DISTRIBUTED PROCESSING
AT CHAMPION INTERNATIONAL
A Case Study of Software Productivity

Christine V. Bullen
Stephen E. Humphrey
G. Allen Moulton

September 1979

CISR No. 50
Sloan WP No. 1090-79
DISTRIBUTED PROCESSING
AT CHAMPION INTERNATIONAL
A Case Study of Software Productivity

Christine V. Bullen
Stephen E. Humphrey
G. Allen Moulton

September 1979

CISR No. 50
Sloan WP No. 1090-79
This paper reports on a case study of improved software productivity at Champion International Corporation. The improvements in productivity are related to the use of a package of software development tools called Champion Distributed Processing System (CDPS). The background of central and distributed data processing at Champion is outlined. CDPS is described at length and related to some other techniques for software development. Champion's approach to estimating software development jobs and measuring productivity is reviewed. Conclusions drawn from the case interviews address the extent of improved software productivity at Champion and the role of CDPS in these improvements.

SOFTWARE PRODUCTIVITY

Software productivity is currently a major concern in the DP industry. As the cost of hardware decreases, more and more of the DP budget is made up of people costs. DP budgets as a whole have been expanding rapidly due to increased demand for services. As a result, top management attention and concern has been focused on the DP function - resulting in an even greater interest in the efficient and effective management of the function.

Positive achievement in the area of software productivity could play an important role in improving both the efficiency and effectiveness of data processing. However, the term "software productivity" itself
has many definitions, interpretations, and measurements.

The Center for Information Systems Research (CISR) is undertaking a major research program in this area. One of the proposed outcomes of such research is a thorough definition of software productivity. However, it is necessary to create a working definition in order to carry out the research. Productivity in an economic sense, is a measure of useful product output per unit of resource input. In the software productivity area the outputs are applications software systems to support business functions. The inputs of chief concern are professional labor. Productivity can be increased in two ways: either by reducing the labor required to develop an application system, or by improving quality control - reducing the scrappage rate for systems produced. The quality control aspect is of overriding significance but often ignored in discussions of software productivity.

One track of software productivity research at CISR will address these major issues: 1) What are the best ways of measuring productivity, and 2) what factors influence the quality of the productivity? Organizations have used various measures in the past that they have felt reflected improved productivity, for example, lines of code produced in a fixed period of time, reduction of the number of bugs or the time to debug, reduction of the overall time to produce a finished product, or ability to re-use the product in many areas of the company (common systems). All of these are relevant measures, but no one is convinced that he or she is using the "right" measure. In the course of this research we at CISR hope to shed light on the possible relevant measures and factors effecting the quality of software productivity and to report on the success or
failure of their use in various corporate environments.

Champion International has lived through various stages of centralized, decentralized, and distributed processing modes. Experience and application of sound management principles has enabled the establishment of a consistent and workable strategy for DP at Champion.

CDPS was developed at Champion International Corporation as the means for better supporting and fulfilling the needs of a large, scattered user population. Committed to creating systems and support which increase the competitive edge in the marketplace for its distributors, Champion is developing a system which has several qualities which seem to enhance the productivity of the overall DP function at Champion.

CDPS has attracted attention because of its reputation of enhanced software productivity. However, productivity enhancement was not Champion's major goal in developing this system. Champion's major goal was to create a system which solved the operational needs to design and successfully run a distributed processing environment. CDPS was developed at Champion to facilitate the design and operation of applications in the distributed environment. CDPS has been used for application development since early 1977 at Champion, when development of the working version began. Champion has also used COBOL and other tools. The CISR research team was interested in comparing the past performance in software creation to the current performance using CDPS. The Management Information Services staff provided the investigators with their project records and the opportunity to discuss the validity of the productivity claims.

The CDPS concept has several qualities associated with software productivity.
1) As a transaction processing system, it combines the building blocks of traditional processing environments (i.e., DBMS, OS, TM) into a unified system which is hardware portable. Reassembly of the CDPS building blocks is unnecessary in the event of a hardware switch. Hardware portability can improve the productivity of the DP function in that major system rewrites can be avoided during hardware changes. The fact that CDPS works and is continuing to work may be the most significant measure of productivity.

2) As an application development tool, CDPS utilizes a language, called Champtalk, which integrates the pieces of this system, and is also easy to learn. This simplicity of language ranges from ease of learning and fewer errors in coding to the production of fast, efficient programs.

The purpose of this study is to 1) trace the development and implementation of CDPS at Champion International Corporation, 2) outline the characteristics of CDPS which make it an interesting development tool, 3) to describe the project control system at Champion, i.e., productivity measures, planning standards, and 4) to point out where the gains in productivity appear attributable to the CDPS system as it is implemented at Champion.
METHODOLOGY

This study was undertaken through a series of interviews conducted at Champion International and at TOMINY, the co-developer of CDPS. These included at Champion: Vice-President of Management Information Services, Manager of Technical Services, two Directors of Business Systems, and various internal staff members who have assisted in the implementation of CDPS at Champion. (See Figure 2, Organization of Management Information Services.)

BACKGROUND OF CHAMPION INTERNATIONAL

Champion International Corporation is a multi-divisional manufacturing and distribution firm, headquartered in Stamford, Connecticut. At approximately $3.5 billion annual sales, its primary business is forest products — including building products, paper products, and packaging materials. (See Figure 1, Champion International Corporation.) Management Information Services is located in Hamilton, Ohio, and the VP of MIS reports to the Corporate Senior VP for Control. Primary authority for cash management and financial control for the firm is centralized under the Corporate Controller.

The Timberlands Division is the forest management group. With responsibility for tree nurseries, seed orchards, and mature timberlands, its role is the growth, harvest, and management of the softwood resources.

The Champion Papers Division manufactures and distributes both coated and uncoated papers, milk cartons and juice containers, paperboard, and envelopes. Of the twenty-four manufacturing plants and the fifty sales
CHAMPION INTERNATIONAL CORPORATION

CHIEF EXECUTIVE OFFICER

TIMBERLANDS

BUILDING PRODUCTS
  - Champion Building Products
  - Weldwood of Canada

SENIOR VICE PRESIDENT CONTROL

PAPERS
  - Champion Papers
  - Champion DairyPak
  - Nationwide Papers
  - Champion Office Products
  - Federal Envelope

PACKAGING
  - Hoerner Waldorf

Management Information Services

Tax

Audit

Corporate Controller

FIGURE 1
CHAMPION INTERNATIONAL CORPORATION
MANAGEMENT INFORMATION SERVICES

SENIOR VICE PRESIDENT
CONTROL

VICE PRESIDENT
MANAGEMENT
INFORMATION SERVICES

DIRECTOR
BUSINESS SYSTEMS
DEVELOPMENT

DIRECTORS (2),
BUSINESS SYSTEMS
SERVICES

DIRECTOR
DATA PROCESSING
SERVICES

MANAGER
TECHNICAL
SERVICES

MANAGER
TELE-COMMUNICATIONS

MANAGER
ADMINISTRATION

FIGURE 2
and distribution centers which make up this division, there are a variety of different businesses which have become part of this division through acquisition. Sales account for one-third of corporate sales volume.*

The Champion Building Products Division, comprised of 70 mills, 50 sales offices, and 100 distribution centers, manufactures and distributes plywood, lumber, and fiberboard, among others. Commanding about one-half of Champion's sales volume, this division's major business is the former U.S. Plywood - which merged with Champion Paper in 1967.

Champion Packaging Division is the former Hoerner Waldorf Paper business. Responsible for the manufacture and distribution of packaging materials such as paperboard, pulp, gift wrap, and paper bags, Hoerner Waldorf has seventy manufacturing plants and over one hundred sales and distribution locations. Packaging accounts for the remaining one-sixth of corporate sales.

TOMINY

TOMINY grew out of the consulting contract awarded to the developer of CDPS in 1976. Its initial role was to architect and manage the development of an operating system, programming language, and file management system for the IBM Series/1 which would fit the needs and philosophy of the Management Information Services group at Champion. The founders of TOMINY are former staff members of Cincom (developers of the

TOTAL database management system). TOMINY has been developing versions of CDPS to run on other hardware - IBM 370 computers, the Intel 8086, the Sentinel microcomputer, and the Honeywell Level/6. In addition, CDPS is currently available through IBM as a Series/1 IUP (Installed User Product).

DATA PROCESSING AT CHAMPION BEFORE CDPS

In 1967 Champion Paper Company merged with U.S. Plywood, and the latter business became the Building Products Division of Champion International. It was hoped that through a series of smaller acquisitions in the furniture, carpeting, and paper-related businesses that Champion International could combine the marketing expertise from Champion Papers with the massive distribution network in the former U.S. Plywood to make the new Champion grow.

Consequently, the MIS group faced the job of trying to absorb and consolidate the DP needs of disparate businesses into the common service of the MIS group at corporate level. In 1967, with intentions to streamline the order entry function in the Champion Papers Division, MIS planned the installation of an on-line, centralized order entry system in Hamilton, Ohio - intending to install a Burroughs 3500 mainframe. Centralization of the order entry clerks and the function was intended to improve the production planning process. However, the lack of available software ultimately forced the cancellation of the Burroughs project. Centralized order entry was accomplished in mid-1971 using an IBM 370/155 at Hamilton. The results were good. Production planning was vastly enhanced and order processing was made more efficient and effective.
The substantially increased MIS visibility resulting from this leading-edge on-line system and the experiences in managing the increasing complexities of hardware and software downtime shaped the present Champion philosophy of "distributed processing". Today, for example, Champion will not design on-line transaction processing systems to operate on the same computers used for heavy batch processing. The response requirements of users of on-line systems ultimately necessitate low CPU utilization. Job scheduling for heavy batch load, on the other hand, attempts to fill up every available machine cycle. The conflicting demands of these sorts of loads have led Champion to the policy of off-loading interactive transaction processing to small dedicated mini-computers. The small computers are interfaced to the large central machines where large databases or heavy processing is required.

In 1972, a pilot IBM System/3, card-based batch system was installed in the Office Products Business of the Papers Division. For the distributors, a mini-based system could be justified in terms of personnel savings and savings through better calculation of the complex pricing formulas and inventory management in the remote offices. However, the System/3 was batch-oriented and proved to be too slow. Distributors required almost immediate turnaround; shipping documents had to be ready within two minutes of the order entry.

The System/3 also required specialized operator training and skills not usually available in field offices, where high personnel turnover was the norm. Champion learned that distributed processing per se was not sufficient to achieve their objectives. A field system should be interactive and self-documenting, enabling new operators to make productive
use of the system while learning. Subsequent systems would be built in a
tutorial, menu-oriented style on interactive display terminals with sub-
stantial error checking and correction procedures built into the transaction
entry process.

The distribution businesses were still manual. In 1971, MIS
investigated the feasibility of a centralized network of 200 dumb terminals
to automate the order processing functions. The distributors, though, were
responsible for their own profit margins. Forcing them to relinquish con-
trol over their own order processing would, in their view, mean forcing
them to give up control over their profits - an unacceptable risk for any
distributor. Furthermore, a centralized processing operation risked down-
time for everyone simultaneously and increased management complexities in
the central DP shop. One executive, a World War II veteran, remarked
"Never park the fighter planes in a group". Moreover, the cost of moving
data back and forth through the communication lines from the distribution
areas would be extremely costly and needless. The centralization concept
was discarded.

In 1972 MIS began the search for an appropriate mini-based system
for the distributors by looking at all of the mini manufacturers. The ob-
jective was to select a mini-computer with many of the attributes found on
mainframes: i.e., screen entry, large disk capacity, good stand-alone
capabilities, and telecommunications. The field of choice at that time was
limited to Four Phase, Singer, and DEC. Each looked technically good, but
the availability of hardware service for the distributors in the remote
areas was critical to Champion. Singer/Friden had landed a major contract
with Sears Roebuck, whose sites were also widely distributed, and service
looked acceptable. The contract went to Singer, and between 1973 and 1975 twenty-six machines were installed in several divisions.

In December of 1975, Singer abruptly left the business, and Champion began the search for a new system. The strategy was to find the mini-computer hardware which fit the MIS philosophy at Champion. For Champion, three factors were paramount:

1) It was clear that the primary objective was not to maximize the CPU usage of any particular machine, but instead to minimize the complexity of the system and improve clerical productivity - i.e., do not mix application systems; implement one application per machine, and avoid the snow-balling effect of adding new applications to existing machines. Since cost per CPU is about $9,000, the cost/benefit was favorable even at the warehouse level. This also avoided the problem surrounding the old centralized mainframe approach.

2) Just as important was to build common systems where practical. Develop the software centrally, and then distribute the processing to the remote sites; use the expertise of the MIS staff in Ohio to build systems which are operable by the processing site staff - inexperienced users. This approach would include the distribution of the databases, necessitating powerful database software.
3) The longevity of the software must be assured through hardware independence. After the Singer experience, Champion would not risk allowing the vendors to ignore this problem. Champion would build its own operating system for a mini, its own file management facilities, and its own programming language — or it would dictate these conditions to a vendor.

During the search for a new piece of hardware, both DEC and Datapoint systems seemed to offer the best foundation for Champion. Toward the end of the search, however, IBM announced the Series/1 computer sans operating system. IBM came in low bidder on the hardware and won the contract in late 1976.

Champion decided to gain control of its own destiny by developing its own operating software to manage the distributed network. TOMINY was then hired by Champion to architect the development of the operating system, the programming language, and the file handlers. IBM systems engineers, under contract with Champion, became part of the joint project. The outcome was CDPS.

DEVELOPMENT OF THE CDPS SYSTEM

There was no operating system on the Series/1 in 1976, no programming language, and no applications. Champion and TOMINY jointly developed a prototype of CDPS on the IBM 370/155 at Hamilton. The prototype was capable of compiling and executing all CDPS instructions, but did not
run on-line. Interaction with a terminal was simulated with card input and printed output. This testbed took about four months to complete before work began on the Series/1 operating system and the applications. The 370 testbed enabled simultaneous development of the operating system and applications, thereby reducing overall development time by nine to twelve months. Programmers used punch cards to simulate screen entry to the CDPS applications.

The application programs were finished before the Series/1 CDPS system was complete - the two major applications took about seven months to complete, and the CDPS system was complete in about one year. The applications were then debugged on the Series/1 interactively. Most of the applications worked well immediately, with the exception of some screen formats which needed to be rearranged from the card simulations.

Currently, Champion plans to have about 150 installations for the Series/1 in various locations by 1981. Approximately 60 are already in operation. In addition, a new real time CDPS-developed system is now aiding in monitoring the paper milling process and labelling paper rolls.

The Building Products business distributes plywood, lumber, logs, and particle board through its network of 150 regional sales offices and warehouses. These distributors stock and sell the building materials to lumber yards, retail chains, and to housing and furniture builders. The major jobs performed at these sites are taking of orders, filling these orders, invoicing the customers, maintaining inventories, and assisting the plywood mills in production planning. Historically, this has meant heavy paperwork, complicated manual pricing formulas, and difficult pro-
curement planning for the business.

The installation and operation of Series/l computers in each of the sites automates these previously manual paper processing jobs for the sales offices and distributors. Order entry is accomplished interactively by a clerk at the distribution center. The system generates the necessary shipping documents and keeps track of the orders. Upon shipment of the items a delivery transaction is entered into the computer, again interactively. The system updates the inventory status in the distributor's warehouse and generates invoices. At some sites pricing is computed by the system. Record keeping in the CDPS system closely resembles the manual files used previously for manual paperwork processing. Each of the sites transmits the orders, inventory status, and sales information to the central IBM mainframe site in Ohio each night. The information is then available for production planning and accounting purposes. All cash collection and accounts receivable are centralized at the corporate level and dealt with on the mainframe, with credit data transmitted back to the warehouse locations nightly.

CDPS enabled the successful development of these applications by providing a programming environment which simplified the development process. Programmers at Champion compiled the applications in batch mode on the 370/155 and then tested and debugged on-line on the Series/l. Using the Champtalk language, they have a menu-based interactive programming support tool - similar to the mode of operation of the application program itself. Programs are written centrally in Hamilton, and object code is distributed to the sites. Hence, system development can be centralized or distributed. Use of the screen interface facility (described
later) enables the programmer to format and lay out the CRT displays that the clerks will use for order entry and other operations. As will be seen below, the screen layouts become an essential part of the system documentation. CDPS provides a utility for printing screen layouts in a form suitable for direct binding into documentation. In one large system design document reviewed at Champion, CDPS had been used to design, edit, and prepare the documentation of hundreds of screen layouts - the screen formatting was complete and on the machine at no extra cost before the project was even approved. (See appendix for example.)

Champion uses CDPS to facilitate the incorporation of on-line operational documentation into distributed application systems. Because of high clerical personnel turnover in the field, extensive formal training - either through instruction or written materials - would be prohibitively expensive. With CDPS the use of carefully tuned menus and other screen formats is particularly simple, allowing the application designer to plan and implement a dialogue between the clerk and the application to teach a new clerk the ropes. The menu approach is not unique to CDPS; CDPS provides a style whereby it is the natural outcome of all development projects. Since the menu-based dialogue is used by all software development people in their own use of the system, they develop a style of building applications systems that way as well. This is not as trivial a point as it may seem. Applications development people who are limited to card input and debugging through dumps have a great deal of difficulty empathizing with clerical staff at a terminal. In this case the instincts and choices made by developers will often fail to work effectively in practice. No amount of formal system development life cycle documentation
and planning can substitute for having a good feel for what it is like to sit at a terminal and use a computer as an integral part of your job. With CDPS the developers use the Series/1 in the same mode as the users. Moreover, the database is structured in terms that are easily related to manual filing systems. Personnel in the user departments can readily comprehend the system that is being designed for them. Because the menu-based dialogue and the simple database are readily understood by local personnel there is no need for computer specialists at hundreds of distributed sites.

CHARACTERISTICS OF CDPS

CDPS is an integrated package of application development tools oriented toward on-line transaction processing systems for small computers. The design criteria emphasized, 1) distributed processing, 2) that application systems serving different user departments should not be placed on the same machine. With a properly arranged distributed processing configuration, the only common component that could be shared between different applications would be the processor and some memory - the components that have seen the most rapid drop in prices. Champion has estimated that the monthly saving from hardware sharing between two user departments would amount to a few hundred dollars. This hardware cost savings, in Champion's opinion, would be more than offset by the increased cost of coordination between the two departments.

Because of the Champion design criteria, the original implementation of CDPS on the Series/1 was built with constraints on the number of
different databases that could be effectively operated on a single configuration. The scheduler also tends to saturate between seven and twenty-five tasks or terminals, depending upon the hardware configuration. If the distributed processing philosophy is adhered to, the limitations on single configuration complexity do not interfere with the capability of the full network. The constraints in the Champion implementation of CDPS were deliberately imposed by Champion MIS management to enforce programming standards. Champion believes that the standards and constraints have fostered the improvement in development productivity. The limitations imposed by Champion, however, are not inherent to the basic design of CDPS.

CDPS provides the following facilities that are required for building on-line transaction processing systems:

1) a programming language (Champtalk) and runtime support library;

2) a database management system (File Handler);

3) a screen interface (Screen Handler) with mapping support to relate locations on the screen to processable data structures;

4) a print control facility (Print Handler) that permits the integration of report heading and detail lines during output;

5) a task monitor, memory manager and scheduler.

The relationship among the components of CDPS are shown in Figure 3. CDPS is unique among presently available packages in providing the full range of facilities in one integrated fabric. In a traditional IBM mainframe environment, an installation might use CICS for the screen
interface and some of the scheduling function; the operating system itself for the remainder of the scheduling, memory management, and runtime monitor services; COBOL as a programming language; TOTAL or DLl as the database system; and a combination of application code and operating system utilities to produce reports. On top of this combination of components would be the requirement to effectively use OS JCL and possibly TSO command language. One important comparative advantage of the CDPS design is that the application developer needs to learn only a smaller set of internally consistent tools rather than a combination of systems and languages, none of which were designed to go together. The experience at Champion supports the hypothesis that CDPS would have a much shorter learning curve than traditional methods. Champion has found that new programmers can operate effectively in CDPS within a few weeks, whereas months are required to indoctrinate even experienced people in the traditional COBOL approach. CDPS also gains advantage from its limitations. Since CDPS has a restricted range of alternative methods for designing and writing a program, the designer and programmer spend less time considering alternatives.

1) The Champtalk language is frequently described as "COBOL-like". It was also described as "very close to Singer Assembly language". Although seemingly contradictory, both statements may well be true. Champtalk looks like an assembly language in that each statement consists of a label, an operator, and a list of operands, such as

```
UPDATE MOVE INPCOD,PLCOD GET THE PATIENT CODE
```

Assembly language has always had an appeal as a simple
CDPS Application Configuration

(CDPS components are shaded)

FIGURE 3

Diagram showing application configuration with shaded components.
language with few structuring rules. Statements are simply listed one after another. The order of appearance is the order of execution except for branching instructions. The procedural simplicity of assembly language, however, has been counterbalanced by the necessity to learn arcane rules for accessing and manipulating data and by the limitations on ability to specify structure in data. A different sequence of assembly code is usually required to add two fixed point numbers together than to add two floating point numbers, and conversion between data types requires strange-looking code. Data definition in assembly language is usually limited to listing names of fields with the space occupied and possibly an indication of data type. The data type information, if present, cannot be used directly to generate correct code sequences. Field names are short (usually no more than eight characters) and grouping of fields into larger units is difficult to express.

Champtalk, although it looks like an assembly language, has eliminated most of the deficiencies of traditional assembly languages, and it has achieved many of the capabilities of a higher level language. The language is compiled into interpretive code, permitting the "object" code to be transferred from one
machine to another and executed by the interpreter on the target machine. At Champion, most compilations are done on the 370 mainframe with execution on the Series/1.

The interpreter also permits compact storage of relatively complex operations. For example, the ADD instruction in Champtalk can add any two data types and store the result in a third. The data type conversions, selection of appropriate code sequences, and exception handling are handled by the interpreter. The programmer can express the action simply - in terms that are almost identical to a COBOL ADD verb. The difference is syntactic: where COBOL uses long words to express syntax, Champtalk uses punctuation. The syntax of Champtalk permits the same programs to be physically shorter. In circumstances where the more compact expression would be harder to understand, comments can be inserted to explain the action. The terseness of expression in Champtalk is probably an advantage over COBOL. COBOL is verbose in areas that are of little importance - such as explaining a single ADD statement - but provides no automatic method to assure documentation of the intended effect of significant sequences of statements.

In structuring data, Champtalk provides full access to the capabilities of the database management
system. Program working storage is also expressed in the same form as database structures. Not only is the power of the language enhanced, but the programmer need only learn one form of data structure expression. Access to the database management system is integrated into the Champtalk language. In the usual COBOL/TOTAL or COBOL/DLI environment, the programmer would write different codes to access a different database manager. Moreover, a CICS or IMS task manager would also impose unique coding requirements. These differences inhibit the transferability of most COBOL code. Since CDPS is identical in language and functionality on all machines, a Champtalk program is considerably more portable than most on-line database-oriented COBOL programs can be.

The one area where Champtalk compares unfavorably with modern higher level language (not COBOL) is in structured programming. Champtalk provides no IF-THEN-ELSE or loop control statements. These kinds of facilities could be provided with a simple macro-processor on top of the language. Or, if required, a higher level language could be built on top of the Champtalk base. TOMINY is working on a subset COBOL compiler for CDPS.

Because of the integrated character of CDPS, Champtalk is also used to serve the same function
as job control language on a batch mainframe system. A Champtalk program can cause another task to be executed by storing its name in a data structure shared with the supervisor. Hence, menu programs assist the entry clerk or other terminal operator in choosing the next action to take place. The control of execution is expressed in exactly the same manner as any other Champtalk program.

Champtalk programs are usually short - performing the functions required for a single transaction. The philosophy is that a program is executed as a result of a terminal operator's entry, responds to that entry, possibly reconstructs the image on the screen and resets the name of the program to respond to the screen, then terminates while the system waits for operator entry. This style is similar to the intended style of transaction processing under CICS. Ideally, this design approach would permit fully parallel programming and testing of all transactions. All status information between programs must be stored in the database - in a working storage section when not in the permanent records. It does take practice to design a system in small component transactions rather than larger programs. Observation at Champion compared to other installations suggests that the integrated nature of Champtalk and CDPS make it easier to train people in
the transaction philosophy and easier to keep them operating in this mode.

2) The **database management system** (File Handler) is similar to many other hierarchical database systems. The File Handler can deal with any hierarchical database structure, but may express and store the inter-relationship between records in a different fashion from other systems. The CDPS view of the database is described by analogy to paper storage media. A record corresponds to a piece of paper, containing lines which in turn are made up of fields. Records are stored together in a file, but may be accessed by an index. Secondary indexes may be used to find records from other characteristics. Links between records must be symbolic - i.e., by storing the value of an indexible field. The symbolic links eliminate chain pointers that have been grafted onto traditional database systems. The designers claim that the CDPS file Handler is far superior in performance to earlier database systems, such as TOTAL. It is also clear that the elimination of pointers has simplified the structuring of data. The result is less time in the development cycle required for "optimizing" pointer usage and finding obscure bugs resulting from pointer changes. The database also becomes more modular. A file is entirely self-contained and could even be
moved to a different machine if that were desirable.

The limitations on the File Handler are its restriction to very short names for fields and records and its orientation to on-line direct access as opposed to sequential access. The limitation on name size is of little importance unless a global data dictionary across a large system or group of systems is required. The tendency in large systems with small names is to use numeric names instead of mnemonics, reducing the readability of programs, introducing subtle typographical errors as bugs in programs, and requiring greater external documentation. If the File Handler were to be enhanced, we would recommend wider field names and a language syntax that permitted hierarchical reference to fields in data structures in the manner of PL/1. The inefficiency of the CDPS File Handler for sequential processing is of little importance when CDPS is used for on-line applications. Batch applications to consolidate data or post end of day records, however, run very inefficiently. Champion avoids this problem where possible by transmitting data to the mainframe and executing the consolidations there.

3) The screen interface and mapping support permit the design of screens separate from the program. The screen itself is used to lay out the screen format.
A utility converts the data entered into a template to drive the screen handler. The utility may also be used to prepare documentation of the screen design for a system - even prior to programming. In one system design observed at Champion, the production screen designs were incorporated in the proposal document - and the templates were already available before programming started. This technique can substantially reduce redundant work between documentation and programming.

4) The **CDPS Print Handler** is an interesting innovation in report printing. The program identifies lines as heading or detail lines. The print handler counts lines, deals with exceptional situations on the printer, and automatically inserts headings at the top of each physical page. Removing the line count and heading function from application programs substantially simplifies the task of writing a report.

5) The **scheduler** is closely related to the interpreter for Champtalk programs. In fact, the only application programs on the system are the Interpreter and the File Handler. The scheduler manages system memory using the File Handler to page data and object code in and out. The scheduler also manages the sharing of processor time between different tasks. The scheduler was designed to operate on a small machine with a
limited number of terminals. Scaling up to a larger machine with a larger number of terminals and tasks is limited by the algorithms and data structures designed for the small machine. Champion does not face this problem to a great extent because of the distributed processing philosophy that limits the number of different things done on a single box. However, were it desirable to use CDPS on a mainframe with more than two dozen terminals, the scheduler would have to be re-sized and perhaps rewritten entirely. TOMINY says that it has accomplished this change in its IBM 370 and Interdata versions of CDPS.

MIS MANAGEMENT AT CHAMPION

As Champion went through the stages of acquisition and then divestiture of its subsidiaries, the MIS Department spent considerable effort first consolidating, then separating the DP functions in the subsidiaries. In the late 70's, as the company has become more of a unified entity, MIS has been able to concentrate more on supporting the business. At one time, 70 percent of MIS activity was spent on maintenance and special reports generally ordered by lower level management. The ratio has been shifted so that today 70 percent goes into new systems development. To insure that business priorities are followed, every request for MIS services requiring more than one man-month must be approved by both the VP of MIS
and a divisional executive with appropriate financial authority. Champion has succeeded in involving the top management of each division both in setting priorities and in approving service requests. For this reason, minor requests, of interest only to a small number of lower level managers, may be passed over in favor of projects aimed at major assistance to the business. Moreover, the divisional executives are more involved in planning the resources required to support their activity.

In preparing a request for divisional approval, MIS analyzes the problem and prepares a report including a description of the proposed activity, estimated development and operational costs, estimated cost savings, and other benefits. Cost savings are limited to actual measurable out of pocket savings - such as whole positions eliminated. All other savings and benefits are described as intangible. Development costs are estimated using the standard costing rules described below. The division is charged standard cost, so that variances in assigned personnel or performance will be managed within MIS, not passed through to the division. On approval, a project team is assembled and the project re-estimated by the project manager and the project team based on the personnel actually assigned. Project performance is regularly tracked, using the PC/70 system, against the project team estimates. Periodically, the standard costing formulas are reviewed against actual project performance and revised where deviations occur. There is an additional aggregate tracking of standard cost against actual, since transfer payments to MIS are based on the standard cost, and MIS operating costs are determined somewhat by actuals. There seem to be enough other elements driving both the MIS budget and expenditures that this latter check on the standards is limited
to detecting only gross aggregate variances.

Standard cost project estimates at Champion are based on an analysis of the project that identifies the files to be processed and the programs that must be written. From discussions at Champion it was evident that project designs were aimed at moderate-sized programs with a transaction orientation. Champion views the system development job as consisting of two kinds of work - programming tasks and system tasks. Programming tasks involve the detailed design, coding, and checkout of individual programs. System tasks involve all the other activities that must be done to complete an application system - for example, system design, file layouts, systems integration, system documentation, and user liaison activity. Project development manpower requirements are based on estimates of programming task hours, system task hours, and special considerations, such as a new user or a new hardware or software environment. Programming time is estimated from a measure of complexity of the program determined by the following formula:

\[
\text{Program Weight} = (\text{Input Files})^2 + (\text{Output Files})^2
\]

Complexity is then determined from the program weight by the following tables:

<table>
<thead>
<tr>
<th>Complexity</th>
<th>Program Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple</td>
<td>up to 2</td>
</tr>
<tr>
<td>Medium</td>
<td>3 to 17</td>
</tr>
<tr>
<td>Complex</td>
<td>18 and up</td>
</tr>
</tbody>
</table>

A simple program by this formula is one with only a single input and a single output file. A complex program is one with more than five
files to deal with either as inputs or outputs. The original estimating formula for programming hours, derived in 1975, did not distinguish between COBOL programs and those written in another language. The first distinction made was for MARK IV report programs on the mainframes. Later it was found that Champtalk - the CDPS programming language - ought to be classified with MARK IV rather than COBOL.

The present rule for estimating programming time from program complexity is:

<table>
<thead>
<tr>
<th>Program Complexity</th>
<th>Programming Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>COBOL</td>
</tr>
<tr>
<td>Simple</td>
<td>30</td>
</tr>
<tr>
<td>Medium</td>
<td>60</td>
</tr>
<tr>
<td>Complex</td>
<td>80</td>
</tr>
</tbody>
</table>

These estimating formulas are necessarily rough, but have been studied and revised several times based on actual experience. The use of Champtalk (a full programming language with database capability) or MARK IV (a report generator) has proven to require 33 percent less programming time than the same programs done with COBOL. It should be noted, however, that the design of programs and files is also somewhat dependent on the expected programming language and database system. In particular, there is the impression that a project design for CDPS might have a larger number of less complex individual programs than the same functionality designed for COBOL. This impression has not been validated with empirical evidence, since no such experiment has ever been performed. With considerable extra effort, one could dig through the PC/70 data matching up projects to attempt to construct an experiment out of the projects that have actually been done.
This level of data analysis was not done in the present study. The attached chart shows the results of limited post-hoc analysis of project data from seven Champion systems. The reduction in programming time with CDPS is clear, although the experiment was not rigorous enough to prove the significance of the result.

System task hours do not show the same differential between CDPS and COBOL. The estimating formula for system hours is based on the number of COBOL programming hours and the number of programs in the system as a whole:

<table>
<thead>
<tr>
<th>Number of Programs</th>
<th>Percent of Total Programming Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - 4</td>
<td>60%</td>
</tr>
<tr>
<td>5 - 9</td>
<td>90%</td>
</tr>
<tr>
<td>10 - 19</td>
<td>120%</td>
</tr>
<tr>
<td>20 and up</td>
<td>150%</td>
</tr>
</tbody>
</table>

The estimates for system time do not presently show any improvement when using CDPS; however, this may be due to a lack of sufficient data to detect the difference. As more projects are done with CDPS at Champion, it might be worthwhile to check back to see if a differential has appeared. CDPS may also actually require more system time proportional to programming time because of the additional human engineering and built-in operator documentation required in a distributed system. It would be expected that this extra attention to detail in the development process would reduce subsequent maintenance and operating costs.

Champion MIS management did not seem to be very concerned about whether development costs would be markedly lower by use of CDPS. The
## CHAMPION INTERNATIONAL PROGRAMMING STATISTICS

### COBOL VERSUS CDPS - SEVEN SYSTEMS

<table>
<thead>
<tr>
<th>Type Programs</th>
<th>COBOL</th>
<th>CDPS</th>
<th>COBOL vs CDPS Average Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No. Programs</td>
<td>Average Hours</td>
<td>No. Programs</td>
</tr>
<tr>
<td>Simple</td>
<td>71</td>
<td>62</td>
<td>131</td>
</tr>
<tr>
<td>Medium</td>
<td>9</td>
<td>125</td>
<td>111</td>
</tr>
<tr>
<td>Complex</td>
<td>3</td>
<td>289</td>
<td>25</td>
</tr>
<tr>
<td>Total</td>
<td>83</td>
<td>77</td>
<td>267</td>
</tr>
</tbody>
</table>
decision to use CDPS in a project depends more on the operational improvements to be derived from distributed processing than on reduced development costs. In other words, CDPS is the distributed processing alternative at Champion and the choice being made is between a centralized mainframe system and distributed processing, not between COBOL and CDPS.

SUMMARY

This Champion distributed processing example illustrates one view of productivity enhancement in a data processing environment. Our research confirms that there are aspects of MIS management and the use of CDPS which can be viewed as enhancing software productivity in certain ways.

1. Centralization of system development enables successful development of common transaction processing systems at a reasonable cost. Champion uses the skill and experience of its Hamilton staff to maintain control over projects and to ensure quality of each system in a distributed environment.

2. Distribution of the hardware to the one-hundred-fifty businesses lowers uncontrollable downtime risk for each distributor. The businesses are not subjected to system failures at the central DP site since each business has its own hardware. Data transmission costs are also reduced.

3. Simultaneous development of CDPS applications and
and CDPS system software reduced overall development time by seven to twelve months. Use of the System 370 in Hamilton allowed Champion to have the CDPS applications ready for debugging on the Series/1 as soon as CDPS became operational on the Series/1.

4. **CDPS limits the complexity of the design process.**

This integrated package of application development tools, being aimed at on-line transaction processing, is intentionally limited by design criteria which dictate limited numbers of databases and terminals at each site. These limitations enable Champion to fulfill its basic MIS philosophy of one application per system. These limitations ensure minimal system complexity -- and consequently decreased probability of development failure.

5. **CDPS enables hardware independence.** System longevity became a major issue at Champion after the Singer experience. MIS management undertook development of CDPS with TOMINY in order to minimize the risks associated with tying a system to one hardware vendor. Ideally, then the programs and data would be portable.

CDPS has proven portable across IBM equipment at Champion, i.e., from a 370 mainframe to a Series/1 configuration. Moreover this system portability
should extend to other hardware as well. CDPS programs run on the Intel 8086 and the Honeywell Level/6 systems, so CDPS can be termed "hardware portable". However, this portability is limited by the data in any particular configuration. CDPS does not guarantee data portability as easily as code portability: for example, data stored in EBCDIC cannot be used directly when a CDPS application is transferred to an ASCII machine. However, the complete data description contained in a CDPS database has enabled TOMINY to develop an automatic data conversion program for moving between unlike machines.

6. At Champion new programmers become proficient in Champtalk faster than in COBOL. Where Champtalk takes a few weeks to learn, COBOL takes months. There is even some evidence, although undocumented, that CDPS is easier to learn for a new programmer than an experienced COBOL programmer.

7. Application testing and debugging proceed faster in CDPS than in the COBOL environment. CDPS provides interactive tools for examining the data used and modified by each transaction, while COBOL debugging is done from trace statements and dumps. CDPS can also be used to shorten the recycle time in debugging by interactive compilation. The potential benefit of interactive com-
pilation cannot be judged from the Champion experience, since most compilations are done in batch on the 370. This differential is difficult to gauge at Champion because CDPS applications and COBOL applications were programmed in batch mode during development of CDPS.

8. **CDPS is less complex than other aggregations of programming tools available.** CDPS programmers avoid the complexity of coding a mainframe computer with JCL, COBOL, CICS, report generators and other systems. CDPS in contrast is simple, consistent, and intentionally limited in scope for the distributed environment.

9. **Although CDPS clearly increases programming efficiency, there is no proof from this study that major differentials exist in non-programming (system task) time between COBOL and CDPS.** It is possible that CDPS system time is relatively longer due to increased on-line documentation of the systems in the distributed environment at Champion. Furthermore, including the documentation in the system itself though (menu-based interaction) assures the end-users that systems are actually documented.
APPENDIX

Example of CDPS System Development*

SYSTEM DESIGN

The programmer first plans the overall system including CRT screen design, data storage requirements, report layouts, and program operation. Then, the DBOS development tools are used to enter this information into the computer system. Turn on the system and a master menu, like the one in Figure 1, is presented.

If it is desired to enter new programs, modify programs, or perform other development functions, option number 1 should be selected. The General Systems Development menu is then displayed, which gives a more detailed list of options (Figure 2).

SCREEN DESIGN

In this case, a screen is to be created, and option number 4 is selected. A new screen is displayed and the programmer types the exact layout (Figure 3).

The information to be displayed by the computer is entered exactly as it is to appear to the operator. Wherever the operator is to type in information, an underline _ is typed on the screen, as for example, after PATIENT CODE.

After the entire screen has been prepared, press the enter key and the information on the screen is stored. No additional screen programming is required.

Changes can easily be made and a print-out obtained for incorporation in the user’s manual.
DATA BASE DEFINITION

The data base is defined next by the Data Definition Language (DDL). The file is set up to contain a number of records, one for each patient in the hospital. Each patient is identified by name and room number. The record on each patient is subdivided into two portions or line types. The first line type (01) contains the identifying information about the patient. The second line type (02) contains the type of illness with the hospital entry and estimated exit dates. In this sample program only one line type 02 is used. However, multiple line type 02’s could have been used if it were desired to keep a history on each patient.

The file is named HOSP and has room for 30 patients. The primary key is the patient code (PCOD) and there is space allocated for 30 different patients. The patient is identified by two fields, patient name (NAME) and room number (ROOM) which are 10 and 4 characters respectively. The second line type (02) contains three fields of 10, 6 and 6 characters.

Since it may be desirable to access the file also by room number or type of illness, secondary references are defined.

PROGRAM VARIABLES

The purpose of this section is to set up the input and output data streams and the program constants for each task (normally one CRT). In this example, the screen called CRT, is defined first. The variables in the top line of the screen are defined in the first three statements. Then the variable fields are named and sized.

Next, the data base for line 01 is named DBLIN1. Again, a convention is followed whenever a file is accessed. In this convention there must first be an area of 12 characters reserved for the data base manager. This is followed by a function code (such as read, write, etc.), status code, file name, space for forward and backward pointers to be used by the data base manager, key field name, and actual key to be used. Next there must be a list of the variables that are to be used from the given line, followed by the variable field names and sizes.

The HOLD command means that no one else may update this field while it is being updated from any one CRT.

Similarly, the commands to read line 02 of the file (DBLIN2) by the primary key (PCOD) and to read the file by the secondary key (ROOM) are set up.

Finally, the constants, screen messages, and other variables (literals) to be used within the program are defined.
MACRO NAMING - MAIF

PROGRAM NAME - MAIF

GENERAL HOSPITAL APPLICATION

THE DEMOIMATION PROGRAM IS WRITTEN TO SHOW HOW EACH IT IS TO BE RUN AT THE DR AND WITH AN IDEA OF WHAT IS NEEDED. AN APPLICATION CAN BE PROGRAMMED WITH THE DR FOR THE APPLICATION TO BE GENERATED. THE SKIP FOR THE APPLICATION PROGRAMM. THE SCREEN DESIGN. THE END OF THE APPLICATION IS WRITTEN. ALL OF THESE TASKS CAN BE DONE IN LINE AND THE APPLICATION RUN.

VARIABLE DEFINITIONS

THE INPUT/OUTPUT AREA NEEDS TO BE SET UP

DRIVE: 1
01 FILLER 10C FREEMAN KEY LINE 1
02 FILLER 20F FUNCTION (READ FILE)
03 FILLER 20F FUNCTION (READ FILE)
04 PLOC 15C LOCATION POINT 1
05 FILLER 6C LOCATION POINTER 1
06 FILLER 6C LOCATION POINTER 2
07 FILLER 6C LOCATION POINTER 3
08 FILLER 6C LOCATION POINTER 4
09 FILLER 6C LOCATION POINTER 5
10 FILLER 6C LOCATION POINTER 6
11 FILLER 6C LOCATION POINTER 7
12 FILLER 6C LOCATION POINTER 8
13 FILLER 6C LOCATION POINTER 9
14 FILLER 6C LOCATION POINTER 10
15 FILLER 6C LOCATION POINTER 11
16 FILLER 6C LOCATION POINTER 12
17 FILLER 6C LOCATION POINTER 13
18 FILLER 6C LOCATION POINTER 14
19 FILLER 6C LOCATION POINTER 15
20 FILLER 6C LOCATION POINTER 16
21 FILLER 6C LOCATION POINTER 17
22 FILLER 6C LOCATION POINTER 18
23 FILLER 6C LOCATION POINTER 19
24 FILLER 6C LOCATION POINTER 20
25 FILLER 6C LOCATION POINTER 21
26 FILLER 6C LOCATION POINTER 22
27 FILLER 6C LOCATION POINTER 23
28 FILLER 6C LOCATION POINTER 24
29 FILLER 6C LOCATION POINTER 25
30 FILLER 6C LOCATION POINTER 26
31 FILLER 6C LOCATION POINTER 27
32 FILLER 6C LOCATION POINTER 28
33 FILLER 6C LOCATION POINTER 29
34 FILLER 6C LOCATION POINTER 30
35 FILLER 6C LOCATION POINTER 31
36 FILLER 6C LOCATION POINTER 32
37 FILLER 6C LOCATION POINTER 33
38 FILLER 6C LOCATION POINTER 34
39 FILLER 6C LOCATION POINTER 35
40 FILLER 6C LOCATION POINTER 36
41 FILLER 6C LOCATION POINTER 37
42 FILLER 6C LOCATION POINTER 38
43 FILLER 6C LOCATION POINTER 39
44 FILLER 6C LOCATION POINTER 40
45 FILLER 6C LOCATION POINTER 41
46 FILLER 6C LOCATION POINTER 42
47 FILLER 6C LOCATION POINTER 43
48 FILLER 6C LOCATION POINTER 44
49 FILLER 6C LOCATION POINTER 45
50 FILLER 6C LOCATION POINTER 46
51 FILLER 6C LOCATION POINTER 47
52 FILLER 6C LOCATION POINTER 48
53 FILLER 6C LOCATION POINTER 49
54 FILLER 6C LOCATION POINTER 50
55 FILLER 6C LOCATION POINTER 51
56 FILLER 6C LOCATION POINTER 52
57 FILLER 6C LOCATION POINTER 53
58 FILLER 6C LOCATION POINTER 54
59 FILLER 6C LOCATION POINTER 55
60 FILLER 6C LOCATION POINTER 56
61 FILLER 6C LOCATION POINTER 57
62 FILLER 6C LOCATION POINTER 58
63 FILLER 6C LOCATION POINTER 59
64 FILLER 6C LOCATION POINTER 60
65 FILLER 6C LOCATION POINTER 61
66 FILLER 6C LOCATION POINTER 62
67 FILLER 6C LOCATION POINTER 63
68 FILLER 6C LOCATION POINTER 64
69 FILLER 6C LOCATION POINTER 65
70 FILLER 6C LOCATION POINTER 66
71 FILLER 6C LOCATION POINTER 67
72 FILLER 6C LOCATION POINTER 68
73 FILLER 6C LOCATION POINTER 69
74 FILLER 6C LOCATION POINTER 70
75 FILLER 6C LOCATION POINTER 71
76 FILLER 6C LOCATION POINTER 72
77 FILLER 6C LOCATION POINTER 73
78 FILLER 6C LOCATION POINTER 74
79 FILLER 6C LOCATION POINTER 75
80 FILLER 6C LOCATION POINTER 76
81 FILLER 6C LOCATION POINTER 77
82 FILLER 6C LOCATION POINTER 78
83 FILLER 6C LOCATION POINTER 79
84 FILLER 6C LOCATION POINTER 80
85 FILLER 6C LOCATION POINTER 81
86 FILLER 6C LOCATION POINTER 82
87 FILLER 6C LOCATION POINTER 83
88 FILLER 6C LOCATION POINTER 84
89 FILLER 6C LOCATION POINTER 85
90 FILLER 6C LOCATION POINTER 86
91 FILLER 6C LOCATION POINTER 87
92 FILLER 6C LOCATION POINTER 88
93 FILLER 6C LOCATION POINTER 89
94 FILLER 6C LOCATION POINTER 90
95 FILLER 6C LOCATION POINTER 91
96 FILLER 6C LOCATION POINTER 92
97 FILLER 6C LOCATION POINTER 93
98 FILLER 6C LOCATION POINTER 94
99 FILLER 6C LOCATION POINTER 95
100 FILLER 6C LOCATION POINTER 96
101 FILLER 6C LOCATION POINTER 97
102 FILLER 6C LOCATION POINTER 98
103 FILLER 6C LOCATION POINTER 99
104 FILLER 6C LOCATION POINTER 100
105 FILLER 6C LOCATION POINTER 101
106 FILLER 6C LOCATION POINTER 102
107 FILLER 6C LOCATION POINTER 103
108 FILLER 6C LOCATION POINTER 104
109 FILLER 6C LOCATION POINTER 105
110 FILLER 6C LOCATION POINTER 106
111 FILLER 6C LOCATION POINTER 107
112 FILLER 6C LOCATION POINTER 108
113 FILLER 6C LOCATION POINTER 109
114 FILLER 6C LOCATION POINTER 110
115 FILLER 6C LOCATION POINTER 111
116 FILLER 6C LOCATION POINTER 112
117 FILLER 6C LOCATION POINTER 113
118 FILLER 6C LOCATION POINTER 114
119 FILLER 6C LOCATION POINTER 115
120 FILLER 6C LOCATION POINTER 116
121 FILLER 6C LOCATION POINTER 117
122 FILLER 6C LOCATION PO
PROGRAM PROCEDURES

In the procedures section, the logic of the program is defined in simple straight forward instructions. All the code generated is re-entrant and therefore can be used by multiple tasks

The following instructions are all that are required in the HOSP example.

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRTROU</td>
<td>Read CRT</td>
</tr>
<tr>
<td>CRTWTU</td>
<td>Write CRT</td>
</tr>
<tr>
<td>COMP</td>
<td>Compare A and B</td>
</tr>
<tr>
<td>BE</td>
<td>Branch if A and B equal</td>
</tr>
<tr>
<td>BNE</td>
<td>Branch if A and B are not equal</td>
</tr>
<tr>
<td>MOVE</td>
<td>Move data from A to B</td>
</tr>
<tr>
<td>FILE</td>
<td>Execute file command</td>
</tr>
<tr>
<td>BLANK</td>
<td>Set the field to blank</td>
</tr>
</tbody>
</table>

The CRT screen is read by the CRTROU command. The function field is examined to determine the type of command and the program branches to the appropriate section to add, delete, update, etc. the file.

Within the program for each section, the following steps are generally followed:

- Set asterisks to LOC1 and LOC2 for the file processor.
- Move the data from the input screen fields to the output file fields.
- Execute the file command (such as Add, Read, etc.)
- Check to see if the file command was executed successfully or if some difficulty was encountered.

One such problem might be a command to update a non-existent record.

- Move the data from the file fields to the screen fields and write the output screen by the CRTWTU command.
- Go back and be ready to read the next screen.
DISP
MOVE +RF,PPANC
MOVE +ASTRA,POCO
MOVE +ASTRA,POCO
MOVE INFCCD,PPCOD
FILE DRLN2
COMP PSTAT, +ASTRA
BE DISPO2
DISPO2
COMP PLOC1, +ASTRA
BE GOOD
MOVE INFCCD,PPCOD
MOVE PHNAME, INNAME
MOVE FROMIN, INROOM
MOVE +RF,LPUNC
MOVE +ASTRA,LOCC1
MOVE +ASTRA,LOCC2
FILE DRLN2
COMP LSTAT, +ASTRA
BNE DISERR
BNE GOOD
MOVE ILLNESS, ILL
MOVE END, ENTER
MOVE EXITDATE, EXIT
B GOOD
SEC BLANK ILL
BLANK ENTER
BLANK EXIT
MOVE +ASTRA, INNAME
MOVE +ASTRA, LOCC1
MOVE +ASTRA, LOCC2
MOVE INFCCD, SHROOM
FILE DSPE
COMP LSTAT, +ASTRA
BE GOOD
MOVE NAME, INNAME
MOVE SPCCD, INPCOD
CRTWU CRT
CRTDU CRT
B RDSEC
UPDATE MOVE INFCCD, PCI0D
MOVE +RF,LPUNC
MOVE +ASTRA, LOCC1
MOVE +ASTRA, LOCC2
FILE DRLN2
COMP LSTAT, +ASTRA
BE UPLINO2
UPLINO2 MOVE +RF,LPUNC
MOVE +ASTRA, LOCC1
MOVE +ASTRA, LOCC2
MOVE INFCCD, PCI0D
FILE DRLN2
COMP LSTAT, +ASTRA
BE DELEN
DELEN MOVE +RF,LPUNC
MOVE +ASTRA, LOCC1
MOVE +ASTRA, LOCC2
FILE DRLN2
COMP LSTAT, +ASTRA
BE DELEN1
DELEN1 MOVE +RF,LPUNC
MOVE +ASTRA, LOCC1
MOVE +ASTRA, LOCC2
MOVE INFCCD, PPOCD
FILE DRLN2
COMP PSTAT, +ASTRA
BNE DELEN
BNE DELEN1
FILE DRLN1
COMP PSTAT, +ASTRA
BNE DELEN
B GOOD
ENDX CRTWU DONE
END END PROC

SET TO READ THE FILE
SET POINTER 1
SET POINTER 2
GET THE KEY FIELD
READ LINE 1
0.. WAS THE READ GOOD
Y.. PROCEED LINE 2
N.. SET NO-NO MESSAGE
GO WRITE THE MESSAGE
0.. 11 IT THE LAST RECORD
Y.. SET THE END STATUS
GET THE NAME
GET THE ROOM
SET TO READ LINE 2
SET POINTER 1
SET POINTER 2
READ THE SECOND LINE
0.. WAS THE READ GOOD
N.. WRITE THE ERROR
MOVE ILLNESS TO THE SCREEN
MOVE ENTER DATE
MOVE EXIT DATE
GO WRITE THE GOOD SCREEN
BLANK THE SCREEN FIELD
BLANK THE ENTER FIELD
BLANK THE MESSAGE AREA
SET POINTER 1
SET POINTER 2
MOVE IN THE KEY FIELD
READ THE FILE BY ROOM NO.
0.. WAS THE READ GOOD
N.. WRITE THE ERROR
0.. IS THIS THE END
Y.. WRITE THE SCREEN
GET THE NAME
GET THE PATIENT CODE
WRITE THE SCREEN
READ THE RESPONSE
READ THE NEW DATE
WRITE THE NEW DATE
0.. WAS THE READ GOOD
N.. WRITE THE ERROR
0.. WAS THE READ GOOD
SET TO READ LINE 2
SET POINTER 1
SET POINTER 2
GET THE KEY CODE
READ THE RECORD
0.. WAS IT A GOOD READ
Y.. CONTINUE
N.. SET ERROR MESSAGE
N.. SET ERROR MESSAGE
GO TO WRITE THE MESSAGE
GET SET TO WRITE THE RECORD
GET THE NEW DATE
WRITE THE NEW DATE
0.. WAS THE READ GOOD
N.. WRITE THE ERROR
0.. WAS THE READ GOOD
GET SET TO READ LINE 2
SET POINTER 1
SET POINTER 2
GET THE KEY CODE
READ THE RECORD
0.. WAS THE READ GOOD
Y.. CONTINUE
N.. GET THE ERROR
GO TO DISPLAY IT
SET THE DELETE FUNCTION
DELETE LINE 2 FIRST
SET TO READ LINE 1
SET POINTER 1
SET POINTER 2
GET THE KEY CODE
READ LINE 1
0.. WAS THE READ GOOD
Y.. CONTINUE
N.. GET THE ERROR
DELETE LINE 1
DELETE LINE 1
GO DISPLAY THE GOOD MESSAGE
WRITE THE LAST SCREEN
END OF THE PROGRAM
END OF THE PROCEDURE