPEN, CLOSE and GET commands. However there is no need for MCBAS to install a JP instruction (195) as this is done by LBASIC on initialisation. Lines 570–600 can therefore be deleted.

Note that MCBAS replaces the vectors for LBASIC's OPEN, CLOSE and GET commands, ie, these commands become inactive. This is of no real consequence because it is unlikely that they will be required during use of MCBAS — the program is usually used only once for each BASIC program.

MCBAS uses other exits within the GET command. However the original DOS function is restored immediately after use.

All necessary changes have now been made. The program can be assembled as MCBAS/CMD.

LOADING MCBAS V1.1

Obviously LBASIC must be installed before MCBAS can be executed, and so typing "MCBAS" at the LDOS level to auto load and execute the program is pointless.

Boot-up the system and enter the command

```
MEMORY (HIGH=X'nnnn')
```

where nnnn is one less than the value in the program ORG line (line 50). Now enter the following command line:

```
LBASIC CMD "L","MCBAS/CMD:d"
```

where d is the drive number. LBASIC will be loaded and executed and MCBAS will be loaded from drive d. Upon the READY prompt LBASIC is active and MCBAS is resident in protected high-memory. Type:

```
NAME
```

and the title message will be displayed. The MCBAS OPEN, CLOSE and GET commands are now in force.

CONVERSION OF VGBAS1

VGBAS1 adds thirteen extra commands to standard BASIC ranging from REPEAT-UNTIL to CANCEL and EXEC. A change of program location is necessary to avoid overwriting LBASIC. The program can, as with MCBAS, go anywhere, but I chose 8000H.

There are many references to '6000H' within the program (including the program ORG in line 10) that must be changed to '8000H'. If you are using the MISOSYS EDAS 352 Editor/Assembler this can be accomplished in one go by the command:

C /6000H/8000H/

The DEST table (line 6500) contains other address references not quite 6000H. Change the addresses in the following lines to their new 'high' addresses:

6750 from 5FFFH to 7FFFH
6780 from 5FFFH to 7FFFH
6800 from 6002H to 8002H
6880 from 5FFEH to 7FFEH
7180 from 5FFBH to 7FFBH
7260 from 5FFBH to 7FFBH

The relocation of the program is now complete.

As with MCBAS the 'spare' NAME command can be used for easier manual initialisation of the program. Insert the following lines:

```
5    ORG    41E8H
6    JP     INIT
```

These lines install the NAME command vector to the VGBAS1 initialisation routine. Also insert these lines:

```
6212    LD    HL,1997H
6213    LD    (418FH),HL
```

This is to reset the NAME vector so that further use will result in a syntax error.

VGBAS1 uses many other DOS exits. However at the time of writing it was easy to incorporate code to allow it to run with LEVEL III BASIC. I discovered that the same code allows smooth operation with Smal-LDOS. Hence no further changes are required to the listing. the program may be assembled as VGBAS1/CMD.

LOADING VGBAS1 V1.1

VGBAS1 cannot be executed until LBASIC has been installed and so an attempt to load the program from LDOS Ready is inappropriate. Since the initialisation routine relocates the main body of code, there is no need to reserve any high memory .Boot up the system and enter the command line:

LBASIC CMD"L","VGBAS1/CMD:d"

where d is the drive number. Upon the READY Prompt LBASIC is active and VGBAS1 is loaded and awaiting initialisation. The command:

NAME

will accomplish this. The title message will be displayed and all VGBAS1 commands are now in force.

CONVERSION OF SPRITE

This program adds five extra commands to BASIC to perform a software simulation of hardware graphics sprites. The commands are NAME, FIELD, KILL, GET and PUT. LBASIC also supports these commands to perform their respective disk functions (except for NAME). The conversions necessary for SPRITE are therefore predominantly pertaining to use of the NAME vector.

In the original listing, lines 10 to 60 were a loading message and an auto-execute facility. Note the alternative method of achieving automatic program execution, viz. to define the *KI and *DO vectors as the start of the program initialisation routine. This method has one disadvantage — a load error and premature halt of program loading may well cause a system crash.

The usual NAME vector installing lines should be inserted, ie:

```
10    ORG    418EH
20    JP     SPRITE
```

This will allow manual initialisation of SPRITE with the NAME command.

Of the five SPRITE commands, four are LBASIC commands and the fifth is the NAME command itself. The best way to overcome this problem was to make SPRITE commands double-worded by prefixing them with "NAME". The initialisation routine could then set the NAME vector to point to an additional routine which checked the second word in the command and passed control to the appropriate SPRITE routine accordingly.

It is now no longer necessary for the SPRITE initialisation routine to set the various command vectors, as the GET, PUT, FIELD and KILL commands perform their respective LBASIC functions. Consequently lines 80–270 can be deleted and replaced with the following:

```
80 SPRITE  LD    HL,SPRCMD
90         LD    (418FH),HL
```

This resets the name vector to point to a new subroutine SPRCMD. This is the subroutine that checks the second word in the command and passes control to the proper verb action routine.

To accommodate the new routine the program module ORG in line 390 must be changed accordingly. The new line is:

```
390    ORG    0F5FEH
```

Remove the END statement by deleting line 2920 and append the following subroutine:

```
2920 SPRCMD PUSH    AF
2930        RST     10H
2940        POP     AF
2950        CP      0AAH
2960        JP      Z,KILL
2970        CP      0A4H
2980        JP      Z,GET
2990        CP      0A5H
3000        JP      Z,PUT
3010        CP      0A3H
3020        JP      Z,FIELD
3030        CP      0A9H
3040        JP      Z,NAME
3050        JP      1997H
3060        END     SPRITE
```

After this short routine has been appended the entire listing can be assembled as SPRITE/CMD.

LOADING SPRITE V1.1.

Again SPRITE cannot be automatically loaded and executed from DOS. Boot up the system and enter:

MEMORY <HIGH=X'F5FD'>

to protect the SPRITE program and enter the command line:

LBASIC CMD"L","SPRITE/CMD:d "

where, as before, d is the drive number. Upon the READY prompt, LBASIC will be active and SPRITE will be resident in protected high memory. To initialise the program type:

NAME

and the title message will be displayed. LBASIC's GET, PUT, KILL and FIELD commands will still work as normal performing their respective operations. The SPRITE commands KILL, FIELD, NAME, GET and PUT Must now be prefixed with "NAME" for them to be executed. For example, FIELD<6,6> becomes NAMEFIELD<6,6>; KILL becomes NAMEKILL, etc. Omission of "NAME" will cause the LBASIC function to be performed.

The two example programs on page 28 of August '84 's CT now become:

```
10 CLS:PRINTCHR$(188)STRING$
(15,140)CHR$(188):PRINTCHR$
(191)" Video Genie "CHR$(191)
:PRINTCHR$(191)" Soft-Sprite    "
CHR$(191):PRINTSTRING$(17,131)
20 NAMEKILL:NAMEFIELD(17,5)
30 NAMENAME1,15360:REM **
THIS LINE IS NOT NECESSARY
40 A$=INKEY$:IFA$=""THEN40
ELSEA=VAL(A$):IFA=0THEN40
50 NAMEPUT1,A,3,1:GOTO40
```

Line 30 is not actually necessary since all sprites are initialised to 15360 start by the NAMEKILL command. The other demo-program is:

```
10 CLS:PRINTCHR$(188) ... ...
STRING$(17,131)
20 NAMEKILL.NAMEFIELD(17,5):P=
15360:NAMEGETP,0:CLS
30 FORY=1TO3:T=P:FORX=
1TO3:NAMEGETP,1:P=P+20:
NEXTX:P=T+<5*6
4):NEXTY
40 GOTO40
```



FROM THE CLASSROOM



Richard Rebain

A new column, opened to observe computing in Britain's schools.

Some weeks ago, at the London comprehensive in which I teach, I watched a group of eleven year olds working with computers. Earlier in the year they had been in the habit of entering the computer room as if a little in awe of the atmosphere of magic (which in any case had entered with them). Now they had come running up the stairs and along the corridor, burning Mr Dunlop's best rubber in an attempt to stop before the reinforced door. Somehow the magic was more mundane, the ogre within less forbidding.

They had begun, at least when this particular ogre had first met them, with LOGO. Most London schools have several versions, we also possess a small remote control robot device, a VALIANT TURTLE. Logo produces line drawings on the screen in obedience to simple instructions of the —

```
FORWARD 15
RIGHT 90
```

— variety. It will also store programs," and today children, we are going to teach the computer some new words". The Research Machines version we were using had lots of facilities, including colour, and the pupils were enthralled. Various writings and conversations had suggested that pupils and LOGO got on best in a "pupil-led" environment. You know the sort of thing: you only show them how to do something when they discover a need for it, and *ask*.

I've always been a tiny bit bothered by this idea. How could they know what LOGO could do unless they'd been shown? Of course if they were shown it would influence the direction of their investigations. It was never Catch-22: naturally the good ol' compromise was fine, until we discovered the Turtle.

It had red eyes and they glowed. Instant, 100%, undivided attention. They had suddenly realised that the boxes were able to influence the real world. After that it didn't matter what I had thought LOGO and the Turtle could do. My credibility depended on finding a way, it still does. It has surprised me just how much I've had to learn along the way. A slightly different implication to "pupil-led" than I had expected.

Incidentally, though we have found the Valiant Turtle most useful, it did take a while to get to know its little ways. It doesn't much care for undulations in the work surface, or bright sunlight, or overlapping sheets of paper (A0 sheets being quite unusual in my school), or to very smooth surfaces.

Somehow those eleven year olds proved so satisfying to teach. They seem never to think of these lessons as work or the computers as teaching tools. When they made mistakes the frustrated chatter was excited, when a success came up it was a personal triumph. Best of all for that hour they were not in a classroom and I was not a teacher. This has been due to the effectivenes of both the software and the hardware.

We have a six station Research Machines Network (RML 480Z microcomputers linked through a server and using double density eight inch floppies as backing memory). More recently we have acquired a 20Mb Winchester complete with a comprehensive package of software from the ILEA Computer Centre already installed and running from a menu. Printing is taken care of via two Epson printers (MX-80 and FX-80); these have worked well with a few glitches; those probably produced by constant use and occasional prying fingers. September will see an update. We have six silicon discs (sideways-RAM) on order which should boost the 480Zs to 256K, another six stations for the Network, an RML Nimbus 16-bit micro, mainly for administration, and another Epson FX 80 printer. Incidentally these school networks are usually equipped with 14 inch Microvitec colour monitors, very reliable and a clear brilliant picture. The whole package has functioned extremely well and I certainly would have no hesitaiton in recommending it, especially for educational use. The pupils have only had one consistent complaint; the lack of 'shoot em up' type arcade games. I realise that this is hardly an educational consideration worthy of note. Still, the pupils in general are very impressed with the performance and imposing appearance of the RML equipment. Many have enquired after prices and suppliers, so have their parents. Its not the prices that put them off, its the answer to questions like "great, but it can't play Pac-man, can it?" I'm almost ashamed to admit that I'm awaiting the publication of the flight simulator, AIRBOURNE, with bated breath. More on that when I get a copy. . .

□ □ □

TEACHING nowadays, especially if your subject is Computing, involves a great deal of instructional work with adults. It's really a little strange at first. The same classrooms, the same desks, even the same teacher. Just the colleagues transplated from the staff room and occupying the desks that only an hour previously the horrendous fourth had lounged in. Almost a kind of nightmare fantasy.

I don't know if the old adage about doctors making lousy patients was founded in truth but to my surprise it hasn't been too much of a trauma. Strangely, its some of the much-lauded advantages of computers in general that are causing some grief. After spending many laborious hours learning to find their way through the vagaries of multiqueststar, or whatever, the satisfaction sometimes changes to bewilderment. Instead of having conquered the computer they find instead that their horizons have been broadened, ". . . heavens, how much more is there to learn?" So different from ". . . yes, well, thats all very well, but it can't do so-and-so . . . can it?" The last delivered hope rather than satisfaction.

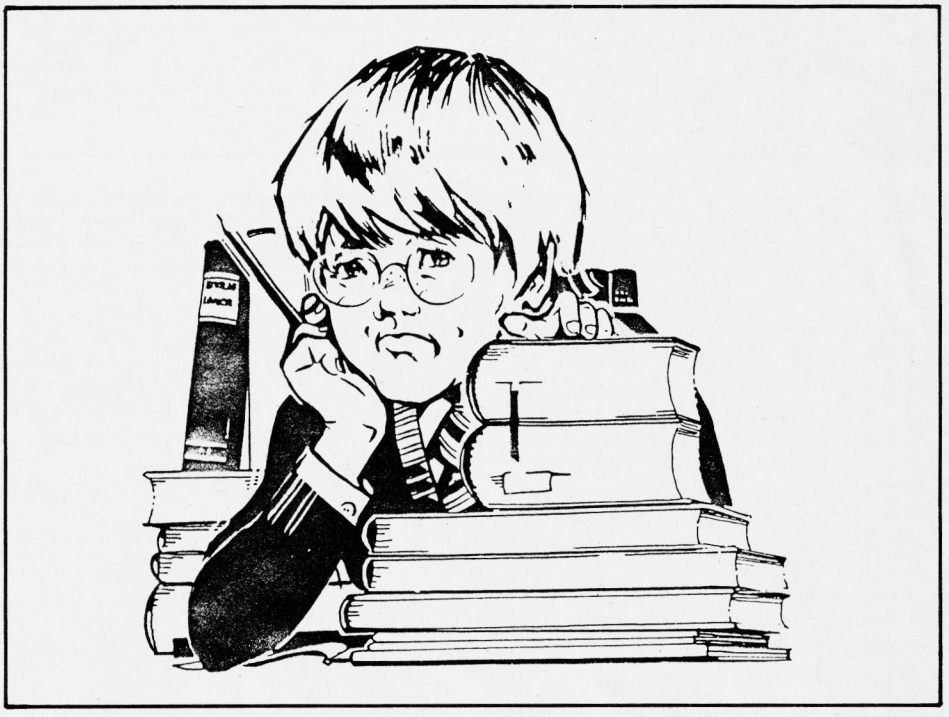
Another of the classic advantages, the computer's predictability, its endless patience, has also roused the adult ire. The adults KNOW the principle of GIGO, garbage in, garbage out. They don't seem to be able to cope with the implications. Children, on the other hand, seem to perceive the principle the other way round. They don't mind getting the garbage out every now and again because they seem to have an almost *innate* impression that the type of garbage they get out tells them something about the mistakes they have made. I don't make these observations in the hope of stirring outraged colleagues into writing letters; I know these scenarios are generalisations, but I'm equally sure that most computer studies teachers have met similar situations. I'm primarily interested in the reasons for the different attitudes and I have a feeling that it's

tied in to two factors: the teachers' expectations of themselves, (coupled with their superiors' expectations of them) and the present nature of the microcomputer.

I feel that the sheer scale of developments has left some teachers with slight feelings of hopelessness. They sometimes make excuses linked to their age or their subject background. They feel insecure because they suspect that superiors within subject areas are themselves under pressure to show that computers are being used, and they regard the enigmatic microcomputer as the root of the new instability in their lives. Little wonder they don't have quite the same attitude as their pupils. Adults in the kind of stress environment that teachers often find themselves, with little or no computing experience, find it extremely difficult to gain competence on the rare three day course that they are able to attend, or perhaps more frequently, on the couple of afternoon sessions at the local regional training centre. It falls more usually on the computer studies teachers within their schools to help them, and they are rarely timetabled for such courses but have to run them before school, lunchtimes and after school.

□ □ □

THE INFLUENCE of computers within education has grown at a staggering rate. Far faster, I suspect, than anyone envisaged. Software within subject areas has not really been adequate for teaching needs, but this is not the problem. There is so much software already that it's a full time job just keeping up. I don't believe that when the whole area of large numbers of sophisticated microcomputers in schools was being mooted, the sheer scope of in-service training required was ever officially recognised. It has grown. The picture is quite different already to some years ago, and given adequate funding should continue to grow. The need is there, more than that, the need is expanding at an increasing rate.



WDSoftware

For the QL

WD Utilities (3rd ed) **(base £5.50)**

PRINT 60-file DIRectory or view it on one screen, one-key LOAD. COPY or PRINT 60 files with one key (allows for namesakes). Multiple FORMATting to prevent corruption by stretching of tape. TOOLkit to give dated, numbered modules in program development. PRUNE old files to release space (one key DELETEs a file). Full instructions in QUILL file. Use up to 6 EXTRA MICRODRIVES (add on your Spectrum ones)!

WD Utilities for CST Disks **(base £8)**

100-file capacity, for CST/Compumatamate disk system AND up to 4 extra microdrives. User-friendly timesavers.

Ref QL (4th ed) **(base £4)**

700 useful QL references in an ARCHIVE file. Too long to share a cartridge.

For Spectrum/QL/BBC

WD Morse Tutor **(base £4)**

From absolute beginner to beyond RYA and Amateur Radio receiving. Adjust pitch. Set speed to your test level (4-18 wpm). Learn from single characters, via groups with wide spaces to random sentences; decrease spacing to normal. Write down what you hear, then CHECK on Screen or Printer (or speech for Spectrum fitted with Currah Microspeech). Also own message, random figures, letters or mixed.

For Spectrum 48K

Tradewind **(base £4)**

Sailing/trading strategy game with graphic surprises.

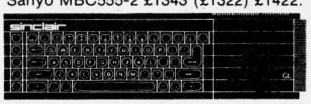
Jersey Quest **(base £4)**

Text adventure with Bergerac and the Dragon, (not disk).

Prices: (incl Europe postage, elsewhere add £1). Spectrum/BBC cassettes, base price only. QL or Spectrum Microdrives. £2/cartridge plus base price. 5¼" floppies. £2 plus base Morse for £11.30. 3½" floppies, £4 plus base. Two or more programs on one medium — pay medium plus base EG. WD Utilities and RefQL for £10.50, but IMPOSSIBLE to mix QL/BBC/Spectrum programs on one medium. Send YOUR cartridge and base price, but FORMAT it FIRST in your DRIVE 1 for compatibility.

WDSoftware, Hilltop, St Mary, Jersey.
Tel: (0534) 81392 Dept CT

BUSINESS COMPUTERS
Apricot F1E £637 (£614) £658. Apricot F1 £894 (£870) £933. Epson PX8 £900 (£872) £892. Commodore PC10 £1595 (£1564) £1664. Commodore PC20 £2573 (£2485) £2685. Sanyo MBC 775 £1920 (£1899) £1999. Cannon A200C £1609 (£1586) £1686. Sanyo MBC550 £723 (£699) £799. Sanyo MBC550-2 £975 (£939) £1039. Sanyo MBC555-2 £1343 (£1322) £1422.



ORIC AND SINCLAIR COMPUTERS
MCP40 Oric printer/plotter £109 (£110) £122. Sinclair pocket TV £97 (£95) £101. Sinclair QL Computer £374 (£365) £386. QL Floppy disc interface £107 (£103) £109. 3.5" disc drive to suit this interface £177 (£176) £196. Sinclair Spectrum Plus Computer 48K £123 (£127) £147.Original 48K Sinclair Spectrum £89 (95) £116. Kit to upgrade the Spectrum to Spectrum Plus £30 (£30) £40. Micrdrive £49 (£50) £60. RS232 interface 1 £40 (£50) £60. Special offer:— Microdrive + Interface 1 + 4 cartridges £97 (£99) £107. Blank microdrive cartridges £2-50 (£3) £4. Spectrum floppy disc interface (See Cumana disc section for suitable disc drives) £97 (£89) £99. Interface 2 £20-45 (£20) £24. 32K memory upgrade kit for 16K spectrum (issue 2 and 3 only) £31 (£28) £30. Spectrum Centronics printer interface £46 (£42) £47. ZX Printer has been replaced by the Alphacom 32 £61 (£59) £72. ZX81 computer £29 (£29) £39.

COMMODORE COMPUTERS
Commodore 128 £269 (£249) £279. Commodore 64 £161 (£159) £189. Commodore 64 + recorder + software £187 (£213) £243. Convertor to allow most ordinary mono cassette recorders to be used with the Vic 20 and the Commodore 64 £9-78 (£9) £11. Commodore cassette recorder £43 (£44) £50. Centronics printer interface for Vic 20 and the Commodore 64 £45 (£41) £46. Disc drive £191 (£186) £217.

AMSTRAD, ATARI AND ENTERPRISE AND MSX COMPUTERS
Amstrad 464 Colour £342 (£348) £388. Amstrad 464 Green £232 (£247) £287. Amstrad 664 Colour £439 (£431) £481. Amstrad 664 Green £331 (£332) £382. Atari 130XE computer £158 (£163) £183. Atari 520ST computer with 3.5" disc drive, mouse, monitor and software £675 (£670) £730. Atari 800XL computer + recorder £120 (£123) £143. Atari 800XL Computer + disc drive £229 (£230) £260. Atari data recorder £34 (£37) £47. Atari disc drive £172 (£171) £191. Atari 1020 printer £93 (£99) £115. Enterprise 64 computer £172 (£170) £190. Enterprise 128 £233 (£229) £249. Goldstar MSX £138 (£138) £158.

ACORN COMPUTERS
Acorn Electron £119 (£119) £139. New 64K BBC Model B Plus with doub. density disc interface £457 (£441) £471. BBC Model B £345 (£333) £373. Acorn disc i/f + DNFS £97 (£95) £100. See below for suitable disc drives. Colour monitor £188 (£228) £268.

CUMANA DISC DRIVES
To suit disc interfaces of Sinclair, Spectrum and BBC B. Single:— 40 track single sided £117 (£120) £150, 40 tr double sided £149 (£149) £179, 80 tr ds £166 (£166) £196. Dual:— 40 tr ss £209 (£211) £251, 40 tr ds £285 (£283) £323, 80 tr ds £307 (£304) £344.

PRINTERS
New Epson LX80 £249 (£249) £282 Tractor for LX80 £25 (£33) £53. Brother HR5 £148 (£152) £184. Brother M1009 £201 (£203) £234. Shinwa CTI CPA80 £218 (£222) £258. Cannon PW1080A £309 (£306) £356. Brother EP22 £125 (£114) £134. Brother EP44 £212 (£208) £228.

SWANLEY ELECTRONICS

Dept CT, 32 Goldsel Road, Swanley, Kent BR8 8EZ, England.
TEL: Swanley (0322) 64851

Official orders welcome. All prices are inclusive. UK prices are shown first and include post and VAT. The second price in brackets is for export customers in Europe and includes insured air mail postage. The third price is for export customers outside Europe (include Australia etc) and includes insured airmail postage.

Computing Today

NEXT ISSUE OUT FRIDAY OCTOBER 11th

BUILDING A DATA BASE —
a bricks and mortar approach
SIDEWAYS ROM —
Superpower's Amstrad sideways ROM card reviewed
COMPUTER CRIME —
the anatomy of the problem
LEAST SQUARES —
approximating polynomials of N'th degree
CAD —
BBC character generation
COMPETITION —
win a Zenith PC

All this and more in the next edition of

Computing Today

Articles described here are in an advanced state of preparation but the circumstances may dictate changes to the final contents.

THE EVOLUTION CONTINUES..

ALGORITHM ANGLES

Geoffrey Childs

I make no apology that my main aglorithm in this article is, when expressed in BASIC just one line long!

```
895 IF SR=1 THEN SR=0:RETURN
```

An algorithm is a posh word meaning: "a method of achieving a result." It is usually applied to a result that we wish to use many times. I am sure that for many readers of this magazine, programming is the main activity with computers. A method that can be applied to programming is an algorithm, whether it is a page of coding or simply an idea that can often be used.

Much scorn is heaped on programmers who do not sit down and plan a program with flowcharts or similar methods. We are told that if we do not discipline ourselves in this way, we may manage to hack out games, but that our methods are not suitable for any form of serious programming. What do they mean by "serious programming"? I suspect it means programming for business, where a firm of consultants will be called in if anything goes wrong. Yet computing has progressed far beyond the point where the only serious applications were for firms like I.C.I. or one of the big banks.

Let us take just one example of serious programming that has no business applications: educational programming. I am sure that readers can think of many other examples, but this one will be sufficient for the purposes of this article.

The first question in a programmer's mind must always be: "What does the end user want?" Certainly, in education, as in all other applications, the first requirement is a program that works. Simplicity of use is of major importance, particularly if the content matter is such that the teacher or user is unlikely to be particularly proficient with a computer. Speed is unlikely to matter much. If the programming style enables the program to run efficiently, it will be adequate in most cases.

There is a need, naturally, for the program to be useful rather than a gimmick, and many educational programs fall down on this score. But, above all, there is a necessity to sustain interest. I defy any programmer who thinks that maximum interest can be provided by sitting down with pencil and paper, working out flowcharts and subroutines, typing the program into the computer, and leaving it at that.

My method is rarely to put anything on paper. I will have had ideas in my head, usually for days or weeks before I touch the computer. In this time a broad design will have been formed. When I have enough time to spare I start to write the program. Still no paper, usually. As I write the program. I live with it, and expect to have many thoughts which will improve something that I have already done, brighten up the next part or indeed add something fundamental to the overall design. During this time I may set a bright idea for the title page which is usually one of the last parts of the program to be written.

I may have started with some subroutines. — I don't break all the rules all the time — but there are times when I have an idea that would be much more simply executed if I had written part of the main program as a subroutine. I could rewrite it, I could copy it out again. What a bore! Programming is my hobby, not my daily round of toil. Luckily there is a simple way out. Set the flag SR, GOSUB into the main routine and simply insert the one line algorithm that I have given.

Is it bad programming? I will leave you to decide. The experts talk about recursion with great reverence and say that it is not possible in BASIC. I suspect it is, but since the main program can be thought of (or even written) as a subroutine, I could use a snobbish name for my method and call it semi-recursion. But that isn't the point, — the method is useful and efficient. If it doesn't satisfy the purists, why should we mind if the program works?

Has anybody realised that because you can do things like this, BASIC in some respects is a very efficient language? Despite what I have said, I nearly always write machine code on paper if I use it, unless I have had too many drinks to read my own writing!

```
800 INPUT "Enter name of file.";F$
810 L=LEN(F$):IF L=0 THEN 800
820 IF L>8 THEN F$=LEFT$(F$,8):L=8
830 G$="":FOR N=1 TO L
840 Z=ASC(MID$(F$,N,1))
850 IF Z=46 THEN N=L
855 IF Z>47 AND Z<58 THEN G$=G$
    +CHR$(Z)
860 Z=Z AND 223: IF Z>64 AND Z<91
    THEN G$=G$ + CHR$(Z)
870 NEXT
880 IF G$=""THEN 800
890 F$=G$+".DAT"
900 REM Rest of program.
```

This is a recent example of the use of the algorithm — if you like an algorithm in itself. The idea is to call for a name for a data file that the computer will accept. Small or upper case is accepted, and if anybody has been 'helpful' by adding .DAT this is taken too. I wrote this in the main program, (yes, it should have been a subroutine, but it wasn't). Later I thought of another option for the main menu which also needed a data file. Just put in line 895 as indicated and GOSUB 800 from the new routine. Easy!

Another recent recurrence of the situation arose when I was writing a suite of statistics programs. Naturally, a standard deviation calculation was included, and it was necessary to call this routine from another, more complex calculation. The original routine had used CLEAR for two reasons, one slightly sloppy, perhaps. Various counters had been used, and the CLEAR resets these to zero if it has been 'forgotten' in the original coding. It also contained arrays dimensioned to the user's taste. The CLEAR allows them to be redimensioned.

But this causes the SR flag to be cleared as well, and may obliterate the return address in some BASICS. The first problem is solved by POKEing a flag instead of setting a variable, and the second by using the dreaded GOTO instead of GOSUB, and RETURN.

Where do you POKE to set the flag? On this particular computer I used location 83, since it is in the middle of the operating system in a row of unused bytes. Nobody's going to mess about there (except me). Another way is to set a LIMIT and use the locations above this as flags. If the BASIC is in RAM (or part of it is), you can POKE into the middle of a message successfully. Beware of this, however! Somebody may be using a modified BASIC. A simple and universal alternative is to write:

```
1 REM This is the last REM in the
program.
```

It is probably true, anyway, if you program like me. It leaves you a location, say 6 above the start of BASIC, that is safe to use as a flag.

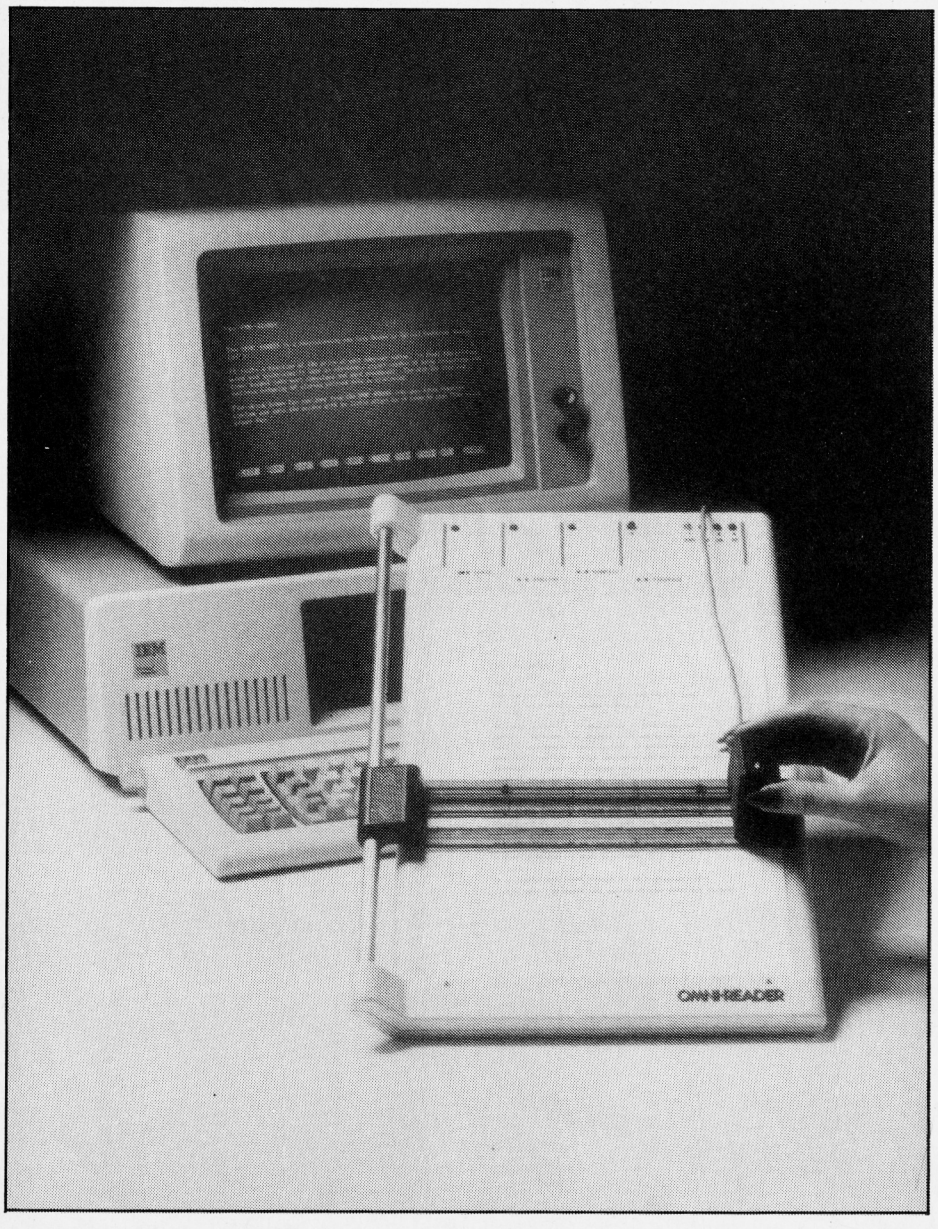
THE OMNI READER

Jamie Clary

An OCR system for less than £10K? You must be joking! Oberon International aren't, and theirs costs less than £1K

The Omni Reader must be one of the more unconventional gadgets *Computing Today* has reviewed in recent months, a welcome change from modems, disc-drives, and other run-of-the-mill peripherals. Not only is it different, but so far as we can tell it is the only optical character recognition system widely available, and it is certainly uniquely priced at £595 + VAT.

The system consists of two moving parts, and a tablet upon which the source document rests. A precision read-head, containing a broad-spectrum light source and an infra-red detector, is moved by hand along a grove in a 'tracking-guide'. The tracking guide is essentially a ruler serving two purposes. Firstly, it assists the user in aligning the read-head accurately to coincide with



vides the system with vital information regarding the position of the read-head and the rate at which the line is being scanned. A 'clock-track', a line of dots and dashes printed onto the ruler, is detected and decoded by the system as the image of the character is lifted from the page. Since no guarantee can be put on a user's ability to move the head smoothly along the page, the system has been designed to tolerate significant variations in the scan-rate and this self-clocking technique copes admirably with such variations. Incidentally, the stated maximum bidirectional scan rate is one hundred and sixty characters per second, which we are told is 2 to 3 times the speed of a 'competent' WP operator (competent WP operators slighted by this remark, please complain to Oberon, not us! - Ed.)

The Omni Reader can be linked to any micro with an RS232C or RS423A interface and appropriate software. Six baud rates, randing from 300 to 600 Baud can be selected via a DIL switch on the tablet's rear panel, and although no serial cable is supplied, all the necessary information required to configure a match for your system is given in the user manual.

Incidentally, it is amusing to note that the maximum scanning rate claimed for the system exceeds its maximum serial transmission rate by roughly 50 characters per second - shome mishtake, shurely?

FIRMWARE

The resident software permits the system to recognise the four commonest typefaces: Courier 10, Courier12, Letter Gothic 12 and Prestige Elite 12. Additional typefaces may be read by downloading software, available from Oberon at extra cost.

Since very subtle variations can occur in the size of characters and the quality of documents, the system can be made tolerant of slight irregularities. LARGE and SMALL are two options, selected by scanning command lines printed on the tablet, which can be used to overcome the effects of photocopiers giving copies of imperfect scale. The POORCOPY mode, sadly not what I at first expected (but then my prose isn't *too* bad) can be used if characters on a document are imperfectly formed.

QUIRKS

Of the few 'quirks' that became evident after using the system and following talks with Oberon themselves, one important nuance was uncovered. The system works by detecting the stark variation between the whiteness of paper and the darknes of a character deposited upon it. In practice, the mechanism is a little more complicated than this. As has been already mentioned, embedded within the read head is an infra-red detector and a broad spectrum light source. While light is *reflected* by the virgin sheet, certain deposits on the paper will *absorb* infra-red, and it is the absorption and reflection of infra-red light that enables the device to function. Unfortunately, only Carbon will absorb light from the correct spectrum! Copy bashed out on your time-honoured portable probably won't make the grade, since good old-fashioned typewriter ribbons carry non-carbon based inks that will *not* absorb infra red. However, all is not lost, since the system will read a photocopy of your latest piece, assuming it's in the proper typeface etc., and most of the cartridges used in modern office typewriters contain carbon-coated or carbon-impregnated tapes.

Although this may at first appear a chronic oversight on the part of the designer, an ability to descriminate between deposits on the paper does offer certain advantages. Not only will the Omni Reader totally ignore coffe stains and the remains of other accidents, but wiping over selected portions of the text with one of those ghastly green or purple highlighting pens will make the read-head merrily skip words, sentences and paragraphs as marked.

Sadly, the Omni Reader will not work with the characters most familiar to us, namely the dot matrix variety. This has less to do with the wide range of fonts that dot matrix can offer, than the formation of a printed matrix as a series of parallel rows of dots, which causes difficulties in recognising the beginnings and ends of individual and successive characters.

FACTSHEET: Omni Reader

Dimensions	120mm high, 270mm wide, 404mm long. Power adaptor 63mm high, 77mm wide, 187mm long
Weight	Omni Reader: 1.2Kg Power adaptor: 0.87g
Rule scan length	190mm max. with standard rule 360mm max. with long rule
Data output format	RS232C, configured as modem, handshake on RTS to control sent data and on DTR, DSR, and DCD to control received data. 1 start bit, 8 data bits, no parity, 2 stop bits.
Baud rates	Rear panel selectable to 300, 600, 1200, 2400, 4800, 9600
Typeface recognition	Courier 10 Courier12 Letter Gothic 12 Prestige Elite 12
Price	£595 + VAT £695 + VAT for complete system including software
Available from	Oberon International Cleveland Road Maylands Wood Estate Hemel Hempstead Herts HP2 4SE Tel. (0440) 3803

CONCLUSION

The Omni Reader is enviably free from direct competition, a detail optimists and pessimists alike will use to equal and opposite advantage when predicting its commercial value. On the plus side, freedom from competition brings instant market domination. Conversely, there isn't much to be gained from comering a non-existent market, and that continuing research into optical character recognition has spawned just one general purpose character reader for the home/office market suggests that the demand for a product of this type is not enormous.

But difficult as it is to judge any 'first of its kind' equipment, the Omni Reader really isn't a bad gadget. A little over-priced, perhaps, considering what else £600 will buy. The important thing to note is that the system *works,* and it is *available,* and until a rival optical character recognition system comes along in the same price bracket, it would be unfair not to suggest considering it.

PRINTERS AT DISCOUNT PRICES



CTP80
80 col thermal printer
Centronics interface only 99.00

MP165
FRICTION + TRACTOR 165CPS
75CPS NLQ MODE 285.00

MANNESMAN TALLY MT80	219.00
CANNON PW1080A	319.00
CANNON PW1156A	435.00
BROTHER M1009	189.00
DAISYSTEP 2000	275.00
BROTHER HR5	99.00
CTP80 THERMAL	99.00
CCP40 PLOTTER	79.00
A4 SIZE PLOTTER	199.00

Shinwa

CP80 PARALLEL	199.00
CPA80 PARALLEL	209.00
CPA80 SERIAL	239.00
CPA80C FOR CBM64	235.00

OTHER VERSIONS AVAILABLE

ALL PRICES INCLUSIVE OF VAT
PLEASE ADD 10.00 DELIVERY

MANY OTHER PRINTERS AVAILABLE.
PLEASE TELEPHONE FOR PRICES OR ASK FOR OUR FULL LIST OF COMPUTERS, PRINTERS, RIBBONS, DISK DRIVES, MONITORS, ETC.

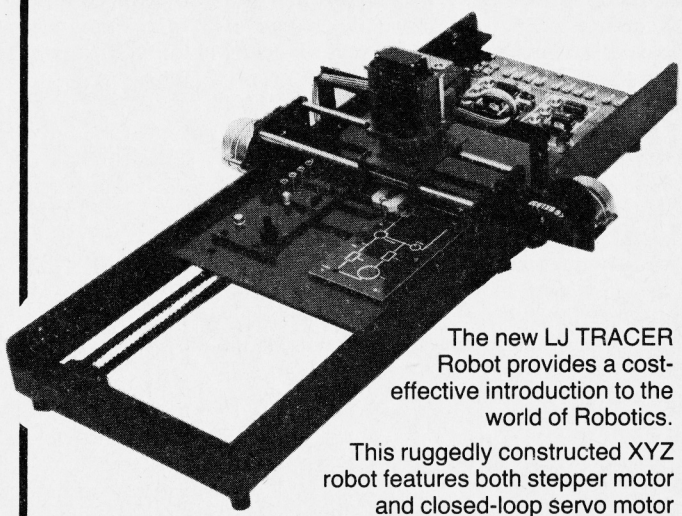
MINNIE MICRO COMPUTER STORE

TO ORDER JUST TELEPHONE WITH YOUR ACCESS/VISA NUMBER AND WE WILL DISPATCH SAME DAY SUBJECT TO STOCK (NOT SUN)

0702•615809

12 EASTERN ESPLANADE, SOUTHEND, ESSEX.

TRACER - A new Robotic Teaching System from LJ



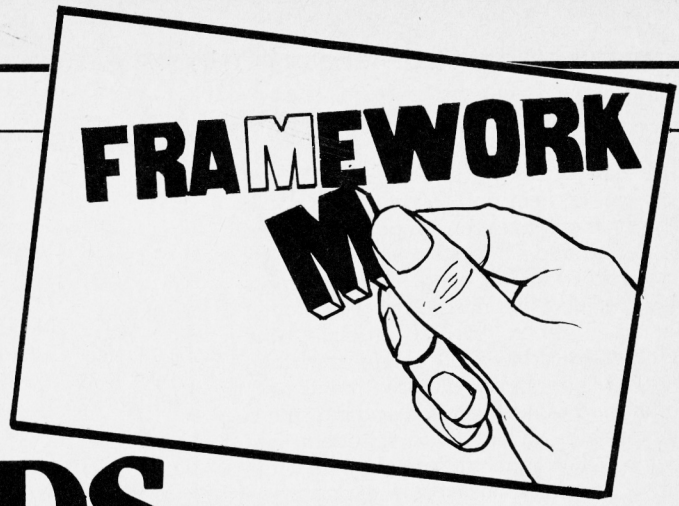
The new LJ TRACER Robot provides a cost-effective introduction to the world of Robotics.

This ruggedly constructed XYZ robot features both stepper motor and closed-loop servo motor drive. The TRACER can be driven by any microcomputer with a suitable TTL level I/O facility.

The TRACER is supplied with a pcb Assembly Task Kit (as shown) and a 3 colour pen-plot kit.

For full details of this and other LJ products send for our catalogue.

LJ Electronics
Francis Way, Bowthorpe Industrial Estate
Norwich, NR5 9JA. England.
Tel: (0603) 748001 Telex: 975504



ADDING BASIC COMMANDS

Geoffrey Childs

"It is easy", one manual comments, "to tailor this BASIC to your needs." Easy? I'll leave that for you to answer! Perhaps it is just a question of *relativity.* Recently I read a claim by somebody that a professional progammer could learn a new processor in two days. This raised much disbelief, except from another gentleman who reckoned it would only take him twenty minutes. O.K., I'll believe them, but I know it would take me a lot longer. I also read somewhere that an average day's output from a professional programmer was 22 bytes. At first sight this looks extremely small, but if you work it out, it simply means that a team of six programmers would write a BASIC interpreter in about six months if they started from scratch.

All this comes down to the question of whether you could improve your own BASIC intepreter. Maybe you don't want to. More likely, you feel that it would be impossible for you to do so. What I want to show you is that although I am not claiming that is is easy for most of us mere mortals, it may not be as impossible as it looks! I will agree that on the two computers I have tried this with, the BASIC has been in RAM Which makes life easier. I have seen BASIC extensions for some ROM based interpreters, such as the Commodore 64, BBC, Spectrum and Amstrad. It is a matter of knowing how and if it can be done of your own computer. In this article I intend to describe what I did on one computer, the Einstein. In the end, I feel the effort was worthwhile but there were many moments when things went wrong and I knew myself for the idiot that others have always taken me to be!

ON LINE

It all started with a telephone call one lunchtime. He said, "I've got a prototype Einstein, and I'm going on holiday for two weeks. Will you take it and write some software if you can?" I replied that I should also be on holiday. "Yes, that's what I mean" His idea of a holiday for himself is very different from his idea of a holiday for me! Still he's the boss of a software house, and works very hard. At least that's what he tells me.

So, that evening I picked up the biscuit-box. We called it a biscuit box because it looked like one, but eventually I used this particular machine for quite a long time and became very fond of it!

At that time the user manuals had not been completely written, so some ingenuity had to be used to discover the capabilities of the machine. This is exactly the challenge I enjoy, and it was interesting to later read the (very good) manuals and find where I was right and where I had missed something. The BASIC that was available was Xtal, which was loaded from disc after going into DOS. Recently, Tatung have produced an alternative BBC type BASIC which will run on the Einstein.

The first problem was that I had never used Xtal Basic before. Maybe this wouldn't have been a great problem with a full manual, but first I had to discover what it could do. The reserved word table must be somewhere in BASIC, so it had to be found. It is always worth doing with a new machine. It is surprising what gets missed out, even in a fully written menu

```
10 FOR N= TO 16000
20 IF PEEK (N) <> 78 THEN NEXT
30 IF N = 16000 THEN END
40 IF PEEK (N+1) <> 80 THEN NEXT
50 IF PEEK (N+2) <> 85 THEN NEXT
60 ?N: NEXT
```

This searches for the three middle letters of INPUT. INPUT will normally be near the start of the reserved word list, so that when you set a value of N, you should be able to find the start of the table without too much difficulty. (You can leave out line 30 if you can put up with a trivial crash at the end.) Another way of finding the reserved word would be to go into the machine's monitor (if it has one) and dump the code, but I am not convinced that this is quicker in the end.

By the time I returned the Einstein, I had discovered what all but two of the keywords did, and decided I was very impressed by the machine and Xtal BASIC. I also decided that there were a few ways in which Xtal BASIC could be improved, and intended to try to do this. This is not a contradiction in terms. A good BASIC is worth improving, whereas a bad one you just hope not to use! I know I may stir up a homets' nest, but my opinion is that Xtal is a better BASIC than the much vaunted BBC . . . All right, all right, but I am entitled to my own opinion!

Not long afterwards I was loaned an (almost) production model, and decided to try out my ideas. There were three tasks to carry out before anything serious could be attempted, and another ongoing one, which a year later I haven't fully completed!

The first was to write a disassembler. It is probably true to say that making alterations to BASIC is three parts disassembly to one part writing code. It isn't particularly difficult to write a BASIC disassembler (or assembler come to that), but it is time consuming. It is one way in which one can become more familiar with machine code. It may not be original, but I think I put one innovation into this one, as it will read code from the right place if a keyword is typed in instead of a number.

WORKSPACE

The next task was to give myself space to work. I wanted an extra 1K, as I felt that this would give sufficient space for the extra coding for the new reserved words, the additions to the address and reserved word tables and the diversions of resident routines that might be needed. This still left over 42K for BASIC programs. XTAL BASIC works on a series of pointers which can be accessed by PTR and listing these suggested which ones would have to be changed to raise the start. Eventually I changed them, checked with a short BASIC program which worked and SIZE was returned correctly. Satisfied with a job well done, BASIC was saved, and reloaded. The start wasn't raised. The cold start routine had undone all my work! So that had to be changed as well. After disassembling this and making several changes (as well as the original ones), BASIC saved and loaded successfully. When I was reminding myself of this in order to write this article, I discovered it could all be done with a single DOKE! On loading BASIC, the machine executes a

cold start routine, which copies the pointers into the addresses required by the interpreter. It is only necessary to adjust the start of BASIC pointer and the routine will make the adjustments required for the other pointers.

The next task was to make space for extra reserved words and addresses. The architecture of the code at the end of the interpreter works like this:

Standard word table	14859
Auxiliary word table	15300
Auxiliary address table	15412
Standard address table	15462
Standard function address table	15598
Cold start boot up	15698
Basic program start	15873

If the auxillary address table is moved up to the old start of BASIC, we have another 50 bytes for the names of new reserved words. We also have room to add addresses to the auxiliary table which will correspond to these words. Copy the auxiliary address table into the new locations by POKEing. Now change the pointer, and, if you like, put 0's into the old locations of the auxiliary address table. Your BASIC should still be working although you have nothing new and have lost 1K of RAM. You will need to change the cold start routine to adjust this pointer before you save this amended version.

Ready to go? Not a bit of it. Now comes the hard work. You do not, of course, need to disassemble the whole BASIC, but you need to know enough of what is going on at certain parts of it. Prepare for a slog, this is what I meant by the "ongoing task".

Eventually I was ready to add new words, I chose REG for the first experiment. It is not very useful as it simply stores the registers in a buffer, but the code is simple. Most of the routine would be used in a later addition: USR X,Y loads register BC with X, executes code from Y, and stores the register results in this buffer. Above all, this word would check that I was getting in and out of the interpreter successfully. You do need to find out which registers must be stored, whether to return with a RET or a JP, and how to add to the reserved word table keeping the end of table pointer at the right place. REG worked fine, for what it was, and now I could be more ambitious.

There were three questions I needed to ask:

- What would I like to add to the BASIC (or amend)?
- What would other users like?
- What did the magazine reviews suggest was needed?

Above all I wanted a FIND and a FIND" which are not *quite* the same, as the former searches reserved words while the latter searches text. I also felt that a TRACE should be added, along with a simpler method of appending programs. Other users would probably like some additional control structures, and I would add REPEAT UNTIL, GOTO and GOSUB labels. LCL, the option to localise variables in a subroutine, was another sop to structure.

EXAMPLE ROUTINE

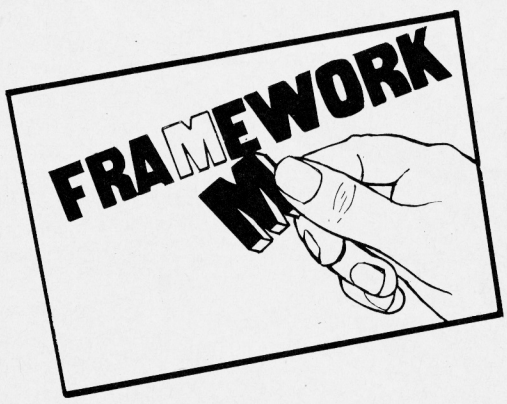
At this stage I shall show you one of my routines. It is for a keyword OVER. If you have never heard of it before, nor had I. Yet it is so simple and frequently used that it amazes me that nobody seems to have thought of it and incorporated it in other BASICS. It simply prints a message at the bottom of the screen for a key-press, and turns over on to the next screen. Haven't you written this routine in BASIC, time and time again?

The coding is simple. It was finding the ROM routines that took most of the time. It is a good example that assembling BASIC is three quarters disassembly.

There were four routines to locate. First place the cursor at the right spot. This is found by looking through the PRINT routine searching for @ and seeing what it does next. You can find message routines at numerous places in the BASIC. INCH (the near equivalent of GET or INKEY in many systems) gives a routine for a keyboard scan. Finally, it isn't hard to work out what CLS will give as a clear screen routine!

I give the routine in decimal, not hex. I'm sorry, I find that it is easier to remember locations that way, even though I suppose it is a little eccentric!

```
16650 PUSH HL
16651 LD DE 5892 (row 23 col 4)
16654 CALL 8889 (place cursor)
16657 CALL 697 (write message)
16660-16681 MESSAGE
16681 CONTAINS 135: 7 for bell, add
      128 for end of message.
16682 LD E 25
16684 CALL 8889 (flash cursor at right
      place.)
16687 CALL 13692 (keyboard scan)
16690 CALL 11122 (clear scan)
16693 POP HL
16694 RET
```



Space does not permit detailing the routines used in full, but these brief descriptions of some of the more interesting ones may be of use.

FIND first of all stores the subsequent 8 (or less) bytes in a buffer specially created for this. If it is not followed by ", the bytes will be in tokenised form, which allows the reserved words, as well as parts of strings to be located. The pointers for start and end of BASIC program are put in the registers and each line is scanned for a match with the buffer. If such a match is found the line number is taken from the stack where it has been PUSHed, into HL, and the routine at 768(dec.) is used to print it on the screen. The routine continues until the end of the BASIC program and takes about 0.3 seconds to search a program of 20K size with perhaps a couple of matches. Some of the routine is also called by the GOTO and GOSUB when a label is used instead of a line number.

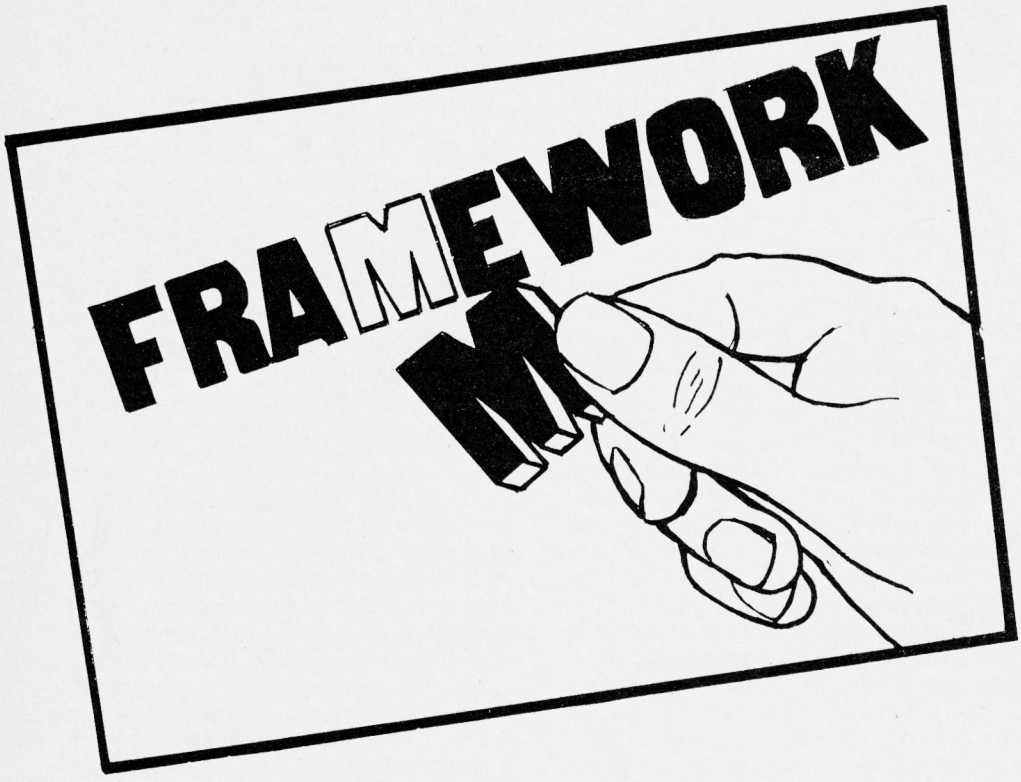
REPEAT UNTIL consists of two parts. The first part claims a buffer of 20 bytes, so that these loops can be nested 10 deep. The repeated statement pushes everything in this buffer up two bytes, and finds the address of the end of the REPEAT statement, loading the lowest 2 bytes of the buffer with this. Anything after REPEAT will be ignored so that REPEAT Countdown, for example, is acceptable for those who like to label in this way. The UNTIL relies heavily on the IF routine resident in the interpreter. There are difficulties in that the sense of a statement is sometimes opposite and that ELSE would be meaningless in an UNTIL statement.

LCL (localise variables) works by a trick, (to be honest an amateur can get away with it, although I'm sure that professionals would frown upon it!) On LCL, any single letter variable has a 9 added, e.g. N becomes N9. At RETURN or a second LCL, all 'single letter followed by 9' variables are turned into single letter ones. Thus a subroutine containing LCL may be called where N=10 in the main program. In executing the subroutine, a FOR N=1 to 1000 loop is run. Upon return from the subroutine, N will have the old value of 10. Both N's will be in the variable table but the interpreter will take the value of the first N. Using LCL means you can't use variables of the type K9 in the main program, but perhaps that isn't a great sacrifice!

OLD needs more rewriting of the existing NEW routine than writing the OLD routine. NEW now saves some pointers and the first 20 bytes of program in an area of memory reserved for this. OLD checks whether the previous program is sufficiently intact, and if so, loads back what has been stored by NEW.

One could not write an article like this without mentioning *bugs!* First of all, there were the bugs in the original Xtal BASIC, — well, perhaps not bugs, but alterations I felt should be made. PSG7, N; a sound generator command, crashed the machine into limbo if you were careless in the choice of N. The MUSIC didn't stop, unless you put in a rest at the end. REM ignored subsequent statements on a line. The first two were easy to cure, but to change the REM would have caused confusion. The answer was to use as an alternative REM which operated only as far as the next colon.

The most intriguing bug that I came across in writing the alterations was in FIND. The routine is meant to report line numbers when the word is found. It did report — 371 *every time!* After correction it only reported 371 some of the time! Clearly the machine was trying to tell me that although in early computing days 42 was the secret of the universe, Einstein's relativity has changed this secret to 371.



Once the alterations had been written I did not find much that had to be altered. Once I was mystified for 24 hours, though. Why couldn't I LOCK my disc programs any more? Changing the reserved word LOC to LCL sorted that out, when I eventually solved the mystery.

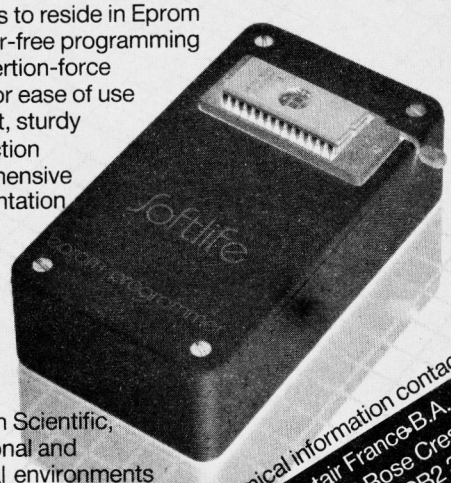
Until very recently I hadn't found any bugs in either the Einstein's Xtal BASIC or my own version for about six months. But suddenly, bugs were having a field day! One in the Xtal, 2 in my own. IF...THEN...FILL doesn't work on Xtal! The reason is that the second token in FILL is the same as ELSE. This would be fiddly, but not impossible to correct. My FIND doesn't work if the program is under HOLD. That would be easy to alter. Just call the MGE routine at the start of FIND. Finally a most peculiar one: If you type: ?A!A, the machine completely hangs up. It is clearly to do with the fact that I used ! as a label indictor for GOTO and GOSUB, but I haven't fathomed this out. Anyway, who in his right mind would write '?A!A'? Actually it came about because of an oddity I noticed: ? 45 II 4 gives 41. ? 43 II 4 gives 47. What is 65 II 3, and why?

The answer is 66; ASC("II")=124, the token for XOR!

Note: The OVER routines were taken from an early version of Xtal BASIC (when the routine was slightly modified, for the later version, you can imagine my horror when I discovered that the CALLs had changed addresses by a few locations!). If you want to use OVER with later versions of Xtal, change 8889 to 8900 (addresses 16654 and 16684), 697 to 699 (address 16657), 13692 to 9924 (address 16687), and 1112 to 11093 (address 16690).

NEW
How To Make Money From Your Software
LEGAL AND BUSINESS ASPECTS
ANNE STAINES LL.M. Barrister
A GUIDE TO
— The legal protection of computer software
— Confidential information
— Piracy and counterfeiting
— Trade marks and 'passing off'
— Your legal rights
— Setting up a business
— Selling through an established software publisher
— Raising finance
— Marketing and selling
— Contracts
— Exports
— Useful addresses
ORDER FORM PAYMENT WITH ORDER
To: ESC Publishing Limited, 25 Beaumont Street, Oxford OX1 2NP
Please supply ___ copy/copies of How to Make Money from Your Software by Anne Staines at £6.75 plus 80p post and packing
Name
Address
ESC Publishing Limited Co. Reg. No. 1335403
VAT Reg. No. 311 4190 09
PRICE £6.75
Available from bookshops or direct from ESC Publishing Limited

Softlife
A LOW COST EPROM PROGRAMMER FOR THE BBC MICRO
- Programs 2764, 27128, 2764A and 27128A Eproms
- Only connection to User Port on BBC
- Operating software supplied in Eprom – no tape loading problems
- ROM format command allows Basic programs and other files to reside in Eprom
- Fast error-free programming
- Zero insertion-force socket for ease of use
- Compact, sturdy construction
- Comprehensive documentation
- Proven in Scientific, Educational and Industrial environments



£67.00+VAT (inc.p&p)
For technical information contact: Alastair France B.A.
Softlife Ltd 7 Rose Crescent Cambridge CB2 3LL

BLOCK CODE

Dan Jonsson

The problem with most computer languages is that they don't discourage, and sometimes even encourage, bad programming habits. Here is a proposal for a new language that makes the programmer behave — and even if it's never implemented as a language, it can be used as a pseudocode to describe algorithms.

The simplest logical structure of a program is a *sequential* arrangement of operations. A simple program, shown below, to point out the square root of a number illustrates this: in this program, the four operations in lines 10 to 40 are executed one after the other, ie sequentially, and then the program ends.

```
10 PRINT "PLEASE INPUT A POSITIVE NUMBER"
20 INPUT X
30 LET Y=SQRT(X)
40 PRINT "SQUARE-ROOT OF ";X;"=";Y
50 END
```

The sequential structure is not the only logical structure available, however. The complexity, flexibility and power of computer programs derive to a large extent from the appropriate use of non-sequential, *conditional* and *iterative,* execution of statements. Facilities for specifying conditional and iterative execution of statements are available in all full-blown programming languages, but in more or less convenient forms.

It is possible to fully implement all kinds of conditional and iterative control structures by means of *jump instructions,* such as **goto** and **if-goto,** after all, this is the only means available for expressing program structure in machine code.

Problems arise when this primitive form of instruction is used directly by the programmer. The basic problem with jump instructions is that they are *too* general and flexible, and leave the programmer too much freedom to construct convoluted, obscure, error-prone, and hard-to-modify code. For these and other reasons, programming languages have been developed which are intended for so-called *structured programming,* where jump instructions are eliminated in favour of structured control structures such as **if-then-else, while-do,** and **repeat-until.**

Structured languages include Pascal, Modula-2, Simula, ADA, C, PL/I, RATFOR, COMAL and others. Since Pascal is the most well-known among these, it will be taken as a representative of structured languages in general in the discussion below. Control structures commonly used in Pascal include **if-then, if-then-else, while-do, repeat-until,** and **for-do.** In addition, Pascal-like languages possess facilities for creating compound statements by bracketing two or more statements with **begin-end,** curly brackets, etc.

In sum, there are four common ways of specifying program structure:

- By the sequential arrangement of statements;
- By jump instructions (**goto, if goto,** etc);
- By explicit control structures (**ifothen-else, while-do,** etc);
- By statement brackets **(begin-end,** etc).

It is worth noting that all four methods of specifying program structure are used in Pascal. Explicit control structures have not really replaced **goto** statements, but have merely been added to the language. In addition, the control structures offered sometimes require **begin-end** statement brackets to be added. This multiplication of features may indicate that the approach taken is not ideal.

One reason for the retainment of **goto** statements (and/or 'structured' jump instructions, eg **break, exit, quit,** etc) seems to be the well-founded view that Pascal-type control structures are not by themselves sufficiently powerful to provide convenient representations of program structures required to solve many moderately complex programming problems. Exiting from the interior of loops and exiting from program segments on error conditions may be cited as two problems that cannot be comfortably handled by means of only Pascal-type control structures.

A NEW STRUCTURE

For some time, I have been engaged in an attempt to develop a new, powerful type of control structure. The qualities I have been looking for are versatility, symmetry, coherence, simplicity, and conciseness — quite a long shopping list, but the 'psuedo-language' presented in this article, **blockcode,** seems to possess these virtues to a great extent. The proposed way of specifying program structure works as follows:

1. Each program contains one or more *statements*. Statements may take the form of compound statements, ie, *statement blocks*. Since statements blocks are themselves statements, they may be nested within one another.

2. Each statement block starts with a *front statement* followed by a colon (:) and then one or more *internal statements* followed by a full stop(.).

3. Front statements start with an optional **do, also** and **else,** followed by an optional condition part starting with **if**. Front statements may thus have eight different forms:

	empty front statement
if	simple conditional statement
do	unconditional **do** statement
do if <cond>	conditional **do** statement
also	unconditional **also** statement
also if <cond>	conditional **also** statement
else	unconditional **else** statement
else if <cond>	conditional **else**-statement

The following simple example of a statement block corresponds to an **if-then** statement:

```
if x<>y:
  x=(x+y)/2
  y=x.
```

4. Control structures are specified not only by front statements but also by the special internal statement **repeat**. The blockcode below, for instance, corresponds to a **repeat-until** structure:

```
do:
  x=x/2.
if x>y: repeat.
```

5. To improve the readability of program text, **so** and **skip** may be used to indicate the location of empty statements.

6. If a program line contains m colons (:) and n full stops (.), then the left margin for succeeding program lines is moved m-n steps to the right if m>n, and n-m steps to the left if m<n.

The last rule generates an automatic indentation of program lines that reflects the logical (hierarchical) structure of the program. Two important concepts, which in reality concern the logical structure of programs, can be defined in terms of the pattern of indentation of program lines. First, the indentation of a statement equals m-n, where m is again the number of colons preceding that statement and n is the number of full stops preceding it. According to the rule just stated, the quantity m-n is reflected in the indentation of the program line containing the statement except when statements on the same line are separated by a colon or a full stop.

Second, if two equally indented statements are *not* separated by a *less* indented statement, they are said to be *connected*. In the program segment below, for instance, the indentation of statements a and c is equal to 0, while the indentation of b, d and e is equal to 1. Two pairs of equally indented statements — a and c, and d and e — are connected, while two other pairs — b and d, and b and e — are separated by c.

```
(a)   if K>0:
(b)     K=K/2.
(c)   if i>K:
(d)     i=i/2
(e)     K=2*K.
```

7. Each **also** statement and each **else** statement must be preceded by an equally indented, connected conditional statement.

8. Each **repeat** statement must be preceded by a less indented **do** statement.

The rationale for 7. should be fairly obvious: **also** and **else** both presuppose an earlier contingency which either did or did not occur. Similarly, the rationale for 8. is that a **do** statement is required to delimit the set of statements that may be repeated.

CONTROL OF EXECUTION

The above rules concerned how to construct synatactically correct statement blocks. Let us now turn to rules specifying how to control the execution of statements in statement blocks by means of front statements and **repeat** statements.

9. Unless otherwise specified in the rules below, statements are executed in sequential order, starting with the first statement in the program.

10. When a front statement S has been executed, execution will continue with the next statement in the program in eight cases:
(a) if S is an empty front statement, or a simple conditional statement with a **true** condition;
(b) if S is an unconditional **do** statement, or a conditional **do** statement with a **true** condition;
(c) if S is an unconditional **also** statement, or a conditional **also** statement with a **true** condition, and the most recently tested condition in a connected front statement was **true**;
(d) if S is an unconditional **else** statement, or a conditional **else** statement with a **true** condition, and the most recently tested condition in a connected front statement was **false**.
In all other cases, the next statement that is not more indented than S will be executed next.

11. When a conditional front statement S is executed, its condition will not be tested in two cases:
(a) if S is an **also** statement and the most recently tested condition in a connected front statement was **false;**
(b) if S is an **else** statement and the most recently tested condition in a connected front statement was **true**.
In all other cases, the condition will be tested.
12. When a **repeat** statement has been executed, execution will continue with the most recently executed less indented **do** statement.

SOME EXAMPLES

Segments of blockcode corresponding to **if-then-else, while-do** and **for-do** control structures in Pascal are exhibited in Listing 1. Also shown are the simple, straightforward blockcode representations of two structures that strain Pascal's powers of expression: a loop with a **premature exit** (requiring an awkward nested **if-then-else** construction).

The representation in terms of blockcode of the program segments in Listing 1 is almost trivial. To translate the BASIC code in Listing 2 into blockcode poses a much greater challenge. This is a merge sort routine (published in Computing Today, July 1984) containing no less than 14 **goto** statements. If it proves possible to translate Listing 2 into structured blockcode with no **goto**s at all, then you will probably agree that something has indeed been proved. (If that's not enough, I suggest that you try to translate the routine into Pascal, with no **goto**s. By the way, it is no fault to use **goto**s in the BASIC code, since the control structures of BASIC are at best the same as those of Pascal.)

It will be instructive to translate the BASIC code in Listing 2 step by step. The strategy used is to succesively indentify and code statement blocks, proceeding from larger to smaller blocks, filling in more and more details.

Starting with the general structure of the routine, it consists of a couple of initial statements plus a main loop:

```
FB=0; PRINT "Sorting"
DO:
  (lines 650-860)
  FB=FB+1; PRINT FB.
IF C(0)<>0.
  REPEAT.
```

Analysing lines 650-860, we find that they contain three major blockcode segments interleaved with some initiating statements. We thus obtain a somewhat more detailed picture of the structure:

```
FB=0; PRINT "Sorting"
DO:
  NA=0; NB=0; NC=0; FA=0
  (lines 650-660)
  (lines 670-750)
  NA=0; NB=0; NC=0; HA=0
  (lines (770-860)
  FB=FB+1; PRINT FB.
IF C(0)<>0:
  REPEAT.
```

Lines 650-660 contain a for loop, which is easily translated:

```
X=0; DO IF X<=1000:
       B(X)=0; C(X)=0.
ALSO: X=X+1; REPEAT.
```

Lines 670-750 represent a considerably more complicated loop. It is nevertheless a fairly straightforward task to write down the corresponding blockcode:

```
DO:
  HA(A)=A(NA)
IF FA=0:
  B(NB)=HA; NB=NB+1.
ELSE:
  C(NC)=HA; NC=NC+1.
SO:
  NA=NA+1.
IF NA<N:
  SKIP.
ELSE IF HA <=A (NA):
  REPEAT.
ELSE:
  FA=(FA+1) AND 1
  REPEAT.
```

Lines 770-860 consitute a quite complicated decision structure, and there are several ways of translating it into blockcode. As there is not sufficient space to go into the intricate questions involved here, I present the representation chosen without further comments:

```
IF B(NB)=0:
  H(A)=C(NC); NC=NC+1.
ELSE IF C(NC)=0:
  H(A)=B(NB); NB=NB+1.
ELSE IF C(NC)>=HA AND B(NB) <HA))
       OR (C(NC)>=HA AND B(NB)>C(NC))
       OR (B(NB)<HA AND B(NB)>C(NC)):
  H(A)=C(NC); NC=NC+1.
ELSE:
  H(A) = B(NB); NB=NB+1.
```

When the above blockcode segments are inserted at their proper places in the program structure, the blockcode in listing 3 is obtained.

Listing 1.

PASCAL

BLOCKCODE

if--then--else

```
if n>1 then
  begin
    f:=n*f;
    n:=n-1
  end
else
  g:=n*f;
```

```
if n>1:
  f=n*f
  n=n-1.
else:
  g=n*f.
```

while--do

```
while x<y do begin
  x:=2*x;
  y:=y+1
end;
```

```
do if x<y:
  x=2*x
  y=y+1.
also: repeat.
```

for--do

```
for k:=1 to 100 do begin
  B(k):=A(k)+d;
  d:=d+A(K)
end;
```

```
k=1; do if k<=100:
  B(k)=A(k)+d
  d=d+A(k).
also: k=k+1; repeat.
```

loop with 'exit'

```
more:=true;
while more do begin
  x:=x+5;
  if x mod 3=0 then more:=false;
  if more then
    begin
      x:=x-2;
      n:=n+1
    end
end;
```

```
do:
  x=x+5.
if x mod 3<>0:
  x=x-2
  n=n+1
  repeat.
```

'exit-upon-error'

```
read(x);
if x<>0 then
  begin
    process(x);
    read(y);
    if y<>0 then
      begin
        process(y);
        read(z);
        if z<>0 then
          process(z)
        else
          error
      end
    else
      error
  end
else
  error;
```

```
do:
  read(x).
if x<>0:
  process(x)
  read(y).
also if y<>0:
  process(y)
  read(z).
also if z<>0:
  process(z).
else:
  error.
```

ADVANTAGES AND SIGNIFICANCE

Among the advantages of blockcode, the first and foremost is its power and versatility: it can be used to emulate all common control structures and then some. This means that the elimination of **goto** statements should be a practical proposition at last. The automatic indentation of program lines in a way that reflects the logical structure of the program is another important advantage, especially for 'top-down' programming. The use of **also** is another innovation, and I would like to draw the attention to the elegant symmetry between **else** and **also**. On the whole, I feel that the economy and elegance of blockcode speak strongly in its favour.

Now, while intellectually blockcode may be a success, what are its practical uses and practical significance? One application that comes immediately to mind is to construct a compiler or interpreter for a blockcode-type programmmig language. This should not be too difficult; for instance, a stack may be used to keep track of truth values of previously tested conditions on different levels of indentation and/or addresses of previous **do** statements on different levels of indentation .

Such a language may or may not arrive some day, but in any case it is possible to use a blockcode-type pseudocode to describe algorithms and to design programs. The next step would then be to translate the pseudocode into a suitable programming language. It is actually quite simple to represent blockcode-type structures by means of **goto** statements. This translation could be preformed by a programmer, or automatically by a separate program, a 'pre-compiler'. (As stated in the introduction, it is only when **goto**s are used 'freely' that they may be considered harmful. It is quite legitimate to use them to represent a precisely delimited set of control structures in accordance with precisely specified rules.)

A hidden virtue of blockcode may be pointed out in this connection: data streams 'consumed' or 'produced' by a computer process may be described in terms of blockcode-type 'data programs'. 'Input-output-oriented' structured programming (eg, 'Jackson Structured Programming' and 'Warnier Orr Programming'), where program structure basically constitutes an extension of the structure of input and/or output data, requires a common formalism that satisfies this requirement of dual applicability.

Finally, while common flowchart are good for providing visual representations of code containing **goto** statements, they are not suitable tools for 'top-down' structured programming. The logical structure of a program written in blockcode can be visualized, however, by means of *blockcharts*, a kind of 'structured flowchart'. I shall make no attempt, though, to present blockcharts here — they deserve a separate article.

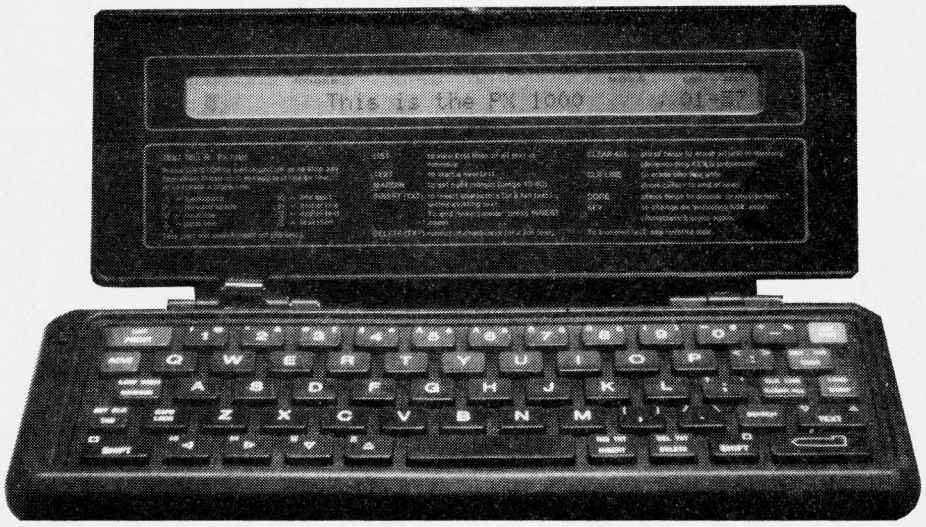
Listing 2.

```
640 FB=0:PRINT "Sorting"
650 NA=0:NB=0:NC=0:FA=0
660 FOR X=0 TO 1000:B(X)=0:C(X)=0:NEXT X
670 HA=A(NA)
680 IF FA=0 THEN 710
690 C(NC)=HA:NC=NC+1
700 GOTO 720
710 B(NB)=HA:NB=NB+1
720 NA=NA+1:IF NA>N THEN 760
730 IF HA<=A(NA) THEN 670
740 FA=(FA+1) AND 1
750 GOTO 670
760 NA=0:NB=0:NC=0:HA=0
770 IF B(NB)=0 THEN 840
780 IF C(NC)=0 THEN 830
790 IF B(NB)>=HA AND C(NC)>=HA THEN 820
800 IF B(NB)>=HA THEN 830
810 IF C(NC)>=HA THEN 840
820 IF B(NB)>C(NC) THEN 840
830 HA=B(NB):NB=NB+1:GOTO 850
840 HA=C(NC):NC=NC+1
850 A(NA)=HA:NA=NA+1
860 IF NA<=N THEN 770
870 FB=FB+1:PRINT FB:IF C(0)=0 THEN RETURN
880 GOTO 650
```

Listing 3.

```
FB=0; PRINT "Sorting"
DO:
  NA=0; NB=0; NC=0; FA=0
  X=0; DO IF X<=1000:
    B(X)=0; C(X)=0.
  ALSO: X=X+1; REPEAT.
  DO:
    H(A)=A(NA).
  IF FA=0:
    B(NB)=HA; NB=NB+1.
  ELSE:
    C(NC)=HA; NC=NC+1.
  SO:
    NA=NA+1.
  IF NA<N:
    SKIP.
  ELSE IF HA<=A(NA):
    REPEAT.
  ELSE:
    FA=(FA+1) AND 1
    REPEAT.
  NA=0; NB=0; NC=0; HA=0
  IF B(NB)=0:
    H(A)=C(NC); NC=NC+1.
  ELSE IF C(NC)=0:
    H(A)=B(NB); NB=NB+1.
  ELSE IF (C(NC)>=HA AND B(NB)<HA)
       OR (C(NC)>=HA AND B(NB)>C(NC))
       OR (B(NB)<HA AND B(NB)>C(NC)):
    H(A)=C(NC); NC=NC+1.
  ELSE:
    H(A)=B(NB); NB=NB+1.
  FB=FB+1; PRINT FB.
IF C(0)<>0:
  REPEAT.
```

A MICRO MANQUÉ



C. Straw

For readers who flunked their French 'O' Levels, *manqué* means 'what might have been but is not'. The Text Tell PX1000 pocket text processor *might* have been a portable micro, but it isn't.

To be completely honest, when I first set eyes on the sub-miniature PX1000, the first thought to enter my head was 'hey, this is just what I've been waiting for'. For some time I had toyed with the idea of risking an overdraft on a portable micro-come-wordprocessor. Foremost in my mind was the Epson PX-8 (reviewed CT November '84 — Ed). I had the fortune of attending the launch of this little beast and fell in love with it immediately. However, a price tag of almost £1000 was way beyond the limits of my quite shallow pockets. The PX-8 appealed for several very good reasons, one being its very good looks (so important for the self-conscious scribbler) which gave it a definite edge on its rather too functionally-styled brother, the HX20. I soon came to terms with the fact that I'd never be a PX-8 owner, so it was with considerable enthusiasm that I siezed the chance to play with the PX1000. But could I see myself paying £655 to become the *owner* of a machine capable of nothing more than word-processing?

WHAT YOU GET

For £655 you get:

- Text Tell PX1000 (in a soft, mock leather case looking not unlike that of an electric razor)
- an AC/DC power supply/charger
- Twin core cable for dumping and loading text to and from tape
- three core cable for attachment to serial printer
- 20 page instruction manual

For the purposes of this review: we were also supplied with the optional Text Tell PXP thermal printer. Unfortunately, we were not supplied with either of the two cables which made it impossible to test the tape storage prowess of the machine or its capacity for sending data down an RS232 line, so I am unable to comment on its performance in these two areas.

The one thing you immediately become aware of with the PX1000 is the beauty and neatness of its design. The black satin finish of the case is both functional and pleasing to the eye. The keyboard is extended QWERTY with two levels of shift on the upper (numeric) row (see photo). Certain related keys are colour coded to indicate a similarity in the functions they perform. For example, the standard alpha-numeric set is in black, whilst all keys concerned with outputting data to and from the display (PRINT and LIST), tape (LOAD/SAVE), and other remote units via the built-in modem/acoustic coupler, are coloured grey. All wordprocessor function keys, ie. cursor up/down/left/right are chocolate-brown. This is more useful than might be obvious at first, since the one-centimetre square keys are quite closely arranged and with such a small keyboard it is important for the user to descriminate between keys without undue fuss.

As is almost standard with portables with built-in displays, the hinged lid of the unit contains the 40 column by one row LCD display. The area just below the display is occupied by a potted guide to using the machine, giving details of all the word-processing functions and commands.

A yellow key situated at the top right-hand corner of the keyboard switches the display on, and also acts as a brake if an operation (ie. transmission of data) is to be terminated before reaching the end of its natural cycle. The display switches itself off automatically after 50 seconds, which can be a little annoying if you are a contemplative writer — I regularly found myself switching the machine back on after a brief spell of day-dreaming — but this does help to conserve precious energy, stored, incidentally, in NiCad cells permanently fixed inside the case.

WORDPROCESSING

The PX1000 has a fairly standard set of wordprocessing functions. Text entry is quite normal, and in this sense the machine is much like a typewriter, but for the barely audible 'beep' as keys are pressed, and the carriage return/line-feed issued at the end of each line.

As has already been mentioned, the wordprocessing functions are accessed via the chocolate-brown keys to the left and right of the alpha-numeric keys. Moving anti-clockwise from the top-left, with each key's shifted function in brackets, the functions are as follows:

PRINT (LIST)	PRINT outputs an entire text 'block' to the printer. Text is stored in blocks, numbered 1 to 99. A block can store a maximum of 7500 characters (7500 characters is the machine's total capacity). As text is being sent to the printer, the words 'Please wait' are displayed informing the user that the machine is temporarily tied-up with an operation. Printing can be prematurely terminated by pressing the yellow ON (STOP) key, but as often happens, the printer continues to churn out everything inside the machine's printer buffer — something that never ceases to amaze me! LIST simply prints-out the first line of every occupied block to remind the user of what is currently held in the PX1000's memory.
RCVE.	This function puts the PX1000 into readiness for the reception of data via its acoustic coupler. Once RCVE is pressed, the 'Ready to receive' prompt is displayed, along with the machine's spare capacity. The manual gives full information on the use of this function, but we were unable to test it without a second unit.
MARGIN (LOW/HIGH)	The MARGIN key allows positioning of the right margin. The default column width is 40 characters, identical to the LCD display length. Columns up to 80 characters wide are permitted, with a lower limit of ten characters. LOW/HIGH sets the baud rate for data transmission via the modem/acoustic coupler. Three baud rates are supported; 300, 600 (V23) and 1200 baud.
TAB (SET/CLR)	Tab stops are initially set at every eighth character. Additional tab stops can be set and cleared by successive actions of the SET/CLR key.
INSERT (INS TXT)	Text can be entered and edited in either the Insert or Overwrite mode, INS TXT permits text from other blocks to be inserted into the current document — a sort of down-market Cut and Paste.
DELETE (DEL TXT)	Moves the cursor back one space and deletes the character in that position. DEL TXT deletes all of the text within the currently selected block. Since this is a fairly drastic measure, you *do* get an opportunity to change your mind, if you so wish . . .
SEARCH	The fairly standard wordprocessing search facility is available with the PX1000. Hit the SEARCH key and the prompt 'SEARCH FOR + PRESS AGAIN' appears on the display. The search string is limited to eight characters only, but this should be enough to ferret-out most words.
TEXT	The TEXT key allows the user to select the current text block. Text blocks can be skipped through in either direction (ie from 8 to 99 or from 8 to 1), and this is achieved by pressing the left or right shift keys in conjunction with the TEXT key.
CLEAR ALL (CLEAR LINE)	Means what it says! Pressing CLEAR ALL twice removes everything from store — obviously a feature to be used with caution! (CLEAR LINE) deletes all text after the cursor on the current line.
DUMP (LOAD)	DUMP puts the machine in readiness to commit text to tape. LOAD prepares the machine to receive text from tape.
CALC (SET)	The PX1000 can also be used as a rudimentary calculator. All of the usual arithmetic operators are present ,BASIC style, and once an expression is entered, followed by a trailing '=', a quick press of the CALC key gives the solution correct to a maximum of 4 decimal places. (SET) defines the depth of indent for solutions given by CALC. (SET) also permits the user to define the maximum number of decimal places displayed for results of CALCed solutions.

IN USE

Although the PX1000 is a pleasure to use, it was slightly worrying to note the speed with which precious storage space was consumed as text was entered. Four small boxes at the top of the display indicate the total amount of store remaining, each box representing 25% of the 7500 character memory. Each time 25% of the store is filled, a box is erased until finally the ***MEMORY FULL*** message is displayed. Colleagues have argued that 7500 characters isn't really enough for 'serious' applications, but by successively dumping portions of a text to tape this problem can be overcome, albeit rather clumsily.

My only real cause for anxiety was the PX1000's annoying habit of switching out of the INSERT mode whilst text was being entered. I often found myself overwriting text which should have been opening up to accept additional copy. Presumably the designers reckoned on Overwrite being the most popular mode, but I think the choice would have been better left up to the user.

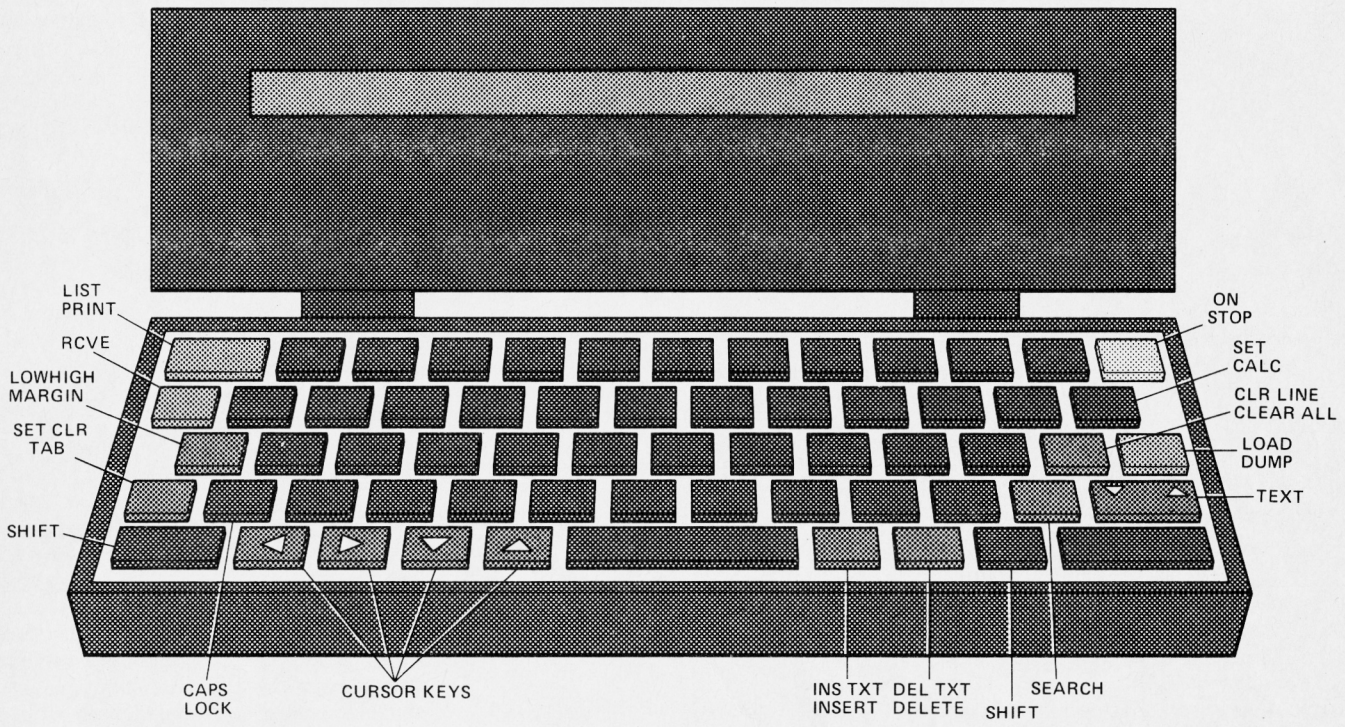
DOCUMENTATION

The 20 page manual which accompanies the PX1000 certainly contains all the information necessary to use the machine properly. However, the layout of the individual sections didn't inspire me at all. Large banner headings are used throughout which at first seems quite encouraging — at least you know where to look for your answers. But the descriptions are so densely packed that occasionally I wondered whether I was still reading the section appropriate to my enquiry. This being said, I have seen a lot worse in my time, but for £655 I would have expected something a little more professionally produced.

FACTSHEET	**Text Tell PX1000**
Price:	£655
Available from:	ACS International Ltd. 175 London Road Camberley Surrey GU15 3JS Tel. (0276) 24434

USERS

The press information issued with the system claims: "The PX1000's potential market is enormous — from the obvious applications that sales executives will find...", (a curious choice), "...to other professionals like journalists, accountants and solicitors." Whilst I would certainly agree that journalists, and perhaps accountants, might certainly make use of a portable text processor, experience has shown that there are none more unwilling to touch a keyboard than those in the legal profession! As for the enormity of the potential market, it rather depends on one's concept of 'enormous' — I would argue that a portable dedicated text processor costing £650+, certainly has potential, but 'enormous' it ain't! This being said, the market for the PX1000 should be obvious, appealing to anyone who needs a portable wordprocessor and who can afford to pay for the privilege. I should make the point that I would certainly purchase a PX1000 if *only* it wasn't a *dedicated* wordprocessor. The PX1000 would have been a better and more attractive product if the wordprocessor was an option in cartridge form. Admittedly, the machine's single line display is useful for little other than wordprocessing. As it is, the machine gives the impression of being a sadly frustrated microcomputer.



AMSTRAD INTERFACES

THIS IS NOT JUST A MODEM, BUT A COMPLETE SYSTEM. NOTHING ELSE TO BUY

★★ MODEM ★★
★ £153.00 ★

Incorporating serial and parallel interfaces, to allow software control of all functions, each feature controlled from basic with the bar commands. Call from m/c or on entering bar modem all controls are menu driven for ease of use, bell/ccitt standards 300/300 600 1200 1200/75 75/1200 full and half duplex. Auto dial and auto answer contact bulletin boards, prestel compatible, software buletin on its own sideways Rom. Unique panel display, it displays what the modem is doing, mode of operation, and digits when auto dialing, standard B.T. plug connector. *Note this modem is not B.T. approved.*

★★ SIDEWAYS ROM ★★
★ £26.05 ★

The unit holds 4 Roms. Each can be 2, 4, 8 or 16K in size incorporating a device to allow slower Roms to be used less than Amstrad suggested 200, that means cheaper Roms. free utility Rom with every unit.

RS232
Communicate with your modem
Talk to other computers
Use serial printers
Split Baud rates
Standard 25 way 'D' connector
£39.96

PARALLEL PORT
Make that Robot move
Control electrical appliance
Twin 8 bit ports
Operates direct from basic
2 x 14 way speedblock connectors
£22.57

8 BIT PRINTER PORT
Make use of that 8 bit printer
Allows character codes
Above 127 (ie 0 to 255)
Plugs in between centronics
Port and printer cable
£17.35

All units are cased and have through connectors
★ Please add VAT ★

COMPUTER HARDWARE & SOFTWARE

KDS ELECTRONICS

15 Hill Street, Hunstanton, Norfolk PE36 5BS
Tel: (04853) 2076

Dept CT

For help with Reading and Writing

01-405 4017

Since 1975, 350,000 adults have been helped to read and write better.
If you want help look for this sign.

For further information
Adult Literacy & Basic Skills Unit
PO Box 213 London WC1V 7ET

WARGAMES AND COMPUTERS

Owen and Audrey Bishop

Part one of a series charting the progress of modern computer wargaming.

People have been playing wargames for thousands of years. They have been played for amusement, or for the more serious purposes of military training, and the planning of military manoeuvers. Alexander the Great, for example, planned his campaigns using models of the terrain, with tokens to represent troops.

The game of Chess can be considered to be an extremely formalised wargame. The origin of Chess is obscure, but most authorities consider it to have been developed in India, several centuries ago. The game, then called *chaturanga*, represented an imaginary battle between Indian armies, with the pieces representing elephants, horses, chariots and infantry *(see front cover — Ed)*. The game differed from modern Chess in several ways. Moves were governed by the throw of dice. Today too, the use of dice (or their computerised equivalent) plays an essential part in simulating the effects of chance (or providence?) on the battlefield. But in the 6th century the strict application of the Moslem laws forbidding gambling caused the use of dice to be dropped from the rules of *chaturanga*. Chess, the descendent of this game, thus has no element of chance in it, making it essentially different from a true wargame.

Another ancient wargame is *Go*. It was being played in China over two thousand years ago. Masters of the game were highly respected and given titles such as *sei* (or Holy Man). It spread to Japan. At the Military Academy in Japan, *Go* was a compulsory study until the 17th century. The game simulates the capturing of enemy-occupied territory by encircling it, and thus still has a strong military flavour today.

More recent developments of wargaming began in the 18th Century, when the map on which the game was played was marked with a grid. There were tokens to represent the forces. These were of diffferent kinds to represent different types of military unit. Each type had a fixed movement allowance to represent speed with which the real unit would be able to move over the terrain.

One of the best known of the relatively recent wargames was invented in 1824 by B.



von Reisswitz, a lieutenant of the Prussian Army and his father. By order of the Prince Wilhelm, this game, known as *Kreigspeil*, was adopted for military training in every regiment of the Prussian Army. The game was played on an ungridded map, and various designs of token were used to represent the troops. The rule-book was a complex one, which is inevitable if a realistic simulation of warfare is to be achieved. In the early forms of the game, dice were used to simulate chance happenings. In later forms, more use was made of experienced umpires to decide the outcome of combat. Had computers been available then, they would undoubtably have had a part to play in the conduct of the game. Von Reisswitz's book on *Kriegspeil*, has recently been translated into English, and is available with a simple kit for playing the game (see address given later).

Wargaming is used today by the greater (and lesser?) powers for military training and planning. Although many people consider that wargaming encourages aggressiveness between nations, there are others, including ourselves, who believe that it has exactly the opposite effect. There is no more effective way than wargaming, of experiencing the futility of war, and being made to realize that in any war *both* sides suffer severe loses. Victory is rarely easy — or certain!

WHAT IS A WARGAME?

The short answer to this is that it is a simulation of warfare. The maps, tokens, and rules represent the battlefield, the troops and the way they would behave in battle. Whether one considers a wargame to be a game in the sense of having entertainment value, is a matter of opinion. In the sense of being a simulated contest played according to rules, with possibly a random element, it is definitely a game.

The popularity of wargaming as a hobby is evidence that most wargames are entertaining and can also be instructive. Part of this is due to the skill of the wargame designers in picking out the more interesting historical contests, or in devising imaginary but ingenious tactical situations for the combatants to resolve.

WARGAMING TODAY

It may come as a surprise to many home computer enthusiasts to discover that there is a large band of wargamers equally enthusiastic and devoted to wargaming. There are national and regional exhibitions and the public meetings often have long queues for admission. There are 'charts' of the most popular (non-computer) wargames. There are wargaming magazines and wargaming tee-shirts. Wargaming is a flourishing hobby, not only in UK, but in Australia, New Zealand, Europe and the USA.

Games based on warfare appeal strongly to many computer owners, as the recent software popularity charts reveal. Conversly, many wargamers believe that computers could help to improve their war-

gaming. Some wargamers are writing programs for this purpose. Yet there seems to be surprisingly little exchange between these two groups of hobbyists. We hope that this short series will help bridge the gap.

The present widespread interest in wargaming as a hobby began in the nineteen-sixties. Although a number of famous persons such as H.G. Wells and Winston Churchill had played and written about wargames earlier than this, the first definitive book on the subject was Donald Featherstone's *War Games: Battles and Manoeuvers with Model Soldiers*, published in 1962. This followed in the late sixties and seventies by many other books by the same author and by others such as Charles Grant. Excluding the more specialist form, *Kreigspeil*, which is less popular, wargames are of two distinct types, board wargames and model (or figure) wargames. The model wargames are those popularised by Donald Featherstone and others. The games are usually designed by individuals or groups of enthusiasts, perhaps published in magazines, and mostly played by club members. A great deal of the activity in model wargaming is at club or 'home' level. The situation is similar to that of the home computer owner who individually or with a group of friends writes games programs for personal amusement and occasionally sends them to a computer magazine for wider dissemination.

By contrast, board wargames are produced by specialist publishing firms and are usually designed and play-tested by teams of professional wargames designers. This situation parallels the publication of computer software by large software houses. Like software, board wargames may occasionally have 'bugs' in their rules!

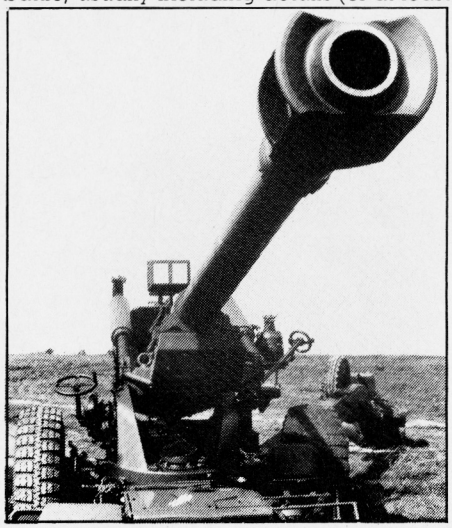
MODEL WARGAMES

The model wargame is played with model soldiers on a table which is usually set out with models of buildings, trees, bridges and other features to represent the terrain of the battlefield. Various scales have to be decided upon before play begins. The scale of the terrain is one obvious factor. Another is the time-scale — what span of time is represented by one game-turn?. Depending on the nature of the game, a turn may represent anything from 5 minutes to a week of military activity. Finally there are the model scales. These present difficulties, because (except in skirmishes involving only a few soldiers) models made to the same scale as the terrain would be too small to handle. Furthermore, few people could afford to buy a million or more model soldiers to field in a major campaign! The solution is that the model soldiers used in wargaming are, in effect, tokens for a larger number of men, This is not to say that gamers would contemplate replacing them with crude blocks of wood. The models are often masterpieces of the model-painter's art.

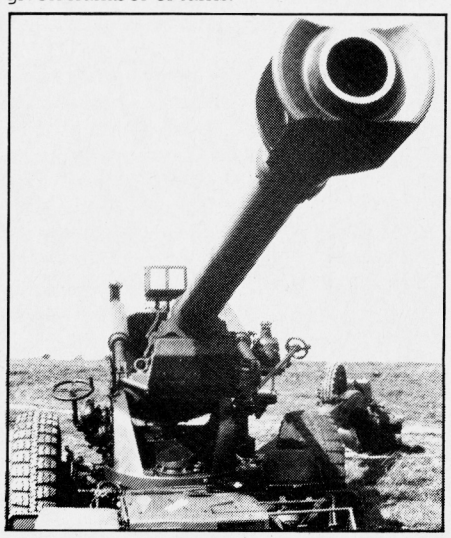
Indeed, modelling soldiers and military vehicles, either from scratch or by modifying pre-cast models, and then painting them with their correct colours and insignia, is a major aspect of model wargaming. To some it is an end in itself. To many it adds immense attraction to the game. The computer is not likely to have much impact on this is creative and visually appealing side of wargaming.

SCENARIO

Every wargame is based on a *scenario*. This is a description of the event leading up to the battle, usually including details (or at least



an outline) of the units taking part and possibly their disposition on the battlefield. If the game is based on an historical battle, much research in history books and other documents will have (or should have!) been undertaken by the person designing the game. The scenario will usually prescribe any special rules for the game, including rules to determine which side has won. The winner is not necessarily the player who kills or captures most of the enemy, or manages to drive the enemy from key strongpoints. In some games a player with smaller and weaker forces may be considered to be the winner if he or she prevents the enemy from advancing beyond a certain point for a given number of turns.



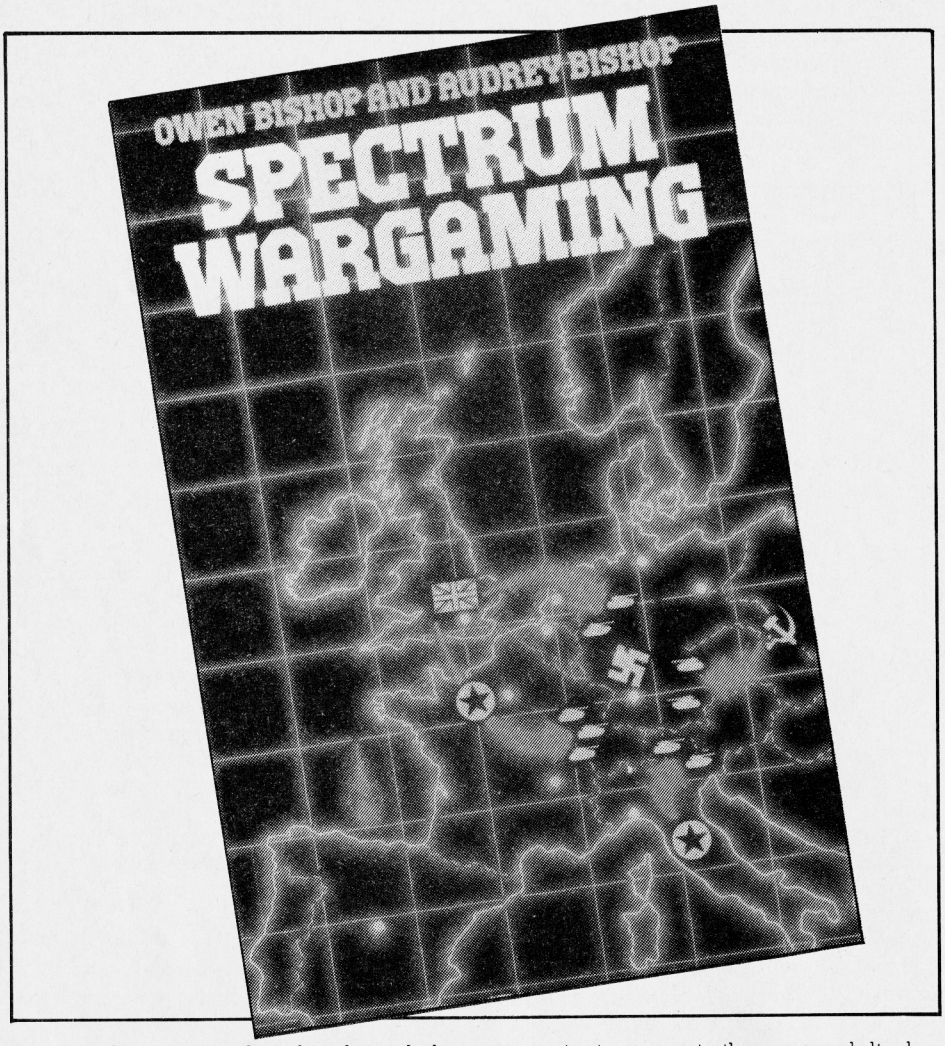
Play is also governed by a set of general rules applicable to the period in which the game is set. There are published sets of rules for the Napoleonic period, for example. These cover the composition of the armies, the rate at which units may move in different types of terrain, and the effectiveness of their weapons .Many of the rules are in tabular form, for example tables of movement allowances (distances) per turn for each type of unit. These distances may be modified if, for example, the unit concerned is wounded, or carrying heavy equipment. Other tables are concerned with the results of combat. These are based as closely as possible on actual field experience with the weapons. In the cinema, the super-sleuth usually manages to shoot the enemy agent dead with a hand-held revolver at a range of several metres. In practice, a revolver is a very unreliable weapon except at point-blank range. Combat resolution tables are based on facts such as this, not on fiction. Similarly, the accuracy of rifle fire decreases with increasing range, and depends on the weapons skill and the mental and physical condition of the soldier firing it. All these factors, and more, are taken into account in the combat resolution tables. In addition there is a random element, usually decided by the throw of dice, to take account of unexpected or unknowable effects that might influence the final result of firing. A chance cloud of smoke or dust might blow across the line of sight just as the trigger is pulled, a gust of wind or the blast from an exploding shell might take the soldier off-balance; or there are just the random inaccuracies inherent in any man-machine combination. As we shall see, the computer can be of great help in this side of wargaming.

It is clear from the above that wargames rules can be extremely complicated. This puts the game designer in a dilemma. If the rules are over-simplified, the game lacks realism. It fails as a simulation. If the rules are excessively detailed and hedged about by provisos and exceptions, they become almost impossible to learn. The game fails as a game. To compromise and produce an authentic yet playable wargame taxes the skill of the wargamer to the extreme. Fortunately there is a sufficient number of designers able to meet this challenge.

One of the delights of wargaming is that the players are free to make up their own rules. This is providing they both agree to play by the same rules! Most wargamers base their game on one of the published rule-books, modifying them appropriately to the scenario they are playing, and to their own personal preferences.

BOARD WARGAMES

The chief source of these is the United States. Games publishers such as Avalon Hill produce dozens of wargames covering all periods. These games cost between £10 and £30 each, which is about the same as a piece of top quality software. If this seems expensive, remember that a typical wargame takes 2 to 6 hours to play and that many sets contain several different scenarios using the same equipment. There are also magazines, such as *The Wargamer*, which include the map and pieces for a wargame in each issue. These are cheaper than the boxed wargames, and many of the games are just as good, though they lack the 'polish' of the boxed games. For example, the accom-



panying literature is less lavish and the 'board ' is of stiff paper, instead of being mounted on card.

The equipment of a typical boxed wargame includes a folded map mounted on card. The map is coloured to represent different kinds of terrain, and marked with rivers, towns, and other essential features. It is usually overprinted with a grid of hexagons. The hexagon, or 'hex' is the unit of distance. The movement allowance of each kind of military unit and the range of its weapons is expressed in hexes. The military units themselves are represented by coloured squares of cardboard (or counters), printed with designs and legends to identify the units and define their fighting capabilities.

The rule book is specific to that one game, rather than applying generally to the historical period, as in the case of table wargaming. The rule book usually provides for several scenarios to be played with the same equipment. Some of the senarios may be playable with a subset of the rule so that you can begin to play without having to learn the complete rule-book.

Board wargaming lacks the visual appeal of model wargaming but, despite this, can become very absorbing. A well-designed game can fully arouse the imagination and the emotions of the players, even though only cardboard counters are involved. One of the snags of board wargaming is the handling of little piles of cardboard squares. This is something that can be avoided by using a computer to generate the map and display symbols on it to represent the fighting units.

COMPUTERS IN WARGAMING

In the third part of this series we shall present the rules and listing for a computer skirmish wargame set in World War II. Before then we will examine the advantages of using computers in wargaming.

As may be apparent from what has been said above, wargaming, whether with models or on a board, involves many routine tasks. Each unit has to be moved individually, and its fighting status (active, wounded, panicked, killed in action etc) updated every turn. Combat between units that are within range of one another has to be resolved by consulting combat resolution tables. Often this is complicated by the fact that various modifiers have to be taken into account, for example, to allow for whether a unit is protected by being inside a building, or whether it is unable to fight effectively because of being wounded. The rolling of dice to add the required random element to the game is another time-consuming activity. Routines such as these can all be undertaken by the computer, with complete freedom from errors, ommissions or bias. The simulation can be improved too. For example, the estimation of the effect of numerous factors in the fall of an artillery shell involves fairly complex geometry. In order to arrive at a result in a reasonable time, wargamers tend to use rough-and-ready methods to determine the chances of a direct hit. With a computer, an exact estimation, which takes all factors into account can be arrived at in microseconds. Similarly, the traditional combat resolution tables can be represented in the program by algorithms for solving suitable differential equations. In short, the computer can not only take over many of the routine tasks of wargaming, but perform them much better.

In addition to acting as a mediator in model wargaming, the computer can provide the means of playing wargames without models if you wish, thus eliminating the expense of buying models, and the need to find a large table on which to play. In this way, it can bring wargaming to those who might otherwise be deterred.

Wargames are not fast games, like the Arcade-style zapping games. They are games of contemplation and planning. If a single turn represents an hour of real time, there is no need to hurry! So the pace of the games program can be geared to that of the human player contemplating the next move. A consequence of this is that a single game can easily take several hours. It may not be possible to complete a game within a single session. With model wargames this presents a problem unless the players have a club room in which the models can be left undisturbed. Writing down the position and status of each unit, packing the models away, and then setting them out again ready for next session, can waste hours which could be more enjoyably spent in actual play. By contrast, if the game is being played on a computer, all details can be saved to tape or disk in a few seconds and retrieved speedily and reliably at the beginning of the next session. Play recommences almost immediately. If players are using models the displays produced by the computer make it much quicker to replace them in their former positions.

Since the computer holds full details of each unit, it can be programmed to make systematic changes in these details. For example, it can simulate attrition, a factor that is often ignored in model or board wargaming. Soldiers who have been in action for days or weeks suffer hardships that eventually undermine their physical strength. Their morale strength may be reduced too, in that they would by more likely to rout or even desert as a result of a relatively minor setback. Such effects are difficult to take account of (and hence ignored) but are easily simulated by a computer. A program can allow for the effects of fatigue, disease, and starvation without any action on the part of the players. All that the players will notice is that numbers and fighting strength are dwindled, morale is decreasing and there may be wholesale desertions from time to time.

So far we have seen how the computer can take over the housekeeping chores of wargaming, and undertake certain other tasks that are normally just too complicated for the wargamer to manage without holding up play unduly. But the computer can do more than this. there are many aspects of warfare

that are beyond the scope of the relatively simple mechanics of the wargame table or board. Two such aspects are hidden movement and independent action. The computer can simulate these, increasing the realism of the game and bringing entirely new dimensions to wargaming. Three such aspects are hidden movement, independent action and two-computer wargaming. We shall discuss these exciting developments next month.

WARGAME SOFTWARE

Those of you who have read so far and are feeling an urge to try a wargame will be wondering what software is available at present. The first wargames software appeared several years ago. The classic program of this era was *Eastern Front*, written by Chris Crawford for Atari. Since then, other games have appeared, some from the USA published by Avalon Hill for Apple, Atari and Commodore 64. Lothlorien has produced some of the best wargames programs in UK.

Although most of the games that have been published are good games in their own right, and their topic is warfare, many of them lack the features of true wargames. They simulate the battle to only the most limited extent, and the essential tactical element of the game is minimal. This is not necessarily a criticism of their publishers and designers. Many popular microcomputers, particularly those which have been available for several years, lack the memory necessary for a wargame program of even moderate complexity. An appreciable amount of memory is needed to store the details of a varied and interesting terrain. More memory is needed to hold information about the armies. Then we need program space for the mechanics of the game — displaying the maps, moving the units and engaging them in battle. To all this designers usually try to add tactical algorithms which allow the computer to take part in the game as an opponent. This is where the wargame programmer comes up against the shortage of memory. Although there have been many sucessful implementations of Chess in which the computer plays one side, wargames are much more demanding. The terrain is larger and more varied than a chess-board. There are more pieces to deal with. There are more choices of move available to them and permissible moves vary from time to time as play progresses. A program that plays in a realistic and effective way against a human opponent is virtually beyond the capabilities of the present generation of home micros. Frequently, the games designers have sought to make up for this by giving the computer larger and stronger forces than those allotted to the human player. They charge ahead whatever the opposition and, if they win, it is by sheer weight of numbers. In some games a few simple offensive or defensive algorithms may be employed by the computer, but these are usually easy to discover and their weaknesses exploited. There is little satisfaction in outwitting a simple-minded adversary!

If you have no one to play against, the best course is to select the '2 player' option (if there is one) and play against yourself. Although this approach may sound strange to computer-games players, wargamers frequently play this way, either from necessity or for preference. Almost all model and board wargames are primarily designed for two or more players (in fact, we know of only one for a solo player). The complexity of the tactics of a wargame is such that trying to exploit the advantages and minimise the disadvantages of each side in turn is a fascinating and enjoyable exercise. At least you have a worthy opponent! Another type of computer wargame is that often marketed as a 'strategy game'. To understand the nature of this type we must differentiate between tactics and strategy. Tactics dominate most model and board wargames. In some games non-tactical factors such as supply of arms are a consideration, but never play an overriding part. Strategy involves marshalling armies, obtaining and managing supplies, influencing governments and populations. Strategy is often most active before the war begins. The aim of a strategy is often to gain a victory without ever having to fight! Since strategy involves numbers of men, tons of food and ammunition, amounts of money, and numbers of supporting voters in the populace, strategy games are well suited to computers. They are easy to program too! But, as games, they often become a boring session of juggling with numbers in order to discover the winning formula. Some strategy games do attempt to incorporate other aspects of warfare, but most of those we have seen have proved very tame affairs in comparison with the cut and thrust of real wargaming.

The above represents our opinion of the computer software we have tried. We enjoyed the simple tactics of Lothlorien's early and popular wargame, *Johnny Reb*. Of the more recent games. we have been most impressed by *Arnhem*, from Cases Computer Simulations Ltd. This game, for the Spectrum, is the one that most accords with our idea of what a computer wargame ought to be like. It has been criticised because it does not take account of supply problems, which were so crucial in the real battle, but we feel this limitation is due to the computer, not the program. When writing computer wargames, the 'Out of Memory' message appears all too often before the program is completed! Obviously, the ideal micro for wargaming is not yet in the stores. Next month we will examine the current selection of popular machines to see which of those available are the best for wargaming.

REFERENCES

Featherstone, Donald, 1970 *Battles with Model Soldiers* (2nd Edn. 1984, David and Charles)

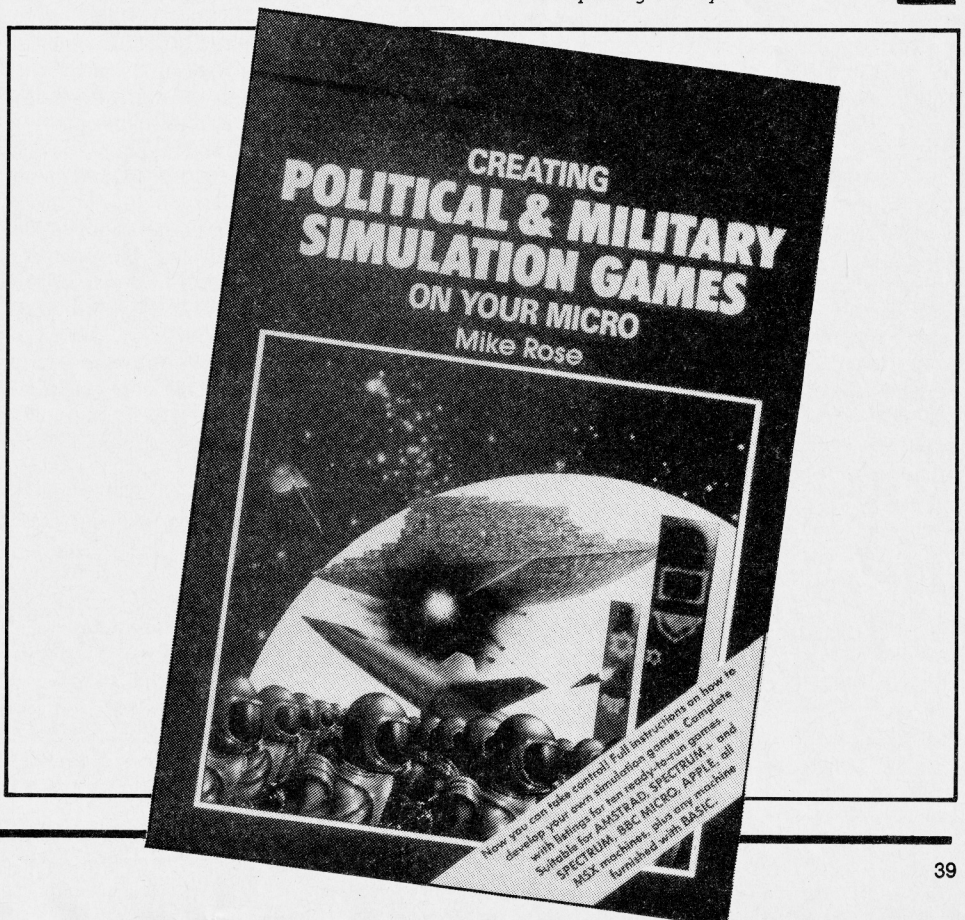
Quarrie, Bruce (Ed.), 1980 *PSL Guide to Wargaming* Patrick Stephens, Cambridge)

von Reisswitz, B., 1824 *Kriegspiel* (Translated and published by Bill Leeson, 5 St Agnells Lane Cottages, Hemel Hempstead, Herts, HP2 7HJ)

A book, *Spectrum Wargaming* by Owen and Audrey Bishop, is available from Collins, price £9.95.

Another book, *Creating Political and Military Simulation Games* by Mike Rose, is also available, from Interface Publications, price £7.95.

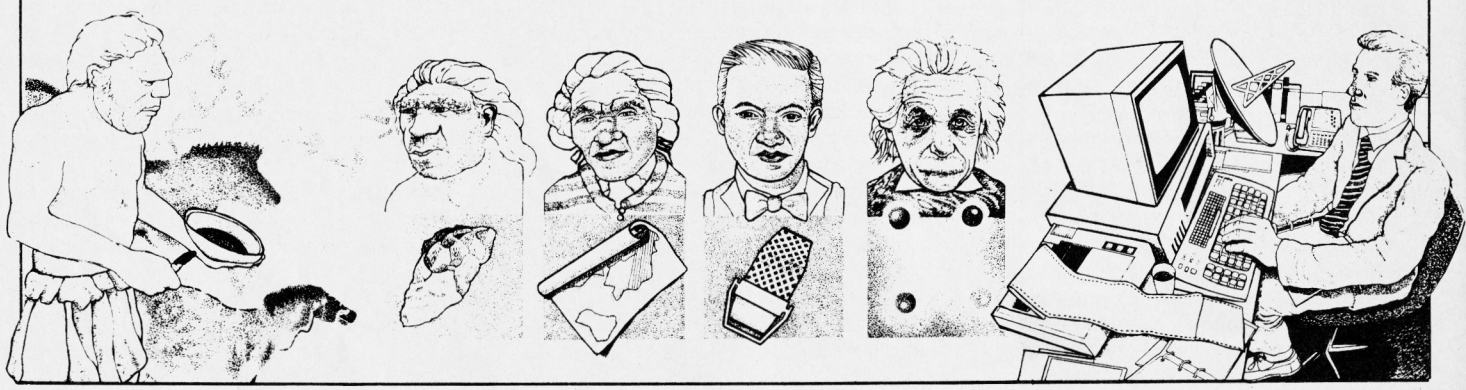
Wargames and Computers continues in the next edition of Computing Today.



LESSONS OF HISTORY

Bill Horne

Part 7: Languages.



For a given hardware system. there is a set of machine code instructions to which the system will respond in a predictable manner. In order to obtain a given series of actions, a series of machine code instructions must be set up in the system store, and this can be a laborious process.

Firstly, there is the need to memorise or tabulate the exact meaning of each instruction. Then, it is necessary to create a skeleton listing of the instructions required. It is not possible to set up the instructions in full at this stage, because some will depend on store addresses that will only be determinable when the skeleton list is complete. It is then possible to go back and fill in the missing data.

When the result has been checked, it must be set up in store, typing the values in by hand. This can be very tedious and error-prone, and checking the result can also be a chore. Since a single error may destroy the stored program completely, it should be saved before any attempt is made to run it, and it may have to be copied back from tape or disc many times before the fault is found.

To simplify this process, each instruction is given a mnemonic, which is a combination of letters that hints at the associated function. For example, LDA or LD A may mean that data is to be loaded into the 'accumulator' or A register. That is easier to remember than a number representing the machine code form of the instruction, and it allows the program to be set up in a more intelligible form. The provision of an 'on-screen editor' allows simple correction of any errors or the insertion of changes.

The resulting data file is then processed by an Assembler program, which reads the mnemonics and interprets them into machine code. It will also recognise 'labels' identifying particular points in the program, and so allow jumps to be made to those points. An Assembler normally works in two passes, as does the manual method, the first pass setting up a skeleton and second filling in the detail.

Even with the aid of an Assembler ,writing in direct machine code can present problems, and it is often more convenient to work at a higher level in the language hierarchy .This can be done in two principal ways. The 'source code' may be interpreted at the time the program is run, or it may be interpreted into machine code form once and for all, the machine code form being used when the program is run.

The best-known 'interpretive' language is BASIC, though it should be understood that this word covers a multitude of 'dialects', some of which may be regarded as sins against the original, which was developed by Dartmouth College. All the variants, however, work on the same principles.

The user sets up a series of numbered statements, which will be executed in numeric sequence unless they instruct otherwise. The statements are stored in a standardised form, some words being stored as numbers ('tokens') to economise in storage space. When the program is run, the content of each statement is scanned. The first word, or its token, will switch the action of the interpreter to a particular section, and what follows will be interpreted accordingly.

Because it is necessary to search for the required interpreter section, and for a number of allied reasons, BASIC programs run rather slowly by comparison with a machine code equivalent, but this is offset by convenience of use, the statements often taking the form of plain English interspersed with mathematical formulae. The ease with which the program can be changed by on-screen editing is another advantage.

Compiling languages are a different matter. They require the 'source code' to be set up according to stringent rules, and the result is 'compiled' by a special program into direct machine code. This can then be run at full machine code speed, the searches required by an interpretive language having been performed during the compiling process.

To say that the compiled machine code runs at full machine code speed may be slightly misleading. Since it lacks the benefit of human intuition, the compiler program does not always produce optimum machine code. For example, it may spend time storing away a variable that does not need storing, because it is an intermediate result which will be used at once.

As a result, it is often possible to scan

through a compiled program and 'optimise' it by removing unnecessary circumlocutions. There are even optimising programs which may do this automatically, but they are unable to compete with human ingenuity in some cases.

The loss of program efficiency may not be large, being commonly quoted as 10%, but that depends on circumstances. In some situations, a very useful increase in effeciency can be obtained by manual optimisation.

Broadly speaking, then, the 'higher level' languages simplify the task of creating a program, but only at the expense of working efficiency. This, it is argued, is acceptable, since it should always be possible to expand the hardware and speed it up to compensate for the loss. While this view is expressed mainly by software engineers, hardware engineers may be disappointed to find that the enhancements which they provide are swallowed up in software inefficiencies.

There is a particular point in this area which deserves special mention. Among more recent microprocessors, some have provided a block of freely usable registers, which allows faster operation in a loop, as the data required can be set in the registers before the loop is entered, saving the need for frequent store accesses during the loop. Unfortunately, some compilers fail to take advantage of this, and the result is that the potential of the better microprocessors is degraded to that of their less effective contemporaries.

In general terms, such phenomena arise in many areas of computer work. Where a more direct approach poses problems for the user, the less direct method reduces system efficiency. This has been compared with power steering on a car. It eases the driver's task, but insultates him from the steering 'feedback' to some extent.

FORTH may be cited as an example of this. A FORTH system provides a number of key machine code modules which, in combination, will perform almost any overall function which is likely to be required. Additional functions can be added if necessary, and the overall program is built up of a hierachy of 'words' which call up sequences of functions which are defined by other words.

By using an ingenious system of direct linkages, FORTH finds its way through the resulting hierarchy very rapidly, and may approach machine code in execution speed. However, it cannot be described as 'user-friendly' language, since it requires the programmer to think very hard and keep track of what is happening. It appeals to those who are ready to work a little harder to achieve high execution speed, but is of little use to those who prefer an easy ride to record-breaking.

In addition to interpreting and compiling languages, a computer will usually have an 'operating system', which looks after the 'housekeeping', monitoring the keyboard, keeping track of screen position, and so on. With most recent computers, the operating system is merged with the BASIC interpreter in a single ROM-borne program, though it has been kept separate in some cases.

The operating system, with or without an associated interpreter, may be more significant than the hardware design in determining machine performance. It may determine what peripheral devices can be used, and is likely to have a key effect on overall running speed.

This is particularly true where extensive graphic colour facilities are provided. These are, in general, recent innovations, and they are a mixed blessing. One consequence is that more Random Access Memory (RAM) is used up, because where a simpler display may require only one store location for each character position on the screen, the full colour graphics system will require nine or more. This may be balanced by reducing the effective screen size, which is in itself a disadvantage for some purposes. However, there may be other reasons for reducing the screen size. It is demostrable that a display with more than 32-40 columns does not show clearly when a standard television set is used as a Video Display. With computers costing £200 or less, it would be strange to build them so that they required the use of a specialised display monitor costing as much again . . .

The erosion of RAM by a colour graphics system has a double impact, because direct manual control of such a system can be tedious and confusing. With perhaps a million dot positions to consider, a process of setting up each dot in turn becomes impossibly laborious, and this is another instance where support software can be useful. There are programs which allow the construction of complex displays with relative ease. Unfortunately, they — like the colour graphics system itself — use up valuable RAM space.

This brief survey of software aids may help to explain why the reduction of software cost per hardware unit proved to be so important in favouring the microprocessor in relation to the bit-slice approach. There are untold masses of specialised software, and each program is dedicated to a particular processor, and to a lesser extent to a specific system using that processor. Each new microprocessor or computer creates a demand for adaptation of existing programs.

This has led to what has been called the 'Fossilisation Syndrome'. This raises opposition to new devices or systems because they are not yet fully supported, but the capital expenditure necessary to alter that situation will not be committed until there is some sell. This leads to deadlock, except in some cases where the new device is sufficiently novel and effective to warrant speculative investment.

The overall value of a computing system is thus dependent on many factors, and its total merit may be very hard to determine by a brief inspection. Examination of a number of new products has shown that there is an initial instinctive reaction, which is often supported by deeper investigation. However, such an assessment springs from experience, and that is rarely available to the first-time user, who may be rather vulnerable to glib sales-talk.

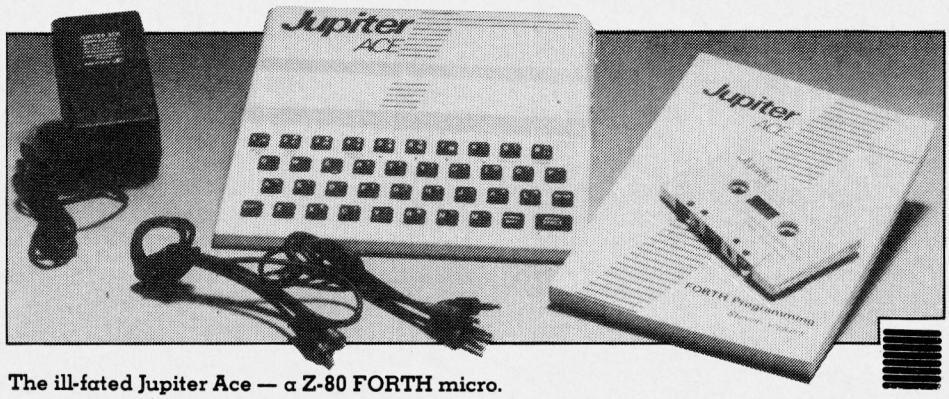
Another problem in this area is that an inexpensive basic machine may prove to be far more costly by the time it has been built up to full capability. The increase in cost may well be swollen by the purchase of apparently useful programs that fail to live up to their advance billing, or hardware extensions that fail to perform well.

This rather depressing picture stems from the fact that the microcomputer industry is still growing, and has yet to settle down to stability and complete integrity. Development has been hampered by a shortage of professional hardware and software engineers, which has encouraged the creation of small enterprises which may produce effective goods, but which lack the backing to do so in a fully economic manner.

The conclusion which must be drawn is that future development must depend on a process by which these scattered independents can coalesce into a smaller number of more viable organisations. This will not be encouraged by the experience of some investors, who have found that the capital they have provided is not always used to good effect, but others have been more fortunate.

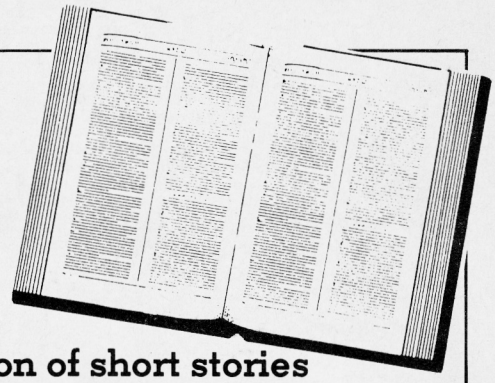
From being a tight-knit consortium of specialists, the British computer industry has expanded into fragments of varying sizes, and while that has produced a measure of weakness, it has also indicated that there is plenty of useful material, if it can only be organised more effectively.

It has been said that there is an optimum size for a business concern, varying according to the type of business involved. It would be fair to suggest that the optimum size for computer companies is still being determined, and it is being determined largely on a basis of the survival of the fittest, but not necessarily on a basis of the fitness of the company's products . . .



The ill-fated Jupiter Ace — a Z-80 FORTH micro.

BOOK PAGE



Garry Marshall

Four books this month, and among them a collection of short stories edited by no less a man than Isaac Asimov.

The first client arrives at the computerised, while-you-wait, detective agency. All he sees when he enters the room is a screen, a video camera and a chair. He wants to locate another man, but all he has to offer the agency is a blurred photograph.

Obviously, there isn't much to go on. But the agency, which is operated remotely using closed circuit television is run by a computer-detective with access to all the world's on-line databases and with the aid of a certain amount of high technology wizardry.

The client is asked to indicate the physical and social characteristics of his target by marking the appropriate choices on the screen with a light pen. When it becomes apparent that the responses are deliberately unhelpful, the light pen is disabled , and the client is asked to indicate his choices by pointing instead. The system takes his fingerprints. By accessing a government computer the client at least, is identified.

Meanwhile, the face on the photograph, and its background, have been electronically enhanced to sharpen them up. Feeding the resulting image to a computer running an expert system produces certain identifications and deductions. The landscape and the architecture of the buildings in the photo allow its location to be narrowed dowm to a small area. The foliage on the trees shows that the photo has been taken recently. The number on the door of the cottage in the picture can be seen. After this, accessing the computer holding the local records for the area in question quickly gives the address of the cottage and the name of the man who has recently rented it. Search over.

All is not what it seems, of course, and our detective gets into the computer at the target's bank and from there there is a chain of events that reveals what his client is really after.

This is the theme of one of the short stories in **Computer Crimes and Capers** edited by Isaac Asimov. The contributions are described as 'stories of crime and suspense set in the high-tech world of modern computers'. They are taken from American science fiction magazines, detective magazines and other collections of short stories. The story outlined above was written in 1978, but many of the others are a good deal earlier, going back as far as 1953.

The point of including our story in the book, of course, is that it is the detective who is the high-tech criminal, for his access to the on-line networks and computers is illegal — he has retained user numbers and passwords from a previous position when his use of them was legitimate.

It is interesting that the story should anticipate an illegitimate use of the computer that is currently of widespread interest and concern. Many of the other stories prefigure current concerns just as accurately, ranging from ways of finding the password to get into a system, through policing and living in computer-controlled societies, to creating foolproof systems.

Science fiction has always been of some help in 'inventing the future'. After all, we can only create that which we can first imagine. In the same way, we may be able to avoid undesirable trends if they can be anticipated. Once done, it won't matter if the anticipating was by a science fiction writer or an academic philosopher. These stories accurately reflect a number of developments that are with us already; there are probably also some that are yet to come. Expert systems, the global village, widespread use of satellites, computer-controlled environments, super-reliable computers and much more are confidently imagined. But although the future uses of computers are imagined accurately, the future computers themselves are not. Physically huge mainframe computers still loom large particularly in the earier stories, which is not so surprising, but also in the later ones.

The matter of inventing the future is closely related to the act, which is very important in computing, of correctly specifying a problem. When the problem concerns what we shall have in the future, its solvers will be creating our future. The importance of formulating the problem correctly is exemplified by the following much-related anecdote from World War Two. Both the army and the RAF needed improvements on their searchlights to detect incoming planes. The army asked for better searchlights, and got them. The RAF asked for better ways of detecting incoming planes, and they got radar.

There is also, I suppose, a certain danger of bringing together a number of stories about ingenious computer crimes, in that it could provide a source of ideas for the budding computer criminal. Current attitudes towards computer crime seem to be somewhat indulgent, based on the idea that the same information is available to those who are its targets as to those committing the crimes, so that the targets ought to be able to use the information for prevention. Whatever else one may say about this attitude, it isn't consistent with attitudes to other sorts of crime.

The film 'The three days of Condor' keeps coming around on the box. It concerns a section of the CIA whose members read books. And books are read for any scenarios that might be useful to the CIA in its operations. Does the Mafia have a similar branch, I wonder, and will this book interest them? In the film, the entire section is wiped out, except for Robert Redford, of course, who rings the CIA for help expressing his bewilderment with the reviewer's lament 'We just read books, for Christ's sake!'. (incidentally, the film is based on a book called 'The six day's of the Condor'. The reason for halving the time span has intrigued me for some time. Does it mean that a film is worth half as much as a book? Or that a film can do it in half the time?)

One of the other stories in the book concerns the use of an intelligent computer by a national security agency. The computer runs what we would now call an expert system. (The story was written in 1963, and the system is called 'Genius'. Our less hyperbolic name could reflect a certain acceptance of such amazing computer-based capabilities.)

The officers of the agency use the system by feeding it all the relevant information they have, after which it delivers its advice on how to proceed. Its advice does not have to be accepted, and the officers can make decisions about what to do for themselves. But the information held by the computer is so comprehensive and its advice so good that they come to rely on it and to accept its advice without question.

The trouble is that, after it has given reliable advice for a long period, it suddenly begins to give advice that turns out to be unreliable and even a cause of danger to the national security. And there seems to be no way to find the cause of the computer's treachery, because it is designed to accept information but not to release it. It communicates only the advice and directives that it deduces from its stored information; not the information itself.

So, the problem is to find out what information the computer has been given to cause it to turn traitor when the information held by the computer cannot be accessed. It is a very

ingenious problem, and one that is closely related to the theory of what is computable. While the struggle to solve it goes on, security is breaking down.

Revealing the solution obviously ruins the story, so you may want to skip the rest of this paragraph, or perhaps just delay reading it while seeking the solution yourself. It turns out that a mole has told the computer that the officers of the security agency are spies. This causes the computer to reason that providing them with correct advice based on their information will be to help a foreign power, so to undermine the national security. Consequently it gives other advice and directives. When the officers of the agency are replaced or, at least, when the computer is told that they are, the system reverts to normal, you may be pleased to know.

The final story in the book is by Asimov himself, and provides and antidote to all the usual stories about how computers will run the world. The computer that runs the world in this story begins to show signs of weariness and, when someone thinks to ask it what it wants most, it replies that it only wants to die because it is fed up with carrying all the cares of the world.

The book is full of well-crafted short stories that are all very enjoyable in their own right, quite apart from any technical or computer-related content. They are also packed with stimulating ideas, although I hope that they don't stimulate any criminals to action.

One small moan. The book is published by Viking. This is an American imprint acquired by Penguin who have decided to use the name in this country. I do think that in the circumstances they might have dealt with references to 'Somersetshire'.

Having started with a description of the activities of a 'hacker', albeit a sophisticated one, it is appropriate to move on to **The Hacker's Handbook**. This book is by Geoff Wheelwright and Ian Scales and is published by Longman: it is not to be confused with Century Communication's book of the same name. I suppose that the Century book will inevitably overshadow that of Wheelwright and Scales because of the publicity given to Scotland Yard's concern about it and the aura of revelations about how to perform illicit activities that surrounds it. This is rather unfortunate, as Wheelwright and Scales have produced a clear and concise book that has been carefully thought out and well produced with some care given to the design and layout of every page. Actually, they have written more than one book. I read the Spectrum-specific version, but there are parallel versions for the Commodore 64 and BBC Micro.

Geoff Wheelwright is a regular contributor to the computer pages that appear in The Times every Tuesday, and Ian Scales has written several books. Their high profile provides a stark contrast to the author of the Century book who felt the need to use a pseudonym.

The book begins with an overview of what you can do when you get your micro on-line, and this provides all the motivation that may be needed to tackle the rest of the book. The next chapter covers, in general terms, the hardware that is needed and explains the principles of how communication is carried out. This is followed by a specific, and detailed, account of how to get your Spectrum on-line which includes descriptions of the software and modems that are needed. The names of manufacturers and suppliers of each are provided.

The remainder of the book covers the services that can be provided or accessed once you are on-line. It deals with bulletin boards, Teletext, Prestel, Micronet and telex. The treatment of what you can do with Teletext (Ceefax and Oracle, if you like) is particularly interesting and different from anything else I have seen, but this part of the book is uniformly good. It includes many illustrations of screens from Prestel, Micronet and Ceefax, and gives examples of dialogues with bulletin boards. This gives authority and conviction to the presentation.

I have to say that I preferred the treatment of the applications to the explanation of the hardware and principles involved. This was partly because I didn't care for the model which they choose to explain the communication process. It consists of two pairs of bicycle handlebars with a brake cable linking the brake lever on one set to the corresponding one on the other. This allows communication to take place by 'braking' at one end, when the brake cable will move the brake lever at the other end, so 'passing a signal' from one end to the other. The model is OK to this point, but the later appearance of a set of handlebars with seven brake levers on it seemed to me to be harder to understand than the process that it was supposed to explain.

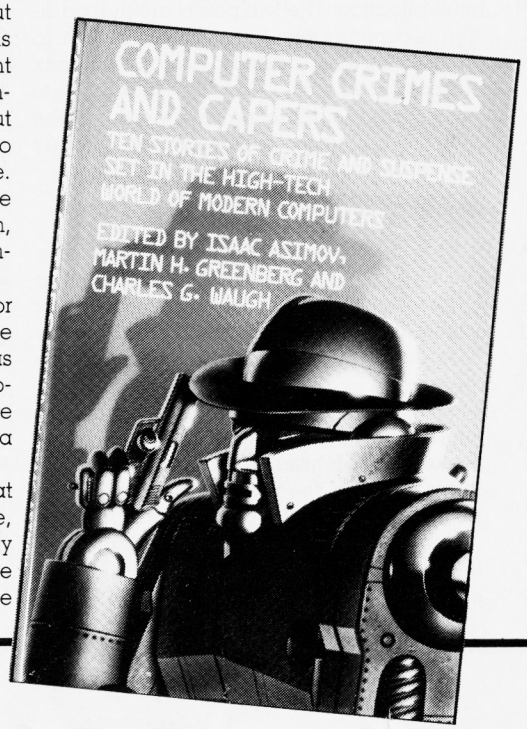
All in all, though, this is a good account of how to achieve on-line communication with a paticular micro and of what you can usefully do once you are on-line.

Computers, Communication and the Community by Christopher Pilley and Margaret Sutherland is an account of what has been done in Scotland to find a role for the new information technologies in the community. It also details attempts at making the community aware of this technology and its potential capabilities. With its own 'Silicon Glen' employing many thousands of people, and with a high rate of unemployment generally, Scotland has more reason than most regions to look for great things from new technology.

Uses for Teletext, Prestel, 'community cable', cable tv and computers, both separately and in combination are considered. Then the booklet comes down to earth with a number of case studies of projects that have been floated, and which mainly centre around micros. They include the Edinburgh Home Computing Club which has the astonishingly simple and sensible aims of gathering computer users together to combat the isolation that is the inevitable fate of the individual computer user, and of introducing to computers those who, for whatever reason, cannot afford one of their own. The Edinburgh Walk-In Numeracy Centre (yes, EWINC) aims to be a centre where adults can drop in for advice and tuition related to numeracy. Micros are used extensively in the tuition, and the Centre has plans for making micros available in the same way to foster computer literacy. Other projects cover the use of computers in the community and to help the disabled.

The booklet shows what one region is doing to help itself to take advantage of the new technology. The message that comes through from each and every case study is that more money is needed to enable more to be accomplished. The cleft stick is that it is incredibly difficult to quantify that which has been achieved, but until this can be shown in concrete terms those who hold the purse strings have an excuse for not loosening them.

Finally, and too briefly, **Introducing CAL** by Keith Hudson is, as its sub-title states, a practical guide to writing computer-assisted learning programs. It shows which topics are suitable for CAL and which are not, and then carefully shows how to prepare a computer-assisted learning package for a suitable topic. It has the enormous merit of showing all the frames produced by a CAL program for a particular unit. There are 99 of them, and it was a brave decision on someone's part to include them all. But the decision is more than justified by the results, which is to give us, in terms as concrete as anything to do with software can be, a demonstration of CAL in successful action.



This month's books are:

Computer Crimes and Capers edited by Isaac Asimov et al (Viking) 253 pages, £10.95

The Hacker's Handbook by Geoff Wheelwright and Ian Scales (Longman) 128 pages.

Computers, Communication and the Community by Christopher Pilley and Margaret Sutherland (The Scottish Community Council) 50 pages, £1.95

Introducing CAL by Keith Hudson (Chapman and Hall) 171 pages, £9.95

THE NATURE OF COMPUTER GRAPHICS

F M Botto

Part two: graphics and matrices

Doing arithmetic on matrices is often tedious and boring, at least by hand, but frequently highly useful in graphics. Fortunately, computers can handle arrays very efficiently, and it doesn't take much trouble to persuade the computer to do matrix aritmetic for you. If you're very lucky, you may even have access to a high-level language that has instructions to do all the boring bits for you. All the same, it's as well to be able to work out what is actually going on for yourself, so that when your program crashes . . .

The reason why matrices are so handy in computer graphics is that you can use a single large array to store the positions of a number of related points — say the outline of a complex shape — and then use matrix arithmetic to move the shape around the screen, turn it around an axis, and many other convenient operations, without having to work out from scratch how to move each individual point.

So to the nitty gritty. The numbers (or, in mathematics, the symbols) in a matrix are called the *elements* and we can refer to each element in the matrix by giving its position. In mathematics text books, you will see matrices written as:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

and here the subscripts label the elements in a fairly obvious way, and one that can be extended indefinitely. A matrix like the one above, with an equal number of rows and columns is called a *square* matrix. A matrix need not have equal numbers of rows and columns; indeed, you can have column (ie, only one column) and row (ie, only one row) matrices if you want.

ADDITION AND SUBTRACTION

For it to be possible to add two matrices (or to subtract them) they must have an equal number of rows and columns. All that you do is to add (or subtract) the corresponding elements from each other: the top left hand element in one array is added to the top left hand column in the other array and this makes the top left hand column in the new matrix. Examples are shown in Fig. 1.

$$\begin{bmatrix} 3 & -2 \\ 1 & 4 \end{bmatrix} + \begin{bmatrix} 1 & 3 \\ 2 & -3 \end{bmatrix} = \begin{bmatrix} (3+1) & (-2+3) \\ (1+2) & (4+-3) \end{bmatrix} = \begin{bmatrix} 4 & 1 \\ 3 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 3 & -2 \\ 1 & 4 \end{bmatrix} - \begin{bmatrix} 1 & 3 \\ 2 & -3 \end{bmatrix} = \begin{bmatrix} (3-1) & (-2-3) \\ (1-2) & (4--3) \end{bmatrix} = \begin{bmatrix} 2 & -5 \\ -1 & 7 \end{bmatrix}$$

Fig. 1 Addition and subtraction of matrices.

There are a couple of important points about the matrix addition and subtraction which may appear obvious but should not be taken for granted, as we shall see with multiplication. The two points are that addition is *cummutative* (ie adding matrix A to matrix B is equal to adding matrix B to matrix A) and *associative* (adding matrix A to matrix B then adding matrix C produces the same result as adding matrix A to the already calculated sum of matrices B and C).

MULTIPLICATION

This is the most time consuming operation, but it is by no means the most difficult to follow. However, it isn't the most obvious of procedures, so don't attempt to attach any logic to it, just accept it and learn to love it.

Two matrices can be multiplied together when the number of columns of the first is equal to the number of rows of the second; if this is the case, the two matrices are said to be conformable.

If two matrices A and B are conformable, and their product is the matrix C, then the elements of C are given by the formula:

$$x_{ik} = \sum_{s=1}^{s=p} a_{is} \; b_{sk}$$

where p is the number of columns of A which is equal to the number of rows of B.

To the non-mathematical, this formula doesn't reveal a lot, so let us proceed by example, firstly using symbols, then using real numbers. For the sake of simplicity, suppose the matrices A and B are square 2x2 matrices, so that their product, C, is also a 2x2 matrix. Figure 2 shows the multiplication of the matrices in symbols. Figure 3 shows the same two matrices multiplied together, but this time matrix B is multiplied by matrix A; note that the resulting elements in their product matrix are different. In general, matrix multiplication is not commutative, ie A times B generally does not equal B times A. We leave it as an exercise to the reader to see if matrix multiplication is associative (in case you can't bear the suspense, it is).

Before going on to some numerical examples, let us attempt a translation of the formula given above. The formula says that the element of the product matrix in the ith row and kth column is equal to the sum of all the elements in the ith row of the first matrix times the corresponding element in the kth column in the second matrix. Looking again at Figure 2, the in the sums that go to make up

$$\underset{A}{\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}} \times \underset{B}{\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}} = \underset{C}{\begin{bmatrix} (a_{11} \times b_{11} + a_{12} \times b_{21}) & (a_{11} \times b_{12} + a_{12} \times b_{22}) \\ (a_{21} \times b_{11} + a_{22} \times b_{21}) & (a_{21} \times b_{12} + a_{22} \times b_{22}) \end{bmatrix}}$$

Fig. 2 Multiplication of matricies, using symbols.

$$\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \times \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} (b_{11} \times a_{11} + b_{12} \times a_{21}) & (b_{11} \times a_{12} + b_{12} \times a_{22}) \\ (b_{21} \times a_{11} + b_{22} \times a_{21}) & (b_{21} \times a_{12} + b_{22} \times a_{22}) \end{bmatrix}$$

Fig. 3 The same two matrices as Fig. 2 multiplied together in the reverse order give a different result, in general.

the elements of C, the new matrix, the first subscript of the old a elements in each sum are all equal to the row of the new element of c and the second subscript of the old b element is equal to the column number of the new element; also where two old elements are multiplied together, the second subscript of the a term is always equal to the first subscript of the b term.

Let us retreat to some numbers: Figure 4 shows two matrices being multiplied together using numbers for the elements; Figure 5 shows the same two matrices multiplied but with the order reversed — notice that this produces a completely different result.

If all the above has just served to confuse, here is a simpler method for multiplying two matrices. Firstly, write down the elements of the first matrix twice:

a_{11} a_{12} a_{11} a_{12}

a_{21} a_{22} a_{21} a_{22}

and multiply each column, from left to right, by the elements of the second matrix, in the following order:

i. 1st row, 1st column;
ii. 2nd row, 1st column;
iii. 1st row, 2nd column;
iv. 2nd row, 2nd column.

This procedure will yield the same matrix product as shown in Fig. 2.

Two matrices do not have to be the same size for it to be possible to multiply them together; indeed, it is only square matrices of the same size that can be multiplied together. As has already been said, the number of columns in the first matrix must equal the number of rows of the second matrix, and the product matrix will have the same number of rows as the first and the same number of columns as the second.

Consider, for example, the two matrices in Figure 6; here, A has two columns and T has two rows, so they are conformable. However, this example is more significant, because the matrix A can be a set of coordinates for the points on a shape and the matrix T can be what is called a **transformation matrix**. The product matrix, A*, contains x and y coordinates that are a mixture of the old x and y coordinates for each point. The proportion of these mixtures is chosen by choosing the sizes of the elements of the transformation matrix, t_{11}, etc.

Take the practical example in Fig. 7a; here a group of coordinates, labelled A,B,C, etc are grouped in a matrix and a simple transformation matrix is applied to them which results in the x and y coordinates being swapped; as you can see in Fig. 7b, the results in a reflection of the shape.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} (1 \times 5 + 2 \times 7) & (1 \times 6 + 2 \times 8) \\ (3 \times 5 + 4 \times 7) & (3 \times 6 + 4 \times 8) \end{bmatrix}$$

Fig. 4 Multiplying two matrices with figures.

$$= \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

$$\begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} \times \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} (5 \times 1 + 6 \times 3) & (5 \times 2 + 6 \times 4) \\ (7 \times 1 + 8 \times 3) & (7 \times 2 + 8 \times 4) \end{bmatrix}$$

Fig. 5 The same two matrices as Fig. 4 in reverse order, giving a different result.

$$= \begin{bmatrix} 23 & 34 \\ 31 & 46 \end{bmatrix}$$

$$\underset{A}{\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \\ x_5 & y_5 \\ x_6 & y_6 \end{bmatrix}} \times \underset{T}{\begin{bmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \end{bmatrix}} = \underset{A^*}{\begin{bmatrix} (x_1t_{11} + y_1t_{21}) & (x_1t_{12} + y_1t_{22}) \\ (x_2t_{11} + y_2t_{21}) & (x_2t_{12} + y_2t_{22}) \\ (x_3t_{11} + y_3t_{21}) & (x_3t_{12} + y_3t_{22}) \\ (x_4t_{11} + y_4t_{21}) & (x_4t_{12} + y_4t_{22}) \\ (x_5t_{11} + y_5t_{21}) & (x_5t_{12} + y_5t_{22}) \\ (x_6t_{11} + y_6t_{21}) & (x_6t_{12} + y_6t_{22}) \end{bmatrix}}$$

Fig. 6 Transformation of coordinates, using symbols.

SCALAR MULTIPLICATION

Any matrix can be multiplied by a single scalar quantity (by a scalar quantity, we mean basically a simple number). For example, if S is a scalar quantity, the multiplication can proceed as in Fig. 8.

This is a convenient method of enlarging or reducing the size of a graphical shape; for example, if we use a 2x2 matrix to contain the coordinates of the ends of a line, multiplying the whole matrix by 2 enlarges the length of the line by a factor of 2.

SCALING IMAGES

It is often necessary to enlarge or reduce the size of a graphic shape. The simplest method is to adopt a scaling factor which operates directly on the position vectors of the points in the shape. The technique is illustrated in the program shown in Listing 1; this program is written in BBC BASIC, but there are no special tricks involved and it should be quite easy to write an equivalent program in any other dialect of BASIC.

An alternative method of scaling would be to use a transformation matrix with just diagonal elements present. Figure 9 gives a simple scaling, while Fig. 9b will also produce a reflection. A program using this technique is shown in Listing 2, and this program is described in the 'Program Description' section; also a flow chart is given in Fig. 10.

As mentioned in the Program Description, the elements of the transformation matrix may be changed by altering a single line (line 560). The elements presently form the matrix shown in Fig. 11 which produces a 2x

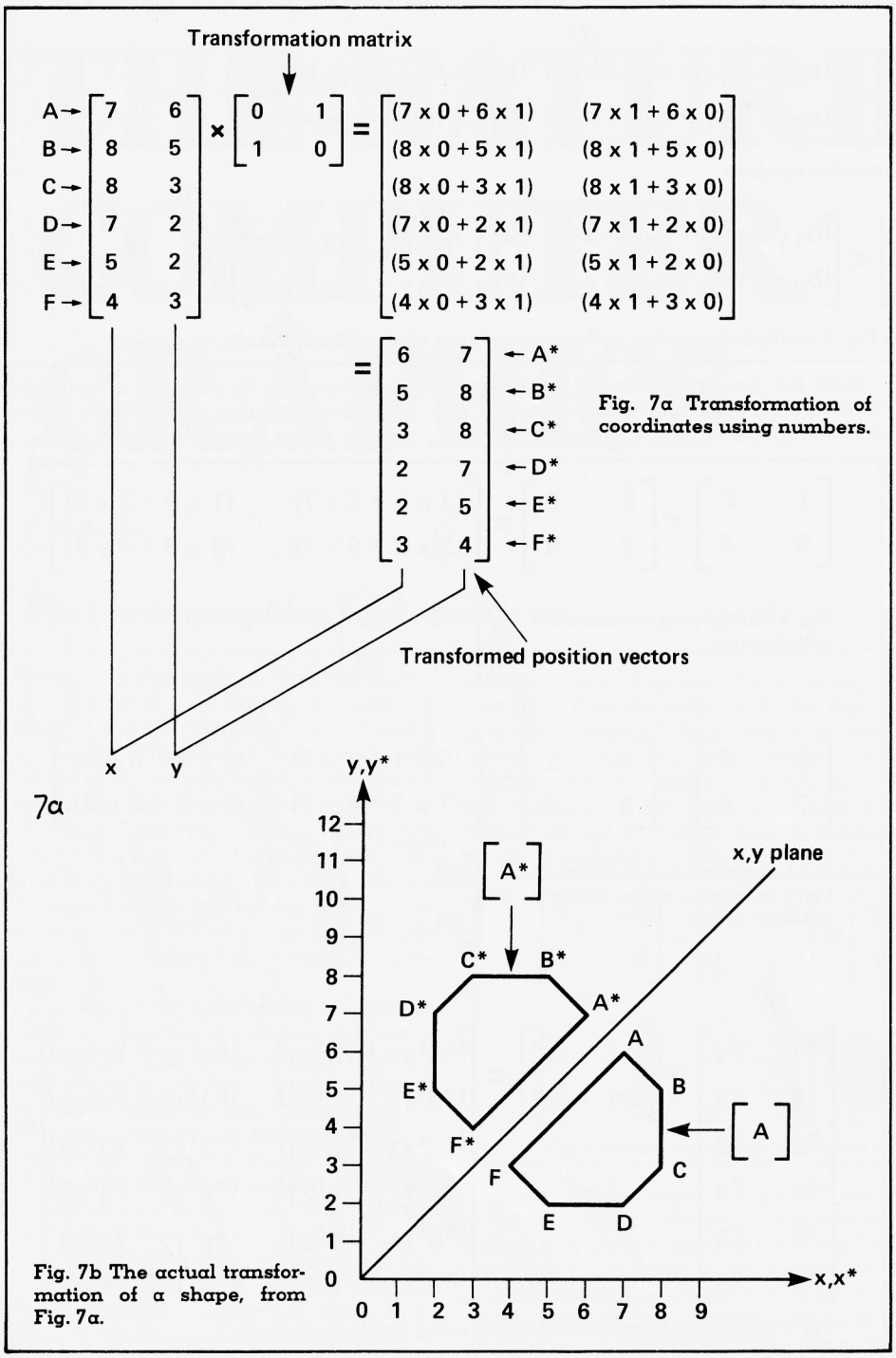


Fig. 7a Transformation of coordinates using numbers.

Fig. 7b The actual transformation of a shape, from Fig. 7a.

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} xS = \begin{bmatrix} a_{11}S & a_{12}S \\ a_{21}S & a_{22}S \end{bmatrix}$$

Fig. 8 Multiplying by a scalar.

enlargement and reflection. If you've typed in this program, it would be a shame to waste the opportunity to see what effect altering the elements would have, wouldn't it?

Fig. 9 Using diagonal matrices to change size: (a) without reflection; (b) with reflection.

$$\begin{bmatrix} S & 0 \\ 0 & S \end{bmatrix} \quad (a) \qquad \begin{bmatrix} 0 & S \\ S & 0 \end{bmatrix} \quad (b)$$

PROGRAM ALPHA

Program Alpha is written in BBC BASIC, and has been structured to promote an easy understanding. The purpose of the program is to simply draw a straight line, and then appropriately transform it. In this program the transformation matrix performs a scaling function. However, the elements of the transformation matrix are easily altered by changing a single line, and this will allow you to experiment with various transformation matrices and perhaps invent a few of your own. Before attempting this, it would be advantageous for you to gain an appreciation of the program operation.

$$\begin{bmatrix} 0 & 2 \\ 2 & 0 \end{bmatrix}$$

Fig. 11 Alternate transformation for listing 2.

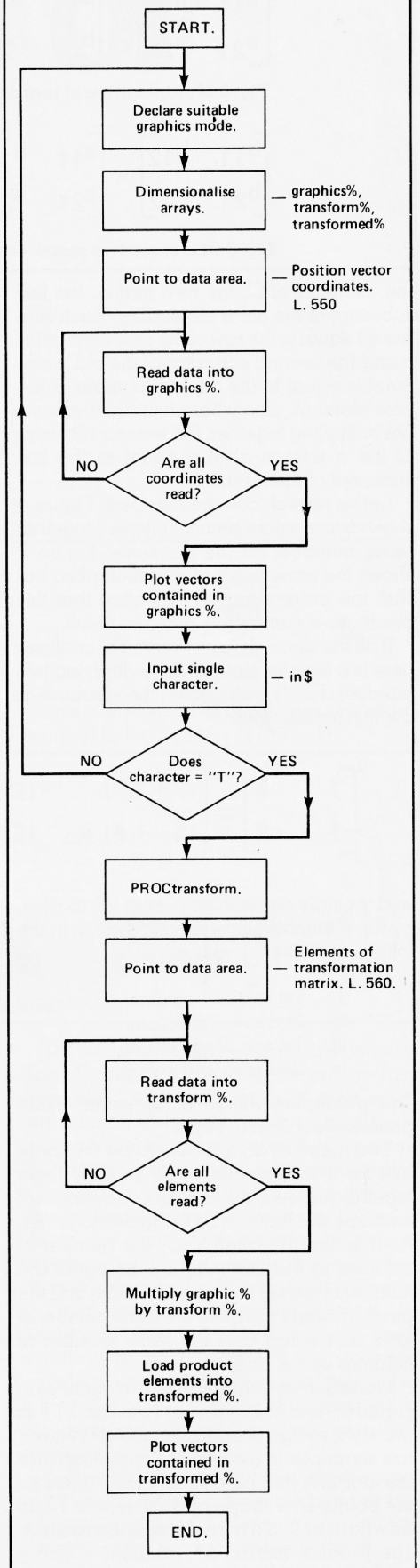


Fig. 10 Flow chart for Listing 2.

Listing 1
A program using scalar multiplication.

```
 1REM ************************************************************
 2REM *                                                          *
 3REM *         Program which illustrates                        *
 4REM *           overall scaling                                *
 5REM *         (Using scaling factor)                           *
 6REM *                                                          *
 7REM ************************************************************
44 PRINT"Press key E for 1.3* enlargement."
46 PRINT"Press key R for .5 reduction."
50MODE 4
60DIM graphics(1,6)
70RESTORE 440
80FOR row% =0 TO 6
90FOR column%=0 TO 1
100READ graphics(column%,row%)
110NEXT column%
120NEXT row%
130MOVE graphics(0,0),graphics(1,0)
140FOR row%=0 TO 6
150DRAW graphics(0,row%),graphics(1,row%)
160NEXT row%
170INPUT in$
180IF in$="E" PROCenlarge ELSE PROCreduce
190REM*********************************************************
200REM*          PROCenlarge follows                            *
210REM*********************************************************
220DEF PROCenlarge
230scalefact=.75
240PROCdraw
250ENDPROC
260REM*********************************************************
270REM*          PROCreduce follows                             *
280REM*********************************************************
290DEF PROCreduce
300scalefact=2
310PROCdraw
320END PROC
330REM*********************************************************
340REM*           PROCdraw follows                              *
350REM*********************************************************
360DEF PROCdraw
370CLG
380MOVE graphics(0,0)/scalefact,graphics(1,0)/scalefact
390FOR row%=0 TO 6
400DRAW graphics(0,row%)/scalefact,graphics(1,row%)/scalefact
410NEXT row%
420REM
430REM*********************DATA BASE*************************
440DATA 700,600,800,500,800,300,700,200,500,200,400,300,700,600
450REM END
```

PROGRAM DESCRIPTION FOR LISTING 2

Line 70-100 Declare graphics mode 4, and dimensionalise graphics%, transform%, and transformed%.

Line 110-160 Point to data area which contains position vectors. Read position vectors into graphics%.

Line 170-220 Plot the position vectors contained in graphics%, and accept a single character (In$). Check In$ to determine whether Key "T" has been pressed. (T=transform)

Line 270-330 (PROCtransform) Point to data area where transformation matrix elements are stored. Read elements into transform%.

Line 380-460 Multiply graphics% * transform% , and read product elements into transformed%. (It's worth looking at this section carefully, as this arrangement will undoubtably help you with your own graphics programs.)

Line 470-500 Plot the position vectors contained in transformed%.

Line 560-570 Data area-Position vectors and elements of transformation matrix.

A more significant program enhancement, would be the introduction of larger data base (position vectors). Thus a more complex graphical shape may be supported. Perhaps you can determine the necessary program adjustments in order to implement such an enhancement.

ACT MICRODEALER

xi APRICOT

CPU	8086
MEMORY	256K RAM
LANGUAGES	Microsoft BASIC, Personal BASIC
MASS STORAGE	No cassette drive Integral Sony 3½" 315K microfloppy disk drive Integral 5 or 10 Mb hard disk
OS	MS-DOS 2.11 with GSX bundled CP/M-86 (not yet available) Concurrent CP/M-86 (not yet available)
KEYBOARD	QWERTY, cursor, numeric pad, function keys
INTERFACES	RS-232C, Centronics, Microsoft mouse
DISPLAY	Monitor (supplied)
GRAPHICS	80 by 24 text with block graphics 800 by 400 high-res graphics under GSX
SOUND	No

Notes. The Apricot xi is a development of the award-winning Apricot, and replaces one of the latter's disk drives with an integral hard disk, providing vastly increased storage with faster access. Memory may be expanded in 128K increments to a maximum of 768K. The languages and operating systems mentioned above come bundled (except for Concurrent CP/M) and four software tools are also bundled, including an asynchronous package for use with the optional modem card.

HAMPSHIRE

TIMATIC SYSTEMS LTD
The Market, Fareham.
Tel: (0329) 239953

For the complete range of Apricot hardware and software. Also dealers for Zenith, Memotech. For future information call or ring anytime.

SCOTLAND

SIRIUS
is alive and well and supported at
ROBOX
(Office Equipment) Ltd,
The Scottish Computer Centre
Anderson Centre, Glasgow
041-221 8413/4
34 Queen Street, Edinburgh
031-225 3871

WEST MIDLANDS

Q data limited
The Black Country's specialist in micro-computing. Full range of ACT Apricots and IBM personal computers.
The Limes, High Holborn, Sedgley, West Midlands.
Tel: Sedgley (09073) 62331

CBM MICRODEALER

Notes: The Commodore 64 is a popular micro with a great deal of games software available. There is also some business software available.

The Commodore 715B is the top model in the 700 range of business machines.

TO FILL THIS SPACE PHONE CAROLINE ON 01-437-0699

NASCOM MICRODEALER

NASCOM 3

CPU	2 MHZ Z80	**DISPLAY**	40 or 80 column 25-line display
MEMORY	8K or 32K inbuilt RAM (expandable to 60K)		
LANGUAGE	Full Microsoft BASIC	**GRAPHICS**	High resolution graphics with 8 foreground and 8 background colours (400 x 256 pixels) Double density graphics with 2 colours (800 x 256 pixels)
MASS STORAGE	Single or twin 5.25" disc drives 350K capacity per drive		
OS	NAS-DOS or CP/M 2.2		
KEYBOARD	Full size QWERTY	**SOUND**	No
INTERFACES	RS232 and 16-bit parallel		

SHARP MICRODEALER

SHARP MZ-3541

CPU	Z80A (two), 80C49
MEMORY	128K RAM, 8K ROM
LANGUAGE	Sharp BASIC
MASS STORAGE	Twin integral 5¼" floppy disk drives, total capacity 1.28 Mb
KEYBOARD	QWERTY, cursor, numeric pad, function keys
INTERFACES	RS-232C, Centronics, interface for extra external floppy disks
DISPLAY	Monochrome monitor, colour optional
GRAPHICS	80 by 25 text, 640 by 400 high-resolution graphics
SOUND	Single channel

Notes: The Sharp MZ-3541 is aimed at the businessman. RAM is expandable to 256K, while two disk drives may be added externally to complement the integral pair. Colour is only possible with the optional graphics expansion RAM. One Z80 handles the main CPU activities while the other handles peripheral activities. The third processor handles the keyboard. The availability of CP/M means a ready supply of business software.

LONDON

SHARPSOFT LTD.
Specialists in all Sharp software and hardware.
Sharpsoft Ltd, Crisallen House, 86-90 Paul Street, London EC2. Tel: 01 - 729 5588.

COMPUTAMART

AT A GLANCE...AT A GLANCE...AT A GLANCE...AT A GLANCE...AT A GLANCE...AT A GLANCE...

CHESHIRE

Computer Junk Shop
We Buy, Sell, Break Computers & Peripherals.
10 Waterloo Rd. Widnes, Halton. Tel: 051 420 4590.

TYNE AND WEAR

HCCS ASSOCIATES
533 Durham Rd., Low Fell, Gateshead. Tel. Newcastle 821924.
Open: 6 days 9am-5.30pm (Sat 10am-5.30pm). Specialists in: Acorn, BBC, Video Genie, VIC 20.

MIDDLESEX

SCREENS MICROCOMPUTERS
6 Main Ave., Moor Park, Northwood, Middx.
Tel: Northwood (09274) 20664
Telex: 923574 ALACOL G.
Official Dealers for: Acorn, Atari, Amstrad, Apricot, Commodore, Dragon, Einstein, Memotech, Oric, Psion, Sirius, Sanyo & Sinclair.
Open 6 days per week

LONDON

LEABUS
legal and busines software
Specialists in wordprocessing systems (multi-lingual wordprocessors etc) based on the Apricot Computers.
Open 9am–6pm. Telephoe anytime.
114 Brandon Street, London SE17 1AL.
Telephone: 01 708 2756.

SUSSEX

GAMER
24 Gloucester Road, Brighton.
Tel: 0273-698424.
Open: Mon-Fri 10am-5.30pm, Sat 9am-5.30pm.

SOUTH LONDON

CROYDON COMPUTER CENTRE
Authorised Acorn Service Centre
29a Brigstock Rd., Thornton Heath, Surrey. Tel: 01 - 689 1280
BBC, Acorn, Electron, Genie, Oric, Kaga Microvitek Zenith Monitors. OKI 80, 82A + 84 Printers. Paper, Ribbons, Software etc. BUY-HIRE.

TO ADVERTISE IN COMPUTAMART RING CAROLINE ON 01 437 0699

COMPUTING TODAY

Lineage: 40p per word.

Semi display: £9.00 per single column centimetre
Ring for information on series bookings/discounts.

All advertisements in this section must be prepaid.
Advertisements are accepted subject to the terms and conditions printed on the advertisement rate card (available on request).

01-437 0699

Send your requirements to:
CAROLINE FAULKNER
ASP LTD, 1 GOLDEN SQUARE,
LONDON W1.

SOFTWARE

TURBO PASCAL
Extended Pascal for PC DOS, MS DOS, CP/M£86 and CP/M-80, includes full screen editor, floating point arithmetic, full string handling feature, random access data files, complies faster than IBM or MT + Pascal, requires less than 35K of disk space, 250 page manual and FREE spread-sheet program written in Turbo Pascal.
****ONLY £54.95****
All prices fully inclusive for prepaid orders.
CONGUIN SOFTWARE,
14 GOODWOOD CLOSE, MORDEN,
SURREY SM4 5AW.
No callers please Phone 0524 381423

COLOUR GENIE owners quiz game £2.99 coming soon — Quiz Master £4.99, Fires of Mordor, 41 Pexwood Rd, Tormorden, Lancs.

DISKS

DISKS
3M—TDK—BASF

SS DD 40T	14.64
DS DD 40T	19.99
SS DD 80T	21.25
DS DD 80T	25.25

All Prices Box 10 And Include VAT & P+P.
Send cheque stating:
Qty, Brand and Type to:
CAROUSEL TAPES
"Disks", 3 Park Parade,
Stonehouse, Glos GL10 2DB.
Tel: 0453-82-2151

ALARMS

BURGLAR ALARM Equipment. Please visit our 2,000 sq. ft. showrooms or write or phone for your free catalogue. CWAS Ltd., 100 Rooley Avenue, Bradford BD6 1DB. Telephone: (0274) 731532.

ACCESSORIES

HOME COMPUTER REPAIRS
Look at our fantastic prices on repairs!

BBC B	£27.50
VIC 20	£20.00
COMMODORE 64	£27.50
DRAGON	£30.00
ORIC/ATMOS	£25.00
ZX SPECTRUM	£17.25
ZX INTERFACE	£17.25
ZX MICRODRIVE	£17.25

PLUS OTHERS!
The above prices are inclusive of parts, labour, P&P. All repairs carry 6 months warranty on replaced parts. Extended warranties, peripheral repairs, upgrades etc. etc. All available. Ring for full details (0234) 213645.
ZEDEM COMPUTER LTD
2 Kimbolton Rd, Bedford.

DISK STORAGE BOXES
Anti-static lockable with smoked glass cover, 2 sizes available:—

to store 70 disks	£17.95
to store 100 disks	£19.95

ED 40, B1, Thornhill,
North Weald, Epping,
Essex CM16 6OW

FOR SALE

RARE BLACK BOX 64K 2 × 720K discs. C/PM in Basic - Cobol Wordstar, Epson MX, 132 column printer £650 o.n.o. Tel: 03827 69297.

TO FILL THIS SPACE RING 01 437 0699

COMPUTING TODAY
CLASSIFIED ADVERTISEMENT — ORDER FORM

If you have something to sell now's your chance! Don't turn the page — turn to us!
Rates of charge: 40p per word per issue (minimum of 15 words) + 15% VAT. Please state classification and post to: **COMPUTING TODAY, CLASSIFIED DEPT., 1 GOLDEN SQUARE, LONDON W1.**

Please use BLOCK CAPITALS and include post codes.

Name (Mr/Mrs/Miss/Ms) .....
(delete accordingly)
Address .....
.....
.....
Signature ..... **Date** .....
Daytime Tel. No. .....

Please place my advert in **COMPUTING TODAY** for . . . . . issues commencing as soon as possible.

BBC and Electron

OXFORD PASCAL



Oxford Computer Systems (Software) Ltd.
Hensington Road, Woodstock, Oxford OX7 1JR, England
Telephone (0993) 812700 Telex 83147 Ref. OCSL

Compilers like these don't grow on trees

Oxford Pascal is Fast

Oxford Pascal compiles down to FAST COMPACT P-code, giving you the real speed and power of Pascal, together with the ability to compile very large programs.

Oxford Pascal is Standard

Oxford Pascal is a full extended implementation of Standard Pascal. This means that you can compile any Pascal program (subject to size), written on any computer, anywhere.

Oxford Pascal is Compact

Because it compiles into P-code, Oxford Pascal reduces programs into the most compact form possible. In fact it allows you to pack more code into your BEEB than any other language, and should your programs become too large, you can still use the CHAIN command to overlay limitless additional programs without losing data.

Graphics & Sound Extensions

In addition to the entire Pascal language, Oxford Pascal features a whole range of Graphics (all modes) and sound extensions designed to make maximum use of the BBC Computer. Oxford Pascal also provides numerous extensions such as hexadecimal arithmetic and bit manipulation instructions.

Oxford Pascal in Education

In Education, Oxford Pascal is fast becoming a *de facto* standard. It is already the most popular Pascal on the Commodore 64, and will soon be released for the Spectrum and the Amstrad. In fact, Oxford Pascal will soon be available for 90% of the computers installed in the U.K., and is already available in German, French, Swedish, and American versions. **Students and teachers alike find that it makes sense to use a standard implementation of Pascal across the whole range of educational micros.** Call us for details of our generous educational discounts.

Manual

Both these compilers come with a manual which has been carefully designed, not only as a quick reference guide, but also as a full tutorial for those new to Pascal.

Resident and Disc Compiler

Oxford Pascal comes in two forms:

For Tape Users...Oxford Resident Pascal. A compiler located largely in ROM which is available at any time. Programs can be written and compiled on the spot without disc or tape access, and compilation is fast enough to make using the compiler much like using the BASIC interpreter. Thus, learning Pascal is a simple interactive process. Some 15K of memory is available for user programs, the remainder being reserved for compiled object code.

For Disc Users...Oxford Disc Pascal offers all the above PLUS...a full disc compiler which is capable of using the WHOLE memory for Pascal object code, it is supplied with a powerful LINKER, allowing you to break large programming tasks down into separately compilable, easily-manageable files.

Friendly Error Messages

Many compilers produce little more than an error and line number to help correct mistakes in Pascal programs. Oxford Pascal however, gives you one of 49 friendly and informative error messages. Messages which not only indicate the reason for an error, but also print out the line in question with a pointer to the exact position where the error was detected.
Run-time errors are reported using line-numbers from the original source-program, with a full explanation of how the error occurred.

Powerful Editor

With Oxford Pascal there is no need for you to learn how to use a new Editor. Pascal programs can be entered in exactly the same way as BASIC programs, without the need to learn any new commands. When you are used to using Pascal, you will find our extensions to the Standard Editor even more useful. What is more, Oxford Pascal allows you to mix BASIC and Pascal together, in much the same way that you can mix BASIC and assembler. In fact you can, if required, mix all three together...BASIC, Pascal and assembler...in one program.

Stand Alone Code

Unlike other compilers, Oxford Disc Pascal allows you to compile on the BBC and then relocate your program so that it will run on any BBC. The relocated program will run without a Pascal **ROM** and can be loaded and run from tape or disc just like any other program.

This means that you can distribute or sell your software freely and without the need for **ROMs**, to run either of the above machines.

Price/availability matrix

	BBC 'B'	C64	SPECTRUM
DISC	£49.95	£49.95	Available July 1st 1985
CASSETTE	£39.95	£22.95	

All prices are inclusive of VAT
Please add £2.00 for postage and packing.

Oxford Compilers — The Future

During the next year, we at Oxford will be releasing a series of language implementations such as C, and Modular 2, for the BBC, and other popular micros.

These compilers are being built, using the most modern techniques in **automated** compiler construction, and will bring to the micro-user, a level of robustness and efficiency, only now becoming available to mini and mainframe users.

Oxford...
the **Compiler Compilers.**

Oxford Pascal Order form. Please make cheques payable to OCSS Ltd.
Please rush me my copy of Oxford Pascal. I enclose £.....
including £2 postage and packing
My computer is BBC 'B'☐ C64☐ (please tick as appropriate)
I would like my compiler supplied on DISC☐ CASSETTE☐
Name .....
Address .....
.....
Postcode .....
Telephone .....

MORE ELEPHANTS TO TRUST





ELEPHANT printer ribbons, head cleaning disks and computer cleaning kits are now added to the ELEPHANT family to provide you with a total computer supplies package. Together with ELEPHANT MEMORY SYSTEMS disks – certified 100% error free and problem free and guaranteed to meet or exceed every industry standard – ELEPHANT is now more than ever the trusted brand that gives you the best from your computer.

Dennison

ELEPHANT NEVER FORGETS

Dennison Manufacturing Co. Ltd.
Colonial Way, Watford, Herts WD2 4JY, Tel: Watford (0923) 41244, Telex: 923321

- **France:** Soroclass, 45, rue de l'Est - 92100, Boulogne.
Tel. Réseau de Distribution: 605.98.99, Administration des Ventes: 605.70.78, Telex: EMS 206 436 F
Germany: Marcom Computerzubehör GmbH, Podbielskistr, 321, 3000 Hannover 51, Tel: (0511) 647420, Telex: 923818
Italy: King Mec SPA, Via Regio Parco 108 BIS, 10036 Settimo, Torinese, Tel: (011) 800.93.93, Telex: 211467 KIN MEC-I
Other Countries: Dennison International Company, 4006 Erkrath 1, Matthias-Claudius-Strasse 9, Telex: 858 6600