COMERIAN PRESS
LLANDYSUL

COMPUTER SCIENCE: PAST, PRESENT, and FUTURE

Inaugural Lecture of the Professor of Computation delivered at the College on October 29, 1968

by

D. C. COOPER
B.SC., PH.D. (London)

UNIVERSITY COLLEGE OF SWANSEA

UNIVERSITY COLLEGE OF SWANSEA

COMPUTER SCIENCE: PAST, PRESENT, and FUTURE

Inaugural Lecture of the Professor of Computation delivered at the College on October 29, 1968

by

D. C. COOPER

B.SC., PH.D. (London)

COMPUTER SCIENCE : PAST, PRESENT AND FUTURE

THE fact that there has been a tremendous increase in the number of computers and in the areas of their application during the last twenty years must now be known to most people. The first stored program electronic digital computer was completed in 1948 ; in 1950 there were about twenty such machines in existence, in 1960 about 10,000 and today around 70,000 with perhaps another 25,000 on order. Whilst many of today's computers are much larger and more expensive than previous machines, they are so much more powerful that the cost per computation has shown a fall as dramatic as this increase in their numbers.

The number of different applications of computers has also greatly expanded, so much so that now they affect our lives in many ways. In the commercial field for example they are used for such purposes as accounts keeping, payroll calculations, sales analysis, the keeping of stock records, the recently launched GIRO system, the analysis of surveys (perhaps a general census or traffic flow). All science and engineering disciplines use computers in many different kinds of calculations and computers are now being used directly in design processes, perhaps with the aid of some kind of two-dimensional communication devices. Computers find application in many Arts subjects, for example archaeological analyses, the analysis of historical records, authorship problems in linguistics and library cataloguing. Artistic endeavours themselves have a new medium in the computer itself, a recent exhibition in London was entirely devoted to the Computer and the Arts with many examples in which machines of some kind either produced the art form or were themselves part of it.

This is a very incomplete list : I don't intend this lecture to be a catalogue of computer applications but rather I hope I may take the widespread use of computers as

an established fact. During these last twenty years I believe there have been several landmarks in the development of computers and computing. I wish to describe some of these and then devote most of this lecture to one particular event which is taking place at the moment, and will turn out to be as significant as any of the previous landmarks.

The important event which began this great expansion was the introduction of electronics, the use of valve circuits instead of electro-magnetic mechanisms. This brought about an increase in speed of around 100, of itself perhaps not so striking but it broke through a barrier. Instead of machines being used mainly as devices much less prone to mistakes than ourselves, we could now perform calculations with their aid which would have been very reluctantly performed, if at all, without their use. Thus new methods could be used and new problems tackled. Of course this factor of 100 is insignificant by present day standards ; we now have a further increase in speed by a factor of at least 10,000.

At the same time as the introduction of electronics a second landmark emerged, indeed without it the increase in speed would have been of little use. This was the concept of a stored program. By this is meant that, instead of us telling the machine what it has to do at every stage, we store within its own memory system a complete list of instructions. These instructions will enable the machine to work at its own speed, rather than at the slow rate with which we could give information to the machine about its course of action. This stored program would have to include many different sets of instructions allowing the computer to select the correct set depending on the circumstances arising in the particular job. These stored programs may be readily changed, thus giving us general purpose devices.

Another significant event was the development of high level languages (sometimes called problem oriented languages, or automatic programming languages). As we have to provide the computer with its list of instructions,

or program, we need some kind of language in which to express this program. The basic language which computers are built to understand is very primitive indeed. It is a simple language in the sense that each individual ' sentence ', i.e. instruction, can say very little. Expressing a practical procedure in this language is a tedious operation subject to many mistakes. What we do then is to invent a language closer to our actual problems, a language which is easier to use than the basic machine language hence making the task of writing a program much quicker and less error-prone. Having designed this language we can provide a program to translate from this language into the basic machine language. Such programs are called compilers and I shall be referring to these again later. The provision of a compiler is a complex task, but once this is done everyone else's programming is greatly simplified. An example of an early programming language which is still the most widely used in the scientific area is Fortran ; an early British language which made its mark was Mercury Autocode.

Related to the concept of a high level language is another important concept, that of data structures. Our machines are built to operate on a few simple kinds of data, usually either one list of numbers or one list of alphabetic and numeric characters. Our actual problems are usually concerned with more complex data structures, for examplem any small lists, perhaps lists of lists, files containing records and each record further divided, several objects of different kinds with different properties such as might occur in simulating traffic flow in a computer, and so on. The early high level languages made a start on this problem by defining as part of the language system more useful data structures, and then the computer would provide a way of representing these more complex structures in terms of the machine's own structure. Particularly significant was a simple method devised for handling lists whose length could vary in a way which could not be predicted.

An engineering development which provoked a landmark in computer development was the complete transition from valves to transistors. From the computer users' angle an important result was the greatly increased reliability achieved, which made machines so much easier to use. Also important was the decrease in physical size and power requirements thus allowing either more convenient smaller computers or more sophistication without a corresponding increase in other factors.

A further event of great significance in computing was the development of operating systems of various kinds. In the early years each particular job for a computer had to be individually dealt with by a computer operator. If a job is to run on a computer for some time, perhaps an hour or more, then the time the operator takes to get the computer ready for the job is not very significant. However, if there are several small jobs to be run the computer could be idle a good proportion of its time, perhaps waiting for the operator to put some cards into the card reader, or to type in the necessary information about the job, or just waiting for the operator to tell the computer to start the next job. Most of this waiting time can be eliminated if a program can be stored permanently inside the computer telling the computer how to organize its own flow of work.

The first such systems were "batch processors," they enabled an operator to provide the computer with a complete batch of jobs which the computer then ran one after the other with no stops. A later development was "multi-processors" in which, rather than the computer running one job after another, it could decide to have several different jobs at various stages of completion. The advantage of this is that if one job gets held up for some reason (perhaps waiting for a magnetic tape device to be correctly positioned) then the computer can continue with some other job. Such an operating system also enables a computer user to be brought into the system. He can be given some communication device and

watch the progress of his program. If necessary, he could stop it, or change it in some way, or give it further data. We then have what is called an on-line system, a development of great importance. However, many problems remain to be solved before we can provide completely general purpose operating systems. Such complete systems have been implemented, but so far they have tended to be rather inefficient.

Another landmark in computing is, I believe, the publication of the Algol report. Algol is a high level language, designed mainly for scientific applications, which appeared sometime after the Fortran language. It is a practicable language used widely, although not as widely as Fortran for reasons unconnected with the merits of the two languages. However, its significance lay in the precision with which the language was defined and the coherency and consistency of the language itself. It was not a hotch potch of rules and exceptions as previous languages had tended to be. It set high standards for its day, although we now know various lacks and some inconsistencies in it.

I do not think anyone in the computer field today would deny the significance of the landmarks I have described, although there are certainly other events which could lay claim to being of great importance. There are certainly some in particular application areas, for example, in Numerical Analysis or Artificial Intelligence, and there are also some in the engineering side of computer design about which I am not proficient to comment. Indeed both hardware (i.e. engineering) and software (i.e. programming) developments have merged in some of the above events. However, rather than discuss further these events, or describe others, I now propose to turn to a very important landmark which I believe is being created now.

This is the emergence of a new discipline with its own goals, techniques and especially educational programmes. Computer Science is the most common name for this

subject, and I shall refer to it by this name, but other names are used, for example, Computing Science, Information Science, Systems and Communication Science.

Again I shall start with some facts to illustrate this development. Results of surveys published last year indicate that in the U.S. in 1964-65 there were about 150 colleges of various kinds giving degrees in Computer Science, or whose content involved a substantial amount of Computer Science ; the figure is now around 400 and includes all the major U.S. universities. In this country a report issued last year stated that eleven British universities and six colleges of technology gave B.Sc. degrees either wholly in Computer Science or joint programmes (usually with mathematics or electrical engineering). I know of three more not on that list and of others who are planning such programmes. A recent advertisement by a well-known bank offered scholarships to enable students to get a degree in Computer Science. At least one examining body allows Computer Science as an A-level subject.

Many recent issues of scientific journals have included articles about Computer Science education ; they have contained surveys, proposed syllabuses, attempted definitions and recommended degree programmes. A very good book with the title *University Education in Computing Science* has just appeared including many facts and opinions. Several conferences devoted to the topic have been held ; both the U.S. and British computing societies have very active committees considering the problems of education in Computer Science.

The Science Research Council, which supports and gives grants for scientific research, has ten separate committees to cover different areas ; one of these is for Computing Science. The U.S. National Academy of Sciences has established a Computer Science and Engineering Board, showing incidentally in its title that Computer Science is involved as much with engineering thought as with scientific theory.

What is Computer Science ? It is easier to say what it is not. It is not a subject based on the binary system, an impression often given in popular books and television programmes. I have more than once heard computers given as the reason for teaching the binary system in schools. Teaching different scales of notation is valuable for the demonstration of a mathematical concept, it is no basis for teaching computing principles and programming. The two important ideas to bring out in first courses are the concept of a program and the concept of representing data of one kind in terms of data of another kind. It is not a subject based on Fortran—or any other particular programming language. Whilst in a first course it is important to teach a particular language, care must be taken not to leave the impression that computers can only do that which can be expressed in one particular language.

Perhaps the only possible definition of Computer Science, and indeed of many other subjects including mathematics and electrical engineering, is that it consists of that collection of activities which are studied in Computer Science departments (or Mathematics departments, or Electrical Engineering departments). There is now a lot of material which is concerned with computers, with computational processes, with methods for storing data, with methods for retrieving data, with languages for communication with machines, with problem solving techniques, with electronic circuit design, with modelling processes, with models of computers, with efficiency of processes, with communication links and networks. None of these individual areas is a new area of study, but all are being studied now with more depth and interaction than in the past. All are connected with the representation and processing of information in some way, be it numbers, lists, formulae, English text or anything else, and with the different ways information may be stored and manipulated. Many individual studies could certainly be classed as Mathematics, others as other subjects, but it is misleading to think of the whole study as being a branch of

Mathematics, or of Electrical Engineering, or of any other subject. Computer Science is an amalgamation of theory and practice to a level probably not present in any other subject.

Whether these diverse sides can be viewed as parts of some coherent whole remains to be seen, whether this new subject contains enough firm foundation of lasting benefit also remains to be seen. I strongly believe it does, there are many others of this opinion and of course, some with the opposite view. I think we are a lot clearer now than we were only three years ago, but still some way from a firm demonstration that we have a new coherent discipline. However, I believe time will indeed show this.

Now to be more explicit about the subject. There seem to be three sides, a theoretical side, an engineering and design side and a methodological side. Let me now discuss each of these in turn.

On the theoretical side there are several different facets. I have mentioned before the problems of data structures and their representation. Even though our computer can only represent its data in a simple manner (perhaps as a single list of numbers) yet our actual problems have much more structure. Does this mean the computer cannot help us ? Of course not, it is rather easy to think of ways of representing these more complex structures as list of numbers, we can use certain numbers as markers, counters, or pointers to other information. Indeed there are many ways of doing this, and this creates the problems. Is this way better than that way ? How easy is it to retrieve information, or to change it ? How do we define one structure in terms of another ? What is a suitable language for expressing this ? What are the consequences of storing information this way ? To tackle this kind of problem we need a framework in which we can make precise these vague questions, i.e. a mathematical theory.

Then again we have language problems. We must tell computers what to do and also tell each other what we

have done. As I indicated before, we can, to a large extent, design our own language ; having done this, a compiler must be provided to translate from this language into the machine's own language. Many such compilers have been written. The first compilers were rather ad hoc devices and only computer experts could produce them. Now we understand a lot about the constituent parts of a compiler, about possible techniques and the relations between them, we know restrictions on language design that allow the writing of efficient compilers, we know suitable data structures and languages in which to write our compilers. All this is now being taught as standard material to undergraduates. However even with all this knowledge one does not automatically produce a good compiler, the knowledge has to be used in the right way. But the understanding is there, enabling better systems to be produced than before and much more easily.

Theory can be applied to particular problems. Thus we can try to prove results such as that one program does the same thing as another program, but more efficiently ; or that the results of this program satisfy some particular property ; or even that no program can exist to solve this problem. Several results such as this last proposition are known and perhaps the most easily explainable is concerned with algebraic expressions. Suppose we provide ourselves with a system for doing algebra in a computer (several such exist) we then might wish to write a program to test if two expressions were equal (in the sense that $(a+b)^2=a^2+2ab+b^2$). It has been proved that, if our algebraic expressions are allowed to contain trigonometric and logarithmic functions, no such general equality testing program can exist. Several similar results are known, including some in the theory of languages and compilers mentioned earlier. In simple cases we can indeed prove facts about particular programs, but we are far from being able to consider most of the programs which arise in practice. We need other theories and more powerful techniques.

On the theoretical side there has also been much work on various models of computation. Theoretical computers, which model some or all of the behaviour of actual computers, are defined and then investigated. For example they can be classified according to their structure or according to the kind of problems they can solve, or the relations between different computers may be investigated.

I have now discussed some of the theoretical problems of Computer Science ; this side is important but it is only half the story. So let us now turn to the second aspect, the engineering and design side.

By this I do not only mean the actual design of computers and their components. Rather I mean the use of the kind of theory we already have and of know-how gained by experience in order to design and implement useful systems. These systems may be a complete computer with its many interconnecting units, a compiler, an operating system, a simulation of a factory or any task performed on a computer.

It is very pleasant if we have a theory telling us how to write our program or design our system. But we usually do not. Still the system has to be designed. Here good design principles and experimentation come in. We are not sure which method will work, or which combination of units is best. So we try various ways, perhaps by writing different programs or perhaps by simulation. We try out methods. How do we choose representative data on which to test our program ? How do we draw conclusions from our results ? How does one analyse the problem in the first instance ? What particular data structures or programming language should we use ? We have to go by the experiences of others, collecting evidence for best approaches and perhaps at the same time carrying out theoretical investigations guided by our experiences.

On this engineering side we must develop good approaches, we must teach, as far as we know it, the right way to tackle large complex problems, we must

teach good experimental techniques. Physics or Chemistry students, for example, are taught these techniques ; it seems equally important to establish in computer users good programming philosophies. Indeed this is essential if a user's results are to have any validity as it is easy to get answers from a machine, but difficult to have any confidence that they are right.

As well as this program design side there is also the equipment side. What are the requirements to be satisfied by a computer system ? How can these be met ? Where should compromises be made ? Economic considerations will play an important part. Is this extra sophistication worth the price ? If we use this design philosophy what will the consequences be and what will their effect be on users ? How can we implement this idea in an economic way, or mass produce this device ? This side is clearly important, but as I have a rather limited knowledge of the electrical engineering side I shall have to leave my remarks at this vague level.

I have now discussed the theoretical and the engineering side of Computer Science, so now let us turn to the third side, the methodological side. In Computer Science we can recognise common structures or processes which come perhaps from some particular application area or perhaps from several such areas which have a common pattern in the way they use the computer. Studies of these particular processes can therefore be undertaken.

One such methodology is numerical mathematics, the study of processes for solving mathematical equations of various kinds. We must study both their theoretical properties and also their computational properties, taking into account the fact that computers only compute with numbers in an approximate way. This, for scientific computer users, is a most important study required in many application areas, however it is not basic to Computer Science. Computer Science techniques and theory are of interest and relevance to numerical users, but they are of much wider relevance. It is none the less

an important methodology, and Computer Scientists must know something about it.

Another methodology is concerned with the techniques used in business data processing. How do we keep information on files, how is it referred to, what about its security both from accidental damage and also from unauthorised access ? A large amount of practical experience and wisdom has been built up and should be known to Computer Scientists.

List processing techniques is another example, this time of a particular kind of process which finds application in many areas. Often a particular task can be expressed most easily in a language which allows operations on lists of symbols. These symbols could be of different kinds, perhaps numbers, perhaps characters, but very frequently they are themselves names of other lists, thus giving rise to a complex entity referred to as a list structure. In many cases these lists will need to be changed, perhaps have extra symbols added to them or deleted from them. Some kinds of programming languages are more suited than others for this kind of application. Different ways of representing such list structures in computers are known and experience has been gained in many ways.

Another important area, and one which has grown up with computers, is the subject referred to as Artificial Intelligence, or Machine Intelligence. This is a vague term covering many different problems, but these problems usually involve complex situations in which some kind of pattern recognition techniques are needed to pick out relevant facts. This then leads to the machine forming one or more plans of action, trying these out and if unsuccessful trying something else, forming some other plan, or eventually perhaps just giving up. Specific investigations include the simulation of human behaviour in chosen problem solving tasks, the design of intelligent robots, the playing of games such as chess, the proving of mathematical theorems, the design of a machine that understands instructions given in English, the co-operation

of a machine and a human in solving suitable problems. All these are activities normally thought of as requiring intelligence of some kind, the problems are diverse yet some common useful methods have been recognized. One important side to this activity is to investigate ways whereby machines can learn to improve on their performance. Several such tasks have been programmed, for example a draughts playing program which learns by the games it plays and robots of various kinds which learn to avoid obstacles.

There are several other important methodologies, for example linguistics and simulation, but I do not intend to discuss these further.

The main point of this lecture is that the emergence of Computer Science as a new discipline is with us now, and that in years to come will be regarded as an important event in Computing. However this is not to say that the kind of work I have been talking about has not gone on since the first days of computers. Indeed it has, and perhaps it is interesting to look at some early articles in the computing literature.

In Volume I of the Journal of the Association for Computing Machinery, issued in 1954, we find an article with the title ' On single versus triple address computing machines '. Here simplified models are set up of two computers which refer to their data in different ways, and then economic comparisons made. Other early issues include a very comprehensive article on sorting techniques, this compares many different methods for sorting information stored on machine accessible files according to some ordering criterion. There is also an article on optimising programs showing how the efficiency of a program on a particular type of computer can be mechanically improved. The first volume of the British Computer Journal, issued in 1958, contains an article on parallel programming, introducing an idea now familiar to all computer people in which is explored the possibility of computers performing more than one operation at a

time. Most early articles are straightforward descriptions of particular applications, but articles suggesting the use of new programming languages soon occur.

Computer Science research has been with us from the first days of computers, but of course it goes back much further than that. Indeed the physical existence of computers is not necessary for much of the work I have been describing, work concerned with information structures and processes. We have to go back at least to Euclid for the concept of a program—for what else is his various constructions for performing given tasks using only a pair of compasses and a straight edge ? Much theoretical work on the definition of a computation, and on what could be computed and what could not, was carried out before the first electronic digital computer was switched on. But there is no doubt that the coming of the computer has given this kind of work a tremendous impetus, so much so that now it is recognized by many that it is no longer adequate to embrace it within existing subject boundaries. There has been a great increase in activity over the last few years. Good books, essential if teaching of the subject is to spread, are now appearing, where previously we only had a mass of programming manuals. Reasonable curricula for University courses have been published and are being given serious attention by various bodies, although it is certain that these first syllabuses will change. However we have made a start.

What of the future ? Clearly in spite of the increased activity much remains to be done. In this country there is a need for people. I can myself see the need for educators and researchers and judging from the output of commercial computer manufacturers, they too have a desperate need for the right kind of person. Every scientific subject needs some people willing to become familiar with Computer Science and look for its applications in their own subject. Computers provide us with a powerful new tool, and create a need for more people to consider the implications. It is hard to get good people

and the money to support them for the purpose of thinking about how to solve problems, rather than spending all their time solving given problems. In the U.S. there are now several very good institutions with Computer Scientists performing research and providing good education. These departments, instead of the handful of people they had some years ago, now may have 20 or more people devoted to Computer Science type work. In this country it is still not easy ; we have potentialities in several places, we have some departments and educational programmes. Many of these I believe even the originators would agree are rather a mixed bag of topics lacking coherency and with gaps. I do not blame the institutions in any way, the time to start is now and things will be that way when one starts something new. Gradually we shall see new blood coming into these departments to fill the gaps and to contribute to the setting up of better integrated programmes.

I believe the situation will be much clearer in, say, six years time when modification and revisions have taken place and when we have had more experience. Then the existence of this new subject will be in no doubt.

There are several problems of great practical importance being worked on now which will be solved by then. Perhaps the most obvious at the moment is concerned with the design of efficient operating systems and with better communications facilities, both more flexible equipment and more flexible languages. Memories are now becoming available, which are capable of storing millions of items of information, information which can be retrieved at random and in the order of a millionth of a second. This could have a big effect on the future ; we do however need new concepts and theories so that we may organise this kind of memory. The conversational use of computers, that is the ability to have immediate access to a computer in order to use it as a direct extension

of one's own thinking, is now available to increasing numbers of users, but is still in an early stage of its development. Much more flexibility and consequently more power will come. Within the next decade we shall see the emergence of a few robots capable of quite sophisticated interactions with their environment. This will involve much Computer Science research, the organising of large amounts of data and the finding of relevant facts. It will involve computer engineers and Computer Scientists working closely together.

It is clear that Computer Science will flourish more in some centres than others, although the basic ideas will be available everywhere. However there is one obvious requirement, and that is the provision of a suitable computing environment. Computer Science in its demands on computer time is like many other subjects : we have projects requiring no computer time, projects requiring moderate amounts of time and projects requiring vast amounts of time. However there is a big difference, the computer and all that goes on around it is of central importance to our work. Computer Science research will flourish even less well than other subjects in an insufficient environment, for example with a badly overloaded computer or a badly administered services side, even though the work itself might not be making much demands on computer time. Whether or not our research interests need much computer time we shall need quite large amounts of time if we are to fulfil our educational function.

A recent U.S. report puts it this way. It is hard to see how a master's programme in Computer Science can be conducted without the use of a centre of such quality that its costs would be about one million dollars a year (or a Ph.D. programme with costs of three times that amount). These figures include rental of the computer or computers. I believe we can provide good facilities at much less than these figures—indeed we must do so—but the principle implied here is right.

In this lecture I have tried to give you some feel for this new subject—Computer Science ; for its present state and for my views about it. There are areas of Computer Science about which I should have said more, perhaps there are some about which I should have said less. In places I have been vague, but to have said more would have involved a whole course of lectures. I believe these are exciting days, days in which we see the emergence of a new discipline, a discipline which interconnects with all others in many ways, yet a discipline which is developing its own motivations and ways of thinking.

I look forward to seeing what the future will bring. Perhaps the best way will be for me to attend the inaugural lecture of some future Professor of Computer Science and I shall be very interested to hear what he will have to say.

GOMERIAN PRESS, LLANDYSSUL