Proposal for a

Center for the Study of Information Processing

submitted by

Carnegie Institute of Technology

to the

Advanced Research Projects Agency
of the
Department of Defense

Allen Newell
Institute Professor
Systems and Communication Sciences

Alan J. Perlis
Director, Computation Center
Professor, Mathematics

Edward R. Schatz
Dean of Research

October, 1964
(Proposal submitted April 27, 1964)

Proposal for a Center for the Study of Information Processing

This document proposes the establishment of a Center at Carnegie Institute of Technology for the study of the science of information processing. It was submitted to the Advanced Research Projects Agency in April, 1964, for consideration at their request, in support of a request for funds. It describes in broad terms what such a center would be like -- what problems it would tackle, how it would proceed in its work, and how it integrates into the larger intellectual community of Carnegie Tech.

We start with a broad discussion of the goals of the proposed center and an attempt to place it in context. From this follows a more substantive discussion of the science of information processing, its current state and important problems. We then take up the major issues that condition the organization and operation of the center.

The Goals Of The Center

The continuing, fundamental aim of the center is understanding the nature of information processing. It must be so. We abide by the faith that out of deep understanding of a part of nature arises both control and application. Further, the attempt to produce a science in an area that not long ago was purely a technology -- to have first rate intellects devote their professional lives to it, to train young men to work in it, to raise centers devoted to its exclusive study -- implies a belief that there is much to discover and deep understanding to be had.

Any attempt to define the field of information processing (called by this or any other name) by its inherent subject matter encounters difficulties. Clearly, the field is concerned with information, the systems that process and transform it, and the way it is used to control, integrate and coordinate other systems. But then the fields of control systems, information theory, computer engineering, programming, automata theory, documentation and information retrieval, linguistics, dynamic programming, modern logic, statistical decision thoery, and others as well, come one and all to be included. This is an ungainly collection of subfields, comprising men who know little of each other's work, perspective and goals, with boundaries that have been determined by the history of parent disciplines and the flowering of specific techniques.

No institute can exist to study <u>this</u> field! Selection is necessary: a narrowing of the focus to some coherent wedge. Even the attempt to state the problems of the field, as we do below, adopts some point of view conditioned by one of the subfields, and misstates the total as some god would see it. This is the state of the science and we make no apology for it. It does mean that our statement of basic problems is itself a description of the viewpoint and commitment of the proposed center.

Matters are even worse. All intellectual activities presume information processing; all systems require communication and control. As complexity increases and as performance demands become more stringent, the information processing component of all these activities and systems becomes critical and must be more deliberately studied and engineered. Eventually, recourse must be

had to the range of devices used to build information processing systems -- to computers, sensors, communication equipment, etc. The professionals in the substantive fields begin perforce to be specialists in information processing: conducting research, designing systems, etc. Thus the science of information processing pervades and interpenetrates all other fields.

No institute will range throughout so vast a domain. Of all the substantive fields and regions of intellectual endeavor, only a few will be represented in any center. Selection, mostly historical and fortuitous, will be necessary. More important, there is no way in which research on information processing can be the exclusive concern of a center, the way physicists are the only ones who do physics, or biologists the only ones who do biology. Important research on information processing -- important in its own right and important in terms of the center's own specialized interests -- will occur throughout the scientific community. This is emphatically the case at Carnegie Tech, as we shall see later.

It might be thought that the issue is one between the science and its application: there is the pure science of information processing (to be done in a center) and the applied science (to be done in the various disciplines). In mathematics, for instance, which also pervades all fields, we do talk of pure mathematics and of applied mathematics. But information processing is much more closely tied to its applications. The new ideas and insights that move it towards being a science arise mostly from applications rather than from

any self-generative character of the pure science of information processing. (The same is often argued of mathematics as well, but that is a different story.)

To summarize: Any center is narrow by comparison with the whole. More important, a center is better characterized by its selections of specific problems, methods and attitudes, than by its most general goals. Almost a decade of research in information processing at Carnegie Tech has produced an approach which still seems to its major participants to be extremely productive. The proposed center will be, by and large, a continuation of this approach. We are oriented towards "software" -- towards programs written on digital computers. We are empirical, in that we believe in constructing programs that do things, and in learning about information processing from the difficulties of construction and from the behavior of the resulting programs. We are theoretical, although not so much by a dependency on formal models (such as automata theory), as by trying to formulate the essential nature of information processing. Thus many, although by no means all, of the tasks for which we build programs are selected for the understanding they yield and not for their usefulness in applied work. This particular combination of theory and empiricism stems from the view that the key problems today in the science of information processing are those of discovery, formulation, representation, and immediate generalization -- and that we are not yet at the place of building very elaborate or formal mathematical structures that are significant.

As in all sciences, we devote a major effort to tool building and to the understanding of what tools are needed. Tools, in this field, are largely

programming languages and programming systems. Since the most important application of the science of information processing is the development of these tools of programming, the concern with tools can be made to yield double dividends. Thus, we try to pose our tool building tasks so they shed light on fundamental scientific questions.

The work in information processing at Carnegie Tech has strong ties with the behavioral and management sciences as well as with the natural sciences. We are concerned with how the human processes information, both in isolation and in organized contexts. In fact, one of the other major applications of a science of information processing is to permit finally a deeper understanding of how humans think, learn and decide as well as to supply improved tools for processing increasingly complex problems. The intertwining of concern with artificial systems and natural systems -- whether artificial and natural languages or artificial and natural problem solving systems -- is mutually beneficial, and will continue to characterize the approach of the proposed center.

The Science Of Information Processing

With the general discussion of the previous section to put the work of the proposed center in context, we can now describe what appears to us to be the main problems and avenues of attack in developing a science of information processing.

Problems of Discovery.

Take any area of application of programming -- arithmetical calculation with matrices, numerical control of machine tools, theorem proving in logic, translation programs for programming languages -- any area whatsoever. The first programs that are written for the area are narrow in application and particular in conception. They tend to be just great aggregates of code. They may be impressive, if they do important tasks not previously done by computers or if they exhibit great exercises in ingenuity and tricky coding. But they often do not appear to the outsider to be more than a technological stunt. However, an important process has been started, which is the main path of advancement in our science. Subsequent programs generalize the range of application, or change the internal representation to increase time or space efficiency, or simply clean up the program to make smoother or neater or more esthetic certain of its substructures. Eventually, the structure of the task environment, its relations to other tasks, and its demands for processing become transparent. It becomes abidingly clear how the data structures and interfaces should be represented, and exactly what choices there are with the relevant properties of each. It becomes clear what generally useful processes of intermediate complexity should exist and how to build complete sets of these from which to construct total programs. When this stage is reached, and the important features of the class of programs have been stated in concise terms, it can fairly be said that a real addition has been made to the science of information processing. The scientific knowledge is contained, not in a formal

theory, but in the knowledge, however expressed, of the structure of a class of information processing tasks and the programs that can do them. The validation of this scientific knowledge lies in the fact that operating programs run and accomplish the tasks in the area, and that this know-how for the area can be communicated to new programmers, who can then produce new programs of high effectiveness in the area without themselves recapitulating all the mistakes and grotesqueries of the past. The degree of progress is often obscured because the end product, being a transparent way of structuring the task and its processing, makes it difficult to understand how muddy the issues once were. As in all science, the progress is contingent: the set of ideas generalizes only so far; new tasks cannot be handled, though everyone was sure they were easy; and so on.

This pragmatic process of achieving clarity of structure is not by any means the whole story in the development of a science of information processing, as we shall see. Yet currently and for some time to come it is the main source of new ideas and new starting points. Since the approach at the start seems so often to be brute force, particular, and task bound, it is worth illustrating some important advances which developed this way.

Example 1: Algebraic compilers. The construction of the programs that translate programs of algebraic statements into machine code has followed exactly the routine described above. The original compilers (e.g., FORTRAN) were impressive achievements, but were extremely complex and obscure. They, and the other early efforts, gave birth to the myth that is still with us: the

many man years it takes to construct a compiler. Yet within five years, through the use of parenthesis free languages (so-called Polish notation), the development of notations for expressing the syntax of a programming languages (so-called Backus Normal Form), and the development of translation schemes utilizing push-down lists (or stacks, as they are also called), an important part of the translation process has become crystal clear. This part is the syntactic analysis of the source language (i.e., of the language as written by the user). The problem of the final production of machine code from an appropriate internal representation is not yet so well understood, and is in fact an area of much current concern. Even so, the total task of constructing a new compiler has become only a modest task, as opposed to the horrendous one it once was -- and fortunately so, since the pressure for new languages increases.

The construction of translators, although it illustrates the general point, is thus an especially important task area. New programming languages are the tools for building information processing systems: Hence tools to build translators are tools to build tools, with all the multiplication of effect without multiplication of effort that is implied. Consequently, any center for the science of information processing will have a major effort in this area. Our own has involved the development of a special programming language for specifying syntactic analysis. We have used a similar scheme to build up a language for specifying code production, that at least pushes the translation problem back another step. From this and similar efforts by others

it is a safe prediction that soon the production of translators for languages inherently no more complicated than ALGOL, FORTRAN, or COMIT will become a routine task.

Example 2: Formula manipulation. The task area is that of manipulating algebraic formulas -- of simplifying, differentiating, factoring, integrating, constructing, etc. Languages such as ALGOL or FORTRAN do only arithmetic, and extension into the area of handling the literal expressions would yield a vast increase in power. Formidable manipulation problems in the natural sciences would become simply computations. This area is less far along than the area of translator construction. A few systems have been built utilizing isolated sets of subroutines for doing a few basic operations, and some of these have been used to yield useful results. A program has been written that performs integration, but as a demonstration program and an example of more general symbolic processes. Our own work involves embedding a scheme in ALGOL (called FORMULA ALGOL) which permits the complete integration of arithmetical and symbolic processes and should be one more step beyond existing programs. Already the line of development is sufficiently apparent in the work of ourselves and others to forsee the eventual incorporation into every algebraic language of symbolic manipulation without great additional complexity. On the other hand, no clarification of the task analogous to the use of stacks in translation has yet occurred.

Example 3: Monitors. By a monitor is meant a routine that controls the computer processing of many different jobs. Many monitors have been written.

One impetus is the time lost in a very fast machine by requiring the human operator to do any of the processing in initiating each new job. A second impetus (just coming into full force) is the problem of handling many users simultaneously in time sharing mode. Yet a third comes from systems which have more than one processor and require a monitor to coordinate and assign jobs. In spite of the number of monitors, their importance, and the first class talent that has gone into their development, we are still at the beginning of the trail. Each monitor is an organization unto itself, each just a monstrous piece of code. No structure has yet emerged; no consensus exists on what the crucial design considerations are, from which the rest of the organization logically follows. While we do not have progress to report yet, this is an area in which research will be pushed.

Example 4: Chess languages. We humans use a wide collection of linguistic devices to communicate with one another. Some small set of these (including some important ones) have been incorporated into programming languages, whereby we communicate with machines. The devices left out we either do not understand, are too difficult to implement, or are not thought to be worthwhile. In order to explore a new linguistic device it is sometimes expedient to move to a new task area in which it can be implemented and is useful. Thus, one may start the process of understanding. Based on an initial use, other programming languages will incorporate it in different ways, or by different underlying processes, until it becomes clear what the function of the linguistic device is and how it affects the other parts of languages.

An example is some work we are doing in developing a chess language that permits one to designate objects in the chess situation simply by stating the various relationships it satisfies. The translator must analyse the various relationships and their interaction, select an appropriate method to find an object satisfying these conditions, and construct the program to carry out the method. In general, such an analysis is exceedingly hard (and is, in fact, what good programmers are paid to do for a living). For instance, in an algebraic language to permit the user to simply write down "the x such that $F(x) = 0$," for some general class of functions, F, requires that the translator be able to solve the equation for x. In general this is too hard a problem, and consequently, current translators are not capable of it. But in chess the relationships are sufficiently tractable that we hope to be able to characterize generally how the analysis and program construction chould go. It is a good bet that once we do the analysis for chess, we will see ways to incorporate similar facilities in more limited ways into classical command languages.

There are other examples of this road to discovery: list processing, simulation languages, numerical control, graphical languages (just starting with programs such as SKETCHPAD) and so on. Each starts particular and gradually becomes a vehicle for revealing a little more of the structure of information processing. Each tends to solve, with varying degrees of success, problems in a multiplicity of new areas: list processing for pattern recognition now an essential part of high energy physics research; numerical control for rocket experimentation; graphical languages for automating design, etc.

Problems of Integration.

The approach to discovery described above is divisive: a multitude of separate strands. Clearly there is much common structure. In fact, whatever is abidingly true in the nature of information processing must occur again and again in all these separate efforts. If you will, these efforts, when successful, represent the elementary generalizations of our science. The problem is to extract from then the really fundamental characteristics and techniques of information processing;i.e., to formalize. The difficulty, of course, lies in the separate genesis of each strand. Each was made by differnet researchers responding to different formulations, and was shaped to the nature of particular task areas.

There are at least two ways to go about an integration. One can develop a formal theory of information processing systems, including the development of the associated mathematics. This route is well traveled; the going is rough but progress is by no means impossible. Indeed, some of the most impressive dividends of this approach were given to the field as its birthright. In 1937, before any of the technology existed that has made the study of information processing an imperative, Turing formulated the concept of a universal computer and proved that not all things were computable.

Such formal theories consist of postulated basic scheme for the representation of all information and a set of primitive information processes out of which all more complex systems must be fashioned. Their virtues lie in

in being able to establish extremely general properties of information processing systems within the model; their major difficulty is limited power on relevant problems. They cannot yet give any answer to efficiency, optimal design, correctness, etc., when asked about programs of practical complexity. We are not working much in this area. On the other hand, good talent is being applied by others, and the power of this approach will steadily increase in time.

An alternative to developing formal theories is to incorporate all of the important features of the separate strands into a single unified programming structure. This is a much more pragmatic approach than the formal one. As in the processes of basic discovery, its success is measured by the clarity of structure achieved, validated by useful programming systems applicable to a diversity of new problems. Note that we are not talking about constructing a monitor or supervisory structure in which all the separate special programming systems can co-exist as separate equals. We mean a single linguistic structure with mechanisms for creating and assigning names, abbreviating, delimiting lexical units, etc. Within this there should be a single set of mechanisms for organizing processes.-- a single command structure. Particular data representations with their proper operations should be incorporated as neat subsystems with clean interfaces between subsystems.

This process is not a simple one, nor one that can be hurried or done by fiat. Let us illustrate. We are currently very active in this enterprise, using ALGOL as a basic vehicle and producing a growing system which we can call EXTENDED ALGOL. Now consider COMIT, a programming language developed for linguistic analysis, and containing a number of unique features. How does one

extend ALGOL, so that it has all the additional powers (of expression and of processing) of COMIT without at the same time duplicating many of the functions already in ALGOL -- and in COMIT in quite different ways? One cannot just embed COMIT in ALGOL as an undigested monstrous subroutine. One must find instead that natural elegant extension of ALGOL which accomplishes the important new functions that COMIT has revealed.

One sees here clearly the complementary role of separate, almost isolated, efforts to build programming systems in new territory on the discovery side, and the effort to build a single universal programming system on the integrative side. We wish to emphasize this dual process, since we feel that it is the main road to the development of a science of information processing during these early, somewhat progmatic decades.

Problems of Proof.

The programming profession is engaged at all times in constructing large and complex symbolic structures that purport to do things and to have certain properties. This program claims to invert matrices; that program claims to post an accurate and complete status summary for a commander's view; a third claims to translate any program in ALGOL into a machine language subroutine that accomplishes the same processing as the ALGOL program. How do we know they accomplish their tasks?

The current answer, of course, is that we don't know. Instead, we are prepared to continually debug our programs. However, we do know that a large proportion of programming effort is spent in this debugging; but also that no really complex program ever becomes completely error free. One of the major

indications of the scientific maturity of information processing will be the development of direct techniques of verifying that programs are correct and have various properties. In short, we must learn to prove assertions about programs. By phrasing the task as one of proof, we wish to emphasize that a program is in fact a mathematical object, and that the techniques to be used are mathematical ones.

Again, there are two approaches to the problem of proving things about programs. Starting from the formal theories of information processing systems, mentioned above, the concern with proof techniques is already uppermost. As we stated earlier, the difficulty is that there appears to be such a long way to go before we are able to prove things about the complex algorithms that interest us. However some rather general things can be shown. For instance, there cannot exist any single general scheme that will determine the correctness of arbitrary programs. The second alternative is to tackle complex algorithms directly, not working within a completely formalized system, but trying to prove things more in the style of an eighteenth century mathematician. Again, consistent with our general approach, we have been more concerned with this latter possibility.

Programs are indeed very large and complex. Consequently, proofs about programs are also complex. Invariably they involve the extensive analysis of different cases -- as when one has to ask how the program would behave if the input data had a certain property (or if it didn't). However, the internal

relationships in the program upon which the proofs depend may not be very deep or subtle, and it may be possible to construct programs that will carry out this case analysis. These will be true theorem proving programs; they will try to fill in the detail of proof schemas suggested by human mathematicians. Since the case analysis may run into thousands of cases -- each of which must in some sense be discovered by the mathematician trying to prove the theorem -- it may be that these mechanical aids will be an essential ingredient in developing proof techniques for programs and will provide a good example of the power of man-machine cooperation. We have been doing some work in theorem proving programs that use case analysis, in order to explore this possibility.

Problems of Efficiency.

It should take no special argument to assert that the efficiency of a program in processing information is one of its most important features. A science of information processing should certainly have some techniques for assessing efficiency and for knowing whether some theoretical limit is being approached. Such concepts as the conservation of energy and (much closer to home) the channel capacity for information convince us of the power of such general laws. We believe in them in programming too. For instance, all programmers believe that you can trade time for memory and time for equipment. But to date, no good theoretical formulations of this trade-off exists, and it can be used only as the roughest qualitative guide. Except in some special areas, such as sorting, the problems of efficiency must simply be put down as future scientific business, but one whose study is critical.

Problems of Representation.

To be processed, information must be represented in some concrete form. The representation determines how much space must be used to hold a given collection of data. It also determines the processes required to write information into the representation, to read information out of it, and to maintain the information. Given a task to be programmed, the variety of ways to represent the information in it are not well understood. Yet what representation is chosen will determine the program's efficiency and flexibility and differently so at different times of the program's use. It is a commonplace that the hard part of programming a new task is creating and selecting the representation; most of what follows is routine coding.

Having pronounced the importance of representation, and by implication asserted that a science of information processing must contain a theory of representation, we confess that matters are in a very primitive state theoretically. We are still at the stage of the natural historian -- discovering new representations under odd rocks and gradually collecting them and cataloguing their interesting properties. An important example of this is list structures. These were developed in connection with a specific task area (problem solving programs) and have since become a well understood and widely used representation. However, there exists no wider framework within which to relate list structures to the various other existing representations or from which to generate new representations in any systematic way to meet new requirements.

Problems of Problem Solving Power

The following is elementary: a computer delivers raw processing power; how it is shaped to useful ends depends upon the program. Assured though we are by basic theoretical results that we can do anything we can specify, there are narrow limits to our understanding of how to achieve complex ends through complex processes. One major part of a science of information processing is concerned with advancing our problem solving power.

As elsewhere, formal theory about problem solving is worse than rudimentary. Instead, progress is achieved by selecting task areas that, while well defined, demand excessively difficult and unknown processing, and then trying to construct programs that do these tasks. The process is quite analogous to that described in the sections on discovery and integration. However, techniques of programming are to be distinguished from techniques of problem solving. One does not (yet) try to incorporate into an integrated programming system, such as EXTENDED ALGOL, the concepts that have arisen in constructing theorem proving programs. This separation is only temporary, reflecting a difference of starting point -- a concern with linguistic mechanisms, such as naming, as opposed to a concern with reasoning mechanisms, such as heuristic search. The distinction disappears as soon as we ask our language processors to do any substantial amount of inference and selection of methods of solution.

The whole area of problem solving makes a pretty illustration of the discovery-integration process. Many separate problem solvers have been written; the tasks have usually been idiosyncratic and non-utilitarian -- games, puzzles,

and areas of elementary mathematics. Some of the individual results have been impressive. Out of these separate efforts has emerged a clear understanding of two related important problem solving mechanisms: the formulation of a problem as the search through an exponentially growing tree of possible solutions; and the role of various heuristics to narrow and guide the search so it can be successful within reasonable processing times.

A current problem in this area is that of generality: how to build programs that can do a wide range of tasks. In normal applied work the interest focuses on a specific task, and in the interests of efficiency the program is adapted as much as possible to the particular features of the task. As a consequence, little experience exists about the organization of general problem solvers; and when the need arises to build programs with fairly general capabilities (as in command and control environments) we are stuck with extrapolating techniques adapted to an opposite goal. This limitation of our past experience to highly specialized programs lies at the root of the myth about the inherent rigidity of programmed systems. One approach to this problem, and an approach we are taking, is to take a single problem solving program (called GPS) and try to make it solve a wide range of problems (about ten). GPS initially proved theorems in elementary logic. It is being extended to various puzzles, elementary mathematical manipulations, more advanced proof procedures, etc. As with most of the work on problem solving, the tasks themselves are not yet of practical importance; also, one could probably design for each task a separate program that would be more efficient. But by trying

to use the same set of mechanisms for all the tasks we increase the efficiency with which we learn the natural generalizations of program organization and descriptions of task environments. This approach is the direct analog of the attempt to build an EXTENDED ALGOL, except that it is at a more primitive and experimental state.

Problems of Mass Information.

The problems of storing and accessing information undergo radical change as one goes from fairly modest amounts of information to massive files. Partly this is due to a radical discontinuity in the nature and relative speed of the accessing mechanism -- from random access in microseconds to serial or cyclic access in tens of milliseconds (and recently to pseudo-random access in tenths of seconds or seconds). Much more constrained strategies of accessing must be used and great care given to the rationale and timing of each access. These special features alone would be enough to dictate concern with the classes of processes involved. They are perhaps most of what is involved in handling large well organized files (as in most business data processing).

But the more general problem of large amounts of information is that the program (i.e., the programmer) knows very little about the data he wants. In a small program, the programmer knows exactly what information is stored where and how it relates to the task of his program. But in extracting information from an encyclopedia, an intelligence file, or even a table of integrals, only the most general features of the data are known. The

relevant data could be in the file in many different forms, fractured in many different parts.

Concern with this problem of organizing complex retrieval systems is conditioned by the accessing characteristics of the physical store, but would exist even if we had billion word random access memories. The problem is intimately involved with natural language. It is not that we must use English, because (say) we want to be easy on the user. Rather, natural language is the only universal vehicle known for expressing arbitrary information. The formalized languages that will eventually develop to handle large wide ranging masses of information will contain most of the important mechanisms of natural language, and will develop out of the study of natural language. The problem of massive information is also intimately involved with mechanized inference techniques, so that a fact can be gleaned from fragments of data in the store that have never before been brought together. One can pursue these inference problems along the same line as the problem solving programs -- working with formalized systems, rules of transformation, and so on. Alternatively, one can work at developing internal models, such that as each parcel of information is added to the system it is assimilated into the internal model. When requests are made of the system, the information is obtained by an investigation of the current state of the model. This latter approach currently seems very profitable, both to us and to other workers in the field, and some of the early work in this direction was done at Carnegie Tech in the form of question-answering programs.

The imminent advant of large scale commercial systems possessing enormous storage will make mass-information handling and retrieval problems, hitherto a specialized task area, standard fare for all large scale computer users. This flood to fill and use available storage will be arranged with languages that have increased naming, state, and search sophistication. The design of excellent I.R. languages cannot wait much beyond 1967.

Problems of Communication.

A language is Janus-like: one face toward each communicant. When both are human and substantially identical one is hardly aware of the two-faced nature. When there is great inequality -- a mother to her child, a boy to his dog, a man to a computer -- we view the language only in terms of the weaker. The problem is to get the computer to understand the instructions we give it; the man can shift for himself. We have already partially learned our lesson with respect to computers; the whole development of programming languages reflects an attempt to ease the task of the human. But at each step the course is roughly the same: a new linguistic feature is proposed (for human benefit) and the problem is how to construct the systems to interpret it and use it efficiently. All the science is built around the machine side of the language problem; the "new linguistic features" come of themselves, full blown, out of the minds of those experienced in the programming art, or these struggling with a problem.

Still, the human is an information processing system in his own right with his own special properties and limitations. At some point, when our

ability to construct programming languages becomes sufficiently great, we will need to know a great deal more about the human and his communication problems. What are his languages? How does he interpret linguistic expressions? What stages of gradual refinement and definition do his intentions go through on the way to becoming a set of instructions to be given to a computer? What is the role of mnemonic symbols for him, and how complex can they become before they cease to be helpful? All these questions are fundamentally psychological and linguistic, and they have interest in their own terms. Our concern with them here is as applied psychology -- as the human engineering of language and communication with the computer.

Let us give two examples of work in this area that we have an interest in pursuing, and where we have given some preliminary consideration to the problems. The first concerns on-line continuous communcations (so called conversational mode) between man and computer. We are currently pushing this problem, and like everyone else we are proceeding by designing systems, putting them into operation, discovering the difficulties and revamping the system. But it is clear that the human has difficulties thinking clearly in real-time -- witness the difference between spoken and written English. Ultimately (after the first exploitation of the new powers opened up by the hardware), the communication language must adapt to these limitations of the human. One could believe this would happen eventually just by successive approximation. One could proceed a good deal faster by understanding in some depth the nature of the human limitations, and then with this understanding in hand, designing language systems to reflect it.

The second example stems from searching for alternatives to on-line interaction: Perhaps the human could simply communicate his rough plans to the machine, and let the computer fill in the details. This puts the computer in the role of problem solver again, but the problems would not be very difficult (that is what it means to be a "detail"). However, before one can worry about the computer program, one needs to know a good deal about the nature of human plans -- about what it is that the human can communicate while he is still vague about the details. (The GPS program referred to earlier, used to simulate human problem solving, has already given us some information about this.)

We have ranged rather widely over approaches and problems in the science of information processing. The pattern adds up to a substantial and integrated attack on the science, and the one that will characterize the center in the coming few years. But to return to the theme in the earliest section, much has been left out. Some of that missing reflects our focussing on the center itself and not on the total Carnegie Tech environment. Although we have mentioned the work on psychology, using simulation as a major tool, we have underplayed it. And we have not mentioned at all the work in management science nor the task areas that arise out of physics' relatively new concern with data processing. But there are still other areas that currently have no strong representation at Carnegie Tech: the translation of natural languages; the problems of pattern recognition in various difficult and noisy environments (handwriting, picture interpretation, biological analysis); the problems of error correcting and detecting codes. We have not mentioned computing hardware

even once! In one sense there is space for these researches in our earlier description: the tilling of special areas provides the initial glimpses of important structure. But no particular subset are essential to the center's own program of research and so the choice of which ones exist and flourish in our total environment will be time dependent on more external factors. The matter of hardware is a special case and we will treat it more fully in the next section.

Organizational Issues Concerning The Center

Given the picture of the substantive orientation of the proposed center, we can turn to questions of organization. There are a few major issues and points of view, each of which requires discussion. Once these have been set forth, the actual details or organization and growth follow simply and without further justification in the final section.

Relationship to the Carnegie Tech Research Community.

We have already commented at length about the way information processing pervades other fields and the way research on information processing ultimately becomes an integral part of many fields. As a consequence, the center must be "open" towards the entire campus. Research on information processing must grow up and flourish throughout Carnegie Tech because the Center is here, not in spite of it. Nothing would be more fatal than to erect an organization whose natural tendency was to absorb all such research within its confines, or which even operated to create a gradient of sophistication, in

which those "outside" the center have little access to the advances achieved "inside." Thus, in a sense, a very substantial segment of Carnegie Tech should become the laboratory for the study of information processing.

The organizational form that seems most adapted to this goal is a center with a relatively small full time faculty and a very substantial joint faculty. The full time group, consisting ultimately of six to seven members, has its research objectives directly focussed on the science of information processing, substantially within the purview of the previous section. The joint faculty have their interests and motivations split between a concern with information processing and some other substantive field. This "other," of course, may simply be one of the many aspects of information processing that happens to grow up in one of the departments -- e.g., a concern with redundancy techniques, or with hardware devices, or management control systems -- anything that happens to settle more comfortably elsewhere.

The important distinction is the following: We in the Center are committed to a line of research, as outlined; we are not committed to any particular size or composition of the joint faculty. The joint arrangements form an opportunity to encourage research on information processing wherever it most naturally occurs on campus by offering partial financial support for faculty members, support for graduate students, and an official way of blurring the lines of separation between those inside and outside the center.

The current situation at Carnegie Tech is consistent with this organizational form, and supports the proposition that this is the correct way to

organize a center for the study of information processing. We have an inter-departmental doctoral program in Systems and Communication Sciences. The heart of this graduate program is a concern with information processing, but it covers the full range from control systems, through cognitive processes, to logic. The current participating departments are Electrical Engineering, Mathematics, Psychology and the Graduate School of Industrial Administration. There are over fifteen faculty members involved, and about fifty graduate students, mostly in Mathematics and Electrical Engineering, all of whom are going toward the doctorate. In each of these four departments there is already much work on information processing that fits the "joint faculty" concept. Most of the work in Mathematics connected with information processing is directly related to the center, but not all. The arrival of good facilities for formal manipulation will produce an increase in the involvement with advanced information processing techniques. In Electrical Engineering, for example, there has been developed a programming system called SCADS (under ARPA support, in fact) for handling continuous dynamic systems on the digital computer. (This is not the first such programming system of this kind, but appears to be the most sophisticated to date.) SCADS, which was done by two graduate students in EE last summer, is a nice example of many points -- of what graduate students can do if given their head; of the way new systems get born by users fighting with problems, and then get incorporated into the regular system; and of the way information processing further interpenetrates a field, since one of the developers of the system will be doing his thesis on programming aids for analysing non-linear dynamic systems.

Both the Psychology department and the Graduate School of Industrial Administration are thoroughly saturated with a concern with information processing and much of the research in information processing done at Carnegie Tech in the last eight years has been done in these two departments (including the development of programming languages and problem solving programs).

We have mentioned so far only the four departments that happen to be involved in the interdepartmental graduate program. But other departments are also involved in information processing research. The Physics department has just added to its faculty an expert on information processing as related to high energy physics research. There is a substantial research project in Civil Engineering to develop a programming language and system for the area of hydrology. As one final indication, a young post-doctoral fellow from the Chemistry department is being supported by ARPA this year to work on doing formal analysis in chemical energy calculations by computer.

Relationship to Graduate Students and Education.

We have always felt that graduate education is part of the growth of a science -- not just a logistic function to be performed, but an integral part of the process of advancing the science. The act of teaching refreshes and stretches the mature researcher. The impact of the uncommitted and iconoclastic mind of the graduate student forces the continual re-examination of old solutions; and offers the best chance for the emergence of something really new. We fully expect a major part of the research output of the center to come from the work of the advanced graduate students.

A consequence of this viewpoint is that there will be no permanent group of full time non-faculty research personnel at the center. Thus, we will differ from a research institution in that all the research people here will also be concerned with graduate education. There will, of course, be all sorts of arrangements that involve exclusive commitment to research: post-doctoral appointments, visiting appointments, semesters with no teaching duties, etc. But all these are essentially temporary in nature.

In the final section we estimate the number of graduate students who will be associated with the center, either through the full time faculty or the joint faculty. The number is currently about 25 and will increase to about 60 during the next four years. Most of these students are in the Systems and Communication Sciences doctoral program, mentioned earlier. Only a certain fraction of these students will be supported out of the center's funds, since many of the other research efforts on campus have their own support.

Mixing Service and Research.

In all events there will exist at Carnegie Tech a Computation Center, whose function it will be to provide the Carnegie Tech community with whatever computing power it needs. The relationship between the Computation Center and the proposed center for research is a crucial one. The problems are whether, on the one hand, those who just want some computing done, even as part of information processing research, will be short changed, everything being in a constant state of flux due to research in progress and no one in the computation center

caring about anything but research, or whether, on the other hand, no one in the proposed center will get any research done because they are so busy providing service. All solutions endeavor to separate the function in one way or another, ranging at the extreme end to using separate computers or even to creating completely separate organizations. Our proposed solution is to incorporate the Computation Center into the proposed center organizationally, to utilize a single computer facility, and to structure the internal organization of the proposed center to deal with the mixing of service and research.

The most important argument for the use of a single facility is that nothing can be gained by having two. Under all circumstances the vast majority of people working on information processing research are not concerned with modifying the currently operating programming system or equipment configuration. These people have as big an investment in a stable, efficent, smoothly running system as any chemist or physicist, who only wants to get his calculation done. Consequently, in terms of service provided the computer system devoted to information processing research differs not one whit from that devoted to "service computation."

The real problem is how to live with a system which is being advanced almost continually, both in the basic programming systems and languages, and in the equipment configuration. The theoretical answer is easy: simply control the process of change sufficiently well to avoid instability. The capability for doing this may have arrived only with effective time sharing systems, which permit systems work to proceed in parallel with other use under suitable protection. In any event, this is the path we are pursuing with out system and we expect to be able to make it work eventually.

So far we have addressed only one side of the issue: how to assure adequate service while doing research. The other side -- how to assure adequate research while doing service -- must still be dealt with. Our solution here is to divorce research from development and maintenance. No member of the research group (roughly, the faculty and graduate students) will have any operating responsibility. Conversely, there will exist a substantial full time programming staff, whose function it will be to institute change and to maintain the programming systems. This programming group will consist entirely of systems programmers, as that term is currently used in the programming profession. Their work will be information processing oriented as distinct from application programming. It has never been the policy of Carnegie Tech to provide application programming, and no change in this policy is anticipated.

It was stated earlier that no permanent full time reseach personnel would exist at the center. The programming staff under discussion does not violate this statement, and it is quite important to understand the distinctions that are being made. Much programming, even at the highest technical level, is not research. The programming and installation of a language already available on other machines, is not research. The reworking of the ALGOL system already in use to improve its efficiency by a factor of two, is probably not research -- although it may involve some very ingenious programming tricks, and is certainly worthwhile. The improvement of the monitor system by adding new features is not research, even though it may take the best programmer to accomplish it, since storage and timing constraints are so tight. The incorporation

of a new programming system, created in the center as a piece of research, into the current operating system is not research, even though it is a necessary follow thrqugh before research becomes useful.

This last example is worth additional expansion. The typical life history of much research done at the center will be as follows. A graduate student will get interested in some aspect of programming, say in monitor systems. Working with one of the professors as advisor, he develops a programming language in whose terms monitor routines may be expressed, including a technique for translating the symbolically expressed monitor into machine code that meets the various timing constraints of the hardware configuration. Focussing on the essentials, he works with a " pseudo-machine" that he has defined for his own purposes and simulates it on the computer. This pseudo-machine differs from the existing machine in many ways -- simpler in many respects, more complex in a few. At the termination of this research it has become abundantly clear that this monitor language should be used for the Carnegie Tech system. At that point there may be a full man-year of hard development work ahead, for it is unlikely that any such new idea could be applied for the first time without considerable effort. Yet this next man-year is <u>just</u> development work; it is not research. Further, it is inappropriate to insist that this student personally engage in this development. At that point the full time programming staff assumes control, and does the additional work necessary to incorporate it into the system. They will continue to improve its use in the operating system. Meanwhile its limitations become apparent. In coping with them they may make a substantial enough contribution to the programming art so that, in retrospect, research has been done:

revealed as a sideline to the task of making the Carnegie Tech operating system system as good as possible.

The point in presenting such an extended example is to show that there can be a meaningful difference in function -- and even in creativity -- without a difference in intellectual level. The best part of the systems programming world today does not do much research by the definition implicity given above; it does high class engineering.

We have labored this long to establish the legitimacy, in a proposal for a research center, of a large full time programming staff whose function is not research. The large size is dictated by the need to keep the total system stable in the face of continuous advancement in the state of the art. The group is in fact larger than it appears. It is augmented by a substantial number of undergraduates; in fact, it is from the pool of undergraduates that the additional programming power comes to do many of the extra features and provide the necessary extra push for their rapid accomplishment. At Carnegie Tech we have used large numbers of undergraduates as programmers for many years, and have found this to be an admirable practice, paying a very large educational bonus as well. Its one main disadvantage, having students responsible for operating system systems, we propose to rectify by the presence of the full time programming staff.

In general every one at Carnegie Tech involved in information processing research does his own programming -- just as a mathematician does his own mathematics. A very few projects exist of such large magnitude that this is an unreasonable mode of operation. An example would be a large question-answering

program. Consequently, a small number of full time programmers will exist to work on these projects. These are to be considered as full time technicians, and not independent research scientists.

With a single computational facility kept in stable performance with an extremely advanced software system, everyone benefits -- the researcher on information processing and the researcher who only wants his information processed. There is no clean answer to the question of what proportion of the center's cost should be assigned to research and what to service. If all the non-information-processing-research people would vanish, the costs of the center and the facilities would not be much less. If all the information-processors vanished, the cost would be a lot less, although still not small, and those left would get less advanced software. However, ARPA should not support research which is not related to information processing research. The administrative control will, of course, lie with those responsible for the center.

Equipment, Hardware evelopment and Hardware Research.

The center, by inclination, talent and past action, is a center of software research. What is needed on the hardware side? The issues naturally break into the three parts in the heading and we will take them up in that orden

Equipment. Four things determine how much equipment one needs at any time. First is the number of users. Second is the amount of processing per user, both in time and memory. Third is the style of use these users will adopt. And

fourth is the basic proposition in programming research: have available substantially more capacity than one can justify.

The oddest of these is style. It refers to the transition now beginning to a time-sharing mode of operation in which the user is able to keep vast amounts of information in the machine at all times, instantly available at his call. This new style, hurried into existence by ARPA itself, carries with it a large saltus in equipment per man. Large secondary stores, high capacity transfer rates, remote stations, a large increase of time spent by all users in communicating with the machine -- all these add up to a substantially higher "overhead." Since this issue of style relates directly to the researcher's effectiveness in using the machine, it is clear that the center must remain relatively close to the best available in this department (independent of the fact that one part of our research interest is concerned with how to develop this style).

Carnegie Tech is a relatively small environment; consequently the total number of users is relatively small (compared, say, to the University of Illinois or MIT), even when one takes into account the high density of scientists concerned with information processing. The computation requirements of the university background are not consistently excessive. Thus, although large computers are required for information processing research, their total demands lie well below the most powerful systems available in the country. For instance, our current system has somewhat less power (in operations per second) than an IBM 7094. The situation on memory is somewhat different. Programming research, and especially work in problem solving, requires as much random access storage as one can reasonably get. In this last department, we again need to be near the top of what is available.

The final consideration -- the need for "over capacity" -- is based on the need to ignore questions of space and time efficiency in order to get on with discovery. An example will make this clear. The initial, uniform reaction of the programming world to list processing when it was first introduced in 1956 (when big machines had 4,000 words of storage) was incredulity that someone would throw half the memory to store the links to other words. Yet it was precisely this dis-efficiency that allowed many other features to be exploited, and which became less and less important as large memories became available.

Generally, for an active installation the need for computing power has grown exponentially. Our own growth is from an IBM 650 in 1956 to a Bendix G20 in 1961 (an increase in power by around two hundred) to a CDC-Bendix G21 in 1963 (another increase of over four times). This is a growth rate in power of something like three per year, and is in no way unusual. The amount of computing obtainable per dollar is also growing exponentially, and has absorbed some, but not all of the cost of this growth in power. Consequently an increasing investment in hardware is still indicated. That this exponential growth must stop eventually is clear. Yet the continued expansion of our knowledge of how to use the computer in sophisticated ways and the continued change in style in using them (always in the direction of increased leverage for the human mind, hence requiring increased operations per thought) virtually guarantees that the expansion will continue for another decade at least.

Another factor that tends to increase the amount of equipment needed, although by an amount small in comparison with the total growth rate, is the need to balance the system while keeping it stable. The sequence goes like this: At

time T a decision is made for a substantial increase in the equipment configuration, to provide new capabilities, a change in style, etc. This involves a change in the programming systems and work on these is initiated. After the equipment arrives, the new programming systems are installed and operational experience accumulates, it is discovered that the vision at time T of the state of affairs at installation was faulty: the disc capacity is limiting, or translation takes too long to permit on-line compilation, or what not. The performance of the total system falls appreciably below that predicted due to a particular feature that becomes limiting: the system is out of balance. Theoretically, at this point, the system could be balanced by a readjustment of the total configuration keeping total dollars fixed. However, to do so is to unstablize the whole system -- to require reworking of the just finished programming system because many features of the hardware configuration have been altered again. The alternative -- and the one that is always taken -- is to leave the existing configuration fixed and to add some incremental equipment to try to achieve balance. This permits one to make as small a change as possible in existing programming systems. The net result, of course, is to remove the offending limiting factor and to reveal the next one -- thus causing the addition of yet another small increment of equipment and programming system. The root source of this rather wobbly way of business is imperfect vision in the face of a dynamic environment; and while there is no intention to condone it, there does not seem to be anyway to avoid it and still progress.

The net result of these considerations is a projection of the total equipment configuration that rises from a current rental equivalent of somewhat

under a million dollars a year to about two million dollars a year in 1967-68. The exact configuration is given with the budget, including estimated cost. The growth is made up of four parts: a relatively modest increase in central processor power; a more substantial attempt to keep up with the secondary storage requirements; a reasonable growth rate on remote terminals of fairly conventional type; and the acquisition of a random access "bulk" store (half-million words and up) when it becomes available. Although one high performance console is included (1964-65), no other attempt is made to keep abreast of advances in such man-machine communication devices (but see the section on hardware development).

This total configuration is shared between all users and its costs must likewise be shared. These estimates on this division are shown in the budget request.

Hardware development. There is a need for a modest sized engineering group in the center, independent of any major research into hardware. The needs are several. First, it is essential that there exist in the center a first rate knowledge of, and sophistication in, the hardware state of the art. Otherwise one is at a decided disadvantage in any negotiations with the computer industry that involves new or special equipment. Estimations of reliability, ease of hardware adaptation into the existing system, probability of successful delivery on time, and definition of acceptance criteria,are all examples of what must be done in-house, independently of the manufacturer. In a large and experimental configuration, even the question of maintenance cannot be left wholly to the manufacturer.

An informed technical judgment of the seriousness of maintenance difficulties independent of that of the manufacturer's maintenance engineers is essential.

The kinds of needs described above extend both to the writing of specifications for special equipment and the monitoring of any special contracts, and to the estimation of the future configurations both needed and possible in the light of on-going hardware advances.

A more substantial need in terms of the amount of people and budget required concern the design and construction in-house of a small amount of equipment. The argument is simple. Programming research often demands slight equipment additions and modification beyond conventional configurations in order to make certain developments possible. These are first seen by the researchers before the industry is prepared to provide the new gear. A standard example from the recent past is the provision of communication terminals with visual display and graphical input capability. Another example is the addition of a list processing adjunct to a standard configuration, something that makes no sense except at an installation that is doing a great deal of list processing. Yet for such an installation, a modest capital expenditure on a piece of equipment could pay for itself many times over. It could even permit modes of operation otherwise denied, such as conversational modes of interaction with list processing systems. No manufacturer will provide such a piece of gear. It must at least be designed in-house; and, given the judicious use of subcontractors, it might be faster and cheaper to construct the gear oneself.

One final and important example is the construction of interfaces to the existing configuration to allow a new device to communicate with the central

machine. Many of these will arise from the demands of the joint faculty to automate various experimental and exploratory arrangements. These will all be one-of-a-kind jobs that regular manufacturers will not touch. While part of the cost will be born by other sources of funds, there needs to be the whereWithal to meet them half way.

Note that none of these items is research. Even the list processing adjunct may be a perfectly straight-forward wired program machine. It is justified by what it allows for programming research, not because it is an engineering achievement.

It is our estimation that there will be a continuous enough stream of these projects to justify a modest engineering group. As with the programming staff, this group is full time, and is not considered to be a research group. Likewise, we anticipate that they will make considerable use of undergraduate students, which will provide that variable pool of manpower for special projects.

Unlike the situation with the basic equipment configuration, we are not prepared to submit costs for specific projects. One small project, involving the incorporation of the RAND graphical tablet, is currently underway, supported out of the present ARPA funds. But even here, we are not prepared to state future costs, since the current phase involves a design study to estimate what sort of additions are necessary. Consequently, we have placed in the budget a relatively modest lump sum for each year.

Hardware research. While the case for a hardware development group is straight-forward, the question of hardware research is somewhat subtle. The center as organized contains no real effort on hardware research. On the other hand, we

feel that there are strong flows back from programming research into hardware, and we would have been happy to have such an effort an integral part of the center. However, we feel no need to make a major attempt to obtain such an effort in order to "complete" the center: it stands on its own feet as a software oriented group. We shall encourage the growth of hardware research; this is one of the freedoms that the concept of joint faculty gives us. With the engineering development group we will have enough latitude to encourage initial efforts in this direction. But whether Carnegie Tech grows strongly in the hardware direction depends on opportunities we cannot now forsee.

Composition of the Faculty.

The growth in work in information processing over the last two years at Carnegie Tech, made possible chiefly by ARPA funds, has been used mostly to strengthen the research approach along the lines described in the earlier section. However, it should not be concluded that we feel the current distribution of effort is optimal. Further modest growth of the central faculty, as implied by the creation of the center, will permit us to adjust the composition of our effort in ways that seem appropriate to us now.

The first is greater strength in the mathematical theory connected with programming. Our own empirical orientation does not blind us to the slowly increasing relevance of more formal treatment, especially if done in an environment which is empirically informed.

A second area is that of natural linguistics and a concern with human communication. Whether appointments in this area would be in the full time faculty

or jointly with psychology is an open question and in fact irrelevant. But there is a large gap in the Carnegie Tech environment in the linguistics area, and this needs filling.

We have already commented that our plans for the full time faculty do not include scientists working on computer hardware. Yet this should be mentioned here again, since we propose to be opportunistic about it. The most likely candidate is a faculty member whose hardware interest lies adjacent to particular programming strengths of the center; for example, a concern with the communication interface between man and computer, or a concern with ways to embed organizations for problem solving into hardware.

As far as the joint faculty is concerned, we intend to be opportunistic. That is, we believe that we should not actively recruit our own image of an "optimal" composition, but should cooperate with the rest of the campus to encourage good research in information processing wherever it naturally arises.