COMPUTER PHYSICS COMMUNICATIONS 2 (1971) 1-10. NORTH-HOLLAND PUBLISHING COMPANY

THE DELPHI-SPEAKEASY SYSTEM. I. OVERALL DESCRIPTION *‡

Stanley COHEN
Argonne National Laboratory, Argonne, Illinois 60439, *USA*

Received 23 November 1970

DELPHI is a modular computer program designed to aid scientists in their day-to-day research. Originally intended to serve the field of nuclear shell-model studies, it has since become a generalized program for theoretical physics. A user-oriented language called SPEAKEASY, included to make the system easily used, is itself becoming a powerful research tool. This paper, the first in a series, is intended to provide a general description of the program. A brief description of nuclear shell-model calculations is included.

1. INTRODUCTION

The following is an overall description of a project undertaken at Argonne National Laboratory to construct a new tool for theoretical physics. This tool is a new approach to the use of a digital computer as a direct adjunct to the everyday work of a group of active research scientists. In constructing the highly modular computer program described here, the primary objectives were ease of use and ease of growth. That both of these goals have been satisfied is already apparent in the rapidity with which it is being adapted to new types of calculation and in the diversity of the studies that have already been carried out within the system.

Detailed explanation of each part of this system of programs would result in a paper of excessive length and complexity. This paper therefore provides only a description of the functional components of the system and an indication of the way in which they are merged into a powerful new facility. Later papers dealing with the various parts of the system are planned.

The original motivation for the construction of the DELPHI system was the need for a computer program to carry out nuclear shell-model calculations. Such calculations are characterized by the vast amounts of information generated in intermediate stages of the calculation. The many observable quantities can be calculated from these intermediate results. It was therefore necessary to provide a flexible and expandable program that could manipulate libraries of stored information. The difficulties associated with nuclear shell-model calculations are described in the first part of this paper. The rest of the paper is devoted to the technique used to solve these problems in a systematic way. The approach is a general one and one which can be adapted to other applications.

Most present-day scientific programs have been designed to carry out restricted classes of problems with a minimum of execution time for each production run. Such "efficient" programs are intended to minimize the amount of computer time used in a study. In contrast, the project to be described here is intended to minimize the amount of human time involved in formulating a new problem and carrying it to completion. One of the most common questions asked about a computation relates to the amount of computer time required to carry out the particular calculation Much more meaningful questions would be ones that ask about the time required from conception of a problem to its completion and the overall computer time used.

The answers to these questions are not, however, unrelated. The DELPHI system has been designed to quickly provide meaningful results to calculations. That it also does so with a minimum use of computer time should not be surprising. In minimizing the human effort to direct a calculation, one also minimizes the chance for errors. Trivial errors are expensive in computer time as well as in human time.

* Work performed under the auspices of the U.S. Atomic Energy Commission.
‡ Based on a presentation at the Conference on Computational Physics, UKAEA Culham Laboratory, Abingdon, Berkshire, England (1969).

The DELPHI system is an open-ended evolutionary system which is a repository for algorithmic computational tools. It is easily alterable and it can easily be expanded to include features needed to explore new avenues of investigation. Its built-in growth capacity provides users with an ever-increasing set of tools for carrying out research. At the same time it has been designed to cater to the needs of scientists relatively unskilled in the use of digital computers. The facilities provided range from documentation techniques and large libraries of coherently-designed algorithmic subroutines to automatic means of managing the dynamic storage of data. They also include a user-oriented language called SPEAKEASY, designed to make short investigations even more direct. In all cases the design specifications are such as to aid the scientist in obtaining answers to his problems quickly.

2. THE NUCLEAR SHELL MODEL

This is a description of a computer-oriented study restricted to a particular field of theoretical physics. This project has been under way for several years and has grown far beyond its original design specifications [1]. A large number of research scientists have been involved with this system at various times. During the period of its existence many dozens of diverse calculations have been carried out [2], and its existence has had a direct impact on the type of calculation carried out in the entire field of study.

The basic field of application is computations of nuclear structure. This is meant to include an understanding of the properties of the low-lying states of various nuclei and the transitions between such states.

As an example of the type of data considered, fig. 1 shows a typical nuclear level diagram in which the level structure and some transitions involving low-lying levels of a particular nucleus are displayed. A variety of labels are associated with particular levels; some of these (such as spin, isotopic spin, and parity) do not depend on the form of interaction between the particles (i.e., they are geometrical) and others (such as the energy and electromagnetic moments) do. In addition, there are the transitions between levels. These include γ transitions, β transitions, and transfer reactions. These transition rates also depend on the details of the form of interaction between the particles.

The task of the theorist is to attempt to understand the observed information, to produce explanations for the apparent anomalous situations, and to predict possible results for new experiments. The nuclear problem is not an easy one, however, since we are in principle solving a problem involving a large number of interacting particles (the neutrons and protons of the nucleus). The difficulties are compounded by the fact that the form of interaction between the nucleons is not really known except in general terms. It is a short-range, attractive force which is spin dependent and apparently has a very short-range repulsive core.

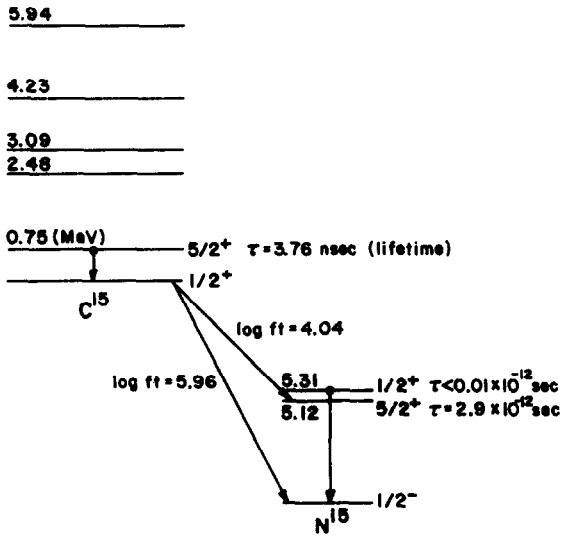

Fig. 1. A typical nuclear level diagram illustrating the type of data involving low-lying states of nuclei.

Even if the forces were known, it would not be possible to directly translate this knowledge into theoretical predictions about nuclear experiments because of the large number of interacting particles involved. For this reason it is essential to have a model that can be used to translate from a specified form of interaction to observable quantities. The nuclear shell model is such a model.

Empirical evidence leads one to suspect the existence of particularly tightly bound configurations ("closed shells") of nucleons analogous to those of the inert gases in atomic cases [3]. The filling of the shells appears to follow a scheme somewhat like that of the chemical case, independently for the neutrons and the protons in the nucleus. The order of filling of the single-particle levels is shown in fig. 2. Particularly stable nuclei exist wherever the number of neutrons or

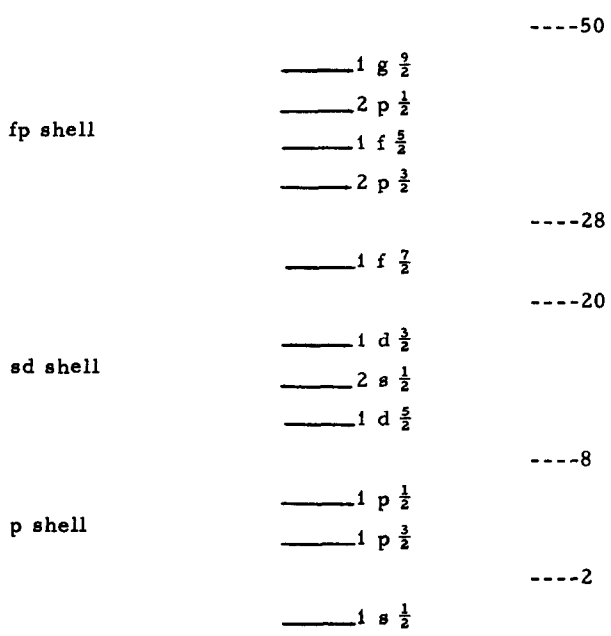

Fig. 2. The order of single-particle levels in the nuclear shell model. The numbers on the dashed lines at the right correspond to the "magic numbers".

protons corresponds to a completely closed shell. This number is one of the so-called "magic numbers". "Doubly magic" nuclei (those that have closed shells for both neutrons and protons) are even more stable. Examples of doubly magic nuclei are:

2 neutrons,	2 protons	α particle, ^{4}He;
8 neutrons,	8 protons	^{16}O;
20 neutrons,	20 protons	^{40}Ca.

To explain the behavior of the properties of the ground state and low excitation levels, we assume that we can treat closed shells as inert cores - i.e., most of the nucleons in the nucleus are to be ignored. The levels available to the extra nucleons can then be limited to a manageable number, and we can work out the consequences. The interaction between the active nucleons is restricted to the form of a two-body interaction, and the predicted values for experimentally observed quantities are obtained.

It should be understood that the assumed interaction is *not* the interaction between free nucleons. Rather it is an effective interaction within this model. This is obviously true since the fictitious inert core is made up of nucleons which also interact. The connection between the effective interaction and the interaction between free nucleons is a separate problem and is also of great interest to nuclear physicists.

A shell-model calculation consists of several steps: (1) a suitable basis is set up to describe all of the possible configurations for the model of interest, (2) for a specific form of effective interaction, the energy matrices of the model hamiltonian are constructed, (3) these matrices are diagonalized, and (4) their eigenvalues and eigenvectors are used to obtain the theoretical predictions for experimentally observed quantities.

If a fit to experimental information is being attempted, the discrepancies between theory and experiment may be used to alter the choice of interaction and an iterative procedure may be carried out.

3. DIFFICULTIES

Having specified the steps in a typical nuclear shell-model calculation, let us now look at the difficulties involved. We restrict ourselves to the seemingly modest problem of an inert ^{16}O core and the three allowed single-particle levels in the sd shell. The basis can be expressed in terms of the allowed ways of distributing neutrons and protons in these levels. Because of the $2j + 1$ projections of angular momentum j for each level, there are twelve possible quantum states allowed for each nucleon in this system. The exclusion principle somewhat restricts the number of ways of forming multi-nucleon configurations by preventing two identical nucleons from occupying the same state.

By simple combinatorials, we obtain the number of configurations allowed for various numbers of neutrons and protons in these levels. Fig. 3 shows the approximate numbers of configurations involved in bases in the sd shell.

One sees that even this rather restricted model leads to an extremely large number of ways of arranging the nucleons in the allowed levels. It is true that various symmetries can be used to select from these arrangements, but such symmetries reduce the sizes of problems only by factors of 2, 3, or 10. One must face the fact that it is a very large task just to enumerate the complete basis. To carry out calculations making use of the basis leads to a sizeable problem in information management.

Full sd-shell calculations can be carried out by hand for simple cases involving only a few nucleons. Larger calculations involving several hundreds of configurations can be done by com-

N_n \ N_p	0	1	2	3	4	5	6
0	1	12	66	220	495	792	924
1	12	144	792	2 600	6 000	9 000	11 000
2	66	792	4 000	14 000	30 000	52 000	61 000
3	220	2 600	14 000	48 000	109 000	175 000	203 000
4	495	6 000	30 000	109 000	245 000	390 000	460 000
5	792	9 000	52 000	175 000	390 000	630 000	730 000
6	924	11 000	61 000	203 000	460 000	730 000	850 000

Fig. 3. The approximate number of ways of distributing N_n neutrons and N_p protons in the sd shell.

puters. Problems really exploring the sd shell are not possible even with the largest machines. To treat a real nucleus with several single-particle levels and several particles outside closed shells by this model is therefore out of the question. The development of approximate methods for solving very large shell-model problems is therefore a must, and the study of the validity of such approximations is of great importance. (Note that these are techniques to obtain approximate solutions of a model of the nucleus and only indirectly approximations to the understanding of a nucleus.)

4. NEEDS OF THE RESEARCH SCIENTIST

The nuclear scientist must be provided with the means of carrying out large-scale shell-model calculations. Because of the size limitations already mentioned, he must also be given the means of developing approximation schemes for solving even larger problems. If done coherently, these approximations can be tested by solving certain problems both exactly and approximately and comparing the results. The scientist must also be provided with the means of carrying out other related large-scale calculations. Finally, the scientist must be provided with the means of formulating and carrying out exploratory types of calculation easily.

This latter need is the one of primary concern to most scientists. By its nature, research implies a constant search for new approaches or new approximations. The ideas are often transient and will vanish if not explored fully. More importantly, if it is difficult to explore new concepts, then the scientists must continually weigh the effort involved in testing new ideas against the possibilities of their being fruitless. The use of a computer in conventional ways greatly increases the tendency of scientists to avoid uncertain exploratory calculations; too much effort and time is involved in setting up new programs.

The overall object of the system being described is to use the power of a computer for the benefit of the scientist in his daily research. It is intended to provide facilities for all of the above-mentioned needs, to do it in a "natural" way, and to develop along with the scientist.

5. THE APPROACH

Several goals were set forth in designing what, as could be seen from the start, would be a large system of computational programs. The most important of these was related to the idea of making use of the computer as a repository for the data generated. Because of the massive amounts of data involved, it would not be possible to manually direct its storage and retrieval. It was also clear from the outset that the system would be constructed piecemeal and that parts would eventually be replaced by more efficient parts.

A modular system was therefore designed with provisions for additions and changes. This capability of evolutionary growth has provided for a coherent development of a new tool for scientists. By now the concept of the system has been greatly enlarged so that the repository includes not only the numerical data but also the computer programs and their documentation. The addition of these features has provided the needed facilities for a true user-oriented system.

While the system is being developed more completely, many exciting new features will be added. The implementation of user-oriented languages is now feasible and user-designed compilers will soon be available within the system.

In the sections that follow, the computational structure of the system is discussed in some detail. While such details are necessary to understand the operation of the system, it should be understood from the outset that users of the system are not really concerned with this structure - rather, they are concerned with the operations that they personally must perform. These relate to the form of communication, i.e., the languages. In all such cases the orientation has been towards providing users with easily-understood and easily-used facilities. The whole approach has been oriented to the trained scientist, willing to use a computer to get results but not really interested in understanding how the computer works or even the details of the programs.

Fig. 4 shows a schematic view of the computational parts of the shell-model system. This block diagram is not a flow chart - all the arrows point both ways! The connecting lines represent paths of information flow; the boxes are major computational modules.

The central box, labeled SYSTEM STORAGE, represents the repository of the system. This repository is a complete subsystem in itself. It is designed to store and catalog data fed to it and to retrieve such data. The retrieval can be by reference to cataloging information. The entire data base for the system, including the catalog, is contained in SYSTEM STORAGE.

In normal use of the shell-model system, the

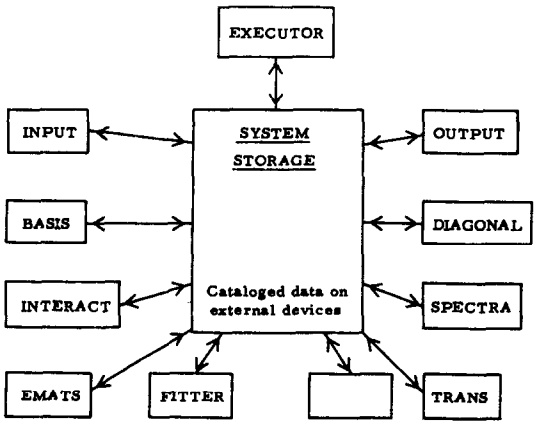

Fig. 4. A schematic view of the computational parts of the nuclear shell-model system.

collection of data in SYSTEM STORAGE carries with it information that describes not only the type of data (integer, real, literal, ...) but also the structure of the stored data (i.e., 2×3 matrix, 40-component array, scalar, ...). It is therefore possible not only to retrieve specific data but also to retrieve all of the descriptive information associated with it. Such data are in a sense program-independent. Programs designed to use the descriptive information along with actual data can operate on any data in SYSTEM STORAGE. This communal form of storage means that major programs can be replaced without disturbing the data base of the system.

All of the stored information is placed on external devices such as disks or tapes. The choice of the particular method of storage is of concern only to the subsystem. The entire subsystem could be replaced by any other subsystem, provided only that it matches the clean interface entries that have been defined. A list of these entries and their operations completely defines the subsystem as far as its function within the larger system is concerned. This clean, modular separation, as will be seen, is the general approach taken in the entire development.

The boxes surrounding SYSTEM STORAGE represent computational programs. (INPUT and OUTPUT will be described separately.) Each of these represents a module designed to carry out a specific type of numerical calculation. They can best be described as mathematical operators. Each is designed to operate on data in SYSTEM STORAGE and produce new data for that repository. Most importantly, they have no communication with each other except through SYSTEM STORAGE.

This communication scheme provides the means for evolutionary changes. New modules may be freely added to the system and old ones can be modified. It is only necessary to be careful in the means of selecting the information from SYSTEM STORAGE and the cataloging of newly created data.

The selection of the data to be operated on is controlled by the executive routine EXECUTOR, which makes such decisions on the basis of the stored catalogs. The executive routine also makes decisions relating to the cataloging of newly created information. All of these decisions are passed on as commands to the SYSTEM STORAGE package.

The final role of the executive routine is to cause a particular computational program to be invoked. In a mathematical sense, then, it has (1) defined the operator (the particular program),

(2) defined the operand(s) (the selected input data), and (3) named the resultant(s) (the output cataloging controls).

Two other special modules exist, one labeled INPUT and the other labeled OUTPUT. These two modules enable the user to communicate with the overall system and to obtain results from the system. The complete separation of these facilities from the computational programs greatly increases the flexibility of the system, New input and output devices can be attached to the system by defining new modules. The fact that all output involves SYSTEM STORAGE also ensures that all meaningful information is available to the repository.

The executive routine receives its instructions from SYSTEM STORAGE. It is therefore possible for directive information to be stored in the repository for later use. This, coupled with the ability of the executive routine to read and control the catalog, provides this routine with a great deal of power in directing computation. In the present implementation, the role of the EXECUTOR is still rather limited. The potential for growth of the present system is most obvious in this routine. The true growth capability is, however, built into the structure of the overall system.

Fig. 5 shows the form of input to the system as it exists today (in the foreground) and as planned (in the background). In each case the rather natural form of the language should be noted. Free-field keyword structure is provided, and every attempt is made to provide an input format that is familiar to the scientist. The rather simple example of the present version illustrates the most straightforward type of run.

This version represents little more than a simplified directive program. The % followed by a name directs the executor to call a particular computational program into execution. The information that follows constitutes the input data for that program. The rather concise example given here is all the information needed to carry out a complete calculation. Its conciseness and the fact that the form of input is so similar to the normal notation of the field favors error-free operation. This is true even though the actual calculation involves a rather large number of separate programs. Diverse types of calculation can be carried out within the system by selecting the correct sequence of calls to the various major programs.

The upper part of fig. 5 shows an alternative form of input presently under investigation. This is also a keyword free-field form, but it is not a

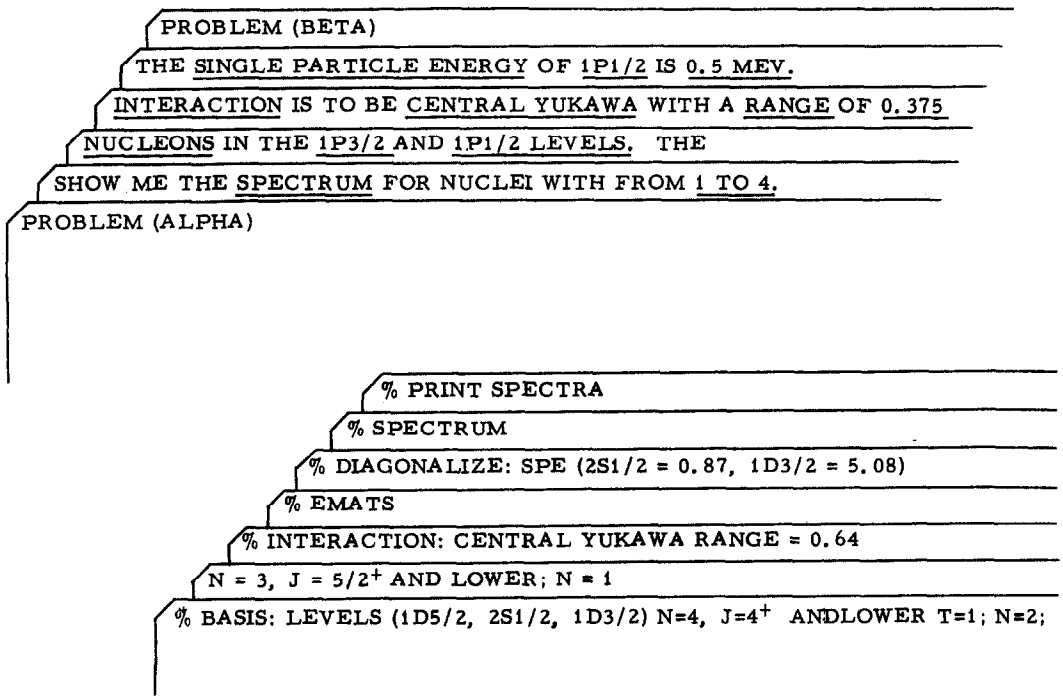

Fig. 5. Typical input. The cards shown in the foreground illustrate the form of directive input currently used. The background illustration is of an inquiry form being investigated for future use.

directive sequence. Rather it is better described as an inquiry. Once read by the executive program, this input sequence could be converted into one similar to the present input form. This in itself would be of interest, but a more powerful feature can be added. Since the catalog of results is available to the executive routine, it can ascertain whether or not certain parts of the calculations have already been done, and thus avoid recomputation.

Indeed it can even attempt to carry out alternative computations to obtain the result by some means other than the obvious direct path. Such alternatives exist even in our present system, but the choices are often made on a purely subjective basis. The ability of the system to investigate alternatives on the basis of accumulated information is of great importance. It could eventually provide the truly important step to a machine capable of learning.

One major defect in most large-scale system developments is the rigid structure for computational programs. It is often easier to reprogram from the beginning than to adapt an existing program to such a system. The particular form of a completely modular long-term storage package of SYSTEM STORAGE makes it easy to design special interface routines to any existing program. Once properly interfaced, such programs can operate under control of the executive system, i.e., as part of the overall system.

The modular form of storage control also enables one to interface any program directly to the data base for operation outside of the system. This means that the entire process can be avoided both for exploratory calculations and while testing new modules. The ability to handle programs in a variety of ways greatly eases the development process.

6. THE STRUCTURE OF A COMPUTATIONAL PROGRAM

Any program that is modified to operate under control of the executive routine will function within the operating system. The constraints for such operations are minimal, and programs of this sort can be and have been incorporated in the present system. However, ease of construction of new major programs is considered a part of the overall objective of our systematic approach. The basis concepts of modularity have been applied to this phase of the problem with rather interesting consequences.

In developing a modular basis for actual computation, the approach evolved into the construction of "clean" subroutines - i.e., ones designed to resemble mathematical operators as closely as possible. A special storage technique called NAMED STORAGE [4] was developed for this purpose and has proved to be extremely powerful. NAMED STORAGE is a specially designed subsystem which is capable of storing data for retrieval by a reference name. Sets of data, each with its associated structure-defining information, are stored in a special dynamic-storage array which has automatic overflow capabilities (This is usually referred to as a virtual memory.) The task of the NAMED STORAGE subpackage is to retain information assigned to it and to be certain that information currently in use is actually in core when needed. This general purpose subpackage is in fact one of the most important units of the overall system.

Each set of data stored in NAMED STORAGE carries with it information about the type of the data (e.g., whether floating-point, alphabetic, or the like) and in addition a description of the structure of the object (e.g., whether it is a 10×10 matrix, a 300-component array, or some other). It is therefore possible to construct operators to operate on named quantities and produce named results. The dynamic-storage scheme makes it possible to decide during execution how to assign storage.

Examples of such routines, called manipulators, are shown in fig. 6. It is important to realize the consequences of having the generalized form of storage associated with these routines. A complete set of normal subroutines does not have the true modular independence available here, since the sizes of objects must be communicated explicitly along with information related to their structure.

In the first example, in fig. 6, for instance

```
CALL MULT('A', 'B', 'C')
```

means multiply A by B and call the answer C.

Manipulators

```
CALL MULT ('A', 'B', 'C')
CALL ADD ('C', 'B', 'D')
CALL PRINT ('A')
-----------------------------
CALL SAVEAS ('PSI', 3, 'A')
CALL IREAD ('PSI', N, 'B')
-----------------------------
CALL JPLUS ('A', 'A2')
```

Fig. 6. Typical manipulative routines used in computational programs of the system.

The general-purpose manipulator MULT is written once and for all (hopefully). In this routine, all of the ideas about the multiplication operation are formulated. It can incorporate as much as desired and can be modified later to include additional meanings for the operation. Such extensions increase the capability of the overall system. The routine MULT can be designed to multiply matrices or arrays or numbers or even literal quantities (if that means something to the users of the system).

Since the descriptive information is available, the manipulator can reject the idea of multiplying a 3×4 matrix by a 12×7 matrix. Instead, it can terminate execution by calling a standard error routine.

The other sample manipulators are also straightforward. The second set corresponds to facilities interfacing NAMED STORAGE and SYSTEM STORAGE. The last set illustrates a special manipulator designed for the shell-model problem. It applies the angular-momentum raising operator to a Slater determinant named 'A'.

Once written, each manipulator is to be viewed as a closed box just as the SIN and COS functions are normally viewed. The ramifications of this will now unfold.

Any manipulator can use other manipulators internally. It is therefore not necessary to repeat a section of programming if it is a common and clearly-defined operation - it is better designed as a manipulator.

The operation of a manipulator can usually be described in a few short sentences. This description in terms of what the manipulator does is all that is needed by a scientist attempting to construct a new computational program. His task becomes one of collecting the basic operations needed to carry out his computation and placing them in proper sequence; he need not construct all the detailed data-manipulation steps found in most programs.

It is important to realize that the number of really basic operations in any field is limited. For the entire shell-model study, no more than a few dozen are needed over and above the ones common to any scientific study. If the construction of new programs is found to require a particular operator that does not exist, it must be programmed. Once this is done, however, it becomes a new tool and is added to the library of the system. (The manipulator library is available to users in exactly the same form as are the trigonometric functions - i.e., as a call library.)

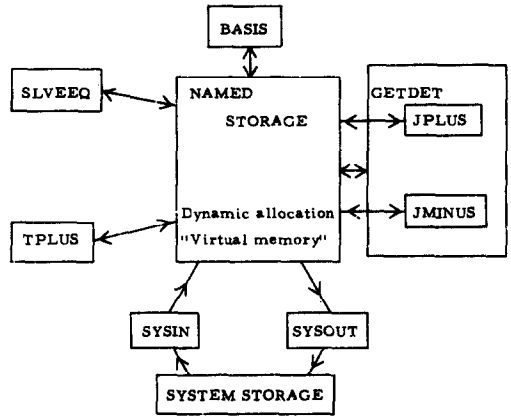

Fig. 7. A schematic view of parts of a typical shell-model computational program. Each box represents a manipulative routine.

The user is therefore able to program his new concept, making use of an ever-growing set of routines designed for coherent use. Since the set of tools is growing, programming entirely new problems is more likely to be the relatively simple task of merging a set of routines than the time-consuming task of extensive programming Fig. 7, which represents one of the shell-model computational programs, illustrates the concepts in a diagrammatic form. Each box represents a manipulator (only a few of those involved in this program are shown). Each manipulator operates on data in NAMED STORAGE and produces new information for NAMED STORAGE. Each is an operator. Several manipulators can be connected together to form a new manipulator.

Special interface routines are provided to transfer information between SYSTEM STORAGE and NAMED STORAGE. These routines, controlled in part by the executive program, perform all of the book-keeping necessary to permanently store and retrieve information sent to SYSTEM STORAGE.

7. SPEAKEASY

The description of the DELPHI system could be stopped here. The user (the scientist) has been presented with a rather straightforward systematic way to construct programs, execute them, and build a library of results. This, in fact, was the intent of the original adventure. A small extension, however, leads one to rather interesting consequences.

Manipulators	SPEAKEASY
CALL MULT ('A', 'B', 'C')	C = A * B
CALL ADD ('C', 'B', 'D')	D = C + B
CALL PRINT ('A')	PRINT (A)
CALL SAVEAS ('PSI', 3, 'A')	SAVEAS (PSI, 3, A)
CALL IREAD ('PSI', N, 'B')	IREAD (PSI, N, B)
CALL JPLUS ('A', 'A2')	A2 = JPLUS (A)

Fig. 8. Manipulative routines and the corresponding SPEAKEASY statements.

Fig. 8 shows the manipulators described previously. Beside each is a statement obviously equivalent to that statement in some sense. Suppose one constructs a special program that can read the input on the right and as a result perform the operation on the left. If one does, one has a new computer language. One designs the operations by designing the manipulators and then uses a standard interpretive program to call them from a concisely written program. A sample program is shown in fig. 9. This SPEAKEASY program evaluates the integrals

$$A_{ij} = \int_{R_{min}}^{R_{max}} \psi_i \psi_j r^2 \, dr, \quad j \leq i \leq 16,$$

where the values of R, ψ_i and ψ_j were previously computed and saved in SYSTEM STORAGE.

The simplicity of the program shown in fig. 9 is deceptive. Hidden beneath those straightforward statements is the entire system that has been discussed already. NAMED STORAGE is directly usable and references to SYSTEM STORAGE are included. The entire mass of stored data is directly available to the user as is the entire

```
R = SAVED (R)
FOR I = 1, 16
P1 = SAVED ( PSI, I)
FOR J = 1, I
P2 = SAVED ( PSI, J )
AIJ = TOTALINTEGRAL (P1*P2*R**2: R )
PRINT ('FOR', I, J, AIJ )
ENDLOOP J
ENDLOOP I
```

$$A_{ij} = \int_{R_{min}}^{R_{max}} \psi_i \psi_j r^2 dr \qquad i \geq j$$

Fig. 9. A SPEAKEASY program to evaluate integrals of the indicated form.

library of modular subroutines. The only real task was designing an interpretive language that actually is readily usable by the scientist. The problem was mostly one of vocabulary, since the operations themselves were already programmed.

This new language, called SPEAKEASY*, is now operational and is rapidly growing in power. Great effort has been expended in providing user-oriented decisions at all places. In designing the new control language, every decision has been made on the basis of the needs of a research scientist. The intent is to provide the scientist with answers to his problem in a form that is as easily read as possible and to provide them as quickly as possible. In all ambiguous cases, the language processor attempts to guess the intent of the scientist and to carry out that operation if at all possible.

The result of having this powerful and yet easily used language available is already apparent. Major problems can and have been solved in a single day. That this is possible should not be surprising. A modern digital computer is after all a very fast and sophisticated device with a vast storehouse available to it. The scientist, on the other hand, is unlikely to utilize entirely new concepts each time he attempts to carry out a calculation. He more likely wishes to make use of some techniques with which he is already familiar.

The more conventional programming techniques require that he enter into details of the calculation that are not really pertinent to the problem he is really interested in solving. He is more likely to make errors in programs because he is forced to look at microscopic details of the program and, more importantly, because he is likely to be thinking about his problem and not the artificial steps he must take to solve it.

SPEAKEASY bridges the gap between the scientist and the computer by forcing the computer to do as much of the mundane work as possible and by letting the scientist concentrate on the real scientific problem. The learning capacity of the system (i.e., the stored data and the open-ended manipulator library) provides the flexibility necessary to significant development of the tool.

A real-time interactive version of SPEAKEASY is also available. This version makes use of an IBM-2250 display station. The combination

* SPEAKEASY is also available as an independent program package for computers of the IBM-360 series. A descriptive manual and the load module are available from the IBM Program Information Department [5].

provides a simple yet powerful means of communicating with a large digital computer. The immediate response in this mode of operation has the advantage that subjective decisions can be made and tested in a short time. Users find that problems that could not be attacked sensibly by any other means can often be solved quickly and easily with this version. Each user of this interactive mode soon tends to view the entire system as a tool especially designed for his particular application. This adaptation therefore has been successful in its goals.

8. CONCLUDING REMARKS

This paper has described a general approach to the use of a digital computer. While discussed from the viewpoint of a specific application, it should be apparent that the concepts apply quite generally.

Micro-modularity, with the implied standardization of structure, is the basis of the DELPHI-SPEAKEASY system. This approach provides the scientific users with an ever expanding box of tools with which to direct a computer. The fact that the modules are designed to be used together means that users are freed from the need to study the inner workings of the routine. The user can easily combine modules into new programs for carrying out specific computations. (The existence of SPEAKEASY shows that such mergings can in fact be carried out automatically.)

The extreme modularity has many useful consequences. Probably the least obvious one is that exploratory calculations will consume less overall computer time. This is true in spite of the overhead which is implied by modularity. The existence of an extensive library of well-tested and easily-merged routines means that much less new programming must be done. The chance of errors is therefore minimized, and fewer actual computer runs are necessary to obtain answers. The fact that only a small effort is required to formulate a new calculation means that scientists are more apt to attempt to use a computer in exploratory ways. It therefore gives research scientists the needed access to computers.

At Argonne National Laboratory, the DELPHI-SPEAKEASY system is being used and developed by a group of scientists. It has already proved its worth, and it becomes increasingly apparent that its existence will greatly affect the future work by this group.

Several papers dealing with parts of the DELPHI-SPEAKEASY system are planned. One hope is that these will stimulate those writing new routines to structure them so that they may easily be adapted into a system such as the one described here. The interchange of programs and subroutines between installations would be much more beneficial if clearly defined standards could be established.

ACKNOWLEDGEMENTS

The system described here has been developed over a period of several years. A large number of people have been involved in its implementation. While it is not possible to give credit for all of the contributions, I would like to acknowledge the contributions of Drs. D. Kurath and J. M. Soper to the shell-model programs. Also I wish to thank Dr. C. M. Vincent for his aid in the development of SPEAKEASY. Mr. Keith Rich of the National Accelerator Laboratory has been responsible for many of the necessary special subroutines used in the system.

REFERENCES

[1] S. Cohen, R. D. Lawson, M. H. Macfarlane and M. Soga, in: Methods of computational physics, Vol. 6, Nuclear physics, ed. B. Adler (Academic Press, New York, 1966) p. 235.

[2] S. Cohen, R. D. Lawson, M. H. Macfarlane and M. Soga, Phys. Letters 9 (1964) 180;
S. Cohen and D. Kurath, Nucl. Phys. 73 (1965) 1;
S. Cohen, R. D. Lawson and J. M. Soper, Phys. Letters 21 (1966) 306.

[3] M. G. Mayer and J. H. D. Jensen, Elementary theory of nuclear shell structure (Wiley, New York, 1955).

[4] S. Cohen, Argonne National Laboratory Report ANL-7021 (1964).

[5] S. Cohen and C. M. Vincent, SPEAKEASY, IBM program 360D-03.3.012 (1969).