The residence halls data management system at the University of Montana 1984

Stephen S. Goheen

The University of Montana

Let us know how access to this document benefits you.
Follow this and additional works at: https://scholarworks.umt.edu/etd

Recommended Citation
https://scholarworks.umt.edu/etd/7897
The Residence Halls Data Management System
At The University Of Montana, 1984

by

Stephen S. Goheen
B.A., Harvard College, 1970

and

Leslie E. D. Hemmingson
M.A., State University of New York at Binghamton, 1976

Presented in Partial Fulfillment of
the Requirements for the Degree of
Master of Science
UNIVERSITY OF MONTANA
1984

Approved by:

[Signature]
Chairman, Board of Examiners

[Signature]
Dean, Graduate School

2/1/85

Date
COPYRIGHT ACT OF 1976

This is an unpublished manuscript in which copyright subsists. Any further reprinting of its contents must be approved by the author.

MANSFIELD LIBRARY
UNIVERSITY OF MONTANA
DATE: 1984
THE RESIDENCE HALLS
DATA MANAGEMENT SYSTEM
AT THE
UNIVERSITY OF MONTANA, 1984

by

Leslie E. D. Hemmingson
in conjunction with

Stephen S. Goheen

Presented in Partial Fulfillment of
the Requirements for the Degree of

Master of Science

UNIVERSITY OF MONTANA
1984
Henmingson, Leslie E. D., & Stephen S. Goheen, M.A., August 13, 1984 Computer Science

The Residence Halls Data Management System at the University of Montana, 1984 (347 pp.)

Director: John R. Barr

This paper is an examination of the process of developing an automated data system for the Residence Halls at the University of Montana and the software engineering tools used in this development. This system was supposed to manage the applications and assignments of the potential dormitory residents. As well it had the additional functions of maintaining the students' campus addresses, generating reports concerning the housing population, and aiding in the analysis of the demographics of the student residents.

First, the original method of operation was analyzed and then the new structured system specification was created using DeMarco's methodology in conjunction with a PSL/PSA representation of the final analysis. Second, the automated system was designed following the methodology of Yourdon and Constantine. It was represented in structure charts and pseudocode. Third, a prototype, consisting of a sub-set of the system, was implemented by a separate implementer. Fourth, a working model of the data entry and updating portion of the system was implemented. And finally, the remainder of the project was 'cancelled' by the ultimate system owner, the University's Computer Center.

This examination includes a detailed explanation of the way in which the software engineering tools were employed and their ultimate value to this project. It also evaluates the successes and failures of the team, especially with regards to the final failure to produce a usable, automated system for the Residence Halls.
ACKNOWLEDGEMENTS

This graduate thesis project was a joint venture from the start to the finish. At every stage of the way Les Hemmingson and I worked in conjunction on it. In this sense it was atypical from the academic perspective. However it much more closely resembled the development of a software project in the real world. In fact this was one of our prime objectives when we proposed to undertake it. I am very thankful to have had the opportunity of working with Les on this project. He provided insights, skills, and a balance which made the project more valuable for me than it would have been as an individual venture. We worked well together, often complementing each other in terms of what we brought to the project.

There are a few individuals whom we would like to acknowledge for their assistance with this project. As is always the case the people associated with a project have a dramatic impact on both its qualitative and quantitative facets. We were most fortunate to have had an opportunity to work with some fine individuals.

Dr. John R. Barr, our thesis committee chairman, provided both helpful direction and more importantly critical review of our work. He was always available and willing to put aside his own work when we needed him.

Ron Brunell, the director of the Residence Halls, was an ideal end-user for this type of project. He had a clear idea of what he wanted to accomplish and he was able to communicate that to us. In the end he was so kind as to make sure that our academic progress was not impeded by the demise of their automated system.

Sandy Fortmann, the administrative clerk for the Residence Halls, was the poor individual upon whom fell the responsibility of having to explain their existing system to us. This she did tirelessly and pleasantly despite the fact that it took time from her already busy day. She was always ready to answer our numerous questions, listen to our suggestions, and thoughtfully evaluate what we were proposing. Without her it is hard to imagine our project having progressed as far as it did especially in the analysis stage.

Ben Forbes, our programming assistant, was a pleasure to work with and know. His willingness to learn and to strive for excellence were a real asset to our project. Beyond his formal involvement during implementation, Ben was often a sounding board and a good friend throughout the entire year.
## CONTENTS

### CHAPTERS

1. INTRODUCTION AND PROJECT OVERVIEW ............................................. 1
2. INITIAL USER CONTACT: REQUIREMENTS ANALYSIS AND PROJECT FEASIBILITY 10
3. PHASE I ANALYSIS: PRELIMINARY DATA FLOW DIAGRAMS .................. 29
4. PHASE II ANALYSIS: CURRENT SYSTEM DATA FLOW DIAGRAMS .......... 46
5. PHASE III ANALYSIS: NEW LOGICAL DATA FLOW DIAGRAMS; PSL REPRESENTATION; AND SYSTEM SPECIFICATIONS 61
6. PHYSICAL SYSTEM ANALYSIS AND SPECIFICATION ....................... 78
7. TOP LEVEL DESIGN ........................................................................... 85
8. EVOLUTION OF THE DATA DICTIONARY ........................................ 102
9. DETAILED DESIGN ............................................................................ 118
10. EVALUATING THE EXPERIENCE OF WORKING WITH A PROGRAMMING ASSISTANT 145
11. PHASE I IMPLEMENTATION: THE PROTOTYPE .................................. 149
12. PHASE II IMPLEMENTATION: THE 'UPDATE' MODEL ......................... 160
13. PHASE III IMPLEMENTATION: THE FUTURE .................................... 180
14. CONCLUSIONS .................................................................................. 185

### REFERENCES

189

### APPENDICES

A. FIRST REPRESENTATION OF CURRENT PHYSICAL DIAGRAMS ............. 190
B. INITIAL CURRENT PHYSICAL DATA FLOW DIAGRAMS ...................... 195
C. FINAL CURRENT PHYSICAL DATA FLOW DIAGRAMS ........................ 200
D. FINAL CURRENT LOGICAL DATA FLOW DIAGRAMS .......................... 205
CONTENTS (continued)

E. INITIAL NEW LOGICAL DATA FLOW DIAGRAMS 208
F. FINAL NEW LOGICAL DATA FLOW DIAGRAMS 215
G. PSL/PSA REPRESENTATION OF ANALYSIS 222
H. DATA DICTIONARY ELEMENTS 240
I. RHDMS - OVERVIEW DESIGN 250
J. BEGIN DESIGN - UPDATE DATA BASE MODULE - INITIAL DATA ENTRY 262
K. CONTINUE DESIGN - UPDATE DATA BASE MODULE - SUPPLEMENTAL DATA ENTRY 274
L. 1022 CODE - THE 'UPDATE' MODEL 291
<table>
<thead>
<tr>
<th>FIGURES</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 RHO preliminary fact sheet</td>
<td>13</td>
</tr>
<tr>
<td>3.1 RHO route of application chart</td>
<td>32</td>
</tr>
<tr>
<td>4.1 Early physical context diagram</td>
<td>49</td>
</tr>
<tr>
<td>4.2 Early physical global diagram</td>
<td>51</td>
</tr>
<tr>
<td>5.1 Early logical overview diagram</td>
<td>63</td>
</tr>
<tr>
<td>5.2 Intermediate logical overview diagram</td>
<td>65</td>
</tr>
<tr>
<td>5.3 Final logical context diagram</td>
<td>69</td>
</tr>
<tr>
<td>5.4 Final logical overview diagram</td>
<td>70</td>
</tr>
<tr>
<td>5.5 Final logical overview diagram - showing automated portion</td>
<td>73</td>
</tr>
<tr>
<td>6.1 1022 test data base sizing graph</td>
<td>83</td>
</tr>
<tr>
<td>7.1 Top level DFD of basic transform</td>
<td>91</td>
</tr>
<tr>
<td>7.2 Top level structure chart - transform centered perspective</td>
<td>92</td>
</tr>
<tr>
<td>7.3 Top level structure chart - transaction centered perspective</td>
<td>94</td>
</tr>
</tbody>
</table>
The completion of a software engineering project (6 - 9 credits) is an essential part of the Computer Science Master's Degree program at the University of Montana. A fundamental premise behind this requirement is that the project extends the educational opportunities available in the traditional classroom setting by involving the student in a situation where the software eventually created has real utility value with real end-user(s) who have real needs and requirements. In other words, the final outcome is a piece of production software instead of something which functions merely as an academic exercise with no life beyond the end of the class. In this way, the project bridges the gap between the artificial, hot-house environment of the classroom and the environment of real-world software development. At the same time, it provides a kind of laboratory where the student can build and test those software engineering skills which are the main focus of his graduate course work, i.e., the analysis, design, and management skills which make the difference between programming and software engineering.
This document represents the record of such a software engineering project. It is both a history of the work done to adapt an operation to computers and an analysis of the tools, methodologies, and techniques used to accomplish that task. In the most general of terms, the project involved something for which computers are commonly used - taking a currently existing manual system and automating some part of it, making it more efficient, adding the capabilities to do things not possible in the manual mode, but not radically restructuring or altering the logic of the original manual system. Among the formalized tools and methodologies used in the course of this project were DeMarco's [1] data flow oriented analysis concepts, Yourdon and Constantines' [4] structured design techniques, and PSL/PSA, [2] a software tool used for formalized problem statement and analysis.

The structure of this document is essentially chronological. However, while the focus moves from analysis through design, and finishes with implementation, relevant issues are discussed in the context of the particular phase in which they were raised. It seems to be a fact of software engineering projects in general that activities most relevant to one phase often seem to occur during another. As a result, the chapters for the most part
represent a mingling of narrative description and project analysis. We felt this was more appropriate than attempting a separation of those modes because that more closely reflects how the project was actually done.

**The Main Members Of The Cast**

This project was initiated in the fall of 1983 and it carried through the summer of 1984. The work was done by Leslie E. D. Hemmingson and Stephen S. Goheen in partial fulfillment of the requirements for a Master's Degree in the Computer Science at the University of Montana. It was done under the direction of Dr. John Barr, chairman of our graduate committee and Associate Professor of Computer Science. The scope of this project was somewhat larger than most. We were, to the best of our knowledge, the first here at the U. of M. to the make the project a team effort.

**What We Wanted**

Before we actually began to look for a specific project, we had a number of general criteria we wished to satisfy. Some of these could be characterized as physical requirements. For example, we had decided we wanted to work as a team and so, with regard to size, one of our require-
ments was for the project to be big enough to require two people, but at the same time something that could be managed within one year. Another requirement was that the project have a business, or at least business-like orientation, with a real user with real needs, independent of the Computer Science Department or the University's Computer Center. We wanted the programming to be applications level and user oriented. We specifically wanted to avoid doing a project with systems level or hardware dependent orientation. Finally, and most importantly, we wanted a project which would give us experience in all phases of software development, particularly with more regard to analysis and design than to coding. The Residence Halls project has come as close to fitting these requirements as we could have hoped.

Working For The Residence Halls Office

In more specific terms, the subject of this project was the University of Montana's Residence Halls Office. Every year they received more than 2,500 applications for dormitory housing. These forms were in four parts, containing more than 60 pieces of information which were processed and recorded manually in a complex system of card files and indexes. The focus of the project involved converting this manual system to an automated
data base, and providing means by which this information could be accessed through various reports to make the process of assigning rooms more efficient.

Initially, as we began our analysis, our very first impression was that some sort of improved data organization system was needed. However, as we pursued our analysis, using DeMarco's data oriented techniques, we came to realize that the Residence Halls Office was in fact already maintaining, albeit manually, a relatively sophisticated data base of information, arranged so as to reflect the various relationships between important elements of data, and complete with cross references and cross indexes (although these were not always totally formalized). The constraints confronted by the Residence Halls Office seemed to be more the result of the large volume of data to be handled and limitations of manpower rather than the result of any lack of necessary or appropriate organization.

As we progressed further in the analysis phase of this project, we moved toward some kind of data base management system as the appropriate solution. The data the Residence Halls Office was managing was kept in a variety of physically separate, logically flat, paper files, and it seemed perhaps some program like the University's
"System 1022" Data Base Management system might be the right vehicle to both combine all that information into one computer file and provide the necessary avenues of access. During this stage, we focused more and more on fully understanding both the contents of each physical file and the various relationships between those files, with an eye toward being able to construct a single data base file which would both reflect that information and ensure a correct transformation of the one form to the other.

Eventually, we came to understand that more was involved than just transforming one kind of data base, the Residence Halls Office's physical files, into another, a computer System 1022 Data Base File. One obvious complication involved simply the logistics of getting the information into the computer file in the first place. The raw data from the Residence Halls Office's multiple part application forms, which once had been simply separated and put into various folders, drawers, or even boxes, would have to be typed in at a computer terminal, and this would necessitate dealing with all the problems of verification and accuracy which go along with this level of data entry. Also, the Residence Halls Office had expressed desires for other capabilities, such as being able to do various counts and correlations, as
well as making rapid retrievals of selected portions of the information which it already possessed, but which previously it had not been able to access effectively because time and manpower constraints had been too great.

Therefore, the general form of the solution we eventually arrived at involved two segments. One was a unified, logical file, in the form of a System 1022 Data Base, constructed so as to maintain all the information and relationships of each of the separate physical files, and organized to reflect as best as possible the physical configuration of the application form. The other was a user interface which would have to accomplish a variety of objectives, but primary among these, in our view, was facilitating the speedy and accurate entry of data.

As we progressed through the design phase of the system, other objectives surfaced which required particular attention. Among those was the need to provide for novice or infrequent users. Since there was a strong possibility that much of the yearly data entry phase of the system would occur within a relatively short time span, it seemed likely this might necessitate, or at least lend itself, to using temporary personnel. Another, as mentioned before, was the need to attend to the problem of providing useful, flexible, organized
data retrieval capabilities, which had been expressed as a strong desire by the Residence Halls Office. As is most often the case with software engineering projects, as we progressed with our design, additional needs and/or desires continued to surface, and we were constantly adjusting and refining the shape of our solution. In fact, even during implementation it was necessary to work to fine tune the design. During the transition from design to implementation we produced a working prototype of the user interface for the data entry function to be reviewed by the user, and we were able to incorporate their suggestions and comments from that session into later versions.

With regard to implementation, two factors in particular contributed to the complexity of that phase. The first was that we were given the opportunity to bring someone into the project to do coding who had not been previously involved. We took that opportunity, even though we were at a very early phase in the design, for two reasons, first because we felt it would be a good test of our design skills, and second because we felt it would add another element of real-world software development to what we were doing. At the same time however, it meant we had to deal with an added layer of complexity and establish and manage another line of communication.
The second factor adding to the complexity of the implementation was that because we had not yet reached the detailed design phase when we accepted a programming assistant, we had to both do design and manage the ongoing coding at the same time. This meant we had to deal with the problem of doing design at a level of detail sufficient to allow code to be produced directly, while simultaneously working to finish the final overall design.

In the last analysis, the dual nature of our graduate thesis project, i.e. the need to both do the work and simultaneously reflect on and evaluate the tools and methods employed in doing it, was what most often provided the greatest source of challenge. It was also most often the greatest source of interest as well.
CHAPTER TWO
INITIAL USER CONTACT
REQUIREMENTS ANALYSIS AND PROJECT FEASIBILITY

Our Introduction To The Project

It was early in the fall of 1983 when we first began to consider how to satisfy the computer science thesis project requirement. Because of other class requirements, our schedule, as we had defined it, was to begin work on the project in the winter of 1984 and finish by the end of the summer or fall of that year. But as mentioned before, we had established a number of criteria for what we wanted in a project, and because we were uncertain what might be involved in finding something appropriate and getting approval, we wanted to try and ensure we had adequate time to look.

In the last week of September, we arranged a meeting with our thesis committee chairman for what we thought would be a preliminary discussion to outline what we wanted in a project, along with other matters of technical logistics. However, during the course of the discussion, our chairman mentioned the possibility of a project which he thought might fit our requirements, but which also might need to be started right away. He was favorable
to the team approach idea. He felt that in fact it had a lot of validity because it was the way most real-world software was developed. While we were concerned about our need to finish other classes that fall and about our tentative schedule not to begin work on a project until the winter, at the same time, if it was what we were looking for, it seemed unlikely we could afford to pass this project up. Therefore, we arranged another meeting for the following week to learn more about it.

During our next meeting, which took place the first week in October, our thesis committee chairman told us the project involved working with the Residence Halls Office to computerize some portion of their operation. Beyond that, he had no more details, but when we expressed an interest, he made a phone call to the director of the Residence Halls Office and arranged an appointment for later that morning.

Initial User Contact
Determining Project Requirements

At this point in our academic progress, we were just starting our study of the software engineering techniques we would later need to complete the project, and in retrospect, it seems likely it was as much good fortune as anything which kept us from taking on a project which
was too big or complex. Ultimately, we had very little time to spend studying the feasibility of the project or to determine a concrete requirements specification before accepting it, and for the most part we operated from intuition and a basic fear of getting in over our heads. However, it would have been very useful during this phase if we had had some kind of formal exposure to the subject of feasibility studies and requirements specification.

At the beginning of our meeting with the director of the Residence Halls, we were given a sheet, see figure 2.1, containing some basic information about what the RHO (Residence Halls Office) desired in terms of completion dates and output. Included on this sheet were two lists of "elements," which, during that first meeting we were able to determine meant the items of information the director wanted to collect and manage. Also on that sheet was a list of "processes needed", which we interpreted to mean desired outputs. It was also on this sheet that the first reference to a System 1022 data base was made, although it was passed over quickly with the proviso that we could use whatever means we wanted for the implementation.
COMPUTERIZATION OF RESIDENCE HALLS APPLICATIONS

2. 1022 base for program June 1, 1984.

ELEMENTS

1. Name
2. Social Security Number
3. Local Address
4. Permanent Address
5. Local Telephone Number
6. Permanent Telephone Number
7. Sex
8. Prepayment - Date Received and Amount
9. Living Option Choice (Coed, Noncoed w/ or w/o visitation, hall, quiet floor)
10. Personal Option Choices (Smoking, Alcohol and Roommate.)
11. Major
12. Prepayment Cancellation - Date and Amount Refunded
13. Quarter in Residence. F W SP SU
14. Parent or Guardian: Name, Address and Telephone Number
15. Handicap
16. Birthdate
17. Class Standing
18. Student Status (New, Transfer, Returning)
19. Marital Status
20. Type of Room (Single or Double)
21. Scholarship or Special Student Status
22. Order number of application received
23. Used and available Space
24. Special Program
25. Address Forwarding

CANCELLATION DATE AND TIME ARE SCORCED

ROOM ELEMENTS / Room Management

1. Single or Double
2. Telephone
3. Room Key I.D. Number
4. Outdoor Key I.D. Number
5. Room Number and Hall

PROCESSES NEEDED

1. Alpha Rosters w/any or all elements
2. Hall Rosters w/any or all elements
3. List of spaces in use - individual hall
4. List of empty spaces available - individual hall
5. Summary Numbers of any elements.

Figure 2.1
RHO preliminary fact sheet
At that time, one of the things we had learned about requirements analysis was that the needs and wants of the user should be determined, in as much as it was possible, without regard to what the final vehicle of implementation might be. The point was to have as few preconceptions as possible. At the same time, considering that our final system was implemented using System 1022, and, considering that we actually knew very little about data bases or data base management systems at that time, in looking back it seems it could have been a critical mistake that so little attention was paid to this item. With regard to determining overall project feasibility, once again it is our opinion that not having had prior exposure to any defined methodologies was an area of weakness which, fortunately, did not adversely affect the final outcome. It is our strong belief that the excellent general analysis techniques we learned and used during the analysis phase of the project probably compensated somewhat for that lack.

We were told that the June 1st deadline for final system start-up was somewhat flexible. They expected to get the majority of their applications between then and the end of August, and they wanted the system ready to process those applications. At the same time however, they were receiving a steady stream of applications
all the time, so a week or two one way or the other was not critical. However, the January deadline for designing new forms was critical because it was a date set by the printing office. One of the wrinkles of this project was that the director of the Residence Halls also wanted to design new forms which would be more in tune with his new computerized system. He felt it was necessary for us to coordinate our analysis with the production of these new forms. Suffice it to say that we were not at all sure what would be involved in meeting this requirement. Also, that deadline seemed to be the most troublesome, given our original timetable.

In any case, after the director's initial presentation of the information on the sheet, we spent the rest of the interview asking questions to determine both what was required as well as what was desired from the proposed system. The primary need expressed was the ability to collect and dispense data about applicants for dorm rooms. Specifically, this meant statistical summaries and demographic profiles. The principal need for this information was to enable the RHO to better determine such things as the ratio of female to male applications, or how many of their applicants were first term freshman. As an example, we were told of a problem encountered that fall where, because accurate determinations about
even such simple statistics as the numbers of male and female applicants had not been available, they had allocated space by guess and tradition and had been left with a situation where they had male students in cots, eight to a room in basements, while there were entire floors empty in some female dorms.

Another need expressed was that of being able to get up-to-the-minute information about such things as a student's campus address and phone number. As an illustration, we were told of the how the Residence Halls Office often functioned as a kind of back-up campus directory service during the first couple weeks of fall semester, which, it was felt, needlessly used a great deal of their time. It was explained that information about a student's campus address and phone number was integrally connected to their room assignment, and that during the process of making that assignment, such information was recorded on a portion of their original application form. At one time this part of the form had been intended to be returned to the student, but because it also served as the most up-to-date record of that information, it was kept in a box to be used later as a reference. The problem was that during the early portion of the fall semester, before a general university phone register was published, very often when requests for such information...
came through the campus switch board, they were simply routed to the Residence Halls Office. Having to constantly search their box of cards as a directory assistance service seemed to the director to be a very inefficient use of their time, especially when he could envision being able to have a computer very quickly print up a complete alphabetical listing of all dorm residents with their addresses and phone numbers for the switch board operators.

Other interests expressed included the possibility of producing mailing labels, and of being able to produce yearly summary reports of the demographics of the dormitory population, so that trends with regard to things like the overall age, sex, or return rate of residents could be compared from year to year. Also, discussed was the capability of producing intermediate listings of available dorm space to aid in the room assignment operation. We were told they did not want any kind of automatic room assignment system, but rather wanted a way to store preliminary room assignment information as it was determined so that organized reports could be produced for use in making further assignments. They also wanted the system to provide a year-end summary of all their data, both to archive in lieu of all the application forms
they were currently keeping, and so their data base could be purged and started fresh the following year.

We would like to make an important point with reference to this last item. We were told that under no circumstances would they want to keep information from one year to the next. As it later turned out, this was not totally correct. But we did not become aware of this fact until we were into the implementation phase of the project. In this particular circumstance, the oversight turned out not to be too critical. The information they needed to retain was fairly minimal, and a small modification allowed us to handle the situation. We do not mention it so much as an example of a failure, but rather as an illustration of how important it is to pursue as intensively as possible a full understanding of the user's wants and needs. We asked the question straight out, and the answer was given clearly and unequivocally, and yet, the information was incomplete.

It is clear to us looking back from this point in the project that conducting a thorough requirements analysis and feasibility study should be a critical part of any overall software engineering methodology. As such, sufficient time and resources should be allowed for doing these things in the setting of the graduate project.
And yet, for a variety of factors, not the least of which is the sense of urgency most students feel to find a project and begin working on it, it is unclear how much emphasis is put on this phase. In our case, we spent no more than a week considering before we officially accepted this project. One wonders if, once a project was started, it could ever be abandoned because it later turned out to be infeasible and still be considered a valid project.

**Determining The Range Of Users**

As the interview progressed, one area which we focused on with particular interest was determining who all the users of this system might be. Overlooking or not correctly understanding the role of some peripheral user we knew could result in big problems down the road. The director of the Residence Halls told us he wanted a separate system for use by his office alone. Some of his output he felt could be useful to other departments, but he stressed the notion of wanting a system for his use to satisfy his needs. The primary user he identified as his administrative clerk, along with other members of his staff.
However, included in the category of other departments (to whom some of his output might prove useful) was the business office, the campus operator, and the food service. During the course of our analysis, we classified these among the external interfaces to the system. What use the campus operator would have for his information has already been mentioned. In the course of talking about the involvement of the food service, it turned out that in order to live on campus, the resident must also subscribe to a meal plan. The director envisioned being able to provide food service with an accurate list of all dorm residents which they could use to enforce that requirement.

The involvement of the business office was less clear. We were told about how every dormitory application had to include a $100.00 deposit to hold their room. This deposit had to be recorded both by the Residence Hall's Office and the University's accounting department. If an application was withdrawn before a certain date, that deposit was refunded. After that date, the applicant was eligible for partial refunds dependent on the exact date. This information had to be coordinated in some way. Also, there was the matter of coordinating refunds for withdrawal from the dormitory after occupancy. While we did not get a clear understanding of how the
RHO interacted with the controller's office during that meeting, we noted it, along with the whole question of defining the community of peripheral users, as an area requiring further study.

A final area which came up for consideration during this initial discussion was the nature of the relationship between the RHO and the University of Montana's Computer Center. While we did not at that time have a detailed understanding of the precise position occupied by the Computer Center with regard to the production of outside systems, we did know that they were likely to play some role in the final outcome of this project. The director of the Residence Halls mentioned the name of someone he considered his contact in the computer center. He said he thought it possible that the data collected by the Residence Halls Office might prove useful to this individual for a registration system of some kind he was working on. This too, was not clear to us and we made a note to contact him at the Computer Center to investigate further this potential user.

The role of the Computer Center also came up in the context of discussing what physical resources might be needed for this project. In particular, the question of the availability of storage space on the University
computer was a matter of some concern. The director's expressed preference was to have his programs and data on the University's computer. He discussed the possibility of purchasing a micro system, but it seemed a less desirable option. We specifically asked about whether or not his office had, or was entitled to, any disk storage space on the computer, and his response was that disk space should not be a problem.

Our notes from that meeting indicate that as the discussion continued, we pursued our perceptions that disk space was a scarce commodity, and that without adequate space the system he was envisioning would not be functional. At that time we could not say exactly how much space might eventually be needed, but the director seemed quite confident that when we could make that determination with a fair degree of certainty, he would be able to acquire the necessary space.

It is clear, after working on this project for almost a year, that we could not have had, at the time of that meeting, as complete an understanding of the nature of the Computer Center's position regarding the development of independent projects which use their resources as we have now. Also, our user undoubtedly had good reasons for his certainty that space would be forthcoming when
the exact needs were determined. Once again, this is a point we dwell on not in order to find fault or place blame. Our firm belief is that a necessary aspect of a project such as this is to analyze and discuss what worked and what did not, to look as carefully at what went wrong as at what went right. However, given that interactions with the Computer Center, including problems with the availability of disk space, eventually resulted in this project not being put into operation, a more careful examination of what may in fact have been a very critical exchange seems worthwhile.

In DeMarco's discussions of analysis techniques, he stresses the importance of identifying all the users of any proposed system. He divides users into three classes, and while a more detailed discussion of his concepts and terminology is left for subsequent chapters, it is relevant to mention at this point that one of his classes is the system owner. Our assumption in this case, without having examining it critically, was that the Residence Halls Office was that user. What 20-20 hindsight showed was that the Computer Center was in actuality the system owner.

Again, the point is not that the analysis was done poorly or inadequately. On the contrary, we felt that by the
time we started our design, we had done a thorough analysis, and we still believe that 95 percent of our analysis effort was excellent. That we were just beginning our study of analysis techniques at the same time we were conducting our analysis must be considered a contributing factor. However, that we failed to fully comprehend the critical role of the Computer Center, only further underscores the importance of both the analysis phase itself as well as the importance of pursuing analysis beyond even the point where it seems finished. What might seem like a small oversight in the early phases of a project can have magnified repercussions by the time of completion.

Deciding On Project Feasibility

We spent more than an hour and a half in this first interview with the director of the Residence Halls. The following day we met again with our chairman to discuss what we had learned and to begin to consider whether this project was feasible and within the scope defined for a thesis project. We identified three major phases of work and discussed what kinds of milestones we might set. The first phase would involve an analysis of the current situation, during which time data flow diagrams and an analysis document would be produced.
It was felt this should be complete by December 83 or January 84. The second phase would involve design, and the hope was that if we could complete it by the end of March, we could do implementation during spring and finish the write-up over the course of the summer. In addition to doing the work, we would be required to document each step in the process. We took the following week to consider.

As mentioned before, one of our main concerns was how the schedule of this project would fit with our own schedule. During the course of our deliberations, we came up with three factors which we believed contributed to making the project feasible. The first was our conclusion (which, as it later turned out, was wrong) that the Residence Halls Office would be virtually the sole authority in terms of the actual use of the system. It seemed the role of other potential users was confined to simply having some use for whatever output this system might produce. Our belief was that not having to devote large amounts of time coordinating the needs of a variety of different outside users would simplify analysis sufficiently to perhaps allow us to complete it in a single quarter.
The second factor was that we felt we had been able to classify the user's requirement into two distinct categories, those which were essential for the system to be of any real value, and those which would be nice to have, but which could be added later if time permitted. To make this distinction would require the user's approval, but on the basis of our first interview, the likelihood of that seemed good. The director of the Residence Halls had seemed extremely flexible about the difference between what he needed and what he would like to have. This classification would allow us to divide the work in such a way that if we ran out of time before all features were implemented, we would still have a viable, working system to deliver, which brings us to the third factor - something we thought of as staged development.

We felt if we could identify early enough what the main functions of the system would be, we could prioritize those functions according to which were necessary to implement first. Our thinking was that we could do the design and implementation in stages. This meant we would likely be doing further development at the same time the system was being used. But it also meant that we would be more likely to provide the RHO with something useful, even if not totally finished, by the date required. We knew our first goal would have to
be providing whatever was necessary for the new application form, but beyond that we did not yet know enough about the system to select that first function. Still, it seemed like the appropriate way to proceed. So, on the basis of these thoughts and assumptions, we decided to accept the project.

Conclusion

This phase was probably far more important and warranted more attention and time than we had to spend on it. It was, in our opinion, the weak spot in our project. We made a number of observations and assumptions, both good and bad, which had a significant impact on how the rest of the project would progress. The differentiation of the user's requirements into classes, as well as the idea of doing the design and implementation in stages according to function was what made it possible to do the project at all. At the same time, the oversights and incorrect assumptions made during this stage would grow into much bigger problems by the time the project came to its conclusion. Given the time allowed, and our minimal background as far as a methodology with which to approach this phase, we are satisfied with the work done. At the same time, more class work and study of the methods and techniques employed in doing
effective, systematic, and complete requirements analysis and feasibility study before actually beginning the project, could possibly pay big dividends in projects yet to be done.
CHAPTER THREE

PHASE I ANALYSIS: PRELIMINARY DATA FLOW DIAGRAMS

Introduction

The next step was to undertake a detailed analysis of the actual functioning of the operation of the RH. We hoped to accomplish two objectives by doing this: first, we needed to fully comprehend what it was that they did and how they did it with their existing system; and second, we had to construct the formal specifications for their proposed, automated system. We chose to employ DeMarco's [1] method of structured analysis in conjunction with a PSL/PSA [2] representation of our conclusions. Together we found these techniques to be very valuable as they helped us to clearly analyze the existing situation and to later specify the new system, especially the relationship between the automated and manual portions.

DeMarco [1, p. 28] recommends a series of steps in order to produce a current physical data flow diagram. These include:

- determination of the context to be studied
- identification of the users affected
- user interviews
- data flow diagramming
- collection of sample data types
- walkthroughs with the user to verify correctness
- publication of the resulting documentation

These are meant to insure that no aspect of the current system is overlooked. We tended to follow these steps as we analyzed the situation at the RH. And we found that they did provide a good method for analysis. We also found that DeMarco's technique offered concrete, specific ways of collecting information, organizing it, analyzing it, and presenting it back to the user for confirmation. We will describe the actual advantages and disadvantages of his methodology as we present the analysis which we did for the RH.

**Initial User Interview**

In our feasibility study, the administrative clerk of the RH was identified as the person who managed the day-to-day operation of their current system of accepting, validating and organizing applications and finally making the actual room assignments. Consequently we scheduled our initial interview with her to begin to get a first hand look at their operation, an as yet undetermined part of which was to be automated.

We decided to take heed of DeMarco's suggestion that the
assessment of the operations be from the viewpoint of the data as opposed to that of any of the users [1, p. 27]. Therefore we were prepared to focus our search for a clear understanding of the current system on what the actual data was, what happened to it, and where it went during its stay in the RH system.

There were two objectives which we hoped to accomplish with this approach. First, of course, we wanted to understand the current system. And second, we wanted to be able to build a verifiable model of the current environment. This model would be the current physical data flow diagram and it would be the basis for the later logical models, which would eventually represent the new specified system. This meant that the physical model would have to reflect the user's concept of their operation, so that they would be able to validate its correctness and, indirectly, our understanding of their current system.

In our first meeting with the administrative clerk she presented us with a chart, which she had prepared to describe the typical steps they went through with an application. See figure 3.1. This representation was a blend of data and process orientation with a heavy dose of control information. Consequently we did not rely on it very much. Instead we asked her to begin to describe the
Figure 3.1
RHO route of application chart
life cycle of an application from the perspective of the application itself.

After providing us with some actual application forms, instructions for completing housing applications, and a brochure describing campus and community living possibilities, the administrative clerk began to go through what actually happens when a prospective student initiates the application process. At different times we found the description beginning to wander away from the data's viewpoint. However, it was always possible to smoothly return to the data perspective of the RH, as long as we remembered our objective. It was obviously not the administrative clerk's preference to describe her work in terms of pieces of data. In fact she tended to want to focus on what process occurred next, as opposed to what happened to the data next. However, with us keeping her focused, she was able to provide a very thorough picture of most of the data flows within their office. In fact looking back, she provided us with more information in that initial interview than we were really able to assimilate. Many of the points of clarification which we asked for later had been explained at the initial interview. However, we had been so saturated with information that we had not clearly grasped her earlier descriptions.
From our point of view we found DeMarco's methodology to be a very good one for questioning and directing the flow of the analysis. We liked the structure which the data orientation provided us as we used the analysis of the data flows to lead us to an understanding of the complexities of the day-to-day operations of the RH with respect to housing applications.

It is also interesting to note that the two of us used two different techniques to record the information presented during this initial interview. One used a more standard textual, outline approach, while the other employed a rough version of data flow diagramming itself to keep track of the new information. The text approach appeared to be better for listing the composition of the data flows at different stages, but the data flow diagramming technique was superior when it came to representing the actual flows themselves.

**Preliminary Data Flow Diagrams**

After the initial user interview with the administrative clerk we produced some very rough data flow diagrams. These fell into two categories: the first were small sub-sections and pieces of the total system; the second were overview diagrams, which tried to begin to put the entire RH system into some unified format. It is significant to note that
the act of producing these diagrams forced us to either make sense from our interview notes or admit that we did not yet understand some aspect of their system. It was not easily possible to bluff ourselves into believing that we knew what was going on when we did not.

Using our initial feasibility study and this interview, we were able to identify more clearly at this point some of the interfaces to the RH application and assignment processes. They included the admissions office, the business office, the dormitories, the housing secretary, and of course the students themselves. We also knew that the University's Computer Center would be involved in any automated system for the RH, but we did not comprehend until much later what a factor it would be. As it turned out, in DeMarco's terminology [1, p. 14] the Computer Center was the third level user, the system owner. The director of the RH was the second level user, the responsible user; and the administrative clerk was the first level user, the hands-on user.

At this stage we each felt that there were some areas in which we had confidence that we understood; there were other areas in which we knew that we were confused; and there were some final areas in which we had disputed opinions about what was going on. We had not yet collaborated heavily on
our analysis, preferring to maintain some independence and, consequently, two distinct points of view.

**Follow-up User Interview**

Our second interview with the administrative clerk was a combination of three elements: more data gathering; education about the nature of the data flow approach to system description; and a tentative presentation of our current understanding of the workings of their system for confirmation and clarification.

Our main focus at this meeting was to gather more information on those aspects of the RH system where we agreed about our confusion. And we also wanted to try to unearth any new areas as yet unturned. We began by describing back to the administrative clerk their own system as we understood it to normally work. This allowed us to further acquaint her with the data flow perspective as we used it to describe the typical application and assignment process.

We found DeMarco's techniques of data organization to be very effective when used to re-describe an existing system back to the user. The administrative clerk was able to clearly spot any discrepancies between our version and the
reality. We also found the data flow diagram approach to be flexible as we were easily able to alter any data descriptions or flows which had to be revised.

The re-description process pointed out some major areas which needed further clarification. For example the use of floor plans in the assignment process had never been discussed previously. When we came to trying to restate how the actual assigning of rooms occurred, it was apparent that we did not have the complete picture.

We also used this second meeting to collect information on all of the exceptions to the normal process. A great deal of this information had been briefly explained to us by the administrative clerk earlier, but we had not been able to assimilate it with all of the other new material presented at the initial interview. These exceptions included cancellation of applications, withdrawal from the residence halls, late applications, room or dorm changes after assignment, special billings, scholarship and fee waivers, and non-student residents. Now that our understanding of the normal process at the RH was firming up, we were in a much better position to grasp these exceptions and to figure out how they fit into the total picture.
Initial Physical Data Flow Diagrams

Since we had collected a major portion of the information needed to analyze the application and assignment processes at the RH, we set about converting our rough notes into more formal data flow descriptions. In keeping with our earlier objective of maintaining two perspectives for a while, we worked independently to translate our notes and rough diagrams into more complete representations according to DeMarco's methodology.

The result of having two perspectives became apparent when we sat down to explain our diagrams and the rationalizations for them to each other. Once again the diagrams clearly showed where we did and did not agree. The benefit came as we worked to resolve the discrepancies. Either through discussion or by re-checking with the administrative clerk, we were finally able to work towards an agreement on our understanding of their system and on our representation of it in the current physical data flow diagram. By doing this we had come to a good understanding of the RH system. Needless-to-say there were still some surprises waiting for us down the road but the framework had been built, which would allow us to integrate these changes into our later analysis and design. See Appendix A.
Integration of Form Design into the Analysis Phase

From our initial feasibility study we knew that the RH was planning on redesigning their application form. They were looking for a new form, which would gather some slightly different information than their existing one, and which would fit with their proposed, automated system. It was obvious to all from the start that the new form would be transitional and that there would most likely be a revised version once the new system was in place.

We now were beginning to have a better feel for the application's use within the existing system and we were facing a two month deadline for ideas on the new form and/or system. With this pressure we decided to focus attention on the relationship between the form and the information which the RH needed to collect. Our analysis was benefited by this increased attention to the actual data elements which were to appear on the applications form. As we came to understand them, we also better understood the entire system which the RH used to process applications and make assignments. Ideally the entire form design process would have come much later in the life cycle of this project. We realized that we were going to be forced to draw conclusions and make recommendations without a complete understanding of what would actually be the final designed system. However
this form design did aid us in our analysis of their current method of operation.

As a preliminary step in the analysis of the application form, we contacted three western universities, Oregon State University, the University of Oregon, and Seattle University, to see what type of housing application systems they were using. We were particularly interested in the type of form they employed and the amount of information which they tried to capture on the application itself. We found that two asked for only minimal information consisting of name and identification number, while the third wanted an entire multi-page questionnaire.

We determined that the RH’s needs lay somewhere in between the two extremes, which our admittedly small sample unveiled. We could see the advantage of the highly condensed form, which obviously minimized data entry and also required access to centralized information which the university would already have gathered. But we felt that this type of form was not feasible for the RH at this time. The reasons for this fell into three categories. First, the director of the RH had a six month deadline for the implementation of the new system, which he wanted to meet. Second, the RH wanted more specific information than the University on each student for their internal purposes. And
third, the director had informed us that their housing application often provided more accurate or more timely information than found in the rest of the University data files. These complications appeared to dictate some hybrid automated system with much increased complexity, which could never have been implemented within the required deadline.

Consequently, we decided to follow the director’s recommendation and to presume that the RH’s system would be a local one in terms of data gathering and that their new form would continue to be the source for their data collection. This would mirror their current method of operation and it would make the switch to an automated system smoother. We held out the hope that in the future their data processing could be integrated with the rest of the University’s. And in fact, we hoped to be able to analyze, specify, design, and implement their new system so that this transition would not only be possible in the future but fairly smooth as well.

In conclusion then, our quick survey of some other housing systems’ application forms did not turn up any easy solutions. It did provide, however, the first seeds for thought with respect to the possible future integration of some of the RH’s data processing with the rest of the University.
Application Form Analysis and Design

In order to analyze the existing application form and the needs which the new form would have to meet, we began to examine the old form with respect to the data which it contained. In essence we broke it up into physical and logical data flows. The original was actually a double sided form broken into four cards by perforations. These cards each served different multiple purposes within the existing system. The first card was sent to the Housing secretary in order to enter the student's $100 pre-payment in the University's financial records; the second became the RH’s alphabetical listing by dormitory; the third eventually became the dormitory’s list of their residents; and the fourth was a dorm notification to be sent back to the student. However, the second and third cards also served as the temporary numerical priority file before they were dispersed to their final destination. Because of this there was much redundant data even on this 'one' form. There also did not appear to be much of a logical relationship between the requested information and where it appeared on the form.

The thrust of our efforts was to decrease data duplication, to logically organize the information, and to make sure that all of the information which the RH was going to need would be accounted for on their new form. We started with the
list, see figure 2.1, which the director had given us in our very first meeting, and our notes from our interviews with the administrative clerk and we arranged a meeting to specifically work on the new form. We wanted to take the data elements which our early analysis had identified and to incorporate them logically into the new form.

From that meeting and our subsequent analysis came a proposed new form divided into three sub-parts. We had tried to offer a form which would make sense both in its entirety and also in any of its three parts. We were able to eliminate the old first part entirely because we envisioned its function being performed by a computer listing of the students who had sent in their $100 pre-payment. Therefore the new first part replaced the old second part, the new second replaced the old third, and the new third replaced the old fourth. Our intent was to improve the system of collecting information, while minimizing the disruption and confusion, which we knew would surround the implementation of a new automated system, and which would only be compounded by radical redesign of the existing form. This objective was also reinforced by our apprehension at having to analyze and design the new application form before we were even finished analyzing the current method of operation.
In yet another meeting we presented our proposal for the new form's logical division of information to the director and the administrative clerk of the RH. They offered some further suggestions and we ended up with a form which had almost all of the statistical information on one side and all of the housing questionnaire information on the other. At the same time each card was also logically organized in a similar manner. Another recommendation of ours was to consolidate as much of the 'official' data as possible into distinct regions on each card.

The RH then prepared a prototype layout of the proposed new form incorporating the format which we had suggested. We reviewed this prototype with them when it was complete, offered a few revisions, and gave it our approval hoping that we had not completely overlooked some vital aspect of the system which would only surface months down the road.

We felt good about the organization of the new form except for the fact that there was still duplicative data entry required by the applicant and that the form suffered from multiple functions for its different components. The duplicate data entry could possibly be eliminated in the future, however, if the RH ever moved away from maintaining card files. The increased complexity caused by one form composed of three sub-parts, each of which served a
different purpose during different stages in the application process, could also be eliminated in the future by a transition to a more fully automated system. This was beyond what we envisioned happening during the scope of our project. Until then the reality of satisfying the requirement of one form with three distinct but related sub-parts would dictate some inherent inefficiency.

Conclusion

This phase of our analysis was very important as it provided the groundwork upon which much of our later analysis would be based. We had begun the process of creating the current physical data flow diagram. This, when done, would insure that we did have a firm grasp of the actual workings of the current application and assignment processes. It would also serve as the basis for our further analysis, especially the new physical and logical data flow diagrams, which DeMarco's methodology called for next. And because we were inadvertently forced, by the necessity of early form design, to begin to deal with the issue of how the RH's data collection and use would relate to the data processing of the rest of the University, we had a start on the issue of integrated versus independent data processing. This aspect of our project would prove in the future to be of critical importance.
CHAPTER FOUR

PHASE II ANALYSIS: CURRENT SYSTEM DATA FLOW DIAGRAMS

Introduction

Now that we had collected a great deal of information about the operation of the application and assignment processes of the RH, we were ready to begin to create the actual physical data flow diagrams which would describe their current system. With these diagrams we wanted to build a verifiable model of their current environment. This would allow us to present our model back to the director and the administrative clerk of the RH for confirmation. The model would also be the basis for the next step in our analysis, the creation of the current logical data flow diagrams.

We hoped to build a model which would be quite familiar to the users of the current system. It would be filled with data flows and processes described in their terms. It would reflect their concept of how their system worked at present. And it would be full of names and descriptions which they would recognize as their own. This would provide them with many check-points and references to help them correlate easily between their current system and the model which we were building. The fact that we had earlier used the data flow diagram concepts, when working with the administrative
clerk, would also mean that she was not totally unfamiliar with this method.

In meeting with our thesis committee chairman at this stage we determined that in order to complete the analysis of this project, our goal would be to proceed through the following steps:

- Complete the current physical data flow diagrams.
- Get the RH's approval for the current physical diagrams.
- Complete the current logical data flow diagrams.
- Complete the new logical data flow diagrams.
- Get their approval for the new logical diagrams.
- Incorporate in any new additions to their system.
- Get their final approval on our analysis, as represented by the new logical data flow diagrams.

With this sequence in mind we set out to complete the next stage of the analysis phase of our project.

Even though we were now no longer trying to maintain two unique views of the analysis as we had done in the earlier phase, we did tend to employ distinct approaches. One of us began from the start to concentrate on the context diagram, which would delineate the domain of our analysis, and a complete global diagram, which would try to show the entire workings of their system. The other of us started to
concentrate on dividing the processes of the RH into sub-systems according to DeMarco's methodology [1, p. 72]. This partitioning of a large system into sub-system components can be carried out again and again at lower and lower levels until finally you end up with components that can be portrayed with simple data flow diagrams of primitive functions.

**Comprehensive Context and Global Data Flow Diagrams**

Using what we had already determined to be the external interfaces with the RH, one of us worked to identify all of the input and output data flows between these interfaces and the RH. The student interface was split into two logically different entities. Now we were using one interface box to represent the student before they actually took up residence in one of the dormitories, and another to represent them after moving in.

The context diagram which was developed at this stage was overly complete. See figure 4.1. It included all of the distinct inputs and outputs for the RH system as separate data flows. And it also showed some of the more important, totally external, flows between the interfaces themselves. We realized that these flows would have to be consolidated in the future, but for now we wanted to err on the side of
Figure 4.1
Early physical context diagram

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
being too inclusive to try to insure that we had not overlooked anything.

The global data flow diagrams produced at this time were increasingly accurate attempts to capture on one sheet of paper the total application and assignment processes of the RH. See figure 4.2. We knew that this type of total system overview would not normally be possible with most system analyses, but in this case the scope of the project was such that a global diagram was feasible. By collecting all of our information on one diagram we were able to see the inter-relationships more clearly, and we were able to make corrections and additions more easily. We did not plan to present these global diagrams to the users without much more education about what they were about because of the added complexity of having so much information on one page. But we did find that they were very useful for our analysis purposes.

What these global data flow diagrams did show graphically was that the RH system was very file intensive. In DeMarco's terminology [1, p. 57] a file is a temporary repository of data and the RH system was full of them, some more temporary than others. At this point it appeared that the current system used seven files: the deposit record book, the temporary number priority file, the temporary
Figure 4.2
Early physical global diagram
alphabetical file, the floor plan, the permanent dorm alphabetical file, the refund file, and the garbage file. Each of these files did serve its own purpose but there was obviously a great deal of data redundancy, since often a file only existed to meet a single need for data organization.

**Leveled Data Flow Diagrams**

At the same time that one of us was working on the overview diagrams, the other was beginning to try to produce some diagrams which were decomposed into levels. DeMarco suggests this as the way to represent any system that is too large to comfortably fit onto one page [1, p. 72]. It is analogous to the top-down problem solution approach in computer science, where a difficult problem is recursively partitioned into its own sub-problems until the resulting sub-problems are small enough to be easily solved.

In producing leveled data flow diagrams there are a number of conventions to follow. Whenever a process is partitioned into sup-processes, the child data flow diagram must represent exactly the same transformation as the parent with the same net inputs and outputs. All data flows shown entering the child diagram must be accounted for on the parent by the same data flows into the associated process.
This does allow for the use of balanced data flow decomposition where the flow on the parent diagram is broken into separate flows on the child, if the sum of the pieces is exactly equivalent to the original data flow. The result is the parallel decomposition of data and function. As the processes are partitioned in a top-down manner so are the data flows.

DeMarco suggest that the final determination on the amount of partitioning into levels has to be done by the analyst using good judgement on the readability of the diagrams [1, p. 82]. He does offer these observations:

- Listen to the data, let it lead you to a natural, understandable partitioning.
- Partition to create logical, meaningful interfaces.
- Partition as much as possible without hurting the readability.
- Seven is a good artificial approximate for the number of processes on one diagram.
- Clear diagrams are better than ones which follow all of the partitioning rules.

Using these guidelines we produced an initial set of leveled diagrams. They were rough and inaccurate in many respects, but they did represent our first attempts to organize the RH system into a decomposed, hierarchical structure.
Presentation to our Thesis Committee Chairman

With the two different perspectives which we had come up with, we were able to locate and clarify many points of confusion. We had been working on our individual representations and analyses in parallel, checking with each other and serving as sounding boards. Periodically there was the need for us to re-synchronize our efforts and understandings. This insured that we maintained a unified concept behind our analysis, even though we were producing different representations of it. Long discussions/arguments provided the basis for a much better understanding of the complete system which the RH used than either of us would have been able to gather working alone on this project. By the time we had come to a consensus on our final analysis, we were ready to present our diagrams and analysis of the current physical system to our thesis committee chairman.

We planned to use this presentation as a dry-run of the presentation which we would do next for the RH. Obviously it would not be an identical situation because of their different levels of expertise and familiarity with DeMarco’s methodology. As well we needed more of the critical analysis of our own work which our chairman had been providing all along. We presented finished versions of both approaches which we had been pursuing in this stage of our
analysis.

From this presentation the three of us were able to identify some areas where modifications needed to be considered. A few were as simple as the clarification of naming conventions. But others required us to re-think the relationship of one of the processes to the rest of the system. This concerned the manner in which dorm change requests, post-assignment but pre-occupancy, were to be represented. In fact the entire assignment process itself was probably one of the most difficult to understand during the course of our analysis. After careful thought we decided to not restructure our diagrams because we felt that they did best represent the operation of the system.

Presentation to the Residence Halls Office

We used the diagrams in Appendix B to assist us in our presentation of the current physical data flows and our analysis of the RH application and assignment processes. Both the global and the leveled versions of the diagrams were utilized.

At first we took some extra care to explain to the director what data flow diagrams were, how information was presented on them, and why they were so useful in systems analysis.
It did not seem difficult at all for him to grasp either their technique or their intent. We started with the context diagram and proceeded down through the lower levels. As we went, both the director and the administrative clerk were able to understand our presentation and to point out where our representation differed from their actual system. We were then able to make the corrections to our diagrams right then and there with their help.

We had the 'deposit acknowledgement' data flow originating from the 'screen requests and applications' process, when it should have been generated by the 'make assignments' process. We also were not complete in our representation of all of the necessary flows in the sub-processes of the 'modify assignments' process. We had left out some of the duplicate file updating which they normally performed when change or cancellation requests were received. It is interesting to note that one of these areas was the same one which had caused us confusion before, 'post-assignment change'. Now the process seemed to be becoming clearer.

After presenting the leveled data flow diagrams, we showed the director and the administrative clerk the complete global data flow representation of their system. See figure 4.2. With the background they had received, they were able to assimilate this complex version. It did prove especially
useful when discussing the complexities of the multiple files which they were trying to maintain. With this diagram it was obvious that some form of file consolidation was going to greatly simplify their system, but of course we did not have to tell them that!

The Current Logical Data Flow Diagrams

So far everything which we did was an attempt to document and analyze the current environment in the RH application and assignment processes, and to describe it in the terms of the current system. This provided an entire host of references which were very comfortable to the users. Now we were ready to begin to describe the current environment in logically equivalent terms.

Again DeMarco provided us with the techniques needed to accomplish this transformation [1, p. 235]. He outlines the following steps:

- Expand the data flow diagrams to remove the highest level physical characteristics.
- Decompose and normalize to describe the file structure in its most logical form.
- Eliminate procedural and historical characteristics by minimizing the data flows.

The objective is to remove as many of the physical
characteristics of the system as possible replacing them with logical equivalents. DeMarco warns that the derivation of logical equivalents is never totally successful, but it is the goal towards which to work at this stage of the analysis. Considerable improvement in the transformation of the system to a new logical representation will occur if these three procedures are employed.

The obvious starting point for us was the file structure of the current physical system. The seven different files were the result of distinct needs for data organization and retrieval, but they contained a great deal of redundant information. As well the fact that the data on any one student was 'distributed' made the job of collecting specific or statistical information very difficult. We immediately logically consolidated the deposit record book file, the temporary priority number file, the temporary alphabetical file, the floor plan file, and the refund file into one centralized super-file. We left the permanent dorm alphabetical file and the garbage file as distinct entities because of the unique functions which they served. This consolidation was the most satisfying aspect of this stage. It had been crying to be done from the first and it was a relief to rid of so many scattered files.

Next we worked to remove some of the physical
characteristics of the current system by consolidating data flows and processes into their logical equivalents. See Appendix C. Both flows and processes were now much more generic and descriptive of what the intent was at each stage of the current system. Their system fell into that classic transform pattern of data input, data transformation, and data output.

When we presented this set of current logical data flow diagrams to the director and the administrative clerk of the RH, we found them very comfortable with the new representation of their old methods of doing business. It definitely helped that they had both been previously exposed to DeMarco's diagrams and methods. It allowed them to view this logically equivalent model intelligently and without as much confusion as would have been the case in an initial exposure to data flow diagramming. We received their approval on our work with only one minor change required. The 'dorm assignment acknowledgement' flow had been inadvertently left off, yet even in this logical representation they were able to pick up on this omission.

By this time in our analysis it was not too difficult to make these logical changes to the RH system. It seemed as if we had been holding ourselves back from making some of them (i.e. file consolidation), as we finished up the work
on the current physical data flow diagrams. With this stage complete we were ready to start work on the new logical representation, which would incorporate the specified additions to the proposed automated system.
Introduction

The main function for us at this stage of our analysis was to take the current logical data flow diagrams just produced and to transform them into the new logical data flow diagrams. DeMarco describes this as the true guts of the analysis phase [1, p. 257]. This is because it is the first time that one is able to focus on how business ought to be handled, as opposed to how it is being done. This new logical model serves to describe the new system without regard to which portions of it will be automated and which will not. That final determination comes later. At this point it is necessary to try to define the domain of change, which will be all of the parts which must be or are likely to be changed by the new system. With this portion of the current system identified, the analyst must set about to reorganize the old into the new. This is where experience and imagination combine to help the analyst 'invent' the new system.

DeMarco begs off the responsibility for explaining how this is actually done, but he does provide his usual set of
guidelines [1, p. 262]. We found the following to be the most helpful:

- Partition to minimize interfaces.
- Pay attention to names, they should have conceptual integrity.
- Respect conceptual counting limits, i.e. no more than seven processes per level.
- Be complete.
- Respect the data conservation rule.
- Be prepared to start over.

These did help us to tackle the problem of how to best transform our current logical data flow diagrams into a new logical set of data flow diagrams.

**Preliminary Overview Diagram**

Our first thought lead us to see the total application and assignment processes as broken into four major procedures. See figure 5.1. The first filtered all of the raw inputs to the system, such as screening applications and output requests. The second built the RH data base. The third managed the data base in terms of adding additional information or changing existing erroneous information. And the fourth produced the output, either as answers to queries or as actual reports.
Figure 5.1
Early logical overview diagram

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
This represented a major consolidation of many of the previous processes as they were merged into these four logically cohesive components. We soon realized that this version of the system was overly simplified, but it still served as an underlining theme to what we were to do later. Some more decomposition would be necessary in order to capture all of the complexities of the RH system. This was especially true concerning the dual record-keeping, manual and automated, which was specified to occur in parallel for the immediate future. The RH wanted the ability to maintain their card file system as they began to utilize the new automated system. This would provide a measure of security during the transition stage, but it also added complexity to the analysis of their new logical system.

Second Stage Diagrams

Now with an eye on describing the new system and on determining which processes were to be automated, we further decomposed some of the processes of the preliminary new logical diagram. The resulting diagram, see figure 5.2, shows how the input screening, the interface with the business office, the room assigning, and the maintenance of the card file processes were all falling outside of the automated portion of the new system.
Figure 5.2
Intermediate logical overview diagram
The major complication with this diagram was how to represent the relationship between the 'assign rooms' and the 'maintain card file' processes and the two files in the system. Where and when the automated data base and the permanent dorm alphabetical card file were to be updated was not entirely clear. This question was resolved by splitting the 'new dorm data' flow into two. One flow would carry the information to the automated portion of the system, while the other would take it to the manual portion.

This dilemma points to the heart of much of the problem which we had with this phase of our analysis. We knew the RH fairly well by now, and we knew what we felt we could do for them in terms of automation, but the fact that they had to maintain two parallel systems was an added problem in terms of increased complexity. We had to figure out a way to represent these two intertwined operations in our new logical data flow diagrams. At the same time solving this problem would clarify the boundary between the manual and automated processes both to us and to the users at the RH.

At this stage we began to work on an initial PSL/PSA representation of our analysis and diagrams to date. PSL/PSA is a tool for computer-aided systems analysis and design. It is independent of the particular methodology, which it is used to support. It merely provides a mechanism
for capturing in an automated, stylized way the information, which the analysis methodology is generating. We knew that our analysis at this point would certainly have to be modified, but we felt that we were closing in on the task of pinning down the new logical data flow diagrams. For this reason and the fact that we realized that any PSL/PSA representation was flexible enough to be changed, we began to build a PSL/PSA data base of processes and data flows. As a side benefit we knew from previous experience with PSL/PSA that merely employing it would force us to clarify and justify some of the decisions we were making.

As it turned out we did not totally complete a full representation of our total system data flow diagrams in PSL/PSA at that time. This was mostly due to the fact that we were aggressively working on perfecting our data flow diagrams so they were changing too frequently for us to spend a great deal of time on maintaining the automated representation of them. We did, however, use this work later when we set about employing this tool to corroborate and support our final new logical data flow diagrams for the automated portion of the RH application and assignment processes.
Third Stage Diagrams

As the interfaces with the outside became clearer, we were able to consolidate them into three logical entities. There were all of the contacts which were external to the University, such as the potential student and their parents. There were all of the interfaces which were internal to the University, such as the dorms, the food service, and the switch board. And there was the business office with its unique relationship to the application process because of the $100 pre-payment and the validation number.

We were also able to consolidate the input and output flows in the context diagram into logically cohesive sets of data flows. This gave us a much more readable and clear representation of the relationship between the RH process and the outside. See figure 5.3.

At the same time we worked to polish our upper level diagram. See figure 5.4. The analysis was beginning to focus on finer details as we worked towards a final representation of their entire process. Now naming flows and processes was becoming an important indicator of how well we were capturing the RH system. We found that DeMarco's suggestions about the need for honest, single purpose names was very accurate. As we were better able to
Figure 5.3
Final logical context diagram
Figure 5.4
Final logical overview diagram
name the flows and processes well, we developed better diagrams and our analysis became more accurate. We also found that the distinction between the potential automated system and the manual one, which they wanted to continue, was becoming clearer as well. Two new processes were added to the diagrams to show this parallelism.

One new process was 'maintain the card file', which had to keep the permanent dorm alphabetical file accurate and as up-to-date as the automated data base. The other was the 'disperse the temporary card file' process, which handled the separate parts of the applications form as they were sent out to the student, the dorms, and the permanent card file in the RH. Both of these processes were in large part redundant with the work being done by the automated portion of the system. Much of what they accomplished could have been done by some specifically designed reports and a more complete use of the automated data base. However, for a certain transition period they were designed to provide a measure of security and a tie to the old methods of doing business. We hoped that in the future they, along with much of the application form cards, would become an unnecessary appendage, which could be disposed of. This would also complement the goal of the integration of some of the RH data gathering and processing with the rest of the University's system, which we hoped would occur in the
future as well.

After meeting with our thesis committee chairman, we were then able to accurately demarcate the boundaries of the automated portion of the RH system. See figure 5.5. We felt that this representation showed the best configuration of an automated data base working in conjunction with a manual card file system.

**Diagrams of the New Automated System**

We were now ready to construct a set of data flow diagrams which represented only the automated portion of the RH application and assignment processes. This required a major change in the scope of the diagrams because all of the manual processes would be excluded, unless they involved a direct interface with the automated system.

The resulting set of leveled diagrams, see Appendix E, clearly shows the relationship between the automated process and the necessary manual processes at the context diagram level. It also shows the partitioning of the new system into sub-processes and the resulting balanced data flows. By only including the proposed automated system, this representation focuses attention on what the new system actually is specified to do. As well it is clear that the
Figure 5.5
Final logical overview diagram - showing automated portion

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
boundary of the automated RH application and assignment processes is now entirely within the RH itself. There are no direct interfaces to this proposed system outside of the office.

We presented this set of diagrams both to our thesis committee chairman, and to the director and the administrative clerk of the RH. They all approved of the scope of the automated project, the data flow representation, and the accuracy of our analysis.

**PSL/PSA Specification**

We next took our earlier work on a PSL/PSA representation of the total system and modified it to represent the new proposed automated system. This required us to once again consider the naming of some of our processes and data flows. We did change some names in order to have more linguistic and functional consistency. For example the process of 'initial data entry', which had at one point decomposed to 'enter student name/address/number' and 'enter student demographics', now decomposed to 'enter initial student info' and 'enter additional student info'. This more accurately reflected what we wanted the process to do, since we did not want the new system to constrain the RH's timing of data entry. We wanted them initially to be able to
enter as much information about an applicant as possible. Then, if need be, we wanted them to be able to stop data entry and send the application along with the $100 pre-payment off to the business office. And finally they should be able to complete any necessary data entry at a later time.

The PSL/PSA representation was very useful as it provided another discipline to apply to our analysis work. It forced us to look very critically at the composition of our data flows and to describe their purpose in the new system. We also used it as our data dictionary, which DeMarco's methodology expects.

DeMarco defines a data dictionary as a repository of data about data [1, p. 125]. It serves as a central location for looking up information about or definitions of terms in the data flow diagrams, which one does not understand. A data dictionary should meet the following requirements [1, p. 127]:

- Definitions must be easily accessible by name.
- There must be no redundancy in the dictionary.
- There should be little or no information in the dictionary that is contained in other sources.
- It must be simple to update the dictionary.

PSL/PSA meets all of these requirements very well except for
perhaps the third in the way that we used it. All defined terms are ordinarily alphabetized when output on standard reports. Any name redundancy in a definition is trapped by the language, as is any inconsistency between the definition and use of a term. And since this is an automated tool, it is easy to modify your work as you need to, incorporating changes or additions to previously defined elements or relationships. In our use of PSL/PSA we tried to duplicate the data flow representation of our analysis. This meant that we did have information in our PSL/PSA which was the same as that in our data flow diagrams. The relationship of the processes to each other and the decomposition of the data flows were described in both locations. However, we did not feel that this was detrimental to our specification, but rather that it was a valuable asset in that it allowed us two distinct but complementary ways to represent our analysis.

Our final new logical data flow diagrams, see Appendix F, and our final PSL/PSA representation, see Appendix G, served as the structured specification for the new automated portion of the RH application and assignment system. Together they provided the necessary information about what the new system was required to accomplish and how much of it needed to be organized. By having to go through DeMarco's formal steps of producing the current physical data flow
diagrams, then the current logical data flow diagrams, and finally the new logical data flow diagrams, we were assured of understanding the existing system, of having a chance of 'inventing' a good new one, and of presenting a clear specification for this new one. The added benefit of using PSL/PSA also to represent this new specification was apparent to us. It provided our analysis a second perspective which served to double-check and support our data flow diagrams.
CHAPTER SIX

PHYSICAL SYSTEM ANALYSIS AND SPECIFICATION

Introduction

As the end of our analysis approached, we began to think about the need to determine some of the physical requirements of the RH's specified system. Although technically such decisions should wait until the entire system is fully designed, we knew that the reality of this situation forced an earlier consideration of these physical implementation issues. It was apparent that the RH would need an automated data base management system or the equivalent. Our early contact with the University's Computer Center had provided us with the information that they would only support software developed in COBOL or System 1022. Because of the existence of 1022, a sophisticated data base program, as one of the viable alternatives, we started to focus on it to see if it would be capable of fulfilling the needs of the RH's system.

We also knew that, if the system was to be implemented on the University's DEC 2060 in the Computer Center, then the RH would have to be allocated sufficient disk pages to allow them to store the program, run 1022, and build their data base of applicants and later residents. For all of these

78
reasons we felt that it was time to begin to determine how many disk pages might be necessary and to begin to try to get the storage space required.

**User Interview to Size Data Base**

In a meeting with the director and the administrative clerk of the RH we learned that the dormitories could house a maximum of 2450 students at any one time. However, the RH had the need to keep information on more than that many students. They received over 2600 applications in the fall of each year and, until final assignments were made, they needed to keep the information on all of these students.

There was also a normal turn-over of students during the year, especially as the quarters changed. In particular the director estimated that the total number of students in the dormitories dropped each quarter following the fall. However they did pick up new students as well, approximately 300 in the winter and another 200 in the spring.

The director's preference was to keep information on all of the students, who had been in residence at any time during the year, for the course of the year. This would enable him to generate some reports and analysis of his own about the population, which he had had in the dormitories for that
year. This meant that ideally there needed to be sufficient storage for approximately 3100 individuals in the RH data base. At the end of each year the director proposed to delete his data base and to keep his long term records on the students in the form of reports generated by the automated system. Then as new applications came in a new data base would be created for the next year.

The director was willing to concede that, if disk storage was not available for a 3100 student data base, he could delete from the computer all of the departing students at the end of each quarter. Then he would need some interim reports to have when he went to do his year-end analysis. Operating this way would require only enough room for approximately 2500 students at a time.

**Estimation of Disk Space Requirement**

We proceeded to determine, as best as we could, how much space 3100 students might take up in a 1022 data base. We were under no illusions about our ability to produce accurate figures, but we hoped to come up with some rough numbers which would give us at least some idea of what we were dealing with.

After consulting with a professor in the Computer Science
department who had much more experience than we in these matters, we set out to follow his suggestion that we use empirical testing to collect some initial sizing information. This required that we build some test student data in a 1022 data base, and that we observe how the size of the data base related to the number of students entered.

In order to begin to enter information into a 1022 data base it is necessary to build a data dictionary which describes the format and type of each piece of information to be accounted for. To do this we had to make some guesses about what information would be included in each student's record, about what length each field would be, and about which fields would be keyed in the data base. The data dictionary we developed at this stage was for our test purposes only, but we tried to make it as inclusive as possible, both in terms of what was in it and in terms of which fields were keyed. We wanted to err on the side of excess, if necessary.

We then set out to measure how many disk pages would be required for different numbers of students. To enter the specific information for hundreds of unique students would have been too time consuming given the rough, tentative nature of the measuring which we were doing. Consequently, we created one complete student record in a sequential text
file. Then we instructed 1022 to build a data base from this text file of one student. Finally we recorded how many disk pages were required to store this new data base. Next we duplicated the original student record in the text file using a text editor and then we repeated the building process. We continued to do this until we ran out of allocated working space on the computer.

We found that, after an initial period of disproportionate weight, the resulting function was very linear. See figure 6.1. This is not surprising given the way in which 1022 stores its information and given the fact that we were not entering unique student records. This later factor obviously distorted the size of the key tables, which 1022 had to build, but we felt that we had so overcompensated by including so many attributes and by keying so many of them that hopefully we were somewhere in the ball park. This conclusion was also supported by our doing a count of the number of bytes necessary to store each record in a sequential file and then working out some size figures for greater numbers of records in a standard text file.

**Recommendation for Disk Storage**

Our conclusion was that the RH would need approximately 550 disk pages on the University's DEC 2060 in the Computer
Figure 6.1
1022 test data base sizing graph
Center to store a 1022 data base containing information about 3100 students. We made this rough estimate to the director of the RH along with some other estimates of the number of pages which would be needed to store 1022 source code, COBOL source and executable code (if needed for our application), reports, and sequential data entry files.

The total number of pages projected came to just under 1000. We did believe that this was as good a guess as we could provide in this early stage of the project. The director of the RH took our information and sent a formal request for storage space on the University's computer to the Computer Center.
CHAPTER SEVEN
TOP LEVEL DESIGN

Introduction

Five months has passed by the time all phases of analysis had been finished and our work had been verified by the RHO. It had taken almost twice as long as hoped, and with our deadline for having something operational by June rapidly approaching, it was looking like we would have to try to accelerate the design and implementation phases somewhat.

In addition, we had added another complication to the project by deciding to try to work with a programming assistant for coding purposes. In particular, our assistant was an undergraduate, whose work for us would fulfill part of his requirements for an advanced programming course he was taking. We were to supply him with design documents from which he would create working code.

We had formed a good general picture of what the overall shape of our system might be from our analysis, and it seemed logical to translate the processes described in our final data flow diagrams to system functions. We had also decided to try to use System 1022 as our implementation vehicle. We had heard about a 1022
programming language, and in fact the first task we set for our programming assistant was to research this and come back with a report as to its availability, power, and feasibility as an implementation language.

One problem in particular with regard to our decision to use a programming assistant is relevant to the discussion of the design and implementation phases of this project. This had to do with the circumstances of our schedule. Originally, our plan had been to do an overview design of the entire system, then on that basis pick one function in particular as the focus for a detailed design, and finally implement that function in time to meet the June deadline. The remaining functions were to be designed, implemented, and added to the working system as fast as possible during the remainder of the summer.

While it seemed the addition of a programming assistant might help our project in terms of giving us more total time to spend on design, the fact was that when we agreed to accept the assistant, we had not actually done any design. This meant that in order to utilize this extra manpower, we would have to restructure our original design approach. Both top-level and detailed design would have to be done simultaneously with implementation,
and the complication was that we would have to be able to provide our programming assistant with the specifications for modules and functions which could be coded and tested independently, without yet having a complete picture of what the final system would look like. We would have to design from both the overview and the detailed perspective at the same time, which meant we would have to do a great deal of jumping back and forth from task to task, with the result that some of our time and energy would be spent in various coordinating activities. At the same time however, since we had completed classes in various structured design techniques, we had a much better command of the repertoire of design tools and methods we would be using to attack this stage in the project, and so we did have some cause for optimism.

Even though, because of our particular circumstances, it seemed like all levels of design were ongoing at the same time, the discussion of the design stage will be presented in three parts. This first looks at the top-level, or overall perspective. The second deals with the evolution of the main data structure, the System 1022 data dictionary, and the increasing problems of dealing with the Computer Center. The third looks at the detailed design.
The Design Perspective

To start with, we considered the shape of our final system from two different perspectives. The first, which is termed transform centered design, [4, p. 171-201] sees a system as a process or set of processes which "transforms" input — in this case the raw unstructured demographic facts about potential residence hall occupants — into the output — the structured reports and data sets which represent a unified picture or pictures of the residence halls population as a single entity. The second, which is referred to as transaction centered design, [4, p. 202-221] is a perspective which looks at a system as a collection of tools and utilities — or in other terminology, modules and functions — and the transaction (or control) centers required to coordinate them, which allow a user to manipulate and massage his data into the shapes which best suit his needs.

From the first perspective, we viewed our system as needing to monitor and control data input as much as possible, filtering errors, forcing the user at all times to enter at least plausible data, and giving every opportunity to review and modify incorrect entries before putting the data in a data base. After the information was in the data base, the system would have to provide
the user with the means to retrieve the data through a variety of prepared and formatted reports. The user would need the power to do supplementary data input, and to modify data after it has been put in the data base, but attempts would be made to moderate and channel this power.

From the second perspective, we saw our system as attempting to provide the tools necessary to allow the experienced user to custom tailor reports, or query the data base directly with whatever questions occurred to him. The system would allow data to be entered, changed, removed temporarily and returned, or taken out altogether dependent on the needs of the user. Predictably, it was a combination of these two perspective which we used to create our final system.

The First Design Session

Our first actual paper and pencil session resulted in the representation of our system as a single transform, which would be accomplished by a series of as yet only generally defined transaction centers. As mentioned before, the transform involved converting raw data input to an organized data base, and then to formatted output. The transactions would involve the logical organization
of all the possible operations to be done. The steps we went through started with trying to name all the general activities we could think of as necessary for the system to accomplish its transformation. Then we tried to group these activities together into logical sets, using our final, new logical data flow diagrams to aid us. Finally, we drew a top level data flow diagram of the transform and a high level structure chart to represent the basic hierarchy of this transformation. Refer to figures 7.1 and 7.2.

While involved in that process, we also began to consider the question of what to do about the user interface. We ended that first session by identifying two general aspects of the interface which we determined should be the focus of additional work. One concerned the shape and content of the user/system conversations, and the second involved questions regarding the mechanics of the choices.

**Top Level Design - Continued**

By our next session, we had a report back from our programming assistant about the suitability of the System 1022 program language, which is called PL1022. He found its level of sophistication to be somewhat analogous
Figure 7.1
Top level DFD
of basic transform
Figure 7.2
Top level structure chart - transform centered perspective
to BASIC. Control could be transferred either by 'goto' statements or by subroutine calls, but it did not have parameter passing capabilities. Also, it was not capable of any differing levels of scope. All variables were global. It did however, have the capability to declare arrays of string type, and possessed a set of powerful string manipulating functions. It did in fact seem totally suitable, and so we decided to proceed with implementation in PL1022.

In this second session, we determined that we needed to divide our attention between two tasks. One was to describe the overall structure of the system, i.e., what major functions it was going to perform, and determine what order they would be designed and implemented, and the other was to work on the actual physical configuration for the user interface. We continued refining our logical processes, referring to them now as transaction centers. We came up with a list of six, which remained the final list throughout the rest of the project. Those six we called 1) 'Update Data Base', 2) 'Make Room Assignment', 3) 'Dec System Tools', 4) 'Query The Data Base', 5) 'Request Formatted Reports', and 6) 'Exit The System'. We produced a transaction chart with each of our tentative transaction centers shown as separate modules. Refer to figure 7.3.
RESIDENCE HALLS DATA MANAGEMENT SYSTEM

Figure 7.3
Top level structure chart - transaction centered perspective

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Next we introduced some terminology to help manage our increasingly complex hierarchy of components. We decided to refer to our main components, those we had been previously thinking of as transaction centers, as modules. Our sense was that a module identified a logically distinct set of activities, and a system comprised of a set of modules seemed to make good sense. We decided to describe those components which we were beginning to group under each module as functions. Even at this early stage in our design, we conceived of the idea that the primary task of each module would be to coordinate its set of functions, and that each function would be just that, a unique, single action. As our chart grew more and more detailed, the idea of each module being a main menu choice seemed to naturally follow. From there it was a short step to defining a menu for each module which listed each of the relevant function choices. Another reason for using menus for controlling the transaction centers, besides the fact that lists of functions seemed to lend themselves naturally to that type of structure, was our feeling that the proficiency of the different users of the system would most likely vary a great deal. We felt that well constructed menus would allow a fair degree of immediate proficiency even with infrequent use.
We Decide To Focus On The User Interface

At this point, the question came up of whether we should work on the design of the user interface before proceeding with more detail on the structure of the system. We decided we had to turn our attention to working on the interface for two reasons: one because as mentioned before, the shape of our system design to that point seemed to lend itself naturally to a menu driven interface, which gave us a place to start, and two, because we needed to give our programming assistant something to do. We felt we could come up with sufficiently detailed design to give us at least a preliminary look at the main menu portion of the interface, and if we were careful, the coding could be done in such a way as to allow it to serve as a template for the many other menus we felt would be needed before the system was finished. Subsequently, much of the rest of our top level design efforts were focused on the user interface.

As we worked to develop a physical representation of what a typical menu might look like, we also began to articulate what the philosophy of the interface should be. The final result of this session was a tentative design document which outlined that philosophy, detailed the structure of the top-level control module with pseudo
code, and presented the generic shape of all menus, as well as the details for the specific menus for the main controller and the 'Update Data Base' module. See appendix I.

We thought that menu choices should be made with letters instead of numbers because we wanted to be able to provide the proficient user with the option of using strings of initials as a command-string to override the menus if desired. We felt the first letter of each function would be a more convenient mnemonic than strings of numbers. We decided that some sort of special character, like a '/' for example could be used to precede a command line to indicate that the user wanted to bypass the menus. The underlying idea was to create a layered interface which would accommodate both the novice and the experienced user.

In general terms, the interface philosophy articulated during this session involved a combination textual simplicity, flexibility of the use (what we thought of as the layered approach) and above all consistency of appearance. In fact, of all the factors mentioned, we felt the worst thing any interface could do was to make things which behaved in a similar way look different,
or require radically different kinds of responses to similar sets of activities.

We presented this document to our programming assistant, and set him to writing code. In the meantime, we turned our attention to determining what module would be the most appropriate target for our proposed prototype. At some point in the course of everything else that was happening, we had decided that at most, we would be able to implement one or perhaps two of the identified modules by our original deadline. As it in fact turned out, we were only able to finish implementation of about 90% of the 'Update Data Base' Module by June, but while we were working on design, we only knew we had to work as fast as we could and get as far as possible.

Our choice of the 'Update Data Base' module as the target for our prototype was based on the need to be able to at least present the RHO with something they could use by June. Since data entry would be the first thing they would have to do (we had heard by then that they had accumulated over 500 applications already), it was only logical to start there. Also, we still had to answer the question of what the structure of the actual data base file would look like. Focusing on data entry would also force us to design that structure. In fact,
we finally selected as our prototype the initial data entry portion of the 'Update Data Base' module, and much of the rest of our detailed design time for the quarter was focused on finishing that part.

**Conclusion**

During this phase of design, we were constantly forced to deal with numerous problems and distractions. While a discussion of the impact of the increasing involvement of the University's Computer Center is left for the chapter on the evolution of the data structure and detailed design, suffice it to say at this point that this involvement added to the problem of coordinating the various levels of design activity we were trying to carry on at the same time.

One issue of the top level design in particular required some serious rethinking and is a worthy example of the kind of constant juggling we were forced to do. Recall that one of the features we wanted to have as part of our interface was a command override of the menus. We spent a great deal of time discussing how we wanted this to work, both together, and with our programming assistant. However, it turned out that problems with limitations of the language, in particular the absence
of parameter passing capabilities combined with the fact (unknown to us when we first began our design) that no more than five levels of nested calls were allowed, added a great deal to the complexity of implementation. Fortunately, because of our programming assistant, we learned this early enough to make the necessary changes, and eventually we ended up dropping this from our design. What we decided to do instead was to work very hard to limit the number of menu levels to two, and rather than using menus at the function level, use extensive prompts to lead the user through the correct sequence of events.

Finally, the principal design tools we used involved a combination of Yourdon and Constantine structure charts, which were very helpful in clarifying the relationships between the various modules, functions, and subfunctions, and generic pseudo code algorithms of each component, which served as a guide for coding. We did not draw our charts to the level of detail possible using Yourdon and Constantine's techniques, where control, along with the actual parameters and the direction they are passed can be shown. We felt the addition of structured pseudo code made this unnecessary.
We found the concepts of transform and transaction centered systems extremely helpful as a way of beginning, and ultimately organizing the design of the overall system. In particular, the combination of the two perspectives allowed us to develop and maintain a global overview of the system as it evolved, while we worked on detailed design at the function level at the same time. Another way of expressing this was that it allowed us to do both horizontal and vertical design simultaneously.

Somewhere during the course of the top level design, we arrived at a name for this system. We decided to call it, originally enough, the Residence Halls Data Management System, or RHDMS.
CHAPTER EIGHT
EVOLUTION OF THE DATA DICTIONARY

Introduction

In System 1022, the object which defines the structure of the data is called a "data dictionary". Producing one can be considered somewhat analogous to the declaration of a record in Pascal. Each field of the record-like structure, called an "attribute" must be given a unique name. Its type and length must be defined, i.e., text, integer, or real, and the number of characters in the case of text, or the number of places in the case of numbers. Finally each field may be designated as keyed. This is done in a formatted file, separate from the applications code, and when System 1022 runs, this file is used to determine how the actual data is organized in the data base. Information can then be entered into and extracted from the data set on the basis of this structure.

Although they are integrally related, the process of defining the data dictionary is a separate activity from the process of writing the applications program. Our first rough cut of a data dictionary for this project was done while we were working to make an estimate of
the number of disk pages needed by the system. The reasons for focusing on the data dictionary in a separate chapter stem from two bases. One is the fact already mentioned, which is that its definition is a task in and of itself. The other is that major changes made to the shape of the data dictionary, which corresponded to changes in the overall data structure, and ultimately to changes in the shape of the Residence Halls Data Management System (RHDMS) in general, were closely related to an increasing involvement of the Computer Center in this project. Therefore, discussion of these changes provide a good backdrop for discussions of the role of the Computer Center in the project.

The Data Dictionary: Stage 1

Outside of the mock-up made to estimate disk space requirements, our real work on a formal data dictionary began at the same time we were working out our top level design. By the end of April, we had identified and laid out all the fields and defined their approximate size. See Appendix H. Even though this structure would be invisible to the user, we worked to organize it so as to make the data elements reflect, as closely as possible, the actual application form. We did this for two reasons,
both having to do with maintaining the logical consistency between the various representations of the RHO's data.

As mentioned earlier during the discussion of the re-design of the Residence Hall's application form, we had worked to group data elements according to what type of data they were and what was the administrative function involved with collecting them. The data entry portion of our system was designed to prompt the user for data elements from the application form in an order which reflected the structure of that application. Also in the internal workings of our system, as the data from each application was collected, but before it was actually stored in the data base file, it was held in a temporary array in memory, which in turn was constructed to reflect the structure of the data dictionary and the actual application form. By working to ensure consistency across what actually amounted to three separate versions of the same data, i.e., the data dictionary, the internal array, and the physical application form itself, we were actually working to ensure that the translation from one to the other would occur as accurately and completely as possible. This points to the second reason for establishing a logical equivalence between the data dictionary and the other forms of the data.
Something which has not been mentioned previously, but which was of prime importance, was the future maintainability of the system. If this new RHDMS was even remotely successful, it was likely it would be in existence long after we were gone. This meant that someone else would have to be able to make any future modifications. Also, as mentioned in a previous chapter, it was clear that the application form, and the data collected on it, was going to undergo additional changes as the transition from the manual to the automated system progressed, which assured that changes in the code and data dictionary would most likely be necessary on an almost yearly basis for some time. In fact, little did we realize, as we designed that first data dictionary, that radical changes would be coming much sooner than even we had imagined. In any case, to make modification and maintenance as easy as possible, we wanted to be sure that all that was required was to understand one version of the data structure. Even though changes would have to be reflected in at least two places, i.e., the data dictionary and the internal program array which represented it, those changes would be the same in both cases, thus diminishing the chances for error.

The first data dictionary then, was a single entity. It contained room for all the demographic information
supplied by the applicant, all the official data supplied by the RHO, and all the room assignment data. It reflected the fact that, as the director of the Residence Halls had requested, the RHDM S was being designed as a separate, autonomous system for the sole use of the RHO, and reflective of only their needs.

At the same time, on the basis of our estimate of disk space required, the director had sent a memo to the Computer Center in the middle of April officially requesting the necessary space, and towards the latter part of that month, we arranged a meeting to follow up that request with the Computing Services Manager, who is responsible for the administration of disk space on the computer, and the Administrative Information Systems Manager, who is the data administrator for the system. Our sense was that the meeting was just routine. We did not anticipate any problems, but we did in fact encounter some.

The Computer Center Factor

What we discovered was a view totally different from what the director of the Residence Halls had with regard to the relationship between the RHO's proposed new system and the rest of the university's computing facilities.
The Computing Services Manager was reluctant to part with any storage space, citing its increasing scarcity, and the Administrative Information Systems Manager was very adverse to the idea that the RHO was planning to collect and maintain what he perceived as data which to a great extent duplicated what the University was already maintaining. We were somewhat taken aback by these disclosures, since one, the director of the Residence Halls had continually assured us there would be no problems, and two, because we had made several informal contacts with the Computer Center over the course of our work to let them know what we were doing and had heard of no objections what-so-ever.

Most of the discussion centered around the problem of duplicate data. The Administrative Information Systems Manager's position was that any data collected by any office of the University about a significant portion of the student population was University data, and the more separate copies of it maintained, the greater the likelihood that some of it would be inconsistent. Also, given the premium on disk space, maintaining duplicate data meant using disk space unnecessarily. We pointed out that, while some of the more common logistical data collected by the RHO was indeed the same as that collected by the University, such as name, address, phone number,
etc., much of it was unique, and used primarily in the room assignment process. Information such as personal characteristics, visitation preference, roommate preference, and the like were maintained nowhere else.

We asked for suggestions as to how to resolve this problem, and the proposal was made that the RHO use what data it could from the University's master record file, only maintaining a separate data base for what was truly unique to its office. The advantages of such an arrangement would be substantial in terms of the time saved in doing data entry alone, although the RHO would be dependent upon another office for its data, (which, incidentally, was something the director had expressly stated he wanted to avoid). Also, given the prospect that the RHO was often the first contact a student had with the University, there was likely to be a percentage of students which the RHO would deal with, which the University had never heard from before. The problem was what to do with the applications from these people while the university's data caught up. Mr. Lucke's suggestion was that the RHO simply forward its more timely information to the appropriate University department, since they should have it anyway, and it would only be a short time before it would be in the student master record file.
Given a spirit of compromise, this seemed sensible enough, and we begin discussing ways we could redesign our system to extract data from the student master record file as opposed to having it entered at a terminal in the RHO, until we were told that that file was not in a form we could access, and in any case was not open to general access. While there were plans to put that information into a System 1022 file at some time in the future, it was unlikely to be in the near future, which was what we needed.

**The Data Dictionary In A New Form**

Eventually, a second suggestion was put forward, which seemed to meet with approval. The general thrust of it was that we would redesign the data base, modifying the system to deal with these concerns, by breaking it into two separate parts. We would, in effect, create one data base with two data sets. One would contain all the data unique to the RHO's operation, and which would continue to be their responsibility to maintain and under their control. The other would contain all the data which was duplicative of that currently kept by the university. In a sense, it would emulate that proposed System 1022 version of the student master file which was not yet available. It would be necessary
to have at least one field duplicated in both data sets so they could be logically tied together, and it would be necessary to manipulate the two as if they were one, which involved some additional complexity, but which was not outside the realm of reason. Other than that, the data in each set would be totally separate.

Initially, during the first year or perhaps two, the RHO would have to do all its own data entry, but the main attraction to the Administrative Information Systems Manager was that when that System 1022 version of the University's data did become available, the RHO system could dispense with entering and using its own version of that data and begin taking it from the University file. In that way, even though there would be temporary duplication, eventually overall data use and maintenance would be managed in the most efficient way possible.

From our point of view, the principal disadvantage of this solution was that it would increase the complexity of our design at a time when time was beginning to grow short. It would require that we design a version of the RHDMS which initially would do all the data entry, and maintain two logically distinct sets of data in a way which was transparent to the user, while at the same time, we would have to build-in the capability
for the system to be switched at some point in the future
to doing only part of the data entry and maintaining
only one data set while accessing other data from another
source. The RHDMS would not, of course, have the capability
of writing or changing any of the data in that outside
source. So it probably would be necessary to provide
a temporary data set which the RHDMS could write to,
and which could be used as storage for an applicant's
information in the event that the RHO was dealing with
a student which the University in general had not yet
heard from. However, a big advantage to this proposed
compromise was that it would allow us to continue working
on developing the RHDMS, and even more important, it
seemed to be the only way the Residence Halls Office
would ever get one.

Suffice it to say that given the uncertainty with regard
to the question of disk storage needs, the Computing
Services Manager was reluctant to grant all the space
requested. He did however, agree to make an initial
allocation of about a quarter of the requested amount
with a promise to add more after the end of spring quarter
when we would have a more exact idea of the amount needed
given this new wrinkle in our new system, and when other
demands for space had gone down because of the end of
the regular school year. We agreed to meet again with
the Administrative Information Systems Manager when we had worked out the details of our two data sets idea, and that was the end of the meeting. We did not of course realize that in fact no disk space would be allocated to the RHO at that time, or that our proposed compromise would end up being for nothing.

Shortly thereafter, we had a meeting with the director of the Residence Halls during which we reported the results of our meeting with the Computer Center. He seemed satisfied with his initial space allocation, but he expressed reluctance about the idea of having to depend on the University for his data. It was his feeling that very often, a student's first contact with the University was through his office. He did not see any reason why he should send that information away only to wait to get it back at some later time. He was very skeptical about how timely the University's data entry process would be. Also he again expressed his belief that it did not matter if there were discrepancies between his data and the University's, because the system under development was for his office only. We reminded him about those cases which he had mentioned during the early phase of our analysis where other offices of the University would have use for his output. And in any case, it was our opinion that we would be much
better off in the long run if we cooperated with the Computer Center. Accordingly, we proceeded with the task of redesigning the RHDM and restructuring our data dictionary.

**Two Data Sets And Beyond**

As we worked to separate our data into two distinct sets, we determined that for the sake of reliability, we would also duplicate the student's name in both sets. We met again with the Computer Center, and in particular the Administrative Information Systems Manager, to apprise them of what we were doing. We received a list of exactly what information the University had, and the technical information about what shape and format it was in so that our data structure would emulate as closely as possible what would eventually be available. See again Appendix H.

In the end, we became comfortable enough with the concept that we decided to separate another part of our data away from the main into a third data set. This data set was to deal with the information specific to dorm rooms. It consisted of things like the key/lock number, the phone number and so on. The idea was that when someone was assigned a room, instead of storing all...
the room information in the student's record, we would store the student's social security number in a record which described the room and link the two data sets together. In that way we would be able to easily get both general occupancy listings as well as listings of the room assignments for any particular group of students.

In effect, what this meant was that each dorm resident would have three distinct sets of data associated with his record, one which contained his name, address, phone number, and similar information, one which held his personal room preferences and the like, and one which contained the information pertinent to his room assignment. This would give us the flexibility of being able to look at our information from the point of view of either students or rooms. The only drawback to this idea was that the data set of dorm rooms would have to maintained on a permanent basis, which would require no more total disk space, but which meant less flexibility as to how that space was managed. Our sense was that this space requirement would be only moderate since the information in each record was limited.

This idea too, seemed to meet with approval from those at the Computer Center. And of course, at the same
time we were making changes in the final shape of our data, we were reflecting those changes in the detailed design of that portion of the 'Update Data Base' module we had selected as our prototype. We believed it had been a mistake not to have worked with the Computer Center on a more formal and regular basis from the beginning. But we were confident that a potential problem had been smoothed over and that everything was going well.

Our next indication that this was not the case came near the end of May during a meeting with the Computing Services Manager to confirm the status of the RHO's disk allocation. The RHO had arranged to have a computer terminal installed in their office, and we were getting close to installing our portotype for data entry. It seemed a good idea to coordinate both the demonstration and the necessary training in their own computer area. However, it turned out that in fact, no computer space had yet been allocated to the RHO, although the Computing Services Manager had said it would be, and during the discussion it became clear that there was still a reluctance to provide any.

We brought out the point that our revised estimates of space requirements seemed to indicate that the RHO
might need less total space than originally requested. His feeling was that the best course of action was to allocate enough space to get them well into their data entry phase, and after approximately 1000 student records had been entered and we had more accurate information to go on, the final allocation could be made. He said he would give the Residence Halls Office 500 pages of permanent storage, and a 1000 pages of working storage. This we deemed sufficient to get started and we met with the director of the RHO to advise him of the situation. Later, the Computing Services Manager called the director to tell him of the allocation decision, and once again, everything seemed to be under control.

Conclusion

While we worked to change the shape of the data dictionary, we also had to continue with our top level and detailed design. And as has been mentioned before, changes in one required compensatory changes in the other. The most challenging aspect was designing part of our system with what we characterized as a tear away capability. It seemed clear that at some time in the not too distant future, large segments of our system would have to be closed off, while others which did exactly the same sorts of functions, would have to remain intact. This
meant we had to pay very strict attention to making our system modular and to maintaining structural consistency throughout the modules. Some of this will be discussed in more detail in the chapters about implementation.

Ultimately, no space was ever given to the RHDMS, despite our efforts to work out some kind of compromise. It is still unclear exactly what the objections were, but our feeling at this time was that it had something to do with the politics of an administrative service of the University going outside the Computer Center to do its own independent system development. This, however, is only speculation. Suffice it to say that the control exercised by the Computer Center over the physical resources served as a complete block to the completion of the project.

We did not in fact find this out until we had spent another month finishing up the design and implementation of the 'Update Module'. We ultimately held a demonstration of that part of the RHDMS in our development areas on the University's computer, and while it met with approval from the RHO, it was never put into operation.
CHAPTER NINE
DETAILED DESIGN

Introduction

This, the last chapter which discusses the design stage of this project, looks at the evolution of the detailed design. It is not however, a design document. In fact, during this stage of the work, that document was never fully assembled in any formal sense. The fact that we needed to supply our programming assistant, with design work which was converted to code very rapidly, meant our design was produced in pieces which were never collated into any single entity. In a sense, the design was a series of mini-design documents, each of which contained whatever descriptions, discussions, and pseudo code necessary to specify whatever set of functions or subfunctions happened to be needed to continue the implementation. The advantage of this mode was that the feedback loop from implementation to design was very short. This meant problems encountered during implementation could be immediately addressed and dealt with by the designers. At the same time, the disadvantage was that a final, design document which could serve as a record of the entire RHDMS was not produced. See Appendices I - K.
A Template For All Modules

As mentioned in chapter 7, the top level perspective, of the RHDM S was of a number of modules, each of which was intended to be the focal point for a logically defined set of operations. As listed in the main system menu these modules were 'Update Data Base', 'Make Room Assignments', 'Dec System Tools', 'Query The Data Base', and 'Request Formatted Reports'. An additional menu choice was listed as 'Exit The System'. The structure of each of these modules was made to be essentially the same. First the user was informed as to which module he had arrived and confirmation to continue was requested. Then the module went into a loop which displayed the menu choices for all the functions umbrellaed by that module, one of which was to return to the main menu. Finally the function indicated by the menu choice was called. In chapter 7, we talked about our original intention to include a capability to allow the user to enter a command string which would override the need to go through the menus to make function choices, but this had been abandoned because the programming language did not handle this feature well and to force it would have added too much complexity for too little value.

The Specifics Of The Interface
In chapter 7, we had also discussed our interface philosophy and some general principles about the shape of the interface as a whole, whether as a menu or any other kind of prompt or message. These included the concepts of textual simplicity, flexibility of use (the staged or layered approach), and consistency of appearance. In this section, the specific details of the interface are presented.

First of all, conciseness was a concern. If it were necessary to give the user a significant amount of information, we felt a sequence of short messages was preferable to a single long one. We did not want to overwhelm the user with excessive verbiage. At the same time, and along the same lines, we wanted to keep the format of the interaction as open as possible, using blank lines and white space to highlight information rather than conglomerations of characters. For the sake of keeping the code simple, we decided to avoid any kind of fancy terminal manipulation. The only exception to this was the clear screen character.

We also decided to include as an option, the entry of a universal escape character which would allow the user to terminate what he was doing at any point and back up to a previous level or, depending on the circumstances,
continue to the next level. Our messages advised the user at all opportunities that this was an option. Since one of the goals of our interface was to maintain a high degree of consistency in appearance and function so as to create a situation where the most usual responses would become almost second nature, another feature we incorporated involved the availability of a default for any option presented. Most of the questions put to the user involved simple yes/no answers. We highlighted the default choice with square bracket thus, \textit{Y}, which indicated that this choice would be given automatically if a carriage return was entered. Our goal was to identify the most likely path through any part of the interaction, and allow the user to take that path with the fewest keystrokes possible.

\textbf{The Shape Of The Menus}

The basic content of each module menu included a header which identified the menu, i.e., what transaction center or module was currently active, a formatted list of functions which could be called from that menu, along with a command or prompt line for the user to enter his choice. Each function was described with a maximum of two lines of text. The first was the keyword or name of the function, and the second included a description.
of the work the function performed. We felt that most likely, some of the functions would be used only infrequently, and our view was that the description should serve as a kind of reminder of what the function was, rather than an explanation of what it did. Each of these sets of lines were to be separated by a blank line. Since it was necessary to ensure that no menu was too big to fit on a single screen, this limited the maximum number of choices to seven.

Initially, our decision to use the first initial of the function rather than a number as the indicator of the command choice had been made partially because of our intention to incorporate the command string override option into the system. When we later abandoned this feature, we elected to continue using initials to indicate function choice because we felt most of the actual data entry would be alphabetical, which would mean a typist would be more comfortable with the keyboard than the number pad, and also because we wanted to leave open the possibility that some day someone else might want to enhance our interface with this option.

One problem in particular that needed further consideration was how to handle an incorrect entry. We wanted to keep the mechanics of our system as simple as possible,
and therefore we were determined to avoid any kind of complicated terminal control features. Our two choices were to either clear the screen, give an error message and redisplay the menu, or simply scroll the menu with an error message line, let the user try again, and hope he did not make so many mistakes that the entire menu scrolled off the screen. Both approaches had their problems, so we put off a decision until we could actually see what each option looked like on the terminal.

The Design Of The 'Update Data Base' Module

We selected the 'Update Data Base' module as the focus of the first phase of our detailed design for two reasons. The first was that the functions it supervised were concerned only with various aspects of data input, and since that was the first third of the overall transform accomplished by the system, it seemed only logical that it should be the beginning. The second was based on our assessment that much of the actual detailed code required by the system would be associated with the task of data entry, data formatting, error checking and so on, so that having this section done by our coder would put us well on the way to having most of the final code written by the time he was finished with his obligation to the project. Our estimate of the ratio of code needed
for this module as opposed to the rest of the modules turned out to be essentially correct, and was in fact a significant enough amount that our coder was able to actually implement only about half of it. We realized this at a relatively early stage, which was why we reduced the scope of our prototype to include only the initial data entry portion of this module.

The Classification Of The Data Entry Functions

After deciding to focus our design efforts on the 'Update Data Base' module, our next step was to identify those functions it would include. Since its main responsibility seemed to be input, we first looked at the possibility of classifying input into different types. We decided we could identify three types of data entry, depending both on the time and the circumstances of its occurrence.

The first type involved the first time any information about an applicant was entered into the system. In addition to actually accepting data, the principal operations which distinguished this class of data entry from the others involved the creation of a new record in the data base, and ensuring that it did not duplicate any other existing record. Our view was that it should be possible for the user to enter part of the total
data, and interrupt the process at any point for continuance at a later time. We later had to modify this view slightly, but the principal remained the same. We identified this as the initial data entry class of input.

The next type we thought of as supplemental data entry. The activity which distinguished this operation involved the retrieval of a specified record for which some entries had already been made, and completing some or all of its remaining data fields. We envisioned a scenario where the RHO was receiving large numbers of applications on a daily basis, too many to be processed at one time, which needed to be forwarded to the business office so the $100.00 prepayment could be recorded and a receipt issued. Since priority in room assignment was based on a first received, first assigned basis, the RHO would want to open a record for the applicant and enter minimal data so priority could be established. Then the form could be forwarded to the business office and when it was returned, data entry could be finished without too many complications.

We identified a third type of data entry as involving the need to modify, or change data already in some very specific ways. This we thought of as changed data entry, and the scenario used to illustrate this was the case
where some specific error in data entry had not been caught until after the record was entered into the data base, and it was necessary to modify data on a case by case basis. In this class of data entry, we also included the case where an entire record would be removed from the data base. On the basis of these classifications, we attempted to identify a set of functions to handle these types of input. These were 'Add Student Records', 'Complete Student Records', 'Modify A Student Record', 'Delete A Student Record', and 'Undelete A Student Record'. Later, as we refined our classifications of the data, we identified an additional function which we called 'Withdraw/Cancel A Dorm Resident'.

The Classification Of The Data

Eventually we also decided to classify the RHO's data according to whether it came from the student, or was initiated by the RHO itself. This later class we termed 'Official Use Only' data. Within this category of data, we later made a further subdivision. We identified a special category for that set of data which was relevant only in the unusual circumstance that a student withdrew from the dorm prior to the end of the semester, and, because of special extenuating circumstances, a refund
was given, even though normally students would not be eligible for one.

**The Design Of Initial Data Input Processing**

At this point, we were working our way to a more and more detailed picture of the functioning of the system. We selected the 'Add Student Records' as the first function to work on primarily because it was necessary to have data before anything else could be done with it. Our next step was to identify the major parts of the 'Add' (Student Records) function. See the structure chart in Figure 9.1. Its first task, we knew, was to do some kind of input, so we had to decide what the basic shape of the input vessel should be. We decided the basic unit of data entry should be an entire student record (although some fields could be blank or null) because the basic operation of System 1022 was oriented around record structures. We knew it would be necessary to allow for data to be entered or retrieved from single fields of the record, but it would be first necessary to deal with the record as a whole before addressing any individual fields. Therefore, we specified a subfunction, which we called 'Get Student Info', which provided an empty student record for the user to fill out, and
figure 9.1
structure chart for the ADD-STUDENT-RECORDS function
then return that record to the 'Add' function for further processing.

The next question was what to do with the record once data entry was finished. There were two choices. At first glance, the simplest and most obvious seemed to be to place it directly into the data base. That alternative seemed the most direct and, we reasoned, would be the easiest to understand and maintain by a third party. However, there were two factors which mitigated against this. One was the amount of time and processing overhead involved in each access of the data base. Given that at peak times the RHO might have a hundred or more records to enter in a day, that access time could be a factor. The second was the fact that, according to the System 1022 documentation, the data base file itself was most fragile and subject to damage when it was open for updating. Since constant opening and closing was cautioned against, the next best course of action seemed to be to leave the file closed as much as possible and use some kind of buffering scheme. This was the course we took.

We decided to create a temporary sequential ASCII file which would be used to store student records while the user was doing data input. We specified a subfunction
to handle the task of storing them called 'Manage Temp File'. The complexity associated with using a temporary file as a buffer centered around the fact that there had to be an exact correspondence between the way the file was formatted and the structure of the data dictionary, so that the information in the file could be transferred to the appropriate fields in the data base. The student record structure referred to so often during the course of this discussion existed as a physical entity only in the data dictionary. All other structures were logical representations of that form. To ensure correct transfer, the data in the file had to be formatted just like the data dictionary. This meant that every input field needed to have some kind of data in it. Even in the case of a null entry, the field would have to be filled with either blanks or zeros so that when the input structure was written to the temporary file, there would be an exact correspondence between the fields in that file and what the data base expected to receive.

The final job to be done by the 'Add' function was to put the contents of the temporary sequential file into the actual data base. This was a relatively simple operation, but it also required some specific system calls, so we decided to specify a separate subfunction in order to insulate those operations and make future
modifications simpler. This subfunction we called 'Append To Data Base'.

The last stage in the detailed design of the 'Add' function was to specify the tools and utilities necessary for it to actually do its work. The subfunction, 'Get Student Info' worked with whole student records, but since it was necessary to deal with input data at the level of individual fields, we specified a utility which would do initial data input at that level. It had to handle the task of prompting for the data field by field, and using that input to fill the empty student record. It also had to do error and correctness checking as well as interpreting any special escape characters. We decided to build into this utility the capability to force the user to enter at least plausible data, i.e. nine digits in the social security number, no alpha characters in the phone number, the correct number of digits in the phone number, etc.

Despite our best efforts however, it seemed likely that erroneous data would get past the 'Initial Data Input' utility. One capability this utility did not have was to return to a field once an entry had been accepted. Therefore, we decided to specify another subfunction to allow data review. The user would be given the option
of using this utility to look over the total student record and change any fields before it was appended to the temporary file.

One of the big challenges we faced in designing the 'Review' utility involved the problem of how to show the record to the user without doing a lot of fancy screen formatting. The total number of data fields in the record amounted to over fifty, many more than would fit on a single video display screen. Also, we had the question of how to allow the user to designate which field, if any, he might want to change. The solution we devised was to divide the data fields into five logical components of eleven or so fields each, which we called screens. We tried to arrange the fields so that similar types of data would be included in the same screen. The 'Review Data' utility then went into a loop, presenting the screens in a circular fashion, one at a time, giving the user three options: to go to the next screen; to modify some of the fields on that screen; or to stop the review. If the user elected to modify a screen, he was presented each of the fields one at a time with the options of either leaving the data alone, of making a new entry, or terminating the list and returning to the screen. If all the fields of the selected screen were displayed, the entire screen was automatically
redisplayed and the process continued until the user explicitly terminated the utility.

An additional safeguard against bad data getting into the data base was provided by the 'Manage Temp File' subfunction. As it began processing, it gave the user one last chance to abort adding the just entered student to the data base. In fact, it was actually the temporary file to which that record was appended. However, the process of appending it to the data base happened automatically whenever the 'Add' function was exited, and so there was no abort capability at that point.

We decided that the 'Manage Temp File' subfunction also needed two utilities. Since it was critical that the newly entered data be written out to the temporary sequential file in the correct format, we specified a utility which was explicitly responsible for that. In the early phase of our design, when there was only one data set, and therefore only one temporary file, that kind of processing, although tedious, was relatively straightforward. Later, when it became necessary to divide our data base into two sets, it was necessary to use two separate temporary sequential files, one for each set, and so it was therefore also necessary to ensure that the format of both of them corresponded to their
appropriate data sets. The complexity of this utility increased significantly as its duties were expanded from not only formatting the input data for the temporary file, but also deciding which file received which field. The other utility specified for the 'Manage Temp File' subfunction was responsible for the task of actually writing the fields to the temporary files. We called this the 'Append To Temp File' utility.

With the specification of these subfunctions and utilities, the work we did to design the 'Add Student Records' function was essentially finished. In the course of the implementation, we also developed a number of smaller components, things we called tools, each of which did some very specific activity such as check the validity of the entry for the social security number or the phone number, or verify some other specific kind of data field. But these were not included in our working design.

A Later Design Modification

While our programming assistant was busily converting the 'Add' function design to code, we set out to design the 'Complete Student Records' function. We proceeded in much the same fashion as before. First we identified and specified the main subfunctions needed, and then
identified the utilities and tools. The only difference between what we did while designing this function and the 'Add' function, was that while we were in the course of working out the design for this function, we encountered a flaw in the 'Add' function which required us to make some changes in it.

The problem was encountered as a result of the fact that the type of data entry done in the 'Complete' function was essentially different from that done in 'Add'. Since data was being added to a record which supposedly already existed in the data base, it meant we needed some kind of search key to use to find that specific record. It also meant we had to take pains to be sure that the record we retrieved was the correct one.

What this lead us to realize was that we had not been careful enough in the 'Add' function to ensure that each record entered in the data base was unique. After some consideration, we decided to use the social security number to determine uniqueness. As a result, we modified the 'Add' function so that the first data the user was forced to enter was a social security number. Then the data base was searched and if a match was found, the user was informed to whom it belonged, and reprompted for a new social security number. An additional compli-
cation was introduced by the fact that we also had to search the temporary file, since that was a logical extension of our data base and there was a possibility that a number could be in that file but not yet entered in the data base. The fact that System 1022 could not do direct file reads added another complexity in that we had to use a temporary array for checking for social security numbers added during the current session. Finally, since an array is finite, we had to include an automatic mechanism to put the contents of the temporary file into the data base and then clear the array if it ever was filled in a single session.

It was necessary to make these modifications down to the level of the 'Review Data' utility. Since it was necessary to ensure that the user entered a unique social security number to initiate a student record, we realized it would not do to allow the social number to be changed in the course of regular data review. Therefore, we separated the social security from the rest of the data elements and designed a separate utility to handle its review. We also concluded that there was no reason at all to allow the 'Complete' function to even see the social security number, since by that time it would have had to be entered correctly. The final results of this improvement in our understanding and design
was that we realized we would have to provide some function which would explicitly allow the social security number to be changed, complete with all technicalities involved. We felt this was most appropriately within the province of our next function to design, 'Modify A Student Record'.

The Design Of The Supplemental Data Input Processing

In any case, with the matter of unique record identification taken care of, we proceeded with the design of the 'Complete' function. Like the 'Add' function, 'Complete' was intended to cycle through multiple records. See the structure chart in Figure 9.2. It looped, retrieving one record after another until told explicitly to stop. Unlike the 'Add' function however, it did not provide the user with a blank form to fill out. Instead, it retrieved a record already partially filled in and presented the only blank entries for further data entry. The subfunctions specified were 'Retrieve One Record', 'Complete Student Info', 'Append To Temp File', and 'Transact Into The Data Base'. The 'Append To Temp File' utility was the same as used by the 'Add' function. But the 'Transact' utility was logically different from its counterpart in the 'Add' function. This was because of the fact that new data had to be overlaid on top of data from an already existing record in the data
figure 9.2
structure chart for COMPLETE-STUDENT-RECORDS function
base, while in the case of the 'Add' function, it was only necessary to append the entire record to the end.

The complexities associated with the 'Retrieve' utility have already been discussed except for one additional point. Its original design had called for it to be able to retrieve on a variety of different keys, initially only name and social security number were going to be used but other keys were under consideration. After we realized the importance of ensuring that no record was duplicated in the data base and the role the social security number played in this, we decided to allow retrieval on the basis of social security number alone for the time being. Therefore, the 'Retrieve One Record' utility was reduced to a stub which only called another tool actually retrieved a record on the basis of the social security number. We left the original stub in place so that if other modes of retrieval were desired in the future, the structure to support that would already be in place.

The 'Complete Student Info' subfunction is worthy of a little additional discussion. It called on two utilities. The first called was 'Secondary Data Input' which emulated part of the working of the 'Initial Data Input' utility, including the format of the prompts.
It even called on the same error checking tools so that part of the interface also looked the same. As mentioned before, the main difference was that instead of presenting all the data fields one at a time, the student record was searched sequentially and only those fields which were empty were presented for data input. The other utility called was the, by then standard, 'Review Data' utility.

The question may come up as to why we did not just use the 'Review Data' utility by itself to accomplish the supplementary data entry phase of the RHDMS. After all, it did allow for the change of any data field item. And in fact, the answer is that when we designed the 'Modify' function, that is exactly what we did. But in the case of the 'Complete' function, our feeling was that the situation most likely to occur was an initial data entry session with a number of student records, followed by a session at some later time when the rest of the data was entered. The assumption was that usually only a specific set of fields would need further data entry, and the 'Complete' function should proceed to find those as efficiently as possible.

The Rest Of The Data Input Functions
By this time, we had developed a number of multipurpose utilities and tools which we could use in the design of the additional actions of the 'Modify' function. Much of the remainder of the work of constructing it was reduced to a matter of piecing together the correct components. See the structure chart in Figure 9.3.

It has already been mentioned how it used the general 'Review' utility to do most of its work. In addition, it also used the 'Review Social Security Number' utility (which was developed when it became necessary to treat that data element differently), the 'Manage Temp File' utility, and the 'Append To Data Base' utility. It also handled a change to the social security number somewhat differently from other data elements. In that case, after the new number was entered, instead of putting the number into the temporary sequential file, that specific field was changed for that specific record by over-writing directly into the data base file itself. This was again to ensure that only one version of that element ever existed in the data base. The only other aspect of the 'Modify' function which was different from the rest of the functions had to do with the intention behind its use. The purpose of this function was to allow the user to make specific changes to a single record. Therefore it did not process in a loop where
figure 9.3
structure chart for MODIFY-STUDENT-RECORDS function
multiple records could be chanced, but processed only one record per call.

As mentioned previously, a 'Withdraw/Cancel' function was added during the implementation. Its design was much like the 'Modify' function. It retrieved one record, using the utility created to do that task, and then displayed only those specific data elements relevant to the problem of withdrawal or withdrawal from dorm occupancy. The same general style and format as that used in the general 'Review' utility was employed for data entry, and after processing, the record was returned directly to the data base. The other two functions, 'Delete' and 'Undelete' simply involved retrieving the specified record, again using the social security number, and then using System 1022 commands which did what the name of the function describes. However, choosing to delete a record did not remove it from the data base. It only made it inactive. It was marked deleted, but could be returned to the data base later, with the undelete command, and in certain cases, could be included in reports and tallies of general data.

There were other problems which came up while we were working on this phase of the design stage. In a couple of instances, the pseudo-code had to be modified. One
significant obstacle was encountered when we discovered that PL1022 would not allow the nesting of subroutines more than five deep. Our response was to put the functions and some of the subfunctions on the same level as the module controller. This disrupted somewhat the shape of our code and the work we had done to keep our code modular. Another problem involved a bug in the system which required a great deal of work to discover and deal with. However, the discussion of these, and other occurrences is left to the chapters on implementation.

Conclusion

Our actual detailed design ended with the specification of all the functions contained in the 'Update Data Base' module. It had been our intention to continue with the rest of the modules while the data entry portion of the system was in use. However, the fact that the project was terminated as a result of conflicts with the University's Computer Center made further design fruitless. Suffice it to say that in our opinion, the design of this portion of the system represented 80% of the total.
CHAPTER TEN

EVALUATING THE EXPERIENCE OF WORKING
WITH A PROGRAMMING ASSISTANT

Introduction

We were fortunate to have had the experience of working with another person on our project, who was responsible for interpreting our design and coding much of the program. This added a great deal of breadth to our project.

The challenge of design is really two-fold. First, design is the process of arriving at and deciding upon the exact configuration of the solution to a problem. And second, design is the process of specifying or describing that solution in some kind of formalized way, with sufficient detail and completeness so as to facilitate putting it into working code. If the one who specifies a design is also the one who implements it, then all that is really required of the design representation is that it contain enough information to prompt the implementer, alias the designer, to remember the solution that he, the designer alias the implementer, had decided upon in the first place. We welcomed this chance to have our project mimic, as much as possible, the real world in this separation of design and implementation.
In a case like our work on the RHDMS, where one of the objectives and part of the challenge was to apply what we had learned about good, complete design, our goal was to carry out both the problem solving and design representation phases of the design process as though implementation might not occur under our control. Never-the-less, even given this pristine objective, the possibility that the design representation would not turn out to be as thorough or as complete as desired was great unless the design and implementation environments could actually be separated in some way. Having an independent programming assistant working with us, provided us with just such an opportunity. We were given the chance of actually participating in, and putting to good use, that feedback loop which is required between design and implementation. If a portion of our design representation was unclear, we soon found out and were forced to improve it. If we had called for something, which was physically impractical or infeasible, we were given the opportunity of making modifications in our design and of trying again. All of this occurred without us ever having the chance to hack at the code or become engrossed with implementation idiosyncrasies.

The Effect on our Design

Initially we assigned our programming assistant a few
implementation specific research questions, to which we wanted answers. These included problems such as: System 1022's ability to error-check data during entry; menu presentation; the power of 1022's programming language, PL1022; and the question of whether we would be forced to implement routines in COBOL. While our programmer was researching these and running some actual tests with 1022, we set out to begin our design process.

The need to communicate well was evident from both directions. We were fortunate to be working with a very skilled programmer, who was intelligent and resourceful. This meant that he was often able to provide us with real design information, which made our job easier. And that he was able to perceive some problems and work out solutions on his own, such as his creation of data types for the elements of the student record beyond what 1022 allowed. But it also meant that he made some decisions on his own during implementation, which were not ideal. An example of this was the confusing use of the arrays, in which the data elements were shuffled during data entry, error-checking, and preparation for writing to a temporary file.

One of the most valuable benefits of working with a programming assistant was that it allowed the implementation of a working prototype in 1022 quite early on. From this
prototype we were able to test various aspects of the system for overall feel, feasibility, and practicality. An excellent example of this was the user interface. By modifying the prototype in different ways we were able to clarify exactly what type of interface we wanted this system to have. Another example was the updating of the data base, which we were able to test extensively with the prototype. We learned a great deal about some major bugs in one version of System 1022, which would have been disastrous to our designed system if they had gone undetected.

Conclusion

The coordination of our programmer's implementation and our design introduced an added level of complexity to this project, which would not have been there had we not been facing course requirements and deadlines. We were forced to work out the final inter-module relationships, while some of them were being implemented. But this situation did encourage and allow us to produce a prototype of our system earlier than we would have been able to on our own. All in all the fact that design and implementation were separated at this stage was a most useful exercise in a real world aspect of software engineering, and it added to the value of our project despite, or perhaps because of, the complications which it introduced.
CHAPTER ELEVEN

PHASE I IMPLEMENTATION: THE PROTOTYPE

Introduction

Our reasons for attempting to implement a prototype of the RHDMS were many. First of all we welcomed the feedback which would result from having a working sub-set of their system on the computer. This was especially important to us because of our relative inexperience, especially with the System 1022 data base software. Second, we needed to provide our programming assistant some substantive work in order for him to fulfill his course obligations. And finally, we knew that we could not fully implement the entire RHDMS in time to meet their deadline for the beginning of data entry within the next two months. Therefore our decision to focus on a prototype which included upper level system control, the module which allowed initial data entry, and the module which allowed entered data to be reviewed and changed seemed to be an intelligent and pragmatic one.

Some of the following discussion will closely resemble the preceding design explanation. It will however be oriented towards an implementation perspective. The overlap is a natural result of the interrelationship between these two
phases of our project.

**Research of Implementation Questions**

Of primary importance to us were some questions about specific attributes of System 1022 and its programming language, PL1022. We wanted to know if the reality of having to use 1022 was going to cause us to have to alter our design which was not system dependent at this stage. Some of these questions were general in nature, and others were specific to our project as designed so far.

We were concerned about System 1022's ability to do string evaluations and manipulations. We found that PL1022 had capabilities somewhat similar to the BASIC language. There were many string manipulation functions provided, but there was no character-by-character input analysis available. We also learned that PL1022's control commands included if-then-elseif-else, while loops, repeat loops, and sub-routine calls with automatic return. Knowing this we felt confident that we would be able to implement the flow of the algorithms which we were designing without too many strange contortions. We also learned that PL1022 had no ability to call a procedure and pass parameters. This meant that we would have to be extremely careful with the use of global variables, when it was necessary to simulate this function.
As mentioned earlier it was very difficult to implement our command string interpretation idea in the exact manner we wanted. A number of somewhat obscure programming tricks would have been necessary in order to allow the user to transparently jump from one lower level menu, back to the main menu control module, and on to the newly selected function or menu. We felt that this was much too cumbersome and complex to warrant adding command string interpretation to our final system design, and so it was abandoned.

We also researched other questions related to PLl022's ability to read values from text files, the maximum level of nested sub-routine calls allowed, data checking in 1022, operating system calls from within 1022, and minimal screen control. We learned that we would not be able to read directly from text files, that calls could only be nested five levels deep (this was increased to twenty in a later release of 1022), that data type checking was limited, that only a few operating system calls were implemented from within 1022, and that complex screen control would be very cumbersome. With this information we were better able to visualize the form the RHDMS under design was beginning to take.

**Early Prototype Implementation**
We provided our programmer with structure charts and pseudo-code in the Yourdon and Constantine tradition for each module or function as we designed it. We began with the main control module, which initialized the variables, acted as the transaction center by displaying the main menu and calling the chosen modules, and finally did any required clean up before exiting. Our programmer was easily able to make some slight modifications to his earlier menu-driven test model in order to incorporate the design changes, which we had now decided upon.

Then we set out to have our programmer implement the most important portions of the 'Update' module. From our earlier analysis we had determined that adding a new student to the data base was a crucially important function of the RHDM S. Data entry would consume a great deal of their time and it was absolutely necessary that it be done correctly. We required the program to properly accept as much new student information as the user wanted to enter; to do more extensive error-checking than 1022; to present the entered data back to the user for review if desired; to write the student information to a temporary sequential file in the user's area; and finally to transact the temporary sequential file into the existing 1022 data base. Then if all of the student's information had not been entered
initially, the user had to have the ability to complete a student record which was already stored in the data base.

Since the 'Update' module was itself also a transaction center, it was not too difficult to initially take the sub-module, 'Add', and work extensively on it before the rest of the other 'Update' sub-modules were even designed. We realized that there were sure to be lower level functions implemented in 'Add' which would eventually be used by other 'Update' sub-modules. We did not know how much some of these functions would have to be redesigned in order to work in this fashion, but we hoped that by following good programming practices our programmer would leave us with code which we could modify to fit our final design. This was the price we had to pay for starting implementation before the entire system design was complete.

Final Prototype Implementation

Our programming assistant was able to get a working version of the relevant menu control modules and the 'Add' and 'Complete' modules up and running. This system presented the menus properly and allowed the user to enter as much data about the student as they wished. There was a prompt for each field in the record and the user went down the list entering information or skipping the field until they
finished or escaped by entering a special character. We called it the escape character and used the `/` for this implementation. Each entry was accepted into an array location and then it was somewhat preprocessed and checked for correctness of form and type. Our programmer had implemented the structure for complete and consistent error checking, but not every field received it at this time.

After the student's information was entered, the user had the option of reviewing the data eleven elements at a time. They were able to move to the next eleven, decide to change some of the currently displayed eleven, or quit the review process for this student. If they chose to make a change to any of the current eleven elements, for each in turn they were shown the existing value and prompted for a replacement. They could leave it as it was, enter new data which would be error-checked, or quit the element by element changing for this screen. Our intent was to allow easy review and change of mistakenly entered data. We avoided any attempts at screen addressing and slick cursor control because of the limits of PLI1022. However, we felt that having the ability to view eleven elements at a time and only having to deal with linearly walking through eleven at most in order to correct a mistake, was a good compromise.

After data entry and possibly review, the user was asked if
they really wanted to add this student to the data base, in which case the information was written to a temporary sequential file. All of this was contained in a loop, which allowed the user to enter as many students as they liked until they stated that they were done with the 'Add' process. At this time all of the information in the sequential file on each student was put into the existing data base using 1022's transaction processor.

If all of a student's information was not initially entered, the user had the option of calling the 'Complete' module. Then they were asked to supply the social security number for the student, whose record was to be completed. After the number and name were verified as correct, the user was prompted for each empty field in the student's record. When data entry was finished, they were given the option of reviewing all of the data for this student. Then they were asked whether they wanted to return this new information to the data base, and finally whether they wanted to complete another student.

**Prototype Problems**

We ran into some problems during this early implementation stage of our project. There were certain things which we wanted in the program that were not done by our programming
assistant until we pointed them out. They had not been explicitly specified in our design; we just presumed that they would be taken care of. One of these omissions was the echoing back to the user of any bad data, which they had just entered. Another was the need to absolutely prevent weird input from crashing the system. And a third was consistency in what was expected from the user in terms of responses at similar stages in different modules of the program.

In order to get the type of user interface which we wanted, we had to spend some time educating our programmer in the importance of bending over backward to protect the user from themselves and inconsistencies within the program. We realized we had to further specify in our design representation what we expected in terms of a good user interface. We had failed to adequately do this at the start but we were able to pick up on it and make the correction as the coding progressed. Our programmer was very accommodating once he saw the logic behind the need for a really good interface with this type of program.

One important design flaw was discovered during the implementation and testing of this prototype. We had gone to great lengths to design searches into our 'Add' module, which would prevent the user from entering a social security
number for a new student which was already used in the data base. This was essential because of the fact that our two data sets were joined on the social security number attributes. If duplication were allowed, the data base would be corrupted. Our examination of the existing student records worked perfectly well, but we had forgotten about those students who had just been entered in the current session. Their information was patiently residing in the temporary sequential files, waiting to be added to the data base after the 'Add' session was complete. We had not designed a search of these students' social security numbers for duplication. In order to properly protect the data base from the disastrous effects of identical social security numbers, we had to redesign 'Add' to include more thorough duplicate checking. Because PL1022 did not allow reading from text files, such as our temporary sequential files, we could not design a direct search of the previously entered student information. Instead we had to maintain an array containing the names and numbers of all of the students entered in the current session. This array was then always searched as well, and the user was not allowed to duplicate the social security number of any entered student. The importance of and the difficulties associated with maintaining unique social security numbers was beginning to arise as a major concern. It continued to be such during the rest of the design and implementation stages.
This version of 1022 also did not allow more than one use of the transaction processor per scanning of the PL1022 code. This was a major bug in this version of System 1022, and a serious problem for us. We had designed our system to allow the user to move in and out of the 'Add' module as many times as they liked. This was to let the user add a few new students, do something else such as complete an existing student's record, and then return to adding once again. At the end of each session in 'Add', the new student information in the temporary sequential file was to be entered into the data base. With the discovered bug this was not possible. Every second time through, the program crashed with an incomprehensible error message about some problem with the data dictionary! It took us quite a while to actually pin-point the bug as being in System 1022 itself. But after a phone call to Software House [3], the producers of 1022, we learned that we had indeed run into a bug which they knew about. They informed us that there was now a newer version of System 1022 and that in fact the University had been sent the update. They also told us that the newer version would allow nested sub-routine calls down to twenty levels. After some checking we learned that the updated version of 1022 was actually already on the University's computer in a new area. We began to use it and, much to our relief, found that our designed plan to
multiply update the data base now worked perfectly.

**Conclusions**

Our programming assistant was able to successfully present a modified version of the prototype to his course instructor. It was a version which only allowed one data base update per use of the program in order to get around the problem of the buggy transaction processor in the earlier version of 1022. It was pleasing to see how easy it had been to restructure the design of the code to alter the timing of the transaction of the temporary file into the data base. This gave us confidence that our designed program was indeed modular and structured.

We felt that working with our programmer to implement this prototype was a very valuable part of our project. It provided him and us the chance to work in an environment which simulated many aspects of the real world. We learned a great deal about how to communicate design ideas through good representations and follow up examination of the produced code. It is rare in school to have the opportunity of separating the designers from the implementers. This was a unique experience for us. Not only did it add to the breadth of our project, but it also produced a good working prototype of the most crucial portions of the RHDMS.
CHAPTER TWELVE

PHASE II IMPLEMENTATION: THE 'UPDATE' MODEL

Introduction

This phase of the implementation was done without the assistance of an independent implemener. For this reason the overlap between the design and actual implementation stages was greater than it had been previously. In some respects it was good to have more direct control over the coding of our design. One level of communication with all of its overhead had been removed. But on the negative side our focus was now quite scattered between design of new modules, redesign and correction of existing modules, implementation preparation, coding, testing, debugging, and evaluating the performance of the model which we were producing.

It was our goal for this phase to take our prototype and to turn it into a working model of the data entry portion of the RHDMS. It would be an implementation of almost all of the 'Update' module. This required two different types of work. One was the redesign and re-implementation of those modules in the prototype which we were not satisfied with. And the other was the implementation of the rest of the 'Update' sub-modules which would allow the user to modify
the data in an existing student record or to enter withdrawal or cancellation information about a student.

While we worked on these two general goals, we also finalized our expectations for the user interface. We made many minor changes, which standardized the prompts and the default behavior of the system. At the same time we refined the structure of the data. We divided it into four logically different sections in terms of how we wanted to treat each field.

**Modifications to Existing Modules**

Our first priority at this stage was to test the existing program and to debug it a section at a time, until we were confident that what we had worked and worked well. This was made easier because we had our own 'Review' function for examining the dynamic form of the data, the temporary sequential files for the intermediate form of the data, and the 1022 data base itself, along with its own query language, for the final form of the data. We were able to check and double-check our treatment of the data to see if the proper transformations were occurring.

The types of problems we discovered at this time were typically oversights. They included the further need to
process the social security number uniquely, the inconsistent treatment of bad data entry, non-standardized control flow in functions which should have behaved identically, some poor error handling, and non-uniform use of the escape character to terminate a function. Some of these problems effected the quality of our treatment of the data, and others were more relevant to the way in which the program was presented to the user.

As mentioned in the discussion of the detailed design, at a point during implementation we realized that our 'Review' function could not deal with the social security number as it did with the rest of data. When it was only called from 'Add', there was no problem because the data was only dynamically held. It had not yet been entered into the files or the data base, so any change in the social security number during the 'Review' process was fine. With the addition of the 'Complete' module, this was no longer the case. If 'Review' allowed the user to change the social security number, the new information would be appended to the data base but the old record would still exist. Garbage would be left in the data base. Since this was not our designed intent, we broke 'Review' into two functions. One accessed only the social security number. This function allowed the number to be changed and it was only called from the 'Add' module. The other accessed any of the rest of the
data fields, allowing any of them to be changed during their review. It was called by both 'Add' and 'Complete'.

This is an excellent example of the problems caused by beginning to implement a portion of our design before the rest of the design was completed. Because we initially designed 'Review' only for the needs of 'Add', we overlooked the effect of using it in 'Complete'.

Error checking was handled by specific functions, which were called according to the type of the data element. Our implementer had created different data types for each kind of data element and there was a separate error-checking function for each. The prototype did some checking but it was in no way complete or consistent. Because this was important to the integrity of the RH data base, we spent a great deal of time improving these checking functions so that they would catch as many faulty data entries as possible. It was our desire to channel the data input down the right course without over-burdening the system with terribly complex error-checking functions. We knew that we could never prevent all errors, but we did want to do all that was possible to catch as many of them as feasible. This combined with the RHDMS's easy data correction processes would lead to as valid a collection of data as possible, we hoped.
Another area which we discussed during detailed design and which we worked on at this time and for the rest of implementation was consistency in the flow of the program, especially with respect to the default values offered at decision points. In order to minimize the number of key strokes needed to drive this program, we designed default values into almost all decisions. For example, a question might be:

Do you wish to add another student: ([Y] or N)?

The default value 'Y' could be selected by entering a carriage return only. Obviously we tried to determine which decision paths were the most logical at each point. For example, the 'Add' and 'Complete' modules were designed to behave in an identical fashion. But we found that they did not parallel each other in many instances. We tried to locate these inconsistencies and to rectify them. This was very important to us because we wanted to develop as consistent an interface for the user throughout the program as possible. We felt that this would minimize confusion and subsequent errors.

The same concerns we had about the flow of the program from the user's perspective, we had about error messages. We wanted them to be clear, sensible, and consistent. In the prototype they were often too terse or sometimes even given
twice for a single error; once by the called routine and again by the calling routine. We spent the time to make the use of error messages appear logical to the user. Where appropriate we spelled out the correct format for data entry and re-prompted them for it. And we tried never to startle the user with our treatment of the flow of the program upon discovering an error condition.

Originally the escape character was designed to terminate data entry when the user did not want to continue with all of the prompts. During the implementation of this model we expanded another use of the escape character, which had only been partially explored in the prototype. Now we also designed the system to allow the user to 'escape' from almost any prompt. Our intention was to allow them to use it to terminate whatever process they were in the middle of. When appropriate the program would return them to where they began or, in other situations, it would take them to the next step. Again we tried to make the use of the escape character as sensible, consistent, and fool-proof as possible.

Refining the Structure of the Data

Originally all fifty-two fields were treated by the program almost identically except for the social security number,
which we required to be entered and to be unique. Now we decided to partition the data elements into four logical sections for the purpose of program presentation, data entry, and error/validity checking. Each section of data would be handled somewhat differently from the rest depending upon its content. The first section was the social security number. The second was the student information section, including names, addresses, academic status, rooming preferences, etc.. The third was the official data section with the priority number, the validation number, and the pre-payment information. And the fourth was the withdraw/cancel data section, which contained all of the information pertaining to these two functions.

The data was still held in the same array and written to the intermediate files in the same fashion. This change was more a logical one, which allowed us to refine how we treated the different sections of data. The social security number was, even more than before, tested and evaluated for uniqueness. In the 'Add' module no duplication was allowed. In the 'Complete' module there had to be a match with an existing student record. When the user chose to review the student record in 'Add', they were allowed to change the social security number. If they did, we of course had to check the new value for correctness and uniqueness again. When they chose to review in 'Complete', we did not allow
them access to the social security number. This was not the appropriate place for them to try to change a wrong number.

We also used the social security number in 'Complete', and later in 'Modify', to trigger an automatic updating of the data base. For each record the user wanted to 'Complete' or 'Modify', we checked our array, which contained the social security numbers and names of students already updated in this session, to see if there was a match. If there was, then a newer version of that student's information existed in the temporary sequential file. In this case we automatically, forced the transaction of the file information into the data base before we allowed the user to proceed to update that record. This insured that the user was always presented with the most current version of the student record when they went to add more or modify existing information.

The logical separation of the official and withdraw/cancel data from the main block of personal student information meant that we designed changes for some of our existing modules. The entering of data would normally occur in two distinct blocks. First the user was allowed to enter as much personal information as they wished. If they escaped from this or finished, they now went directly to entering official data. If they escaped or finished there, they were
done with normal data entry. This designed change was implemented in the 'Add', 'Complete', and 'Modify' modules. It was also carried to 'Review'. The user was no longer allowed access to the withdrawal or cancellation section of data from these modules. This seemed to make the most sense for the normal manner in which the RH did their business. It did mean that we had to design a new module to handle the entry and review of the withdraw/cancel data. This separation was logical though because withdrawal and cancellation were really exceptions, and they should not have been tagging along with the normal data during the course of its entry.

**Implementing the New Modules**

There were two major modules which were added to the system during this phase. They were both transactions under the control of the 'Update' module. One was 'Modify', which allowed the user to modify or change any data in either personal student information or the official data sections. The other was 'Withdraw/cancel', which was where withdrawal or cancellation information about a student was entered, modified, or merely viewed.

'Modify' was the third major module which the RHDMS needed in order to be complete in terms of data entry. With it
added to the system, the RH could now enter as much initial data as they wanted or had time for; they could return, when they chose, and finish entering the rest of the necessary information; and they could correct any mistakes or change any fields that they needed to. These three processes were the core of the data entry portion of the system. Together they were a working model which satisfied the RH's need to begin data entry.

For this working model we first designed the implementation of the 'Modify' module to utilize the two review functions, which already existed: one for changing the social security number; and the other for the rest of the personal student information and the official data. We thought that this use of existing functions would have the additional benefit of being very familiar to the user. We soon realized that the social security number review function would not work because there was no way to change the social security number of a record with a 1022 transact command, which used this number as the locater value. Therefore, we were forced to redesign how this change could be accomplished. We decided to do it with the 1022 'change' command, which would allow any attribute's value to be modified.

Never-the-less we still ran into difficulties with the social security number when we tried to implement 'Modify'.
This was done to the ever-important need to maintain the uniqueness of the social security numbers in this system. If the user wanted to change a social security number, they had to first provide a valid number, which existed in the database. Then after confirming the correctness of the selected record, they were allowed to enter a new number. If it was also valid and not a duplicate of an existing number, we called 1022's 'change' command in order to alter only the social security number field. The problem was that, when we examined the resulting database, we found errors in the two data sets' numbers and how they were joined. The changed number appeared in the data set containing the unique RH information, but not in the one with the University-wide information. It took us a while to realize that we had initially created a selection set of one student; then we had thrown away that selection set for one of the data sets, when we looked for duplicate numbers in the database; and finally when we attempted to 'change' the number of the student in the selection set, we no longer had both of our data sets pointing at the both halves of the same student's information.

What had happened was that one of the selection sets had been altered by the attempt to find a duplicate social security number in the database. The complexity of implementing this process properly was greatly increased.
because we had chosen to split our data base into two data sets with the hope that this would facilitate integration with the rest of the University's data processing system sometime in the future. This split made the debugging of this logic error much more difficult to accomplish.

The implementation of the 'Withdraw/cancel' module was not difficult. We used the standard user interface, which we had polished in the previous modules. We also borrowed from 'Review' the method of presenting the information and changing it when necessary. This designed duplication had the added benefit of making this module behave like the rest of the system. We liked this consistency, which was getting stronger all along throughout the program.

**Initial User Testing**

We were now ready to have the administrative clerk, the one person in the RH who would be doing most of the data entry, test out our current version of 'Update' model of the RHDMS. We arranged a time when she could bring along some actual housing applications and attempt to enter the information from them into the data base. We hoped to accomplish a number of things with this test. We wanted to observe how well the system worked in use; we wanted to see how easy it was for a novice to interact with the program; and we
wanted to receive advice on what to change in order to make it better fit the user's needs.

Once the administrative clerk overcame her initial reluctance and her desire to have us tell her what to do next, she was easily able to go through the menus and begin data entry. As she entered a number of applications, she informally presented us with a list of modifications to consider. At the same time we began to instruct her about the intent of the designed system in terms of the user interface, the data entry error-trapping, and the protection of the data base, especially with regards to the social security numbers. She seemed to be able to use the system quite easily. There were no rude surprises in the way that the system behaved. She did, however, suggest some changes.

The administrative clerk felt that we should increase the allowable length for the name fields and that we should include in our error message what this length was for the particular field in question. She was surprised to see country as prompt in the address portion because the RH did not ask for it explicitly on their application. We had included it to better duplicate the form of the University's data for each student. There were a few other minor questions and suggestions that she raised such as the removal of one of the categories from the application code.
type, the need for a list of the University's four letter department code, and her desire to be able to include blanks in the quarter applied for field.

Along with these the administrative clerk also made two more substantive requests and one major statement. She requested that some way be added to allow her to duplicate the parent's address from the previously entered permanent address for the student. Apparently it was often the case that they would be identical and it seemed a waste to have to re-type the same information. She also asked us to change the manner in which the physical limitations data was handled. We had designed this field to be a single digit code number representing any limitations, which the student might have. It was our understanding that at present this information was not often used, and so we thought that a code which took up minimal room was the best solution. The administrative clerk requested that we treat this question like we did the question of citizenship; a yes-no field followed by a prompt for more specific information only in the exceptional cases. She thought that this would make more sense to the person doing the data entry and we agreed.

The statement the administrative clerk made that totally surprised us at this late date concerned the application priority numbers. After much debate during the design
stage, we had decided to have this number be totally controlled outside of the automated system. There were just too many complications in trying to have the program assign these priority numbers and in allowing some easy form of operator control over their final value. This would be necessary because there was no assurance that the applications would be entered into the automated system in the same order in which they were received at the RH. And this first-come first-served order was how they wanted to assign the priority numbers.

Knowing all of this, we had designed the priority number field to be an integer attribute, which would contain sequentially larger numbers entered by the user. During this late implementation phase the administrative clerk informed us that the RH had already received applications and assigned priority numbers for the Fall following the coming Fall. This was a complete surprise to us. We had missed this possibility totally during analysis, when it should have been caught. It meant that the RH currently had two applications with a priority number of one, each for a different year. We had planned to aid the process of room assigning by producing lists sorted by priority number. Because of this new information gathered by chance at this late date, we had to seriously rethink the role and form of the priority numbers in this new automated system.
**Final Changes to the 'Update' Model**

All in all we were very satisfied with the initial testing of the 'Update' model. The great pains we had taken to create a sensible and consistent user interface were rewarded when we saw that the administrative clerk did not seem confused by it at all. In fact she hardly even noticed it - the highest form of complement!

We took the suggestions the administrative clerk had given to us and decided which we thought were sensible and feasible to try to incorporate into this model. We easily lengthened the fields because of the way in which our original programmer had stored this information. We decide to leave the country portion of the addresses in the data base because we thought that the RH would use these fields in the future. The minor modifications mentioned for some of the data entry checking functions were also implemented.

In order to allow for the transfer of identical information from the student's permanent address to their parent's address, we created an exceptions test. If the first field of the parent's address received a quotation mark as its data, then all of the address and phone information would be copied from the appropriate permanent address fields. The
user was reminded of this feature by a note in the prompt for the first field in the parent's address.

We also restructured the treatment of physical limitations in the way that the administrative clerk suggested. The user was asked if the student had a physical limitation. Only if the answer was affirmative did they get the prompt requesting the specific limitation; otherwise this field was filled automatically with the code number for no limitation. This later action was necessary to prevent the code field from being presented to the user during a 'Complete' session.

The question of what to do with the priority numbers from different years was a difficult one. There were a number of possible ways to avoid confusing the numbers from different years. We chose as a tentative solution to expand the format of the priority number. It would now include as part of itself the information on the year and quarter to which it applied. Our proposed new numbers would begin with the first two digits representing the year, the third digit the quarter, the fourth a '0' to act as a separator, and the rest to signify the priority order. It would have to be exactly eight digits long in total so that it could still be used properly in numerical sorts to produce listings, which would be automatically grouped correctly by year and
quarter. As an example the nine hundred ninety ninth application for the fall of 1984 would be: 84300999. We did not feel that this solution was ideal but that it would work temporarily while the RH and we considered other better solutions.

**Final Considerations**

We realized at this time that we had quite a few exceptional circumstances during data entry, which would alter the normal progression of prompts for the next field. We decided to consolidate these exceptions into one module, where all of this aberrant behavior would be isolated. We did not want to leave this type of very order-dependent code scattered around the error-checking functions. Therefore we designed and implemented at this late date a module for all of the unique data entry manipulations such as: not prompting for 'other citizenship', if the student is a U.S. citizen; not prompting for physical limitation code, if they have none; and duplicating the permanent address into the parent address fields.

We also tried to design in a method to minimize the pain the user would feel when the University's DEC 2060 computer would crash, as it invariable did especially during the thunderstorm season. Our thoughts included ways at the
beginning of a 1022 session to automatically transact in any residual temporary sequential files, which would not have been added to the data base, if the DEC had gone down during the last session. However we learned from the Computer Center that, if the DEC system ever crashes during a 1022 session, the data base must be presumed corrupted. This meant that the RH would have to make daily (or more frequent during heavy data entry) tape backups of their data base. Then when a crash occurred, they would have to restore their latest version of their data base and re-enter all of the intervening students.

There did not seem to be a better way to handle DEC system crashes, but we were not too concerned because in most normal years data entry was spread out over the spring and summer. This meant that on the average only twenty or thirty new students' applications would be processed each day. If a crash occurred, the burden of restoring the previous backup and re-entering the lost data would not be too great.

**Conclusion**

At this stage we felt that we had a good working model of the RHDMS which would allow the RH to begin to enter the student application information for the current year. All
that remained to make it operational was to get their disk space, which they had been promised, and to instruct them in the use of this 'Update' model. We knew there were still likely to be changes and bugs in it, but we felt confident that it reflected a good implementation of a good design. We were confident that the RH could put this model to use for data entry, while we continued to finish the design and implementation of the rest of the RHDMS.
CHAPTER THIRTEEN

PHASE III IMPLEMENTATION: THE FUTURE

Introduction

The final phase of implementing the rest of our design in order to complete the RHDMS was never begun. We planned to finish the lower level design and implementation of the rest of the system, while the RH used the 'Update' model to enter application data into their 1022 data base. In this way we hoped to gradually build a full working model of their system within three months.

What follows in the rest of this chapter is a description of what we had planned to accomplish during the last portion of our work on this project. It includes what implementation would have happened next; what would have followed that to complete the system; and what we would have provided the RH in terms of user education and support.

The Next Implementation Step

The next item to be implemented was to have been the 'beds' data set. It would have been the third and final set in the RH data base. We designed it to consists of a record for each bed in the dormitories. Each record would have
contained a field for the dorm name, the floor, the room number, the bed number, the room phone number, the room key number, the dorm key number, and a field for the social security number of the occupant of that room. There was to be the option of adding more room information in the future, if it was ever needed to keep track of appliances or other items. All of the items in this data set could have been permanently kept in the data base except for the actual occupant social security number. The task of building this data set would have been tedious, but it would not have been repeated yearly. At the same time it would have been possible to make modifications to the set, if room configurations ever changed.

We planned to have this third data set joined with the other two by the social security number. This would have provided all of the necessary information about a student and their dorm assignment, when their record was selected from the data base. It was to have been almost transparent to the user that the student's data was spread throughout three distinct data sets. In this way we designed the system to only require the entry of the student's social security number into the proper bed record in order to make a room assignment. This would have met the requirements of the RH in terms of their need to extract dorm status information from the RHDMS, while room assigning was still going on.
They expressed a need to be able to know on the basis of room, floor, or dorm how many beds were assigned, how many were still empty, and the number of males and females in each. With the existence of the beds data set, in fact describing the housing configuration of the entire dormitory complex, this information would have been available either from standard, prepared reports or from querying the data base interactively.

We hoped that this information would have greatly helped the person responsible for making the bed assignments. With the RHDMS they would have had the ability to use this general statistical information, but they also would have had the ability to ask much more specific questions of the data base, once they had learned to use 1022's query language. Then they could have found out which female night-owls, who did not object to a smoking roommate and who majored in math, were available as a possible roommate for a smoker from New Jersey.

To go along with this gathering of assignment information, we had designed the 'Assign Rooms' module. Its function was to facilitate the actual selection of a bed for the student and to allow the user to enter the assigned student's social security number into the proper record in the beds data set. This module was distinct from the normal data entry modules,
because of the way in which the assignment process was done during a normal year. First, all of the student information was received, processed, and organized. Then when most of the applications were in and the time was right to begin, the RH started to make tentative room assignments. These were in flux until just before the students arrived. In any case the assigning of rooms did not occur during the initial data processing phase and we designed this system to follow this method.

The creation of the beds data set and the implementation of the `Assign Rooms` module would have completed the next step in the creation of the RHDMS. With them in place the RH would have had most of what they required to manage their data.

**Further Implementation Work**

The other modules, which still would have needed to be implemented to complete the RHDMS, were not as crucial to the early work which the system had to do. That is why they were postponed until the very end. They include the production of standard reports, which the director of the RH had specified in our earliest meetings; the deletion, and possible undeletion, of a student from the data base; some tools, such as the year-end purging of the previous year's
students; and perhaps the ability to query the data base from within the PL1022 program. If all of these could have been implemented, then we would have considered this stage of the project complete.

**User Support**

We intended to be readily available to the RH during their use of the 'Update' model and any more extensive versions of the RHDMS, which we would have provided them during the final implementation phase. We would have supported them by giving them preliminary instruction in the use of the RHDMS, System 1022, and the DEC-20 operating system. We would have been available to answer their questions, to fix any bugs, or to make any modifications in the system which they found they needed. In fact at first we would have served as their user's manual, until we were able to supply them with one.

This support from us would have lasted until they were very familiar with the entire operation of the RHDMS. Then they would have felt sufficiently comfortable with it themselves to use it properly. And they would know it well enough to sign off that the RHDMS was what they had wanted and that we had fulfilled the system specifications with our implementation.
CHAPTER FOURTEEN

CONCLUSIONS

Introduction

We would like to summarize what we were able to accomplish and what we learned with this project for the RH. It, not unlike some other labors of love, consumed almost an entire year of our time, during which it was the primary focus of our attention. There were many aspects of this project which were very good. It provided us an excellent chance to use many of the methodologies which we had been exposed to in the more rarified, academic atmosphere of the classroom, in a real application for a real user. At the same time there were other aspects of this project which were very disappointing. To have worked so hard and long, only to be thwarted at the end by the machinations of the Computer Center, was exasperating to say the least. We had hoped to provide a real product for a real user, but we were unable to implement our automated RHDMS. There was a definite feeling of failure because in the end we had not been able to produce a usable product for the RH and the people there, with whom we had worked on this project.
Accomplishments

The primary goal, which we accomplished with this project, was to have had the chance to work on a major software engineering problem through almost all of its life-cycle. Except for the final implementation stage and the user training and final acceptance, we were able to experience almost all of the different aspects of a substantial software engineering project.

We began with a rough, and overly cursory, feasibility study and requirements analysis. Then we employed DeMarco's [1] structured analysis methodology to produce the system specification, including data flow diagrams and a PSL/PSA [2] representation of our analysis. Following this we worked on the incremental design and implementation of a usable model of the major functions of the RHDMS. We employed Yourdon and Constantine's [4] design methodology with minor modifications to accommodate the added complexity of beginning to implement before the final system design was complete. This was the result of being able to have an independent programming assistant working with us on the implementation of a prototype for our project. Having this programmer made our project more closely resemble an actual software development environment. It was a successful addition to the graduate project, and it was worth a great
deal to us. And we produced a good, usable model, which allowed data to be entered, error-checked, and changed, while it did all that it could to maintain the integrity of the database despite what any user might attempt to do.

**Problems with the Computer Center**

From the very first meeting with the director of the RH, we were cognizant of the need to consider the University Computer Center's role with regards to this project. We failed to adequately deal with this aspect of our project. We mention it in order to emphasize how important it is to try to identify at the earliest date all of the stumbling blocks, which can kill a project in any environment. We failed to perceive that all was not well with the Computer Center and consequently we failed to successfully implement a working system for our users. The Computer Center, it turned out, was what DeMarco refers to as the 'system owner'. In fact it was another interface to the RH system, which had failed to surface in our data flow diagrams. We thought that we were meeting their requirements, and we were completely surprised by their refusal to allow the RH to implement our project.

It is hard to see where things went wrong in our relationship with the Computer Center. Either we completely
missed their signals that we were on the wrong track from their point of view, or they did not even send them until their last minute 'no'. We were aware that they felt responsible for managing the University's data, but we did not think that they would prevent the RH from implementing a stand alone system, designed to allow future integration with other University data processing programs and files.

**Conclusion**

The opportunity to grapple with such a typically real-world, end-user oriented problem was exactly what we were looking for. The chance to apply the tools of our trade to this problem gave us invaluable experience in their application and validity. We were able to apply some of the methodologies, including especially DeMarco's and Yourdon and Constantine's, reflecting the most current thinking concerning the development of software from the software engineering perspective. This was a most worthwhile and educational project for us.
REFERENCES


2. ISDOS Project, Department of Industrial and Operational Engineering, University of Michigan. *Problem Statement Language (PSL) / Problem Statement Analyzer (PSA)*. Ann Arbor, MI: University of Michigan, 1981.


Early physical context diagram
EARLY WORKING VERSION
CURRINT PHYSICAL DIAGRAM
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
APPENDIX B

INITIAL CURRENT PHYSICAL DATA FLOW DIAGRAMS
Context Diagram:

Initial Version
Current Physical Diagram
LEVEL 0:

1. SCREEN-REQUESTS + APPLICATIONS
   - COMPLETED APPLICATIONS
   - REQUESTS
   - APPLICATION PACKET
   - REJECTS
   - DEPOSIT ACKNOWLEDGED REMITTANCE

2. MAKE-ASSIGNMENTS
   - PRELIMINARY APPLICATION PACKET
   - VALUATED APPLICATION PACKET
   - FACT
   - PERMANENT-ALPHA-FILE

3. MODIFY-ASSIGNMENTS
   - FLOOR PLAN + TEMAPRIORITY NUMBER + TEMP ALPHA FILE

INITIAL VERSION
CURRENT PHYSICAL DIAGRAM

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
LEVEL 2: MAKE ASSIGNMENTS

2.1 PREPARE BUSINESS-OFFICE PACKET

Deposit-Record-Book

2.2 MANAGE RE-ASSIGNMENT FILES

Validate-Application-Packet

Temp-Meta-File

2.3 ASSIGN ROOMS

Temp-Priority-Number-File

Permanent-Form-Meta-File

Floor-Plan

REPRODUCED WITH PERMISSION OF THE COPYRIGHT OWNER. FURTHER REPRODUCTION PROHIBITED WITHOUT PERMISSION.
LEVEL 3: MODIFY-_ASSIGNMENTS

3.1 ROUTE-CHANGE-REQUESTS

3.2 CANCEL-ASSIGNMENT

3.3 ARE-ASSIGNMENT CHANGE

3.4 POST-ASSIGNMENT CHANGE

INITIAL VERSION
CURRENT PHYSICAL DIAGRAM

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
APPENDIX C

FINAL CURRENT PHYSICAL DATA FLOW DIAGRAMS
CONTEXT DIAGRAM:

FINAL VERSION
CURRENT PHYSICAL DIAGRAM

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
LEVEL 1:

SCREEN REQUESTS + APPLICATIONS

1. SCREEN REQUESTS + APPLICATIONS

COMPLETE APPLICATIONS
REQUESTS
APPLICATION REQUESTS
REJECTS
SCREEN REQUEST CHANGES
REVISE REQUEST

2. MAKE ASSIGNMENTS

DEPOSIT ACKNOWLEDGMENT
PERMANENT ACCOUNT
PART 11
ARCH ASSIGNMENT CARD
ARCH ASSIGNMENT NOTICE

3. MODIFY ASSIGNMENTS

FLOOR PLAN + TEMPORARY NUMBER +
TEMP ALFA FILE

FIND FILE

FINAL VERSION
CURRENT PHYSICAL DIAGRAM
LEVEL 2: MAKE ASSIGNMENTS

2.1 PREPARE BUSINESS OFFICE PACKET

2.2 MANAGE PRE-ASSIGNMENT FILES

2.3 ASSIGN ROOMS

FINAL VERSION
CURRENT PHYSICAL DIAGRAM

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
LEVEL 3: MODIFY ASSIGNMENTS

3.1 ROUTE CHANGE REQUESTS

3.2 CANCEL ASSIGNMENT

3.3 PRE-ASSIGNMENT CHANGE

3.4 POST-ASSIGNMENT CHANGE

FINAL VERSION
CURRENT PHYSICAL DIAGRAM
APPENDIX D

FINAL CURRENT LOGICAL DATA FLOW DIAGRAMS
CONTEXT DIAGRAM:

FINAL VERSION
CURRENT LOGICAL DIAGRAM
APPENDIX E

INITIAL NEW LOGICAL DATA FLOW DIAGRAMS
Diagram D:

1. INITIAL-QUERY

2. ANSWER-QUERY

3. PRODUCE-REPORT

4. DATA-BASE-INTERFACE

ANALYST-QUERY

ADMIN-ASSIGNMENT

DATA-BASE-CHANGE

SCREEN-DATA-REQUEST

REPORT-RESPONSE

FINAL-ASSIGNMENT-LISTING

INITIAL-ENTRY

NEW LOGICAL Diagram
Diagram 1: Initial Data Entry

Initial Version

Neighborhood Diagram
Diagram 2: Answer-Query

Initial Version
New Logical Diagram
Diagram 3: Produce-Report

Initial Version
New Logical Diagram
Diagram 4: Database Interface

4.1 Make Room Assignment

4.2 Change Data

4.3 Delete Student

NEW LOGICAL DIAGRAM

INITIAL VERSION
APPENDIX F

FINAL NEW LOGICAL DATA FLOW DIAGRAMS
Context Diagram

Final Version
New Logical Diagram
Diagram 0:

1. Initial Data Entry

2. Answer Query

3. Produce Report

4. Database Interface

Arrow directions indicate flow of data and processes.

Final Version

New Logical Diagram
Diagram 1: Initial-Data-Entry

1.1 Enter-Initial-Student-Info

1.2 Enter-Additional-Student-Info

More-Student-Info

Utilization-Number

Student-Info

Priority-Number

Rho-Organization-Data-Base

Final Version

Web Logical Diagram
Diagram 2: Answer-Query

2.1 Select-output-destination

2.2 Make-query response-terminal

2.3 Make-query response-file

Rho-Adax resident-data-base

Final version
New logical diagram
DIAGRAM 3: PRODUCE-REPORT

3.1 PRODUCE-CANNEI REPORT

3.2 PRODUCE-NEW-REPORT

FINAL VERSION
NEW LOGICAL DIAGRAM

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Diagram 4: DATA-BASE-INTERFACE

4.1 MAKE ROOM ASSIGNMENT

4.2 CHANGE DATA

4.3 DELETE STUDENT

FINAL VERSION
NEW LOGICAL DIAGRAM
APPENDIX G

PSL/PSA REPRESENTATION OF ANALYSIS
Name Selection

Parameters:  LANGUAGE=psl  DB=rho.dms  PRINT  PUNCH=#punch.010382
            SELECTION=`BASIC`  TYPE

1 academic_info  ELEMENT
2 assignment_info  ELEMENT
3 parent_address  ELEMENT
4 permanent_address  ELEMENT
5 personal_info  ELEMENT
6 priority_number  ELEMENT
7 residence_halls_info  ELEMENT
8 social_security_number  ELEMENT
9 validation_number  ELEMENT
10 canned_report_request  GROUP
11 canned_report_response  GROUP
12 delete_request  GROUP
13 file_response  GROUP
14 modified_student_info  GROUP
15 more_student_info  GROUP
16 new_report_request  GROUP
17 new_report_response  GROUP
18 query_response_file  GROUP
19 query_response_terminal  GROUP
20 student_info  GROUP
21 terminal_response  GROUP
22 assignment_query  INPUT
23 data_base_change  INPUT
24 room_assignment  INPUT
25 screened_query  INPUT
26 screened_report_request  INPUT
27 valid_student_application  INPUT
28 assign_room  INTERFACE
29 change_data_base  INTERFACE
30 disperse_temp_cardfile  INTERFACE
31 request_data  INTERFACE
32 validate_application  INTERFACE
33 assignment_query_response  OUTPUT
34 final_assignment_listing  OUTPUT
35 query_response  OUTPUT
36 report_response  OUTPUT
37 answer_query  PROCESS
38 change_data  PROCESS
39 data_base_interface  PROCESS
40 delete_student  PROCESS
41 enter_additional_student_info  PROCESS
42 enter_initial_student_info  PROCESS
43 initial_data_entry  PROCESS
44 make_query_response_file  PROCESS
make_query_response_terminal  PROCESS
make_room_assignment   PROCESS
produce_canned_report  PROCESS
produce_new_report     PROCESS
produce_report         PROCESS
rho_dorm_management    PROCESS
select_output_destination PROCESS
rho_dorm_resident_data_base  SET
DEFINE ELEMENT academic_info;
  DESCRIPTION;
  This is information about the student such as academic status,
  student status (new, former, or transfer), and major.
  ;
  TRACE KEYWORD 'Dl';
  CONTAINED IN student_info;
  CONTAINED IN more_student_info;

DEFINE ELEMENT assignment_info;
  DESCRIPTION;
  This is information about the student's characteristics and
  preferences for their residence halls assignment.
  ;
  TRACE KEYWORD 'Dl';
  CONTAINED IN student_info;
  CONTAINED IN more_student_info;

DEFINE ELEMENT parent_address;
  DESCRIPTION;
  This is the student's parents' address and phone number.
  ;
  TRACE KEYWORD 'Dl';
  CONTAINED IN student_info;
  CONTAINED IN more_student_info;

DEFINE ELEMENT permanent_address;
  DESCRIPTION;
  This is the student's permanent mailing address and phone
  number.
  ;
  TRACE KEYWORD 'Dl';
  CONTAINED IN student_info;
  CONTAINED IN more_student_info;

DEFINE ELEMENT personal_info;
  DESCRIPTION;
  This is information about the student such as date of birth,
  marital status, sex, and physical limitations.
  ;
  TRACE KEYWORD 'Dl';
  CONTAINED IN student_info;
DEFINE ELEMENT priority_number;
DESCRIPTION;
This is the number which the \textsuperscript{\textregistered} rho \textsuperscript{\textregistered} gives to each application to show the order in which the actual assignments will be made. It is sequentially assigned in the order in which the applications are received at the residence halls office.

TRACE KEYWORD \textsuperscript{\textregistered}Dl\textsuperscript{\textregistered};
USED BY enter_initial_student_info TO DERIVE rho_dorm_resident_data_base;
CONTAINED IN valid_student_application;

DEFINE ELEMENT residence_halls_info;
SYNONYMS ARE rho_info;
DESCRIPTION;
This is the official information which the residence halls keeps on each applicant besides their priority number and their validation number. It includes application type, pre-payment amount, pre-payment received by, pre-payment date, cancellation refund amount, cancellation date, withdrawal refund amount, and withdrawal date.

TRACE KEYWORD \textsuperscript{\textregistered}Dl\textsuperscript{\textregistered};
CONTAINED IN student_info;
CONTAINED IN more_student_info;

DEFINE ELEMENT social_security_number;
SYNONYMS ARE soc_sec_num;
DESCRIPTION;
This is the student\textsuperscript{\textregistered}s social security number, or their university assigned alternative. It must be unique within the residence halls data base.

TRACE KEYWORD \textsuperscript{\textregistered}Dl\textsuperscript{\textregistered};
CONTAINED IN student_info;

DEFINE ELEMENT validation_number;
DESCRIPTION;
This is the number returned from the business office upon receipt of the application packet and the $100 pre-payment.

TRACE KEYWORD \textsuperscript{\textregistered}Dl\textsuperscript{\textregistered};
USED BY enter_additional_student_info TO UPDATE rho_dorm_resident_data_base;
CONTAINED IN valid_student_application;

DEFINE GROUP canned_report_request;
DESCRIPTION;
This is a request for a report chosen from a list of standard, pre-prepared reports.
DEFINE GROUP canned_report_response;
  DESCRIPTION;
  This is a report, chosen from a list of prepared reports, in a
  previously constructed format.
  ;
  TRACE KEYWORD 'D3';
  DERIVED BY produce_canned_report USING
  rho_dorm_resident_data_base;
  CONTAINED IN screened_report_request;

DEFINE GROUP delete_request;
  DESCRIPTION;
  This is a screened request to remove a student from the active
  data base.
  ;
  TRACE KEYWORD 'D4';
  USED BY delete_student TO UPDATE
  rho_dorm_resident_data_base;
  CONTAINED IN report_response;

DEFINE GROUP file_response;
  DESCRIPTION;
  This is the response to a query where the output is to a file.
  ;
  TRACE KEYWORD 'D2';
  DERIVED BY make_query_response_file USING
  rho_dorm_resident_data_base;
  CONTAINED IN query_response;

DEFINE GROUP modified_student_info;
  DESCRIPTION;
  This is new, modified student information which needs to replace
  old or incomplete information in the data base. It might be
  information about the student or specific information which the
  rho maintains.
  ;
  TRACE KEYWORD 'D4';
  USED BY change_data TO UPDATE
  rho_dorm_resident_data_base;
  CONTAINED IN data_base_change;

DEFINE GROUP more_student_info;
  DESCRIPTION;
  This is the rest of the student information, which might not
  have been entered before the application was sent off to the
  business office.
  ;
TRACE KEYWORD ‘D1’;
CONSISTS OF permanent_address;
CONSISTS OF residence_halls_info;
CONSISTS OF assignment_info;
CONSISTS OF personal_info;
CONSISTS OF academic_info;
CONSISTS OF parent_address;
USED BY enter_additional_student_info TO UPDATE rho_dorm_resident_data_base;
CONTAINED IN valid_student_application;

DEFINE GROUP new_report_request;
DESCRIPTION;
This is a request for a report which has never before been generated. Note: - most likely this will not be implemented.

TRACE KEYWORD ‘D3’;
EMPLOYED BY produce_new_report;
CONTAINED IN screened_report_request;

DEFINE GROUP new_report_response;
DESCRIPTION;
This is a report with out a previously prepared format.

TRACE KEYWORD ‘D3’;
DERIVED BY produce_new_report USING rho_dorm_resident_data_base;
CONTAINED IN report_response;

DEFINE GROUP query_response_file;
DESCRIPTION;
This is a request for information with the response to go to a file for later viewing or printing.

TRACE KEYWORD ‘D2’;
DERIVED BY select_output_destination;
EMPLOYED BY make_query_response_file;

DEFINE GROUP query_response_terminal;
DESCRIPTION;
This is a request for information with the response to go to the terminal.

TRACE KEYWORD ‘D2’;
DERIVED BY select_output_destination;
EMPLOYED BY make_query_response_terminal;

DEFINE GROUP student_info;
DESCRIPTION;
This is the basic student information including name, social security number, various addresses, and residence halls information.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
200 ;
201     TRACE KEYWORD 'D1';
202     CONSISTS OF social_security_number;
203     CONSISTS OF permanent_address;
204     CONSISTS OF parent_address;
205     CONSISTS OF academic_info;
206     CONSISTS OF personal_info;
207     CONSISTS OF assignment_info;
208     CONSISTS OF residence_halls_info;
209     USED BY
210     enter_initial_student_info TO DERIVE
211     rho_dorm_resident_data_base;
212     CONTAINED IN valid_student_application;
213 DEFINE GROUP terminal_response;
214 DESCRIPTION;
215 This is the response to a query where the output is to the
216 terminal.
217 ;
218     TRACE-keyword 'D2';
219     DERIVED BY 
220     make_query_response_terminal USING
221     rho_dorm_resident_data_base;
222     CONTAINED IN query_response;
223 DEFINE INPUT assignment_query;
224 DESCRIPTION;
225 This is a request, made by the person doing the room assigning,
226 for information which they might need to make a good room
227 assignment, such as:
228 - what empty rooms are left on the third floor of Jesse?
229 - are there any M^th majors from Montana who would be
230 able to room with a smoker from New Jersey?
231 ;
232 EMPLOYED BY data_base_interface,
233 make_room_assignment;
234 GENERATED BY assign_room;
235 RECEIVED BY rho_dorm_management;
236 DEFINE INPUT data_base_change;
237 DESCRIPTION;
238 This consists of new information and deletion requests for the
239 data base. Some of the new information might be changes to the
240 information initially entered, or it might be new information
241 necessary because of a 'cancel' or 'withdraw' situation.
242 ;
243     TRACE KEYWORD 'DO';
244     CONSISTS OF modified_student_info;
245     CONSISTS OF delete_request;
246 EMPLOYED BY data_base_interface;
247 GENERATED BY change_data_base;
248 RECEIVED BY rho_dorm_management;
249 DEFINE INPUT room_assignment;
252      DESCRIPTION;
253      This is the actual room assignment made by the exterior process, 'assign room'.
254
255    TRACE KEYWORD 'DO';
256    USED BY data_base_interface TO UPDATE
257    rho_dorm_resident_data_base;
258    USED BY make_room_assignment TO UPDATE
259    rho_dorm_resident_data_base;
260    EMPLOYED BY data_base_interface;
261    GENERATED BY assign_room;
262    RECEIVED BY rho_dorm_management;
263
264    DEFINE INPUT screened_query;
265    DESCRIPTION;
266    This is a question, which has been screened as appropriate, for the data base. Typically it would be a short or informal question and it would not get a prepared, formatted answer. Most of the time the response will be to the terminal.
267
268    TRACE KEYWORD 'DO';
269    EMPLOYED BY answer_query,
270    select_output_destination;
271    GENERATED BY request_data;
272    RECEIVED BY rho_dorm_management;
273
274    DEFINE INPUT screened_report_request;
275    DESCRIPTION;
276    This is a report request, which has been screened as appropriate for the data base. Typically it would be a prepared, formatted answer, either sent to a file or the printer. Most report requests would come from a known list of possible, pre-prepared reports.
277
278    TRACE KEYWORD 'DO';
279    CONSISTS OF canned_report_request;
280    CONSISTS OF new_report_request;
281    EMPLOYED BY produce_report;
282    GENERATED BY request_data;
283    RECEIVED BY rho_dorm_management;
284
285    DEFINE INPUT valid_student_application;
286    DESCRIPTION;
287    This includes the student information, the student demographics, the priority number from the rho, and the validation number from the business office.
288
289    TRACE KEYWORD 'DO';
290    CONSISTS OF student_info;
291    CONSISTS OF more_student_info;
292    CONSISTS OF priority_number;
293    CONSISTS OF validation_number;
USED BY initial_data_entry TO DERIVE rho_dorm_resident_data_base;
USED BY initial_data_entry TO UPDATE rho_dorm_resident_data_base;
GENERATED BY validate_application;
RECEIVED BY rho_dorm_management;

DEFINE INTERFACE assign_room;
DESCRIPTION;
This exterior process uses information from the data base to make the room assignment.
;
TRACE KEYWORD 'CD';
GENERATES room_assignment;
GENERATES assignment_query;
RECEIVES assignment_query_response;

DEFINE INTERFACE change_data_base;
DESCRIPTION;
This exterior process takes requests for changes to the data base and screens them. These might include new information previously not provided, changes to addresses, or status changes.
;
TRACE KEYWORD 'CD';
GENERATES data_base_change;

DEFINE INTERFACE disperse_temp_cardfile;
DESCRIPTION;
This exterior process will take the final assignment listing and place the correct room assignment on each of the parts of the application (cards) and send them to the proper places, i.e. the student, the dorm, and the residence halls card file.
;
TRACE KEYWORD 'CD';
RECEIVES final_assignment_listing;

DEFINE INTERFACE request_data;
DESCRIPTION;
This process would pass query and report requests to the automated process, where they would be answered.
;
TRACE KEYWORD 'CD';
GENERATES screened_query;
GENERATES screened_report_request;
RECEIVES query_response;
RECEIVES report_response;

DEFINE INTERFACE validate_application;
DESCRIPTION;
This exterior process is responsible for providing a valid student application form including the validation number from
the business office.

```
355  ;
356  TRACE KEYWORD 'CD';
357  GENERATES valid_student_application;
358
359  DEFINE OUTPUT assignment_query_response;
360  DESCRIPTION;
361  This is the response to an assignment query, i.e. a question
362  which the person making the assignments might ask to get more
363  information to allow them to make a good room assignment.
364  ;
365  TRACE KEYWORD 'DO';
366  DERIVED BY data_base_interface USING
367      rho_dorm_resident_data_base;
368  DERIVED BY make_room_assignment USING
369      rho_dorm_resident_data_base;
370  GENERATED BY rho_dorm_management;
371  RECEIVED BY assign_room;
372
373  DEFINE OUTPUT final_assignment_listing;
374  DESCRIPTION;
375  This would be the final print out of the students and their room
376  assignments. It would be used for placing the proper
377  information on the respective parts of the card file before the
378  cards were dispersed to the students, the dorms and the rho
379  card file.
380  ;
381  TRACE KEYWORD 'DO';
382  DERIVED BY produce_report USING
383      rho_dorm_resident_data_base;
384  DERIVED BY produce_canned_report USING
385      rho_dorm_resident_data_base;
386  GENERATED BY rho_dorm_management;
387  RECEIVED BY disperse_temp_cardfile;
388
389  DEFINE OUTPUT query_response;
390  DESCRIPTION;
391  This is a response to a screened query. It would not be in
392  a prepared format, but rather in a more raw style. This output
393  should be able to be directed to either the screen or a file,
394  for later printing.
395  ;
396  TRACE KEYWORD 'DO';
397  CONSISTS OF terminal_response;
398  CONSISTS OF file_response;
399  DERIVED BY answer_query USING
400      rho_dorm_resident_data_base;
401  GENERATED BY rho_dorm_management;
402  RECEIVED BY request_data;
403
404  DEFINE OUTPUT report_response;
405  DESCRIPTION;
406  ```
This is a response to a report request. It would be in a prepared format. This output should be able to be directed to either the screen or a file, for later printing, or directly to the printer.

TRACE KEYWORD 'DO';
CONSISTS OF canned_report_response;
CONSISTS OF new_report_response;
DERIVED BY produce_report USING rho_dorm_resident_data_base;
GENERATED BY rho_dorm_management;
RECEIVED BY request_data;

DEFINE PROCESS answer_query;
DESCRIPTION;
This process tries to get an answer from the data base for a query. Such queries might be:
- what is a student’s dorm status
- what is a student’s assignment status

SUBPARTS ARE select_output_destination,
  make_query_response_terminal,
  make_query_response_file;
TRACE KEYWORD 'DO';
EMPLOYS screened_query;
PART OF rho_dorm_management;
DERIVES query_response USING rho_dorm_resident_data_base;

DEFINE PROCESS change_data;
DESCRIPTION;
This process allows the information currently stored in the data base to be changed. It will handle all changes after the 'initial_data_entry' process, including adding new information, modifying old information, or entering 'cancel' or 'withdraw' information.

SUBPARTS ARE select_output_destination,
  make_query_response_terminal,
  make_query_response_file;
TRACE KEYWORD 'DO';
EMPLOYS screened_query;
PART OF rho_dorm_management;
DERIVES query_response USING rho_dorm_resident_data_base;

DEFINE PROCESS data_base_interface;
DESCRIPTION;
This process handles all interactions with the data in the data base after the initial data has been entered. It also is used during the actual assigning rooms process.

SUBPARTS ARE make_room_assignment,
  change_data,
  delete_student;
TRACE KEYWORD 'DO';
EMPLOYS  
  data_base_change, 
  assignment_query; 

PART OF  
  rho_dorm_management; 
DERIVES  
  assignment_query_response USING 
  rho_dorm_resident_data_base; 
UPDATES  
  rho_dorm_resident_data_base USING 
  room_assignment; 

DEFINE PROCESS delete_student; 
DESCRIPTION; 
This process removes a student record from the active data base.  
Note that the record will still be accessible until the data 
base is completely purged. 

TRACE KEYWORD 'D4';  
PART OF data_base_interface; 
UPDATES rho_dorm_resident_data_base USING 
  delete_request; 

DEFINE PROCESS enter_additional_student_info; 
DESCRIPTION; 
This process allows further data entry in case the entire 
contents of the application were not added to the data base 
at the time of initial data entry. 

TRACE KEYWORD 'D1'; 
UPDATES rho_dorm_resident_data_base USING 
  more_student_info; 
UPDATES rho_dorm_resident_data_base USING 
  validation_number;

DEFINE PROCESS enter_initial_student_info; 
DESCRIPTION; 
This process allows one to enter as much information from the 
application packet as possible before the application is sent 
off to the Business Office. The minimal entry must be the 
student's social security number, or their university assigned 
equivalent. 

TRACE KEYWORD 'D1'; 
PART OF initial_data_entry; 
DERIVES rho_dorm_resident_data_base USING 
  student_info; 
DERIVES rho_dorm_resident_data_base USING 
  priority_number;

DEFINE PROCESS initial_data_entry; 
DESCRIPTION; 
This process places all of the student and demographic 
information from the application packet, the residence halls
office, and the business office into the data base. This includes the priority number from the residence halls office and the validation number from the business office.

SUBPARTS ARE enter_initial_student_info;

TRACE KEYWORD 'D0';

PART OF rho_dorm_management;

DERIVES rho_dorm_resident_data_base USING valid_student_application;

UPDATES rho_dorm_resident_data_base USING valid_student_application;

DEFINE PROCESS make_query_response_file;

DESCRIPTION;

This process handles all queries which require their output to go to a file on disk for later printing or viewing.

TRACE KEYWORD 'D2';

EMPLOYS query_response_file;

PART OF answer_query;

DERIVES file_response USING rho_dorm_resident_data_base;

DEFINE PROCESS make_query_response_terminal;

DESCRIPTION;

This process handles all queries which require their output to go to the terminal.

TRACE KEYWORD 'D2';

EMPLOYS query_response_terminal;

PART OF answer_query;

DERIVES terminal_response USING rho_dorm_resident_data_base;

DEFINE PROCESS make_room_assignment;

DESCRIPTION;

This process provides information to the person making room assignments and it also will update the data base with the room assignment as it is made.

TRACE KEYWORD 'D4';

EMPLOYS assignment_query;

PART OF data_base_interface;

DERIVES assignment_query_response USING rho_dorm_resident_data_base;

UPDATES rho_dorm_resident_data_base USING room_assignment;

DEFINE PROCESS produce_canned_report;

DESCRIPTION;

This process allows the user to select from a list of reports, which are in a prepared format.
DEVISE PROCESS produce_jnew_report;
DESCRIPTION;
This procedure, if implemented, would allow the user to custom create a new report.

DEVISE PROCESS produce_jreport;
DESCRIPTION;
This process produces canned reports on demand from the information in the data base. It might also be used to interactively build a new report.

SUBPARTS ARE produce_canned_report, produce_new_report;

DEVISE PROCESS rho_dorm_management;
SYNONYMS ARE rho;
DESCRIPTION;
This process is that part of the `rho` which handles applications for room assignment, makes those assignments, changes those assignments, and produces reports and answers about the assignments and the students for those who need to know. It does this by maintaining a data base of information containing a record for each student.

SUBPARTS ARE initial_data_entry, answer_query, produce_report, data_base_interface;

DEVISE PROCESS trace_dorm_request_request;
EMPLOYS canned_report_request;
PART OF produce_report;
DERIVES final_assignment_listing USING rho_dorm_resident_data_base;
DERIVES canned_report_response USING rho_dorm_resident_data_base;

DEVISE PROCESS trace_dorm_report_request;
EMPLOYS new_report_request;
PART OF produce_report;
DERIVES new_report_response USING rho_dorm_resident_data_base;

DEVISE PROCESS trace_dorm_report_request;
EMPLOYS screened_report_request;
PART OF rho_dorm_management;
DERIVES report_response USING rho_dorm_resident_data_base;
DERIVES final_assignment_listing USING rho_dorm_resident_data_base;

DEVISE PROCESS trace_dorm_management;
SYNONYMS ARE rho;
DESCRIPTION;
This process is that part of the `rho` which handles applications for room assignment, makes those assignments, changes those assignments, and produces reports and answers about the assignments and the students for those who need to know. It does this by maintaining a data base of information containing a record for each student.

SUBPARTS ARE initial_data_entry, answer_query, produce_report, data_base_interface;

DEVISE PROCESS trace_dorm_query_response;
GENERATES assignment_query_response;
DEFINE PROCESS select_output_destination;

DESCRIPTION;

This process determines where the query response will go -
either to the screen or to a disk file.

TRACE KEYWORD 'D2';

EMPLOYS screened_query;

PART OF answer_query;

DERIVES query_response_terminal;

DERIVES query_response_file;

DEFINE SET rho_dorm_resident_data_base;

SYNONYMS ARE rho_db;

DESCRIPTION;

This is the data base file which contains all of the required
information on the students, their applications, and finally
their assignments.

TRACE KEYWORD 'D0';

DERIVED BY initial_data_entry USING
valid_student_application;

DERIVED BY enter_initial_student_info USING
student_info;

DERIVED BY enter_initial_student_info USING
priority_number;

UPDATED BY initial_data_entry USING
valid_student_application;

UPDATED BY data_base_interface USING
room_assignment;

UPDATED BY enter_additional_student_info USING
more_student_info;

UPDATED BY enter_additional_student_info USING
validation_number;

UPDATED BY make_room_assignment USING
room_assignment;

UPDATED BY change_data USING
modified_student_info;

UPDATED BY delete_student USING
delete_request;

USED BY answer_query TO DERIVE
query_response;

USED BY produce_report TO DERIVE
report_response;

USED BY produce_report TO DERIVE
final_assignment_listing;

USED BY data_base_interface TO DERIVE
assignment_query_response;

USED BY make_query_response_terminal TO DERIVE
terminal_response;

USED BY make_query_response_file TO DERIVE
file_response;

USED BY produce_canned_report TO DERIVE
final_assignment_listing;

USED BY produce_canned_report TO DERIVE
canned_report_response;

USED BY produce_new_report TO DERIVE
new_report_response;

USED BY make_room_assignment TO DERIVE
assignment_query_response;
INITIAL ASSESSMENT OF POSSIBLE

RESIDENCE HALLS DATA BASE ELEMENTS
Possible elements for the Residence Halls data base

<table>
<thead>
<tr>
<th></th>
<th># bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>student:</td>
<td></td>
</tr>
<tr>
<td>last name</td>
<td></td>
</tr>
<tr>
<td>first name</td>
<td>36</td>
</tr>
<tr>
<td>middle name</td>
<td></td>
</tr>
<tr>
<td>social security number</td>
<td>9</td>
</tr>
<tr>
<td>permanent address:</td>
<td></td>
</tr>
<tr>
<td>street</td>
<td>20</td>
</tr>
<tr>
<td>city</td>
<td>18</td>
</tr>
<tr>
<td>state</td>
<td>2</td>
</tr>
<tr>
<td>zip</td>
<td>9</td>
</tr>
<tr>
<td>country</td>
<td>18</td>
</tr>
<tr>
<td>telephone #</td>
<td>10</td>
</tr>
<tr>
<td>local address:</td>
<td></td>
</tr>
<tr>
<td>street</td>
<td></td>
</tr>
<tr>
<td>city</td>
<td></td>
</tr>
<tr>
<td>state</td>
<td></td>
</tr>
<tr>
<td>zip</td>
<td></td>
</tr>
<tr>
<td>telephone #</td>
<td></td>
</tr>
<tr>
<td>parent or guardian:</td>
<td></td>
</tr>
<tr>
<td>name</td>
<td>25</td>
</tr>
<tr>
<td>address:</td>
<td></td>
</tr>
<tr>
<td>street</td>
<td></td>
</tr>
<tr>
<td>city</td>
<td></td>
</tr>
<tr>
<td>state</td>
<td>77</td>
</tr>
<tr>
<td>zip</td>
<td></td>
</tr>
<tr>
<td>country</td>
<td></td>
</tr>
<tr>
<td>? telephone</td>
<td></td>
</tr>
<tr>
<td>academic status:</td>
<td></td>
</tr>
<tr>
<td>freshman</td>
<td></td>
</tr>
<tr>
<td>sophomore</td>
<td>1</td>
</tr>
<tr>
<td>junior</td>
<td></td>
</tr>
<tr>
<td>senior</td>
<td></td>
</tr>
<tr>
<td>graduate</td>
<td></td>
</tr>
</tbody>
</table>
student type:
  new student
  former student
  transfer student

quarter applied for:
  fall/winter/spring
  year

date of birth

questions:
  married/single
  male/female

citizenship:
  US
  other:
    name

personal physical limitations

preferred assignment:
  first choice hall:
    name
  second choice hall:
    name

type of visitation:
  coed (unlimited visitation)
  non-coed (unlimited visitation)
  non-coed (limited visitation)

room:
  double/single

roommate preference:
  name

major field:
  name

personal characteristics:
  smoker/non-smoker
drinker/non-drinker
  keep late hours (y/n)

preferred roommate characteristics:
  smoker/non-smoker
drinker/non-drinker
  keep late hours (y/n)
FOR OFFICIAL USE ONLY SECTIONS

type of application:
  vo-tech
  athletic scholarship  ) 2
  vocational-rehabilitation
deferred

$100 pre-payment:
  amount  5
  received by:
    name/initals  3
  date  8

pre-payment cancellation:
  amount of refund  5
  date  8

cancellation:
  date  8
  refund amount  5
  residence halls
  food service

application number (priority #)  5

reg. number (validation #)  10

dorm address:
  single/double  1
  dorm  8
  room  5
  phone #  4

keys:
  room key id number  10
  outdoor key id number  10

=====

TOTAL  424
INITIAL GLOBAL DATA SET

DATA DICTIONARY
Possible elements for the Residence Halls data base

<table>
<thead>
<tr>
<th></th>
<th>LENGTH</th>
<th>TYPE</th>
<th>notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>student:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>last name</td>
<td>17</td>
<td>t</td>
<td>t = text</td>
</tr>
<tr>
<td>first + middle</td>
<td>15</td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>social security number</td>
<td>9</td>
<td>n</td>
<td>n = number</td>
</tr>
<tr>
<td>permanent address:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>perstreet</td>
<td>20</td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>percity</td>
<td>18</td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>perstate</td>
<td>2</td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>perzip</td>
<td>9</td>
<td>n</td>
<td></td>
</tr>
<tr>
<td>percountry</td>
<td>18</td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>pertelephone #</td>
<td>10</td>
<td>n</td>
<td></td>
</tr>
<tr>
<td>parent or guardian:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pglastname</td>
<td>17</td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>pgfirst + middle</td>
<td>15</td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>pgstreet</td>
<td>20</td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>pgcity</td>
<td>18</td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>pgstate</td>
<td>2</td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>pgzip</td>
<td>9</td>
<td>n</td>
<td></td>
</tr>
<tr>
<td>pgcountry</td>
<td>18</td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>pgtelephone #</td>
<td>10</td>
<td>n</td>
<td></td>
</tr>
<tr>
<td>academic status</td>
<td>1</td>
<td>n</td>
<td>(1-4,5,6)</td>
</tr>
<tr>
<td>date of birth</td>
<td>8</td>
<td>d</td>
<td>d = date</td>
</tr>
<tr>
<td>sex</td>
<td>1</td>
<td>t</td>
<td>(F,M)</td>
</tr>
</tbody>
</table>
INITIAL LOCAL DATA SET

DATA DICTIONARY
Possible elements for the Residence Halls data base

<table>
<thead>
<tr>
<th>Date</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4/1/84</td>
<td></td>
</tr>
<tr>
<td>4/26/84</td>
<td></td>
</tr>
<tr>
<td>5/9/84</td>
<td></td>
</tr>
<tr>
<td>5/10/84</td>
<td></td>
</tr>
</tbody>
</table>

Data description of local data set

<table>
<thead>
<tr>
<th>NAME</th>
<th>LENGTH</th>
<th>TYPE</th>
<th>notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>student:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>last name</td>
<td>17</td>
<td>t</td>
<td>t = text</td>
</tr>
<tr>
<td>first + middle</td>
<td>15</td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>social security number</td>
<td>9</td>
<td>n</td>
<td>n = number</td>
</tr>
<tr>
<td>student status</td>
<td>1</td>
<td>t</td>
<td>(N,F,T)</td>
</tr>
<tr>
<td>quarter applied for</td>
<td>3</td>
<td>t</td>
<td>(F84)</td>
</tr>
<tr>
<td>marital status</td>
<td>1</td>
<td>t</td>
<td>(S,M)</td>
</tr>
<tr>
<td>US citizen</td>
<td>1</td>
<td>t</td>
<td>(Y,N)</td>
</tr>
<tr>
<td>other citizenship</td>
<td>18</td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>personal physical limitations</td>
<td>10</td>
<td>t</td>
<td>(0-9 code)</td>
</tr>
<tr>
<td>hall assignment preference:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>first choice</td>
<td>8</td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>second choice</td>
<td>8</td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>visitation preference</td>
<td>2</td>
<td>t</td>
<td>(CU,NU,NL)</td>
</tr>
<tr>
<td>room type preference</td>
<td>1</td>
<td>t</td>
<td>(D,S)</td>
</tr>
<tr>
<td>roommate preference:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>last name</td>
<td>17</td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>first + middle</td>
<td>15</td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>major field</td>
<td>4</td>
<td>t</td>
<td>(MATH)</td>
</tr>
<tr>
<td>personal characteristics:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>smoker</td>
<td>1</td>
<td>t</td>
<td>(Y,N)</td>
</tr>
<tr>
<td>drinker</td>
<td>1</td>
<td>t</td>
<td>&quot;</td>
</tr>
<tr>
<td>nightowl</td>
<td>1</td>
<td>t</td>
<td>&quot;</td>
</tr>
<tr>
<td>preferred roommate characteristics:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>smoker</td>
<td>1</td>
<td>t</td>
<td>&quot;</td>
</tr>
<tr>
<td>drinker</td>
<td>1</td>
<td>t</td>
<td>&quot;</td>
</tr>
<tr>
<td>nightowl</td>
<td>1</td>
<td>t</td>
<td>&quot;</td>
</tr>
</tbody>
</table>

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
FOR OFFICIAL USE ONLY SECTIONS

type of application  
pre-payment:  
amount  
received by  
date  
pre-payment cancellation:  
amount of refund  
date  
withdrawal info:  
date  
refund amount  
application number (priority #)  
reg. number (validation #)
This document contains the overview design of the RHDMS. It includes top level structure charts, menu templates, and the pseudo code for the main system controller module.

"Welcome to the RESIDENCE HALL DATA MANAGEMENT SYSTEM"
There are some underlying concepts and philosophical issues we wish to note.

We started out by looking at the question of design from two points of view. One was the STRUCTURAL DESIGN OF THE OVERALL SYSTEM. By structural we mean how many modules would there be, what would be the function of each module, what would be the relationship between one module and another, and similar considerations. The second point we felt we needed to consider was the actual shape of the USER INTERFACE. Issues dealt with included questions such as, should the system be command driven, menu driven, or some combination of the two? what sorts of conversations should be possible between the user and the system, etc.

To begin our work, we decided to conceptualize the system as a single transform of raw data input to an organized data base, accomplished by a series of transactions. Each transaction would be a menu which would allow the user to select one of the variety of functions. Because we are faced with the need to supply our implementation assistant with meaningful work, we decided to work on the physical design of the transaction menus first.
Top level DFD
of basic transform
This is a preliminary list of points of concern regarding the transaction centers and their corresponding menus.

1. The transactions should be primarily menu driven because the level of proficiency of the different users of the system will most likely vary greatly, and because menus will allow a user relative proficiency with the system even with infrequent use.

2. One question in particular which needs further investigation is how the problem of erroneous menu selections should be dealt with. The two choices initially considered were a.) clearing the screen and redisplaying the menu along with an error message displayed in the command line space, and b) simply displaying an error message, scrolling the menu up some lines and redisplaying the prompt. The problems associated with a clear and redisplay involve the time, and perhaps the terminal specific control characters required, while the problems associated with scrolling the menu is that some of the information provided by the menu would be scrolled off the top. Some compromise might be possible with some number of scrolls done followed by a clear and redisplay if too many consecutive errors are made.

3. In general, the menu should have as basic content a) a header which identifies the menu, i.e. what transaction center is currently active, b) a formatted list of things that can be done within the transaction center, and c) a command/prompt line for the user to enter his menu choice.

4. We would like the option of command override in conjunction with the menus, perhaps by allowing the user to enter the keys for several menu choices in a command string.

5. Menu choices should be done with letters and not numbers to allow the possibility of building up a command line with a series of initials. One note is that: if there were to be a command line, perhaps it could be signified by requiring it be preceded by a slash - '/' , e.g. /UAR.

6. Finally, each menu choice should consist of a maximum of two lines of text, one with a keyword and a one or two word description, and a second with a brief description of the subfunctions available under the province of that menu choice, separated by a blank line. This, combined with the necessity
of an identification header and a prompt line would limit the number of choices available per screen to 6 or 7.
HEADERS LINES
1 or 2 lines to describe major function served by menu

F: First choice:
   Best choice, Principle choice, Number one:

S: Second choice:
   Next best choice, Almost as good, Etc:

O: Other choices:
   Different, Other, More:

K: Additional Choices:
   First, Second, Third:

M: More choices:
   Fourth, Fifth, Sixth:

C: Last Choices:
   Continue, Go On, Move Ahead:

Q: Quit:
   Exit the menu:

YOUR CHOICE:

A generic representation of the basic format of the menus for the Residence Halls Data Management System.
*** MAIN MENU ***
Select Desired Function Or Issue Command String
==============================================

U: Update data base: (options apply to "A STUDENT RECORD")
   Add, Complete, Modify, Delete, Undelete

A: Assign Rooms:
   Check student records, Record assignments:

R: Reports:
   Review on screen, Print paper copy:

I: Inquiry mode:
   Interactive, Canned queries:

T: Tools:
   Files, Print queue, Etc:

Q: Quit:
   End 1022 session:

YOUR CHOICE:

The preliminary configuration for the MAIN system menu.
Pseudo Code for the MAIN CONTROLLER MODULE:

prompt "welcome message"
STOP = false
repeat ( until STOP )
    clear screen
    display main menu
    prompt for choice
    if ( input = 'U' ) then
        call UPDATE module
    else if ( input = 'A' ) then
        call ASSIGN-ROOMS module
    else if ( input = 'R' ) then
        call REPORT module
    else if ( input = 'I' ) then
        call INQUIRY module
    else if ( input = 'T' ) then
        call SYSTEM-UTILITIES module
    else if ( input = 'Q' ) then
        call SYSTEM-SHUTDOWN module
    STOP = true
end if
end repeat
In brief, this system is intended to provide the managers of the campus residence halls with the means to transform the data they acquire from residence hall applicants and eventual occupants into a form they can use for a variety of purposes. Some of those purposes have already been identified and include such things as providing demographic statistics about the sex, age, class standing, and other features which profile the residence hall population, or providing a filing and occupancy report system to use in making room assignments, or providing the means to track the history of the residence hall populations to use to project future needs or occupancy rates of on-campus housing. Other uses will surely evolve as the system is used.

With regards to design, the system can be viewed from two different perspectives. The first, called transform centered design, is that of a process (or set of processes) which transforms raw unstructured demographic facts about potential residence hall occupants into organized reports and data sets which represents a unified picture (or pictures) of the residence hall population as though it were a single entity. The second, referred to as transaction centered design, is that of a collection of tools and utilities, (or in other terminology modules and functions) and the control (or transaction) centers
Top level structure chart - transform centered perspective
required to coordinate them, which allow a user to manipulate and massage his data into the shapes which best suit his needs.

From the first point of view, the system attempts to monitor and control data input as much as possible, filtering errors, forcing the user at all times to make at least plausible data entries, and giving him every opportunity to review and modify incorrect entries before finally putting the data in a data base. After information has been put in the data base, the user can request a variety of prepared and formatted reports to be printed out. The user has the power to modify data after it has been entered into the data base, but attempts are made to moderate and channel that power.

From the second point of view, the system attempts to provide the tools necessary to allow the experienced user to custom tailor reports, or to query the data base with whatever questions occur to him. Data can be entered, changed, removed temporarily and returned, or taken out altogether. With regard to the structure charts provided at this initial design stage, they have been constructed from, and therefore most closely reflect, the transaction centered design point of view.
RESIDENCE HALLS DATA MANAGEMENT SYSTEM

UPDATE DATABASE
MAKE ROOM ASSIGNMENT
DBC SYSTEM TOOLS
QUERY THE DATABASE
REQUEST FORMATTED REPORTS
EXIT THE SYSTEM

MODIFY OCCUPANCY REPORT
FILE HANDLING ETC
INTER-ACTIVE
CANNED CUSTOM
CLEAN-UP INQUIRY STATUS

ADD COMPLETE UNDELETE
DELETE RECORD ASSIGNMENT
SYSTEM STATUS PROCESSED

STRUCTURE CHART - TOP LEVEL VIEW
This document begins the design of the UPDATE DATA BASE MODULE.

It includes the template for the UPDATE menu, as well as the structure chart and pseudo code for the UPDATE DATA BASE MODULE itself. It also includes the structure chart for the ADD function and the pseudo code for the subfunctions GET-STUDENT-INFO, INITIAL-DATA-INPUT, and REVIEW-DATA.

U: Update data base: (options apply to "STUDENT RECORDS")
   Add, Complete, Modify, Delete, Undelete
This system is being designed to consist of a number of modules. Each of those modules is intended to accomplish a logically distinct set of tasks. The UPDATE-DATA-BASE module is intended to provide the tools to actually manipulate the data base itself, with the principal manipulation being the addition and modification of the actual data in the data base. While other modules may include utilities which will manipulate the data base, e.g. the ASSIGN-ROOMS module will connect a student's social security number with some room identification characteristics, one of the intentions of the design is to locate all data input and all direct data base manipulation in this module. Briefly, the structure of this (and for the most part all) the modules is 1.) advise the user he has arrived at this module and ask for confirmation to continue, 2.) go into a loop which displays the menu choices for each of the functions umbrellaed under this module (until the user indicates he wants to exit), and 3) call the function indicated by the menu choice.
*** UPDATE MENU ***

Add To or Modify The Student Records In The Data Base

A: Add:
   put another student record in the data base.

C: Complete:
   put additional data in an existing student record.

M: Modify:
   change data in an existing student record.

D: Delete:
   make a student record inactive.

U: Undelete
   return a student record to the active data base.

L: Leave
   exit this module, return to parent menu.

YOUR CHOICE:

The preliminary configuration for the Update function menu.
UPDATE
DATA
BASE

ADD
STUDENT
RECORDS

COMPLETE
STUDENT
RECORDS

MODIFY
STUDENT
RECORDS

DELETE
STUDENT
RECORDS

UNDELETE
STUDENT
RECORDS

INITIAL
DATA
INPUT

REVIEW
DATA

MANAGE
TEMP
FILE

APPEND
TO D.B.

RETRIEVE
ONE
RECORD

COMPLETE
STUDENT
INFO

TRANSACT
THE DATA
BASE

CHANGE
STUDENT
INFO

ETC
Pseudo Code for UPDATE MODULE controller:

STOP = false
repeat ( until STOP )
    clear screen
    display menu
    prompt "your choice"
    if ( input = 'A' ) then
        call ADD function
    else if ( input = 'C' ) then
        call COMPLETE function
    else if
        ...
        ...
    else if ( input = 'L' ) then
        STOP = true
    endif
endrepeat
Briefly, the structure of the ADD function is that it calls GET STUDENT INFO, and then MANAGE TEMP FILE in a loop until the user indicates he no longer wants to enter student records. GET STUDENT INFO only gets one record per call, because only one record will be eventually be returned to the calling ADD function in turn to be handed to the MANAGE TEMP FILE subfunction. At the same time, GET STUDENT INFO calls the REVIEW DATA sub-function in a loop because it is assumed that the user will want to be able to review the input data until he is satisfied it is correct. When the user wants to stop adding student records, control is returned to the UPDATE menu.
Pseudo Code for The Main Add Function:

output welcome to ADD FUNCTION message
prompt "do you wish to continue or abort"
if "ok" to add then
  CONTINUE = true
  while (CONTINUE)
    CALL GET-STUDENT-INFO
    request confirmation to add student to data base (y/n)
    if (y) then
      CALL MANAGE-TEMP-FILE
    endif
    prompt "another student? (y/n)"
    if (n) then
      CONTINUE = false
    endwhile
  CALL APPEND-TO-D.B.
endif
Pseudo Code for GET-STUDENT-INFO subfunction:

do initialization (variables strings)
call INITIAL DATA INPUT
print "do you want review?"
if yes then
    DONEREVIEW = false
    while (not DONEREVIEW) do
        call REVIEW DATA
        prompt = "review again?"
        if (input = n) then
            DONEREVIEW = true
        endif
    endwhile
endif
Pseudo Code for INITIAL-DATA-INPUT subfunction:

initialize field and prompt
repeat until end-of-fields or <esc> entered
   ACCEPT = false;
   repeat until (ACCEPT = true)
      display prompt;
      read input;
      if input = <esc> then
         ACCEPT = true
      else if type = ok then
         {  
            ACCEPT = true
            increment prompt and input field
         }
      else
         write error messages
      endif
   endrepeat
endrepeat

{NOTE: loop termination at end of last legal field will leave module with field marker incremented one beyond the last legal value}
Pseudo Code for REVIEW-DATA subfunction:

\[
\text{Quit} = \text{false} \\
\text{screens} = 1 \\
\text{repeat until end-of-screens or QUIT} \\
\quad \text{display next screen \{ 11 lines \}} \\
\quad \text{ask for confirmation} \\
\quad \quad <\text{cr}> \text{ or } <\text{y}> \implies \text{ok, increment page} \\
\quad \quad <\text{n}> \implies \text{need to modify screen} \\
\quad \quad <\text{esc}> \implies \text{done reviewing data} \\
\quad \quad \implies \text{QUIT} = \text{true} \\
\text{if n then} \\
\quad \text{repeat until (lines} > 11 \text{ or } <\text{esc}>\text{)} \\
\quad \quad \text{write screen line #1} \\
\quad \quad \text{prompt for replacement and read} \\
\quad \quad \quad <\text{cr}> \implies \text{ok, increment line} \\
\quad \quad \quad \text{new text} \implies \text{change variable} \\
\quad \quad \quad <\text{esc}> \implies \text{goto next screen} \\
\quad \text{endrepeat} \\
\text{endif} \\
\text{endrepeat}
\]
APPENDIX K

CONTINUE DESIGN - UPDATE DATA BASE MODULE -
SUPPLEMENTAL DATA ENTRY
This document continues the design of the UPDATE-DATA-BASE MODULE.

It includes the subfunctions MANAGE-TEMP-SEQUENTIAL-FILE, and TRANSACT-INTO-DATA-BASE.

U: Update data base:  (options apply to "STUDENT RECORDS")
    Add, Complete, Modify, Delete, Undelete
Pseudo Code for MANAGE-TEMP-SEQUENTIAL-FILE subfunction:

for index = 1 to number of fields
  if ( field_data[ index ] = empty ) then
    if ( field_type = integer ) then
      field_data[ index ] = 0
    else if ( field_type = date ) then
      field_data[ index ] = 00/00/00
    endif
  endif
  outfield = {the first} field_length[ index ] {characters}
    { from } field_data[ index ]
  print outfield to tempfile
endfor
Pseudo Code for TRANSACT-INTO-DATA-BASE subfunction:

- Copy 2 data bases to .BAK files
- Open data base files to update
- Transact tempt files into data base files
- Close data base files
- If (1022 error bit set) then
  - Print "serious data base error - seek help etc"
  - Set tempfiles, data base and .BAK files to read only
- Else
  - Delete .BAK files
  - Delete temp files
- Endif
- Print "update complete message"
This document continues the design of the UPDATE DATA BASE MODULE. It details the COMPLETE and MODIFY functions.

It includes the structure chart and pseudo code for the COMPLETE, the MODIFY, and the WITHDRAW/CANCEL functions as well as the pseudo code for the subfunctions RETRIEVE-ONE-RECORD, GET-ONE-RECORD-SSN, and COMPLETE-STUDENT-INFO.

U: Update data base: (options apply to "STUDENT RECORDS")
  Add, Complete, Modify, Delete, Undelete
The COMPLETE function can best be characterized as a supplemental data entry function. Briefly, its structure is to call RETRIEVE-ONE-RECORD, then COMPLETE-STUDENT-INFO, and the APPEND-TO-TEMP-FILES in a loop until the user indicates he no longer wants to do this type of data entry. The final step is to call TRANSACT-THE-DATA-BASE to put the changed records back into the data base. The process is complicated somewhat by the fact that this type of data entry must be coordinated with records that already exist in the data base. This requires a retrieval process with some kind of multiple search and confirmation steps built in so the user can determine that the record he is adding to is actually the desired target record. It also means that provisions must be made for cases when the desired record cannot be found, (or the hopefully unlikely event that an error is detected in the data base) and to ensure that multiple copies of the same record are not entered into the data base.

The principal goal of this function is to allow the user the option to have added student records to the data base with a minimal initial data entry, and then to go back at a later time and complete the data entry. While it would be feasible to also use this function to change those data elements already entered into the
data base (because of the use of the review function allows any data element to be corrected and therefore changed) that is not its primary purpose. As envisioned, this function would provide the capability to repeatedly retrieve the desired records from the data base, allow necessary data input to be completed, and then return them.

However, even as this design was being written up, changes were being contemplated which may result in modifications. One such concerns the need to ensure that all social security numbers in the data base are unique. Because the various data sets in the data base are to be joined on the social security number, and because of problems in retrieving records for completion when the search key used will result in finding multiple records, the ADD function will most likely have to be modified to ensure that no entry is allowed which would result in two records the same social security number. This will also mean that the function which retrieves records based on social security numbers should never find more than one record, and if it does, a very serious error has occurred. Another concerns the probability that, initially at least, only retrieval based on social security number will be done. This means that the module RETRIEVE--ONE-RECORD, the primary function of which was to determine
which field to use as a basis for retrieval, will not be incorporated into this phase, and the COMPLETE function will call the social security retrieval subfunction directly. Other concerns involve the complexity of the search and confirmation process itself and the problem of when to transact the changed record back into the data base.
COMPLETE
STUDENT
RECORDS

RETRIEVE
ONE
RECORD

COMPLETE
STUDENT
INFO

APPEND
TO TEMP
FILES

TRANSACT
THE DATA
BASE

GET ONE
RECORD
S.S.N.

GET ONE
RECORD
NAME

SECONDARY
DATA
INPUT

REVIEW
DATA

CONFIRM
CORRECT
RECORD
COMPLETE FUNCTION

Pseudo Code for COMPLETE-STUDENT-RECORDS function:

prompt welcome message, "do you wish to continue?"
if ( input = <y> ) then
  CONTINUE = true
  DBERROR = false
  ONEFOUND = false
  while ( CONTINUE = true )
    call RETRIEVE-ONE-RECORD
    if ( ONEFOUND = true ) then
      call COMPLETE-STUDENT-INFO
      prompt "confirm addition to data base"
      if ( input = <y> ) then
        call APPEND-TO-TEMPFILES
      endif
      prompt "another student?"
      if ( input = <n> ) then
        CONTINUE = false
      endif
    else if ( ONEFOUND = false )
      prompt "do you wish to continue complete function?"
      if ( input = <n> ) then
        CONTINUE = false
      endif
    else if ( DBERROR = true ) then
      print "strong error message - data base corrupted"
      CONTINUE = false
    endif
  endwhile
  call TRANSACT-THE-DATABASE
endif
**Pseudo Code for RETRIEVE-ONE-RECORD subfunction**

```
prompt "do you wish to retrieve name or S.S.N. (1/2)"
if (input = <1>) then
   call GET-ONE-RECORD-NAME
else
   call GET-ONE-RECORD-SSN
```
Pseudo Code for GET-ONE-RECORD-SSN subfunction:

DONESEARCH = false
while ( not DONESEARCH )
    prompt for S.S.N.
    search for the record in the data base
    if ( only 1 record found ) then
        call CONFIRM-CORRECT-RECORD
        if ( confirmed ) then
            ONEFOUND = true
            DONESEARCH = true
        else
            ONEFOUND = false
            prompt "do you wish to try another S.S.N.?
        if ( input = <n> ) then
            DONESEARCH = true
        endif
    endif
else if ( 0 records found )
    ONEFOUND = false
    output "that number not found in data base"
    prompt "do you wish to try another S.S.N."
    if ( input = <n> ) then
        DONESEARCH = true
    endif
else { more than one record with same ssn }
    DBERROR = true
    DONESEARCH = true
endif
endwhile
Pseudo Code for COMPLETE-STUDENT-INFO subfunction:

initialize necessary variables
field = 1
repeat ( until no more fields ) or ( esc )
  while (temparray fieldlength > 0) and (more fields)
    copy temparray[field] to fieldarray[field]
    field = field + 1
  endwhile
  ACCEPT = false
repeat ( until ACCEPT = true )
  prompt field name to complete
  read data
  test data
  if ( data = ok ) then
    ACCEPT = true
    copy temparray[field] to fieldarray[field]
    field = field + 1
  else if ( input = < esc > ) then
    ACCEPT = true
  else
    print error message, data not correct format
  endif
endrepeat
endrepeat
prompt "do you want to review?"
if ( input = <y> ) then
  DONEREVIEW = false
  while ( not DONEREVIEW )
    call REVIEW-DATA
    prompt "review again?"
    if ( input = <n> ) then
      DONEREVIEW = true
    endif
  endwhile
endif
MODIFY FUNCTION

Pseudo Code for MODIFY-STUDENT-RECORDS function:

prompt welcome message, "do you wish to continue?"
if ( input = <y> ) then
  DBERRPR = false
  ONEFOUND = false
  call RETRIEVE-ONE-RECORD
  if ( ONEFOUND = true ) then
    prompt "do you wish to modify the SSN?"
    if ( input = <y> ) then
      call REVIEW-SSN
    endif
  prompt "do you wish to modify other data?"
  if ( input = <y> ) then
    DONE = false
    while ( DONE = false )
      call REVIEW-DATA
      prompt "are all modifications finished?"
      if ( input = <y> ) then
        DONE = true
      endif
    endwhile
  prompt "add modifications to the data base?"
  if ( input = <y> ) then
    call APPEND-TO-TEMPFILES
  endif
endif
endif
if ( DBERROR = false ) then
  call ( TRANSACT-THE-DATABASE )
endif
WITHDRAW
CANCEL
STUDENTS

RETRIEVE
ONE
RECORD

C/W
DATA
INPUT

APPEND
TO TEMP
FILES

TRANSACT
THE DATA
BASE

GET ONE
RECORD
S.S.N.

GET ONE
RECORD
NAME

CONFIRM
CORRECT
RECORD

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
WITHDRAW/CANCEL FUNCTION

Pseudo Code for WITHDRAW/CANCEL-STUDENTS function

prompt welcome message "do you wish to continue?"
if ( input = <y> ) then
    ONEFOUND = false
    DBERROR = false
    call RETRIVE-ONE-RECORD
    if ( ONEFOUND = true ) and ( DBERROR = false ) then
        call C/W-DATA-INPUT
        if { some data entry has been done } then
            prompt "do you want to add this to the data base?"
            if ( input = <y> ) then
                call APPEND-TO-TEMP-FILE
            endif
        endif
    endif
endif
endif
if { a valid temp file has been created } then
    call TRANSACT-THE-DATA-BASE
endif
APPENDIX L

1022 CODE - THE 'UPDATE' MODEL
GLOBAL.DMD

GLOBAL DATA DICTIONARY FOR RESIDENCE HALLS DATA BASE

This data set contains the information which is university wide.

<table>
<thead>
<tr>
<th>ATTRIBUTE NAME</th>
<th>ABBREVIATION</th>
<th>TYPE</th>
<th>LENGTH</th>
<th>KEY</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATT SOCIAL SECURITY_NUMBER</td>
<td>ABB SSN</td>
<td>INTEGER</td>
<td>LEN 9</td>
<td>KEYED</td>
</tr>
<tr>
<td>ATT STUDENT LAST_NAME</td>
<td>ABB SLN</td>
<td>TEXT</td>
<td>LEN 20</td>
<td>KEYED</td>
</tr>
<tr>
<td>ATT STUDENT FIRST NAMES</td>
<td>ABB SFN</td>
<td>TEXT</td>
<td>LEN 20</td>
<td>KEYED</td>
</tr>
<tr>
<td>ATT STREET PERMANENT</td>
<td>ABB SRPER</td>
<td>TEXT</td>
<td>LEN 20</td>
<td></td>
</tr>
<tr>
<td>ATT CITY PERMANENT</td>
<td>ABB CIPER</td>
<td>TEXT</td>
<td>LEN 18</td>
<td></td>
</tr>
<tr>
<td>ATT STATE PERMANENT</td>
<td>ABB STAPR</td>
<td>TEXT</td>
<td>LEN 2</td>
<td></td>
</tr>
<tr>
<td>ATT ZIP_PERMANENT</td>
<td>ABB ZPER</td>
<td>INTEGER</td>
<td>LEN 9</td>
<td></td>
</tr>
<tr>
<td>ATT COUNTRY_PERMANENT</td>
<td>ABB COPER</td>
<td>TEXT</td>
<td>LEN 18</td>
<td></td>
</tr>
<tr>
<td>ATT PHONE_PERMANENT</td>
<td>ABB PPER</td>
<td>INTEGER</td>
<td>LEN 10</td>
<td></td>
</tr>
<tr>
<td>ATT PARENT LAST NAME</td>
<td>ABB PLN</td>
<td>TEXT</td>
<td>LEN 20</td>
<td></td>
</tr>
<tr>
<td>ATT PARENT FIRST NAMES</td>
<td>ABB PFN</td>
<td>TEXT</td>
<td>LEN 20</td>
<td></td>
</tr>
<tr>
<td>ATT STREET_PARENT</td>
<td>ABB STRPR</td>
<td>TEXT</td>
<td>LEN 20</td>
<td></td>
</tr>
<tr>
<td>ATT CITY_PARENT</td>
<td>ABB CIPAR</td>
<td>TEXT</td>
<td>LEN 18</td>
<td></td>
</tr>
<tr>
<td>ATT STATE_PARENT</td>
<td>ABB SAPAR</td>
<td>TEXT</td>
<td>LEN 2</td>
<td></td>
</tr>
<tr>
<td>ATT ZIP_PARENT</td>
<td>ABB ZPAR</td>
<td>INTEGER</td>
<td>LEN 9</td>
<td></td>
</tr>
<tr>
<td>ATT COUNTRY_PARENT</td>
<td>ABB COPAR</td>
<td>TEXT</td>
<td>LEN 18</td>
<td></td>
</tr>
<tr>
<td>ATT PHONE_PARENT</td>
<td>ABB PPAR</td>
<td>INTEGER</td>
<td>LEN 10</td>
<td></td>
</tr>
<tr>
<td>ATT ACADEMIC_STATUS</td>
<td>ABB AS</td>
<td>INTEGER</td>
<td>LEN 1</td>
<td></td>
</tr>
<tr>
<td>ATT DATE_OF_BIRTH</td>
<td>ABB DOB</td>
<td>DATE</td>
<td>LEN 8</td>
<td></td>
</tr>
<tr>
<td>ATT SEX</td>
<td>ABB S</td>
<td>TEXT</td>
<td>LEN 1</td>
<td></td>
</tr>
</tbody>
</table>

THE FOLLOWING 2 FIELDS ARE SYSTEM PROVIDED, i.e. READ ONLY

| ATT DATE_OF_ENTRY             | ABB DOE      | DATE OF ENTRY | LEN 8 |
| ATT INT_OF_IDENTIFICATION     | ABB IOI      | INTEGER OF IDENTIFICATION | LEN 4 |
LOCAL_DATA_DICTIONARY_FOR_RESIDENCE_HALLS_DATA_BASE

This data set contains the information which is specific to the residence halls.

<table>
<thead>
<tr>
<th>ATTRIBUTE_NAME</th>
<th>ABBREVIATION</th>
<th>TYPE</th>
<th>LENGTH</th>
<th>KEY</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATT SOCIAL SECURITY NUMBER</td>
<td>ABB SSN</td>
<td>INTEGER</td>
<td>LEN 9</td>
<td>KEYED</td>
</tr>
<tr>
<td>ATT STUDENT LAST NAME</td>
<td>ABB SLN</td>
<td>TEXT</td>
<td>LEN 20</td>
<td>KEYED</td>
</tr>
<tr>
<td>ATT STUDENT FIRST NAMES</td>
<td>ABB SFN</td>
<td>TEXT</td>
<td>LEN 20</td>
<td>KEYED</td>
</tr>
<tr>
<td>ATT STUDENT_STATUS</td>
<td>ABB STS</td>
<td>TEXT</td>
<td>LEN 1</td>
<td></td>
</tr>
<tr>
<td>ATT QUARTER_APPLIED_FOR</td>
<td>ABB QAF</td>
<td>TEXT</td>
<td>LEN 3</td>
<td></td>
</tr>
<tr>
<td>ATT MARITAL_STATUS</td>
<td>ABB MS</td>
<td>TEXT</td>
<td>LEN 1</td>
<td></td>
</tr>
<tr>
<td>ATT US CITIZEN</td>
<td>ABB UC</td>
<td>TEXT</td>
<td>LEN 1</td>
<td></td>
</tr>
<tr>
<td>ATT OTHER_CITIZENSHIP</td>
<td>ABB OC</td>
<td>TEXT</td>
<td>LEN 18</td>
<td></td>
</tr>
<tr>
<td>ATT PHYSICAL_LIMITATIONS</td>
<td>ABB PL</td>
<td>TEXT</td>
<td>LEN 1</td>
<td></td>
</tr>
<tr>
<td>ATT LIMITATIONS_CODE_NUM</td>
<td>ABB LCN</td>
<td>INTEGER</td>
<td>LEN 1</td>
<td></td>
</tr>
<tr>
<td>ATT FIRST HALL PREFERENCE</td>
<td>ABB FHP</td>
<td>TEXT</td>
<td>LEN 8</td>
<td></td>
</tr>
<tr>
<td>ATT SECOND HALL PREFERENCE</td>
<td>ABB SHP</td>
<td>TEXT</td>
<td>LEN 8</td>
<td></td>
</tr>
<tr>
<td>ATT VISITATION PREFERENCE</td>
<td>ABB VP</td>
<td>TEXT</td>
<td>LEN 2</td>
<td></td>
</tr>
<tr>
<td>ATT ROOM_TYPE_PREFERENCE</td>
<td>ABB RTP</td>
<td>TEXT</td>
<td>LEN 1</td>
<td></td>
</tr>
<tr>
<td>ATT RMmate_LAST_NAME</td>
<td>ABB RLN</td>
<td>TEXT</td>
<td>LEN 20</td>
<td></td>
</tr>
<tr>
<td>ATT RMmate_FIRST_NAMES</td>
<td>ABB RFN</td>
<td>TEXT</td>
<td>LEN 20</td>
<td></td>
</tr>
<tr>
<td>ATT RMmate_SOC_SEC_NUM</td>
<td>ABB RSSN</td>
<td>INTEGER</td>
<td>LEN 9</td>
<td></td>
</tr>
<tr>
<td>ATT MAJOR_FIELD</td>
<td>ABB MF</td>
<td>TEXT</td>
<td>LEN 4</td>
<td></td>
</tr>
<tr>
<td>ATT SMOKER_SELF</td>
<td>ABB SS</td>
<td>TEXT</td>
<td>LEN 1</td>
<td></td>
</tr>
<tr>
<td>ATT DRINKER_SELF</td>
<td>ABB DS</td>
<td>TEXT</td>
<td>LEN 1</td>
<td></td>
</tr>
<tr>
<td>ATT NIGHTOWL_SELF</td>
<td>ABB NS</td>
<td>TEXT</td>
<td>LEN 1</td>
<td></td>
</tr>
<tr>
<td>ATT SMOKER_ROOMMATE</td>
<td>ABB SR</td>
<td>TEXT</td>
<td>LEN 1</td>
<td></td>
</tr>
<tr>
<td>ATT DRINKER_ROOMMATE</td>
<td>ABB DR</td>
<td>TEXT</td>
<td>LEN 1</td>
<td></td>
</tr>
<tr>
<td>ATT NIGHTOWL_ROOMMATE</td>
<td>ABB NR</td>
<td>TEXT</td>
<td>LEN 1</td>
<td></td>
</tr>
<tr>
<td>ATT APP_PRIORITY_NUMBER</td>
<td>ABB APN</td>
<td>INTEGER</td>
<td>LEN 8</td>
<td>KEYED</td>
</tr>
<tr>
<td>ATT VALIDATION_NUMBER</td>
<td>ABB VN</td>
<td>TEXT</td>
<td>LEN 15</td>
<td></td>
</tr>
<tr>
<td>ATT APPLICATION_TYPE</td>
<td>ABB AT</td>
<td>TEXT</td>
<td>LEN 2</td>
<td></td>
</tr>
<tr>
<td>ATT PRE_PAYED</td>
<td>ABB PP</td>
<td>TEXT</td>
<td>LEN 1</td>
<td></td>
</tr>
<tr>
<td>ATT PRE_PAY_RECVD_BY</td>
<td>ABB PPRB</td>
<td>TEXT</td>
<td>LEN 3</td>
<td></td>
</tr>
<tr>
<td>ATT PRE_PAY_DATE</td>
<td>ABB PPD</td>
<td>DATE</td>
<td>LEN 8</td>
<td></td>
</tr>
<tr>
<td>ATT CANCEL_REFUND_AMOUNT</td>
<td>ABB CRA</td>
<td>REAL</td>
<td>LEN 6</td>
<td></td>
</tr>
<tr>
<td>ATT CANCEL_DATE</td>
<td>ABB CD</td>
<td>DATE</td>
<td>LEN 8</td>
<td></td>
</tr>
<tr>
<td>ATT WITHDRAW_ROOM_AMOUNT</td>
<td>ABB WRA</td>
<td>REAL</td>
<td>LEN 6</td>
<td></td>
</tr>
<tr>
<td>ATT WITHDRAW_SINGLE_AMOUNT</td>
<td>ABB WSA</td>
<td>REAL</td>
<td>LEN 6</td>
<td></td>
</tr>
<tr>
<td>ATT WITHDRAW_MEALPLAN_AMOUNT</td>
<td>ABB WMA</td>
<td>REAL</td>
<td>LEN 6</td>
<td></td>
</tr>
<tr>
<td>ATT WITHDRAW_DATE</td>
<td>ABB WD</td>
<td>DATE</td>
<td>LEN 8</td>
<td></td>
</tr>
<tr>
<td>ATT DATE_OF_ENTRY</td>
<td>ABB DOE</td>
<td>DATE OF ENTRY</td>
<td>LEN 8</td>
<td></td>
</tr>
<tr>
<td>ATT INT_OF_IDENTIFICATION</td>
<td>ABB IOI</td>
<td>INTEGER OF IDENTIFICATION</td>
<td>LEN 4</td>
<td></td>
</tr>
</tbody>
</table>

THE FOLLOWING 2 FIELDS ARE SYSTEM PROVIDED - i.e. READ ONLY

| ATT DATE_OF_ENTRY               | ABB DOE      | DATE OF ENTRY | LEN 8 |
| ATT INT_OF_IDENTIFICATION       | ABB IOI      | INTEGER OF IDENTIFICATION | LEN 4 |

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
SET PROMPT TEXT "1022> ".
LET SYSCASE EQ 1.        ! set to ignore case
!
OPEN GLOBAL LOCAL.
JOIN GLOBAL LOCAL SSN.
!
! Try to transact any data left-over in the '.TMP' sequential files
! into the data base, which would only occur if there was a system crash.
! Otherwise, the files will be NULL and this will have no effect.
! TRANSACT DATA GLOBAL.TMP DESC GLOBAL.DMD LOC SSN UNAPPLIED APPEND NOMSG.
! TRANSACT DATA LOCAL.TMP DESC LOCAL.DMD LOC SSN UNAPPLIED APPEND NOMSG.
!
TY "BE PATIENT, I'M SETTING UP THE DATA BASE." FMT / A / END.
USE PROC.DMC.
PLI022 START.

! INTEGER VARIABLES

LET SSN_POS EQ 1.  ! UNIQUE POSITION IN ARRAY FOR SSN
LET NUM_FEELDS EQ 53.  ! HOW MANY FIELDS
LET END_STUDENT_DATA EQ 41.  ! TO MARK END OF SECTION
LET END_OFFICIAL_DATA EQ 47.  ! ALL EXCEPT CANCEL + WITHDRAW INFO
LET MAX_TEMP_INFO EQ 100.  ! HOW MANY ALLOWED IN TEMP FILE
LET LN_PER_PAG EQ 11.  ! LINES IN A SCREENFUL
DEFINE INTEGER SCREEN.  ! WHICH SCREEN TO DISPLAY
DEFINE INTEGER LINE.  ! THE LINE TO INPUT
DEFINE INTEGER I.  ! COUNTING VARIABLE

! BOOLEAN VARIABLES

DEFINE INTEGER OVER.  ! SESSION IS OVER
DEFINE INTEGER OK.  ! ERROR CHECKING
DEFINE INTEGER IN_BUFF_OK.  ! INPUT LENGTH ERROR CHECKING
DEFINE INTEGER MAIN.  ! GOTO MAIN MENU
DEFINE INTEGER DONE.  ! LOOP TERMINATOR
DEFINE INTEGER AKCEPT_IT.  ! CORRECT INPUT?
DEFINE INTEGER VALID_TMP_FILE.  ! FLAG TO CAUSE TRANSACTION
DEFINE INTEGER DAMAGED.  ! DATA SET DAMAGE
DEFINE INTEGER ABORTED.  ! ESCAPE FROM A MODULE
DEFINE INTEGER DUPLICATE.  ! DUPLICATE SSN IN ADD
DEFINE INTEGER WAIT.  ! NEED TO WAIT IN COMPLETE_FEILDS
DEFINE INTEGER STOP.  ! LOOP TERMINATOR
LET TRU EQ 1.  ! DEFINE TRUE
LET FALS EQ 0.  ! DEFINE FALSE

! ARRAY VARIABLES - FIELD NAMES AND TYPES

DEFINE TEXT 50 FEELDS(53).  ! HOLD FIELD NAMES
DEFINE TEXT 1 FEELD_TYPE(53).  ! HOLD FIELD INPUT TYPE
DEFINE TEXT 20 FEELD_DATA(53).  ! HOLD INPUT DATA
DEFINE TEXT 20 TEMP_DATA(53).  ! HOLD INPUT DATA TEMPS
DEFINE INTEGER FEELD_LEN(53).  ! HOLD LENGTH OF EACH FIELD
DEFINE TEXT 20 TEMP_SSN.  ! HOLD SSN FOR SEARCHING DB
DEFINE TEXT 20 SAVE_SSN.  ! HOLD SSN FOR CONFIRMING MODIFICATION

! TEMPORARY ARRAY FOR INFO DURING UPDATE TO ALLOW SSN CHECKING
! HOLD SSN + NAME OF STUDENTS STORED IN TEMP SEQUENTIAL FILE

DEFINE TEXT 50 TEMP_INFO(100).
! TEMPORARY ARRAYS FOR CHECKING ENTRY CORRECTNESS

DEFINE TEXT 200 IN_BUFFER. ! INPUT BUFFER
DEFINE TEXT 20 CHECK_BUFFER. ! DATA CHECKING BUFFER
DEFINE TEXT 20 TEMP_BUFFER. ! TEMP DATA CHECKING BUFFER
DEFINE TEXT 20 TEMP_ANSWER. ! ACCEPT REPLACEMENTS

! TEMPORARY VARIABLES TO CHECK DATES, ETC. WHEN ENTERED

DEFINE INTEGER TEMP_INT. ! CHECK INTEGERS
DEFINE REAL TEMP_REAL. ! CHECK REALS
DEFINE DATE TEMP_DATE. ! CHECK DATES

! TEXT - RESPONSE FROM USERS

DEFINE TEXT 20 CMD. ! HOLD COMMANDS
DEFINE TEXT 20 ANSWER. ! HOLD ANSWERS
DEFINE TEXT 1 ESCAPE_CHAR. ! THE ESCAPE CHARACTER
   LET ESCAPE_CHAR EQ "/".

! ERROR BEEP

DEFINE TEXT 1 BEEP. ! BEEP CHARACTER
   LET BEEP EQ $CHAR(7).

! SCREEN CONTROL

DEFINE TEXT 2 CLEAR_SCRN.
   LET CLEAR_SCRN EQ "\$E". ! CLEAR SCREEN CODE

! RECORD DELIMITER FOR TEMP FILES

DEFINE TEXT 2 CARRIAGE_RETURN.
   LET CARRIAGE_RETURN EQ $CHAR(13) + $CHAR(10).
!MAIN:

! MAIN CODE LOOP
!
CALL INIT_FIELD_NAMES.
LET OVER EQ FALS.
LET OK EQ TRU.
REPEAT
TY CLEAR_SCRN.
IF OK EQ FALS THEN
  TY "NO SUCH COMMAND - TRY AGAIN" FMT / A / END.
ENDIF.
CALL MAIN_MENU.
TY "ENTER COMMAND:" FMT / A X $ END.
CALL GET_IN_BUFFER.
LET CMD EQ $LEFT(IN_BUFFER, 20).
LET MAIN EQ FALS. ! STAY IN THIS REPEAT LOOP
LET OK EQ TRU.
IF TRIM(CMD) EQ "U" THEN
  CALL UPDATE.
ELSEIF TRIM(CMD) EQ "A" THEN
  CALL ASSIGN_ROOMS.
ELSEIF TRIM(CMD) EQ "R" THEN
  CALL REPORTS.
ELSEIF TRIM(CMD) EQ "I" THEN
  CALL INQUIRY.
ELSEIF TRIM(CMD) EQ "T" THEN
  CALL TOOLS.
ELSEIF (TRIM(CMD) EQ "Q") OR (TRIM(CMD) EQ ESCAPE_CHAR) THEN
  LET OVER EQ TRU.
ELSE
  LET OK EQ FALS.
ENDIF.
UNTIL OVER EQ TRU.
CALL CLEAN_UP.

! END OF MAIN CODE LOOP
!
PL1022 STOP.
INIT_FIELD_NAMES:

! BEGIN STUDENT DATA SECTION

! REMEMBER 'SSN_POS' EQ 1
LET FEELD(1)EQ "SOCIAL SECURITY NUMBER:".
LET FEELD_TYPE(1)EQ "S".
LET FEELD_LEN(1)EQ 9.
LET FEELD(2)EQ "STUDENT LAST NAME:".
LET FEELD_TYPE(2)EQ "T".
LET FEELD_LEN(2)EQ 20.
LET FEELD(3)EQ "STUDENT FIRST NAMES:".
LET FEELD_TYPE(3)EQ "T".
LET FEELD_LEN(3)EQ 20.
LET FEELD(4)EQ "STREET-PERMANENT:".
LET FEELD_TYPE(4)EQ "T".
LET FEELD_LEN(4)EQ 20.
LET FEELD(5)EQ "CITY-PERMANENT:".
LET FEELD_TYPE(5)EQ "T".
LET FEELD_LEN(5)EQ 18.
LET FEELD(6)EQ "STATE-PERMANENT:".
LET FEELD_TYPE(6)EQ "T".
LET FEELD_LEN(6)EQ 2.
LET FEELD(7)EQ "ZIP-PERMANENT:".
LET FEELD_TYPE(7)EQ "T".
LET FEELD_LEN(7)EQ 9.
LET FEELD(8)EQ "COUNTRY-PERMANENT:".
LET FEELD_TYPE(8)EQ "T".
LET FEELD_LEN(8)EQ 18.
LET FEELD(9)EQ "PHONE-PERMANENT:".
LET FEELD_TYPE(9)EQ "P".
LET FEELD_LEN(9)EQ 10.
LET FEELD(10)EQ "PARENT LAST NAME:".
LET FEELD_TYPE(10)EQ "T".
LET FEELD_LEN(10)EQ 20.
LET FEELD(11)EQ "PARENT FIRST NAMES:".
LET FEELD_TYPE(11)EQ "T".
LET FEELD_LEN(11)EQ 20.
LET FEELD(12)EQ "STREET-PARENT (" TO DUPLICATE):".
LET FEELD_TYPE(12)EQ "T".
LET FEELD_LEN(12)EQ 20.
LET FEELD(13)EQ "CITY-PARENT:".
LET FEELD_TYPE(13)EQ "T".
LET FEELD_LEN(13)EQ 18.
LET FEELD(14)EQ "STATE-PARENT:".
LET FEELD_TYPE(14)EQ "T".
LET FEELD_LEN(14)EQ 2.
LET FEELD(15)EQ "ZIP-PARENT:".
LET FEELD_TYPE(15)EQ "T".
LET FEELD_LEN(15)EQ 9.
LET FEELD_TYPE(33) EQ "T".
LET FEELD_LEN(33) EQ 20.
LET FEELDS(34) EQ "RMMATE SOC SEC NUM:".
LET FEELD_TYPE(34) EQ "S".
LET FEELD_LEN(34) EQ 9.
LET FEELDS(35) EQ "MAJOR FIELD:".
LET FEELD_TYPE(35) EQ "T".
LET FEELD_LEN(35) EQ 4.
LET FEELDS(36) EQ "SMOKER-SELF ( Y or N ):".
LET FEELD_TYPE(36) EQ "Y".
LET FEELD_LEN(36) EQ 1.
LET FEELDS(37) EQ "DRINKER-SELF ( Y or N ):".
LET FEELD_TYPE(37) EQ "Y".
LET FEELD_LEN(37) EQ 1.
LET FEELDS(38) EQ "NIGHTOWL-SELF ( Y or N ):".
LET FEELD_TYPE(38) EQ "Y".
LET FEELD_LEN(38) EQ 1.
LET FEELDS(39) EQ "SMOKER-RMMATE ( Y or N ):".
LET FEELD_TYPE(39) EQ "Y".
LET FEELD_LEN(39) EQ 1.
LET FEELDS(40) EQ "DRINKER-RMMATE ( Y or N ):".
LET FEELD_TYPE(40) EQ "Y".
LET FEELD_LEN(40) EQ 1.
LET FEELDS(41) EQ "NIGHTOWL-RMMATE ( Y or N ):".
LET FEELD_TYPE(41) EQ "Y".
LET FEELD_LEN(41) EQ 1.

! END STUDENT DATA SECTION

! BEGIN OFFICIAL DATA SECTION

LET FEELDS(42) EQ "APP.PRIORITY NUMBER:".
LET FEELD_TYPE(42) EQ "N".
LET FEELD_LEN(42) EQ 8.
LET FEELDS(43) EQ "VALIDATION NUMBER:".
LET FEELD_TYPE(43) EQ "T".
LET FEELD_LEN(43) EQ 15.
LET FEELDS(44) EQ "APPLICATION TYPE ( UM, VT, VR, DE ):".
LET FEELD_TYPE(44) EQ "C".
LET FEELD_LEN(44) EQ 2.
LET FEELDS(45) EQ "PREPAYED $100.00 ( Y or N ):".
LET FEELD_TYPE(45) EQ "Y".
LET FEELD_LEN(45) EQ 1.
LET FEELDS(46) EQ "PREPAY RECVD BY:".
LET FEELD_TYPE(46) EQ "T".
LET FEELD_LEN(46) EQ 3.
LET FEELDS(47) EQ "PREPAY DATE:".
LET FEELD_TYPE(47) EQ "D".
LET FEELD_LEN(47) EQ 8.

! END OFFICIAL DATA SECTION
! BEGIN CANCEL/WITHDRAW DATA SECTION
!
 LET FEELDS(48) EQ "CANCEL REFUND AMOUNT: $".
 LET FEELD_TYPE(48) EQ "$".
 LET FEELD_LEN(48) EQ 6.
 LET FEELDS(49) EQ "CANCEL DATE: ".
 LET FEELD_TYPE(49) EQ "D".
 LET FEELD_LEN(49) EQ 8.
 LET FEELDS(50) EQ "WITHDRAW ROOM AMOUNT: $".
 LET FEELD_TYPE(50) EQ "$".
 LET FEELD_LEN(50) EQ 6.
 LET FEELDS(51) EQ "WITHDRAW SINGLE AMOUNT: $".
 LET FEELD_TYPE(51) EQ "$".
 LET FEELD_LEN(51) EQ 6.
 LET FEELDS(52) EQ "WITHDRAW MEALPLAN AMOUNT: $".
 LET FEELD_TYPE(52) EQ "$".
 LET FEELD_LEN(52) EQ 6.
 LET FEELDS(53) EQ "WITHDRAWAL DATE: ".
 LET FEELD_TYPE(53) EQ "D".
 LET FEELD_LEN(53) EQ 8.
!
 ! NUM_FEELDS IS 53
!
 RETURN.
UPDATE:

!  !  !
LET OK EQ TRU.    ! START OUT OK
REPEAT
    TY CLEAR_SCRN.
    IF OK EQ FALS THEN
        TY "NO SUCH COMMAND - TRY AGAIN" FMT / A / END.
    ENDIF.
    CALL UPDATE_MENU.
    TY " ENTER COMMAND:" FMT / A X $ END.
    CALL GET_IN_BUFFER.
    LET CMD EQ $LEFT(IN_BUFFER, 20).
    LET MAIN EQ FALS.    ! DON'T GOTO MAIN UNTIL TOLD TO
    !
    IF $TRIM(CMD) EQ "A" THEN
        CALL ADD_RECORDS.
    ELSEIF $TRIM(CMD) EQ "C" THEN
        CALL COMPLETE_RECORDS.
    ELSEIF $TRIM(CMD) EQ "M" THEN
        CALL MODIFY_RECORD.
    ELSEIF $TRIM(CMD) EQ "D" THEN
        CALL DELETE_RECORD.
    ELSEIF $TRIM(CMD) EQ "U" THEN
        CALL UNDELETE_RECORD.
    ELSEIF (STRIM(CMD) EQ "C") OR ($TRIM(CMD) EQ "W") THEN
        CALL CANCEL_WITHDRAW.
    ELSEIF (STRIM(CMD) EQ "B") OR ($TRIM(CMD) EQ ESCAPE_CHAR) THEN
        LET MAIN EQ TRU.
    ELSE
        LET OK EQ FALS.
    ENDIF.
UNTIL MAIN EQ TRU.
RETURN.
ADD_RECORDS:

!  ADD RECORD SECTION
!
TY CLEAR_SCRN.
TY "******** ENTERING ADD RECORDS ********" FMT / A / END.
TY "DO YOU WISH TO CONTINUE? ([Y] or N):" FMT A X S END.
LET VALID_TMP_FILE EQ FALS.  !  SET TRANSACTION FLAG
CALL GET_IN_BUFFER.
LET ANSWER EQ $LEFT(IN_BUFFER, 20).
IF ($CAPS($TRIM(ANSWER)) EQ "Y") OR -
   ($LEN($TRIM(ANSWER)) EQ 0) THEN
   LET CONTINUE EQ TRU.
   INIT 2 GLOBAL_TMP.
   INIT 3 LOCAL_TMP.
   CALL INIT_TEMP_INFO.
ELSE
   LET CONTINUE EQ FALS.
ENDIF.
WHILE CONTINUE EQ TRU DO
   CALL INIT_VARIABLES.
   CALL GET_STUDENT_INFO.
   IF ABORTED EQ FALS THEN
      TY "SHALL THIS RECORD BE ADDED? ([Y] or N):" FMT / A X S END.
      CALL GET_IN_BUFFER.
      LET ANSWER EQ $LEFT(IN_BUFFER, 20).
      IF ($CAPS($TRIM(ANSWER)) EQ "Y") OR -
          ($LEN($TRIM(ANSWER)) EQ 0) THEN
          CALL MANAGE_TMP_FILE.
          CALL MANAGE_TEMP_INFO.
          LET VALID_TMP_FILE EQ TRU.
      ENDIF.
      TY "ADD ANOTHER STUDENT? ([Y] or N):" FMT / A X S END.
      CALL GET_IN_BUFFER.
      LET ANSWER EQ $LEFT(IN_BUFFER, 20).
      IF $CAPS($TRIM(ANSWER)) EQ "N" THEN
         LET CONTINUE EQ FALS.
      ENDIF.
      ELSEIF ABORTED EQ TRU THEN
         LET CONTINUE EQ FALS.
      ENDIF.
   ENDWHILE.
   RELEASE 2.
   RELEASE 3.
   IF VALID_TMP_FILE EQ TRU THEN
      CALL TRANSACT_INTO_DATA_BASE.
   ENDIF.
RETURN.
COMPLETE_RECORDS:

!  COMPLETE RECORD SECTION
!
TY CLEAR_SCRN.
TY "***** ENTERING COMPLETE RECORDS *****" FMT / A END.
TY "DO YOU WISH TO CONTINUE? ([Y] or N):" FMT / A X $ END.
LET VALID_TMP_FILE EQ FALS. ! SET TRANSACTION FLAG
CALL GET_IN_BUFFER.
LET ANSWER EQ $LEFT(IN_BUFFER, 20).
IF ($CAPS($TRIM(ANSWER)) EQ "Y") OR -
  ($LEN($TRIM(ANSWER)) EQ 0 ) THEN ! CONTINUE
    LET CONTINUE EQ TRU.
    INIT 2 GLOBAL_TMP.
    INIT 3 LOCAL_TMP.
    CALL INIT_TEMP_INFO.
ELSE
    LET CONTINUE EQ FALS.
ENDIF.
WHILE CONTINUE EQ TRU DO
  CALL INIT_VARIABLES.
  CALL GET_RECORD.
  CALL INIT_VARIABLES.
  CALL GET_RECORD.
  IF ( SYSNREC EQ 1) AND ( ABORTED EQ FALS ) THEN
    CALL COMPLETE_FIELDS.
    TY "ADD THIS COMPLETED INFORMATION? ([Y] or N):" FMT / A X $ END.
    CALL GET_IN_BUFFER.
    LET ANSWER EQ $LEFT(IN_BUFFER, 20).
    IF ($CAPS($TRIM(ANSWER)) EQ "Y") OR -
      ($LEN($TRIM(ANSWER)) EQ 0 ) THEN
      CALL MANAGE_TMP_FILE.
      CALL MANAGE_TEMP_INFO.
      LET VALID_TMP_FILE EQ TRU.
    ENDIF.
    TY "COMPLETE ANOTHER STUDENT RECORD? ([Y] or N):" FMT / A X $ END.
    CALL GET_IN_BUFFER.
    LET ANSWER EQ $LEFT(IN_BUFFER, 20).
    IF ($CAPS($TRIM(ANSWER)) EQ "N") THEN
      LET CONTINUE EQ FALS.
    ELSE
      LET CONTINUE EQ TRU.
    ENDIF.
  ELSEIF ABORTED EQ TRU THEN
    LET CONTINUE EQ FALS.
  ELSEIF SYSNREC GT 1 THEN
    TY "DATA BASE IS CORRUPTED - GET HELP!!!" .
    !
    PL1022 STOP.
    !
  ENDIF.
ENDWHILE.
RELEASE 2.
RELEASE 3.
IF VALID_TMP_FILE EQ TRU THEN
    CALL TRANSACT_INTO_DATA_BASE.
ENDIF.
RETURN.
GET_STUDENT_INFO:
!
! RETURNS 'ABORTED' EQ 'TRU' IF TERMINATED
!
CALL INITIAL_DATA_INPUT.
IF ABORTED EQ FALS THEN
  TY "DO YOU WANT TO REVIEW? ( Y or [N] ):" FMT / A X $ END.
  CALL GET_IN_BUFFER.
  LET ANSWER EQ $LEFT(IN_BUFFER, 20).
  IF $CAPS($TRIM(ANSWER)) EQ "Y" THEN
    TY "SOCIAL SECURITY NUMBER =" FMT / A X $ END.
    TY $$SUBSTR(FEELD_DATA(SSN_POS), 1, 3) FMT A "-" $ END.
    TY $$SUBSTR(FEELD_DATA(SSN_POS), 4, 5) FMT A "-" $ END.
    TY $$SUBSTR(FEELD_DATA(SSN_POS), 6, 9) FMT A END.
    TY "IS THIS CORRECT? ([Y] or N):" FMT A X $ END.
    CALL GET_IN_BUFFER.
    LET ANSWER EQ $LEFT(INBUFFER, 20).
    IF $CAPS($TRIM(ANSWER)) EQ "N" THEN
      ! CAN'T ABORT - RESTORES OLD SSN
      CALL REVISE_SOC_SEC_NUM.
    ENDIF.
    CALL REVIEW_DATA.
  ENDIF.
ENDIF.
RETURN.
INITIAL_DATA_INPUT:

! RETURNS 'ABORTED' EQ 'TRU' IF ABORTED
!
LET ABORTED EQ FALS.
LET DUPLICATE EQ FALS.
LET LINE EQ 1.
!
GET THE SOCIAL SECURITY NUMBER

REPEAT
    TY FEELDS (LINE) FMT / A X $ END.
    CALL GET_IN_BUFFER.
    LET TEMP_DATA (LINE) EQ $LEFT (IN_BUFFER, 20).
    IF $TRIM (TEMP_DATA (LINE)) NE ESCAPE_CHAR THEN
        CALL VALIDATE_SOC_SEC_NUM.
        IF ABORTED EQ FALS THEN
            CALL CHECK_TEMP_INFO.
            IF DUPLICATE EQ TRU THEN
                TY ESCAPE_CHAR -
                    FMT "PLEASE TRY AGAIN, OR TYPE '' A '' TO ESCAPE" END.
            ENDIF.
        ENDIF.
    ELSE
        LET ABORTED EQ TRU.
    ENDF.
ENDIF.

ELSE
    TY ESCAPE_CHAR
    LET ABORTED EQ TRU.
ENDIF.

UNTIL (DUPLICATE EQ FALS) OR (ABORTED EQ TRU).

IF (DUPLICATE EQ FALS) AND (ABORTED EQ FALS) THEN

GET THE REST OF THE DATA

LET LINE EQ LINE + 1.
LET DONE EQ FALS.
REPEAT
    TY FEELDS (LINE) FMT A X $ END.
    CALL GET_IN_BUFFER.
    LET TEMP_DATA (LINE) EQ $LEFT (IN_BUFFER, 20).
    LET AKCEPT_IT EQ TRU.
    IF $TRIM (TEMP_DATA (LINE)) EQ ESCAPE_CHAR THEN
        LET TEMP_DATA (LINE) EQ $NULL.
        LET DONE EQ TRU.
    ELSEIF $LEN ($TRIM (TEMP_DATA (LINE))) NE 0 THEN
        CALL AKCEPT_DATA.
    ENDF.
ENDIF.

UNTIL AKCEPT_IT EQ TRU.

UNTIL (LINE EQ END_STUDENT_DATA) OR (DONE EQ TRU).

TY "OFFICIAL DATA ENTRY SECTION" FMT / 8X A END.

LET LINE EQ END_STUDENT_DATA. ! TO BEGIN THE NEXT SECTION

REPEAT
    TY FEELDS (LINE) FMT A X $ END.
    CALL GET_IN_BUFFER.
LET TEMP_DATA(LINE) EQ $LEFT(IN_BUFFER, 20).
LET AKCEPT_IT EQ TRU.
IF $STRIM( TEMP_DATA(LINE) ) EQ ESCAPE_CHAR THEN
  LET TEMP_DATA(LINE) EQ $NULL.
  LET DONE EQ TRU.
ELSEIF $LEN($STRIM(TEMP_DATA(LINE))) NE 0 THEN
  CALL AKCEPT_DATA.
ENDIF.
UNTIL AKCEPT_IT EQ TRU.
UNTIL (LINE EQ END_OFFICIAL_DATA) OR (DONE EQ TRU).
ENDIF.
RETURN.
REVISE_SOC_SEC_NUM:

! 
! 
! LET DONE EQ FALS.
LET ABORTED EQ FALS.
LET DUPLICATE EQ FALS.
LET TEMP_SSN EQ FEELD_DATA(SSN_POS).
TY ESCAPE_CHAR -
    FMT "RE-ENTER OR TYPE \" A \" TO ESCAPE" END.
TY "SOCIAL SECURITY NUMBER:" FMT / A X $ END.
CALL GET_IN_BUFFER.
LET TEMP_DATA(SSN_POS) EQ $LEFT(IN_BUFFER, 20).
IF $TRIM(TEMP_DATA(SSN_POS)) NE ESCAPE_CHAR THEN
    REPEAT
        CALL VALIDATE_SOC_SEC_NUM.
        IF ABORTED EQ FALS THEN
            CALL CHECK_TEMP_INFO.
            IF DUPLICATE EQ TRU THEN
                TY "PLEASE TRY AGAIN" FMT / A / END.
            ELSE
                TY "SOCIAL SECURITY NUMBER =" FMT / A X $ END.
                TY $SUBSTR(FEELD_DATA(SSN_POS), 1, 3) FMT A "-" $ END.
                TY $SUBSTR(FEELD_DATA(SSN_POS), 4, 5) FMT A "-" $ END.
                TY $SUBSTR(FEELD_DATA(SSN_POS), 6, 9) FMT A END.
                TY "IS THIS SATISFACTORY? (Y or [N]):" FMT / A X $ END.
                CALL GET_IN_BUFFER.
                LET ANSWER EQ $LEFT(IN_BUFFER, 20).
                IF $CAPS($TRIM(ANSWER)) EQ "Y" THEN
                    LET DONE EQ TRU.
                ELSE
                    TY ESCAPE_CHAR -
                        FMT "RE-ENTER OR TYPE \" A \" TO ESCAPE" END.
                    TY "SOCIAL SECURITY NUMBER:" FMT / A X $ END.
                    CALL GET_IN_BUFFER.
                    LET TEMP_DATA(SSN_POS) EQ $LEFT(IN_BUFFER, 20).
                END
IF $TRIM(TEMP_DATA(SSN_POS)) EQ ESCAPE_CHAR THEN
    LET ABORTED EQ TRU.
ENDIF.
ELSEIF (DONE EQ TRU) OR (ABORTED EQ TRU).
ENDIF.
UNTIL (DONE EQ TRU) OR (ABORTED EQ TRU).
ENDIF.
IF (ABORTED EQ TRU) THEN
    LET FEELD_DATA(SSN_POS) EQ TEMP__SSN.
    LET ABORTED EQ FALS.
    TY "SOCIAL SECURITY NUMBER CHANGE ABORTED." FMT / A END.
    TY "RESTORING ORIGINAL VALUE:" FMT X $ END.
    TY $SUBSTR(FEELD_DATA(SSN_POS), 1, 3) FMT A "-" $ END.
    TY $SUBSTR(FEELD_DATA(SSN_POS), 4, 5) FMT A "-" $ END.
    TY $SUBSTR(FEELD_DATA(SSN_POS), 6, 9) FMT A END.
    TY "TYPE <RETURN> TO CONTINUE" FMT / A END.
    ACCEPT ANSWER.
ENDIF.
RETURN.
**REVIEW_DATA:**

!
!
LET DONE EQ FALS.
LET SCREEN EQ 1.
REPEAT
  TY CLEAR_SCRN.
  TY SCREEN_FMT 65X "< SCREEN #" A " >" END.
  LET LINE EQ ((SCREEN * LN_PER_PAG) - LN_PER_PAG).
  IF LINE EQ 0 THEN
    LET LINE EQ 1. ! WILL SKIP SSN WHEN LINE IS INCREMENTED
  ENDIF.
  WHILE LINE LT (SCREEN * LN_PER_PAG) AND -
    (LINE LT END_OFFICIAL_DATA) DO
    IF LINE EQ END_STUDENT_DATA THEN
      TY "OFFICIAL DATA ENTRY SECTION"_FMT / 8X A END.
      TY "------------------------------"_FMT 8X A / END.
  ENDIF.
  LET LINE EQ LINE + 1.
  TY FEELDS(LINE)_FMT A " = " $ END.
  TY $TRIM(TEMP_DATA(LINE)).
ENDWHILE.
  TY "------------------------------------------------------------------------".
  TY " 'Y' or <RETURN> ==> YES, GO TO NEXT SCREEN"_FMT / A END.
  TY " 'N' ==> NO, NEED TO MODIFY FIELD/S"_FMT A END.
  TY "ESCAPE_CHAR_FMT "'" A "' "'" ==> DONE REVIEWING DATA" END.
  TY "ENTER CHOICE:"_FMT / A "$ END.
  CALL GET_IN_BUFFER.
  LET ANSWER EQ $LEFT(IN_BUFFER, 20).
  IF $TRIM(ANSWER) EQ ESCAPE_CHAR THEN
    LET DONE EQ TRU.
  ELSEIF $TRIM(ANSWER) NE 'N' THEN
    LET SCREEN EQ SCREEN + 1.
    ! TEST TO END IF SCREEN IS INCREMENTED BEYOND MAXIMUM
    IF SCREEN GT -
      ((END_OFFICIAL_DATA + (LN_PER_PAGE - 1)) / (LN_PER_PAGE)) THEN
      LET SCREEN EQ 1. ! WRAP AROUND TO THE FIRST SCREEN
    ENDIF.
  ELSE ! ANSWER EQUAL "N"
    TY CLEAR_SCRN.
    LET LINE EQ (SCREEN * LN_PER_PAG) - LN_PER_PAG.
    IF LINE EQ 0 THEN
      LET LINE EQ 1. ! WILL SKIP SSN WHEN LINE IS INCREMENTED
    ENDIF.
    REPEAT
      LET LINE EQ LINE + 1.
      TY $TRIM(FEELDS(LINE))_FMT / A " = " $ END.
      TY $TRIM(TEMP_DATA(LINE))_FMT A END.
      TY "REPLACEMENT?:"_FMT A "$ END.
      CALL GET_IN_BUFFER.
      LET TEMP_ANSWER EQ $LEFT(IN_BUFFER, 20).
IF $TRIM(TEMP\_ANSWER) EQ ESCAPE\_CHAR THEN
  ! DO NOTHING, LOOP WILL TERMINATE AT TEST
ELSEIF $LEN($TRIM(TEMP\_ANSWER)) NE 0 THEN
  LET TEMP\_DATA(LINE) EQ TEMP\_ANSWER.
  LET AKCEPT\_IT EQ TRU.
  CALL AKCEPT\_DATA.
  IF AKCEPT\_IT EQ FALS THEN
    LET TEMP\_DATA(LINE) EQ FEELD\_DATA(LINE).
    LET LINE EQ LINE - 1.
  ENDIF.
ELSE
  TY "ITEM UNCHANGED".
ENDIF.
ENDIF.
UNTIL (LINE GE (LN\_PER\_PAGE * SCREEN)) -
   OR ($TRIM(TEMP\_ANSWER) EQ ESCAPE\_CHAR) -
   OR (LINE EQ END\_OFFICIAL\_DATA).
ENDIF.
UNTIL DONE EQ TRU.
RETURN.
MANAGE_TMP_FILE:

! THIS PROCEDURE ADDS THE DATA IN THE FEELD_DATA ARRAY
! TO TWO TEMP FILES. THESE FILES ARE LATER TRANSACTED
! INTO THE DATA BASE FILES.
! CALLED FROM ADD AND COMPLETE SECTIONS OF UPDATE.

LET LINE EQ 0.
WHILE LINE LT 3 DO
  LET LINE EQ LINE + 1.
  PRINT ON 2 $LEFT(FEELD_DATA(LINE),FEELD_LEN(LINE)) FMT G $ END.
  PRINT ON 3 $LEFT(FEELD_DATA(LINE),FEELD_LEN(LINE)) FMT G $ END.
ENDWHILE.
WHILE LINE LT 18 DO
  LET LINE EQ LINE + 1.
  PRINT ON 2 $LEFT(FEELD_DATA(LINE),FEELD_LEN(LINE)) FMT G $ END.
ENDWHILE.
WHILE LINE LT 20 DO
  LET LINE EQ LINE + 1.
  PRINT ON 3 $LEFT(FEELD_DATA(LINE),FEELD_LEN(LINE)) FMT G $ END.
ENDWHILE.
LET LINE EQ 21.
PRINT ON 2 $LEFT(FEELD_DATA(LINE),FEELD_LEN(LINE)) FMT G $ END.
LET LINE EQ 22.
PRINT ON 3 $LEFT(FEELD_DATA(LINE),FEELD_LEN(LINE)) FMT G $ END.
LET LINE EQ 23.
PRINT ON 2 $LEFT(FEELD_DATA(LINE),FEELD_LEN(LINE)) FMT G $ END.
WHILE LINE LT NUM_FEELDS DO
  LET LINE EQ LINE + 1.
  PRINT ON 3 $LEFT(FEELD_DATA(LINE),FEELD_LEN(LINE)) FMT G $ END.
ENDWHILE.
PRINT ON 2 CARRIAGE_RETURN FMT A $ END.
! FORCE IN CARRIAGE RETURN TO DELIMIT RECORD
PRINT ON 3 CARRIAGE_RETURN FMT A $ END.
! FORCE IN CARRIAGE RETURN TO DELIMIT RECORD
RETURN.
MANAGE_TEMP_INFO:

!     ADD STUDENT TO TEMP_INFO ARRAY
!
LET I EQ 1.
WHILE TEMP_INFO(I) NE $BLANKS(50) DO
   LET I EQ I + 1.
ENDWHILE.
LET TEMP_INFO(I) EQ -
   $INSERT($LEFT(FIELD_DATA(SSN_POS), FIELD_LEN(SSN_POS)),-
                TEMP_INFO(I), 1)
LET TEMP_INFO(I) EQ -
   $INSERT($LEFT(FIELD_DATA(2), FIELD_LEN(2)),-
                TEMP_INFO(I), 1 + FIELD_LEN(SSN_POS))
LET TEMP_INFO(I) EQ -
   $INSERT($LEFT(FIELD_DATA(3), FIELD_LEN(3)),-
                TEMP_INFO(I), 1 + FIELD_LEN(SSN_POS) + FIELD_LEN(2))
RETURN.
TRANSACT INTO DATA BASE:

! TRANSACT THE TEMP FILES INTO THE TWO DATA SETS.
! CALLED FROM ADD AND COMPLETE SECTIONS OF UPDATE.
!
! UPON FINDING DAMAGE TO THE DATA BASE, THIS MODULE WILL
! PL1022 STOP
!
FILE COPY GLOBAL.DMS TO GLOBAL.BAK.
FILE COPY LOCAL.DMS TO LOCAL.BAK.
LET DAMAGED EQ FALS.
TY "DATA BASE BEING UPDATED, PLEASE BE PATIENT." FMT / A END.
DBSET GLOBAL.DMS.
IF SYSDAMAGE EQ 0 THEN
  TRANSACT DATA GLOBAL.TMP DESC GLOBAL.DMD LOC SSN
  UNAPPLIED APPEND NOMSG.
  IF SYSDAMAGE EQ 0 THEN
    DBSET LOCAL.DMS.
    IF SYSDAMAGE EQ 0 THEN
      TRANSACT DATA LOCAL.TMP DESC LOCAL.DMD LOC SSN
      UNAPPLIED APPEND NOMSG.
      IF SYSDAMAGE EQ 0 THEN
        TY "DATA BASE HAS BEEN SUCCESSFULLY UPDATED" FMT / A END.
        ELSE
          LET DAMAGED EQ TRU.
          TY "LOCAL DATA SET DAMAGED AFTER TRANSACTION" FMT / A END.
        ENDIF.
        ELSE
          LET DAMAGED EQ TRU.
          TY "LOCAL DATA SET DAMAGED BEFORE TRANSACTION" FMT / A END.
        ENDIF.
      ELSE
        LET DAMAGED EQ TRU.
        TY "GLOBAL DATA SET DAMAGED AFTER TRANSACTION" FMT / A END.
      ENDIF.
    ELSE
      LET DAMAGED EQ TRU.
      TY "GLOBAL DATA SET DAMAGED BEFORE TRANSACTION" FMT / A END.
    ENDIF.
  ELSE
    IF DAMAGED EQ FALS THEN
      FILE DELETE GLOBAL.BAK.
      FILE DELETE LOCAL.BAK.
      FILE DELETE GLOBAL.TMP.
      FILE DELETE LOCAL.TMP.
    ELSE
      TY CLEAR_SCRN.
      TY "EXTREME WARNING - DAMAGE TO THE DATA BASE!" FMT / A END.
      TY "YOU ARE ABOUT TO BE FORCED TO QUIT 1022" FMT / A END.
      TY "GET HELP, THE NEXT LINE YOU TYPE WILL BE YOUR LAST!"
      FMT / A END.
    ENDIF.
  ELSE
    TY "TYPE <RETURN> TO DIE" FMT / A X $ END.
    ACCEPT ANSWER.
  ENDIF.
ELSE
PL1022 STOP. !FORCE RETURN TO INTERACTIVE LEVEL

ENDIF.
DBSET GLOBAL.DMS.
RETURN.
GET_RECORD:

! GET A STUDENT RECORD FROM THE DQA BASE
! FILE. CALLED FROM COMPLETE SECTION OF UPDATE.
! RETURNS 'ABORTED' EQ 'TRU' IF USER ABORTS.
! MOVES INFO INTO TEMP_DATA ARRAY.

REPEAT

REPEAT

TY "SOCIAL SECURITY NUMBER:" FMT / A X $ END.
CALL GET_IN_BUFFER.
LET TEMP_SSN EQ $LEFT(IN_BUFFER, 20).
LET LINE EQ 1.
LET ABORTED EQ FALS.
LET AKCEPT_IT EQ TRU.
IF ($TRIM(TEMP_SSN) EQ ESCAPE_CHAR ) THEN
    LET ABORTED EQ TRU.
ELSE
    LET TEMP_DATA(LINE) EQ TEMP_SSN.
    CALL CHECK_SSN.
    IF AKCEPT_IT EQ FALS THEN
        LET TEMP_DATA(LINE) EQ $NULL.
    ENDIF.
ENDIF.
UNTIL (AKCEPT_IT EQ TRU) OR (ABORTED EQ TRU).
IF ABORTED EQ FALS THEN
    CALL CHECK_TEMP_INFO.
    IF DUPLICATE EQ TRU THEN
        TY "BE PATIENT, WHILE THE DATA BASE IS UPDATED, ".
        TY "THEN YOU´LL BE ABLE TO ´UPDATE´ THIS STUDENT.".
        RELEASE 2.
        RELEASE 3.
        CALL TRANSACT_INTO_DATA_BASE.
        INIT 2 GLOBAL_TMP.
        INIT 3 LOCAL_TMP.
        LET VALID_TMP_FILE EQ FALS.
        CALL INIT_TEMP_INFO.
        TY "OK TO PROCEED" FMT / A / END.
    ENDIF.
    SET FMSG OFF.
    FIND SOCIAL_SECURITY_NUMBER EQ $INT(FIELD_DATA(SSN_POS)).
    SET FMSG ON.
    IF SYSNREC EQ 1 THEN
        TY "STUDENT NAME = " FMT A $ END.
        TY $TRIM(STUDENT_LAST_NAME) $TRIM(STUDENT_FIRST_NAMES) -
            FMT A ", " A END.
        TY "SOCIAL SECURITY NUMBER = " FMT A $ END.
        TY $SUBSTR(FIELD_DATA(SSN_POS), 1, 3) FMT A "-" $ END.
        TY $SUBSTR(FIELD_DATA(SSN_POS), 4, 5) FMT A "-" $ END.
        TY $SUBSTR(FIELD_DATA(SSN_POS), 6, 9) FMT A END.
        TY " IS THIS THE CORRECT STUDENT? ([Y] or N):" FMT / A X $ END.
        CALL GET_IN_BUFFER.
    ENDIF.
ENDIF.
LET ANSWER EQ $LEFT(IN_BUFFER, 20).
IF (SCAPS($TRIM(ANSWER)) EQ "y") OR -
  ($LEN($TRIM(ANSWER)) EQ 0) THEN
  CALL READ_FROM_DB.
  LET DONE EQ TRU.
ELSE
  LET DONE EQ FALS.
  LET FEELD_DATA(SSN_POS) EQ $NULL.
  TY ESCAPE_CHAR -
    FMT "RE-ENTER OR TYPE " ' A " TO ESCAPE" END.
ENDIF.
ELSEIF SYSNREC EQ 0 THEN
  TY BEEP FMT A $ END.
  TY "STUDENT RECORD NOT FOUND FOR " FMT / A X $ END.
  TY TEMP_DATA(SSN_POS) FMT "SOC.SEC.NUM:" X A END.
  TY ESCAPE_CHAR -
    FMT / "PLEASE TRY AGAIN, OR TYPE " ' A " TO ESCAPE" END.
ELSE
  LET DONE EQ TRU.  ! BIG PROBLEMS
ENDIF.
ENDIF.
UNTIL (ABORTED EQ TRU) OR (DONE EQ TRU).
RETURN.
COMPLETE_FIELDS:

! LET THE USER COMPLETE UNFINISHED RECORDS
! CALLED FROM COMPLETE SECTION OF UPDATE
!

! COMPLETE THE STUDENT DATA SECTION

LET LINE EQ 1.
LET DONE EQ FALS.
REPEAT
  LET WAIT EQ FALS.
  WHILE ( LINE LE END_STUDENT_DATA ) AND ( WAIT EQ FALS ) DO
    IF ($LEN($TRIM(TEMP_DATA(LINE)) EQ 0) OR -
      ($TRIM(TEMP_DATA(LINE)) EQ "0") OR -
      ($LEFT($LTRIM(TEMP_DATA(LINE)), 3) EQ ".00") THEN
      LET WAIT EQ TRU.
    ELSE
      LET FIELD_DATA(LINE) EQ TEMP_DATA(LINE).
      LET LINE EQ LINE + 1.
    ENDIF.
  ENDFILE.
ENDWHILE.
IF WAIT EQ TRU THEN
  REPEAT
    TY FEELDS(LINE) FMT A X $ END.
    CALL GET_IN BUFFER.
    LET TEMP_DATA(LINE) EQ $LEFT(IN_BUFFER, 20).
    LET ACEPT_IT EQ TRU.
    IF $TRIM(TEMP_DATA(LINE)) EQ ESCAPE_CHAR THEN
      LET TEMP_DATA(LINE) EQ $NULL.
      LET DONE EQ TRU.
    ELSEIF $LEN($TRIM(TEMP_DATA(LINE))) NE 0 THEN
      CALL ACEPT_DATA.
      IF ACEPT_IT EQ FALS THEN
        LET TEMP_DATA(LINE) EQ $NULL.
      ENDIF.
    ENDIF.
  UNTIL (ACEPT_IT EQ TRU) OR (DONE EQ TRU).
  LET LINE EQ LINE + 1.
ENDIF.
UNTIL (WAIT EQ FALS) OR (DONE EQ TRU).

! COMPLETE THE OFFICIAL DATA SECTION

LET LINE EQ (END_STUDENT_DATA + 1).
! CHECK TO SEE IF OFFICIAL DATA NEEDS TO BE COMPLETED
LET STOP EQ FALS.
WHILE ( LINE LE END_OFFICIAL_DATA ) AND ( STOP EQ FALSE ) DO
  IF ($LEN($TRIM(TEMP_DATA(LINE))) EQ 0) OR -
    ($TRIM(TEMP_DATA(LINE)) EQ "0") OR -
    ($LEFT($LTRIM(TEMP_DATA(LINE)), 3) EQ ".00") THEN
    LET STOP EQ TRUE.
  ELSE
    LET LINE EQ LINE + 1.
  ENDIF.
ENDWHILE.

IF LINE GT END_OFFICIAL_DATA THEN
  MOVE REST OF OFFICIAL DATA
  LET LINE EQ (END_STUDENT_DATA + 1).
  WHILE LINE LE END_OFFICIAL_DATA DO
    LET FEELD_DATA(LINE) EQ TEMP_DATA(LINE).
    LET LINE EQ LINE + 1.
  ENDWHILE.
ELSE ! NEED TO GET SOME OFFICIAL DATA
  LET LINE EQ (END_STUDENT_DATA + 1).
  LET DONE EQ FALSE.
  TY "OFFICIAL DATA ENTRY SECTION" FMT / 8X A END.
  TY "----------------------" FMT 8X A / END.
REPEAT
  LET WAIT EQ FALSE.
  WHILE ( LINE LE END_OFFICIAL_DATA ) AND ( WAIT EQ FALSE ) DO
    IF ($LEN($TRIM(TEMP_DATA(LINE))) EQ 0) OR -
      ($TRIM(TEMP_DATA(LINE)) EQ "0") OR -
      ($LEFT($LTRIM(TEMP_DATA(LINE)), 3) EQ ".00") THEN
      LET WAIT EQ TRUE.
    ELSE
      LET FEELD_DATA(LINE) EQ TEMP_DATA(LINE).
      LET LINE EQ LINE + 1.
    ENDIF.
  ENDWHILE.
  IF WAIT EQ TRUE THEN
    REPEAT
      TY FEELDS(LINE) FMT A X $ END.
      CALL GET_IN_BUFFER.
      LET TEMP_DATA(LINE) EQ $LEFT(IN_BUFFER, 20).
      LET AKCEPT IT EQ TRUE.
      IF $TRIM(TEMP_DATA(LINE)) EQ ESCAPE CHAR THEN
        LET TEMP_DATA(LINE) EQ $NULL.
        LET DONE EQ TRUE.
      ELSE
        ! MOVE REST OF OFFICIAL DATA
        WHILE LINE LE END_OFFICIAL_DATA DO
          LET FEELD_DATA(LINE) EQ TEMP_DATA(LINE).
          LET LINE EQ LINE + 1.
        ENDWHILE.
      ELSEIF $LEN($TRIM(TEMP_DATA(LINE))) NE 0 THEN
        CALL AKCEPT_DATA.
        IF AKCEPT IT EQ FALSE THEN
          LET TEMP_DATA(LINE) EQ $NULL.
        ENDIF.
      ENDIF.
    REPEAT
  ENDIF.
ENDWHILE.
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
ENDIF.
UNTIL (AKCEPT_IT EQ TRU) OR (DONE EQ TRU).
LET LINE EQ LINE + 1.
ENDIF.
UNTIL (WAIT EQ FALSE) OR (DONE EQ TRU).
ENDIF.

! MOVE CANCEL + WITHDRAW DATA
!
LET LINE EQ END_OFFICIAL_DATA + 1.
WHILE LINE LE NUM_FEELDS DO
    LET FEELD_DATA(LINE) EQ TEMP_DATA(LINE).
    LET LINE EQ LINE + 1.
ENDWHILE.
!
!
TY "DO YOU WANT TO REVIEW? (Y or [N]):" FMT / A X $ END.
CALL GET_IN_BUFFER.
LET ANSWER $LEFT(IN_BUFFER, 20). IF $CAPS($TRIM(ANSWER)) EQ "Y" THEN
    CALL REVIEW_DATA.
ENDIF.
RETURN.
READ_FROM_DB:
! READ THE DATA FROM A RECORD IN THE DATABASE
! CALLED BY GET_RECORD
!
LET TEMP_DATA(1) EQ $'TEXTL(SOCIAL_SECURITY_NUMBER).
LET TEMP_DATA(2) EQ STUDENT_LAST_NAME.
LET TEMP_DATA(3) EQ STUDENT_FIRST_NAME.
LET TEMP_DATA(4) EQ STREET_PERMANENT.
LET TEMP_DATA(5) EQ CITY_PERMANENT.
LET TEMP_DATA(6) EQ STATE_PERMANENT.
LET TEMP_DATA(7) EQ $'TEXTL(ZIP_PERMANENT).
LET TEMP_DATA(8) EQ COUNTRY_PERMANENT.
LET TEMP_DATA(9) EQ $'TEXTL(PHONE_PERMANENT).
LET TEMP_DATA(10) EQ PARENT_LAST_NAME.
LET TEMP_DATA(11) EQ PARENT_FIRST_NAME.
LET TEMP_DATA(12) EQ STREET_PARENT.
LET TEMP_DATA(13) EQ CITY_PARENT.
LET TEMP_DATA(14) EQ STATE_PARENT.
LET TEMP_DATA(15) EQ $'TEXTL(ZIP_PARENT).
LET TEMP_DATA(16) EQ COUNTRY_PARENT.
LET TEMP_DATA(17) EQ $'TEXTL(PHONE_PARENT).
LET TEMP_DATA(18) EQ $'TEXTL(ACADEMIC_STATUS).
LET TEMP_DATA(19) EQ STUDENT_STATUS.
LET TEMP_DATA(20) EQ QUARTER_APPLIED_FOR.

! TRANSFORM 1022 DATE FORMAT
IF $LEN($TRIM($'TEXTL(DATE_OF_BIRTH))) EQ 8 THEN
    LET TEMP_DATA(21) EQ -
    $'SUBSTR($'TEXTL(DATE_OF_BIRTH),5,6) + "/" + -
    $'SUBSTR($'TEXTL(DATE_OF_BIRTH),7,8) + "/" + -
    $'SUBSTR($'TEXTL(DATE_OF_BIRTH),3,4).
ELSE
    LET TEMP_DATA(21) EQ $'TEXTL(DATE_OF_BIRTH).
ENDIF.
LET TEMP_DATA(22) EQ MARITAL_STATUS.
LET TEMP_DATA(23) EQ SEX.
LET TEMP_DATA(24) EQ US_CITIZEN.
LET TEMP_DATA(25) EQ OTHER_CITIZENSHIP.
LET TEMP_DATA(26) EQ PHYSICAL_LIMITATIONS.
LET TEMP_DATA(27) EQ $'TEXTL(LIMITATIONS_CODE_NUM).
LET TEMP_DATA(28) EQ FIRST_HALL_PREFERENCE.
LET TEMP_DATA(29) EQ SECOND_HALL_PREFERENCE.
LET TEMP_DATA(30) EQ VISITATION_PREFERENCE.
LET TEMP_DATA(31) EQ ROOM_TYPE_PREFERENCE.
LET TEMP_DATA(32) EQ RMMATE_LAST_NAME.
LET TEMP_DATA(33) EQ RMMATE_FIRST_NAME.
LET TEMP_DATA(34) EQ $'TEXTL(RMMATE_SOC_SEC_NUM).
LET TEMP_DATA(35) EQ MAJOR_FIELD.
LET TEMP_DATA(36) EQ SMOKER_SELF.
LET TEMP_DATA(37) EQ DRINKER_SELF.
LET TEMP_DATA(38) EQ NIGHTOWL_SELF.
LET TEMP_DATA(39) EQ SMOKER_ROOMMATE.
LET TEMP_DATA(40) EQ DRINKER_ROOMMATE.
LET TEMP_DATA(41) EQ NIGHTOWL_ROOMMATE.
LET TEMP_DATA(42) EQ $TEXTL(APP_PRIORITY_NUMBER).
LET TEMP_DATA(43) EQ $TEXTL(VALIDATION_NUMBER).
LET TEMP_DATA(44) EQ APPLICATION_TYPE.
LET TEMP_DATA(45) EQ PRE_PAYED.
LET TEMP_DATA(46) EQ PRE_PAY_RECVD_BY.
! TRANSFORM 1022 DATE FORMAT
IF $LEN($TRIM($TEXTL(PRE_PAY_DATE))) EQ 8 THEN
  LET TEMP_DATA(47) EQ -
    $SUBSTR($TEXTL(PRE_PAY_DATE),5,6) + "/" + -
    $SUBSTR($TEXTL(PRE_PAY_DATE),7,8) + "/" + -
    $SUBSTR($TEXTL(PRE_PAY_DATE),3,4).
ELSE
  LET TEMP_DATA(47) EQ $TEXTL(PRE_PAY_DATE).
ENDIF.
LET TEMP_DATA(48) EQ $TEXTL(CANCEL_REFUND_AMOUNT).
! TRANSFORM 1022 DATE FORMAT
IF $LEN($TRIM($TEXTL(CANCEL_DATE))) EQ 8 THEN
  LET TEMP_DATA(49) EQ -
    $SUBSTR($TEXTL(CANCEL_DATE),5,6) + "/" + -
    $SUBSTR($TEXTL(CANCEL_DATE),7,8) + "/" + -
    $SUBSTR($TEXTL(CANCEL_DATE),3,4).
ELSE
  LET TEMP_DATA(49) EQ $TEXTL(CANCEL_DATE).
ENDIF.
LET TEMP_DATA(50) EQ $TEXTL(WITHDRAW_ROOM_AMOUNT).
LET TEMP_DATA(51) EQ $TEXTL(WITHDRAW_SINGLE_AMOUNT).
LET TEMP_DATA(52) EQ $TEXTL(WITHDRAW_MEALPLAN_AMOUNT).
! TRANSFORM 1022 DATE FORMAT
IF $LEN($TRIM($TEXTL(WITHDRAW_DATE))) EQ 8 THEN
  LET TEMP_DATA(53) EQ -
    $SUBSTR($TEXTL(WITHDRAW_DATE),5,6) + "/" + -
    $SUBSTR($TEXTL(WITHDRAW_DATE),7,8) + "/" + -
    $SUBSTR($TEXTL(WITHDRAW_DATE),3,4).
ELSE
  LET TEMP_DATA(53) EQ $TEXTL(WITHDRAW_DATE).
ENDIF.
RETURN.
AKCEPT_DATA:

! GET AND CHECK INPUT FOR CORRECTNESS.
! VERY ORDER DEPENDENT - FOR EXCEPTIONS.
! RETURNS 'AKCEPT IT' EQ 'FALSE' IF THE DATA IS BAD.
!
LET AKCEPT IT EQ TRUE.
IF LINE EQ (12, 24, 26, 34) THEN
   CALL DATA_EXCEPTIONS.
ELSEIF FIELD TYPE (LINE) EQ "A" THEN
   CALL CHECK AC_STATUS.
ELSEIF FIELD TYPE (LINE) EQ "C" THEN
   CALL CHECK APPLIC.
ELSEIF FIELD TYPE (LINE) EQ "D" THEN
   CALL CHECK_DATE.
ELSEIF FIELD TYPE (LINE) EQ "I" THEN
   CALL CHECK INTEGER.
ELSEIF FIELD TYPE (LINE) EQ "L" THEN
   CALL CHECK LIMIT.
ELSEIF FIELD TYPE (LINE) EQ "M" THEN
   CALL CHECK MARITAL.
ELSEIF FIELD TYPE (LINE) EQ "N" THEN
   CALL CHECK NUMBER.
ELSEIF FIELD TYPE (LINE) EQ "P" THEN
   CALL CHECK PHONE.
ELSEIF FIELD TYPE (LINE) EQ "Q" THEN
   CALL CHECK QUARTER.
ELSEIF FIELD TYPE (LINE) EQ "R" THEN
   CALL CHECK ROOM.
ELSEIF FIELD TYPE (LINE) EQ "S" THEN
   CALL CHECK SSN.
ELSEIF FIELD TYPE (LINE) EQ "T" THEN
   CALL CHECK TEXT.
ELSEIF FIELD TYPE (LINE) EQ "U" THEN
   CALL CHECK STU STATUS.
ELSEIF FIELD TYPE (LINE) EQ "V" THEN
   CALL CHECK VISIT.
ELSEIF FIELD TYPE (LINE) EQ "X" THEN
   CALL CHECK SEX.
ELSEIF FIELD TYPE (LINE) EQ "Y" THEN
   CALL CHECK YES OR NO.
ELSEIF FIELD TYPE (LINE) EQ "Z" THEN
   CALL CHECK ZIP.
ELSEIF FIELD TYPE (LINE) EQ "$" THEN
   CALL CHECK DOLLARS.
ENDIF.
RETURN.
DATA_EXCEPTIONS:

ALLOW DUPLICATION OF PERMANENT INTO PARENT ADDRESS,
VERY ORDER DEPENDENT.

IF (LINE EQ 12) AND ($LTRIM(TEMP_DATA(LINE)) EQ "") THEN
  LET I EQ LINE.
  WHILE I LE (LINE + 5) DO
    LET TEMP_DATA(I) EQ TEMP_DATA(I - 8).
    LET FEELD_DATA(I) EQ FEELD_DATA(I - 8).
    TY FEELDS(I) FEELD_DATA(I) FMT / A X A END.
    LET I EQ I + 1.
  ENDWHILE.
  LET LINE EQ (I - 1). ! TO RESYNCHRONIZE LINE, PARENT WILL INCREMENT
  IF THE QUESTION IS 'US CITIZENSHIP' (24) AND
  THE ANSWER IS YES, THEN DON'T PROMPT FOR 'OTHER',
  JUST FILL 'OTHER CITIZENSHIP' WITH 'N/A'
ELSEIF LINE EQ 24 THEN
  CALL CHECK_Y_OR_N.
  IF (AKCEPT_IT EQ TRU) AND ($TRIM(FEELD_DATA(LINE)) EQ "Y") THEN
    LET LINE EQ LINE + 1.
    LET FEELD_DATA(LINE) EQ "N/A".
    LET TEMP_DATA(LINE) EQ "N/A".
  ENDIF.

ELSEIF (LINE EQ 26) THEN
  CALL CHECK_Y_OR_N.
  IF (AKCEPT_IT EQ TRU) AND ($TRIM(FEELD_DATA(LINE)) EQ "N") THEN
    LET LINE EQ LINE + 1.
    LET FEELD_DATA(LINE) EQ "1".
    LET TEMP_DATA(LINE) EQ "1".
  ENDIF.

ELSEIF 'ROOMATE SOC SEC NUM' (34),
 THEN SET SSN_POS TO 34 BEFORE CHECKING, AND RESET WHEN DONE
ELSEIF (LINE EQ 34) THEN
  LET SSN_POS EQ 34.
  CALL CHECK_SSN.
  LET SSN_POS EQ 1.
ENDIF.
RETURN.
CHECK_AC_STATUS:

!  CHECK ACADEMIC STATUS - A
!
LET CHECK_BUFFER EQ $CAPS($LTRIM(TEMP_DATA(LINE)) ).
IF $TRIM(CHECK_BUFFER) NE -
( '1', '2', '3', '4', '5', '6' ) THEN
  LET ACCEPT IT EQ FALS.
  TY "ERROR - INCORRECT ACADEMIC STATUS ANSWER:" FMT A $ END.
  TY BEEP FMT A $ END.
  TY $TRIM(TEMP_DATA(LINE)) FMT A END.
  TY "PLEASE RE-ENTER." FMT A / END.
ELSE
  LET FEELD_DATA(LINE) EQ CHECK_BUFFER.
ENDIF.
RETURN.

CHECK_APPLICATION:

!  CHECK APPLICATION TYPE - C
!
LET CHECK_BUFFER EQ $CAPS($LTRIM(TEMP_DATA(LINE)) ).
IF $TRIM(CHECK_BUFFER) NE ( 'UM', 'VT', 'VR', 'DE' ) THEN
  LET ACCEPT IT EQ FALS.
  TY "ERROR - INCORRECT APPLICATION TYPE ANSWER:" FMT A $ END.
  TY BEEP FMT A $ END.
  TY $TRIM(TEMP_DATA(LINE)) FMT A END.
  TY "PLEASE RE-ENTER." FMT A / END.
ELSE
  LET FEELD_DATA(LINE) EQ CHECK_BUFFER.
ENDIF.
RETURN.
CHECK_DATE:

!  CHECK DATES - D
!
LET SYSCVTERR EQ 0.
LET SYSCVTMSG EQ 1.
LET TEMP_DATE EQ $DATE(TEMP_DATA(LINE)).
IF SYSCVTERR EQ 1 THEN
  LET AKCEPT_IT EQ FALS.
  LET SYSCVTERR EQ 0.
  TY "ERROR - INCORRECT DATE FORMAT:" FMT A $ END.
  TY BEEP FMT A $ END.
  TY $TRIM(TEMP_DATA(LINE)) FMT A END.
  TY "PLEASE RE-ENTER." FMT A END.
ELSEIF $LEN($TRIM(TEMP_DATA(LINE))) GT FEELD_LEN(LINE) THEN
  LET AKCEPT_IT EQ FALS.
  TY "ERROR - DATE TOO LONG:" FMT A $ END.
  TY BEEP FMT A $ END.
  TY $TRIM(TEMP_DATA(LINE)) FMT A END.
  TY "PLEASE RE-ENTER." FMT A END.
ELSE
  LET FEELD_DATA(LINE) EQ TEMP_DATA(LINE).
ENDIF.
IF AKCEPT_IT EQ FALS THEN
  TY "TRY USING THIS FORMAT - MM/DD/YY" FMT A / END.
ENDIF.
LET SYSCVTMSG EQ 0.
RETURN.

CHECK_INTEGER:

!  CHECK INTEGERS - I
!
LET SYSCVTERR EQ 0.
LET SYSCVTMSG EQ 1.
LET TEMP_INT EQ $INT(TEMP_DATA(LINE)).
IF SYSCVTERR EQ 1 THEN
  LET AKCEPT_IT EQ FALS.
  TY "ERROR - INCORRECT ENTRY:" FMT A $ END.
  TY BEEP FMT A $ END.
  TY $TRIM(TEMP_DATA(LINE)) FMT A END.
  TY "PLEASE RE-ENTER." FMT A / END.
ELSE
  LET FEELD_DATA(LINE) EQ TEMP_DATA(LINE).
ENDIF.
LET SYSCVTMSG EQ 0.
RETURN.
CHECK_LIMIT:

! CHECK PHYSICAL LIMITATIONS - L
!
LET CHECK_BUFFER EQ $LTRIM(TEMP_DATA(LINE)).
IF $TRIM(CHECK_BUFFER) -
    NE ( '2', '3', '4', '5', '6', '7', '8', '9' ) THEN
    LET AKCEPT_IT EQ FALS.
    TY "ERROR - INCORRECT LIMITATION ANSWER:" FMT A $ END.
    TY $TRIM(CHECK_BUFFER) FMT A $ END.
ENDIF.
TY "PLEASE RE-ENTER." FMT A / END.
ELSE
    LET FEEL_DATA(LINE) EQ CHECK_BUFFER.
ENDIF.
RETURN.

CHECK_MARITAL:

! CHECK MARITAL STATUS - M
!
LET CHECK_BUFFER EQ $CAPS($LTRIM(TEMP_DATA(LINE))).
IF $TRIM(CHECK_BUFFER) NE ( 'M', 'S' ) THEN
    LET AKCEPT_IT EQ FALS.
    TY "ERROR - INCORRECT MARITAL STATUS ANSWER:" FMT A $ END.
    TY $TRIM(CHECK_BUFFER) FMT A $ END.
    TY "$TRIM(TEMP_DATA(LINE))" FMT A END.
    TY "PLEASE RE-ENTER." FMT A / END.
ELSE
    LET FEEL_DATA(LINE) EQ CHECK_BUFFER.
ENDIF.
RETURN.
CHECK_NUMBER:
!
! CHECK PRIORITY NUMBER - N
! ITS LENGTH MUST BE 8.
! ITS FORMAT MUST BE:
! YEAR - 2 DIGITS
! QUARTER - 1 DIGIT (1=SPRING, 2=SUMMER, 3=FALL, 4=WINTER)
! DELIMITER - 1 DIGIT (MUST BE ‘0´)
! NUMBER - 4 DIGITS EXACTLY (FRONT FILLED WITH ‘0s´ IF NEEDED)
!
LET CHECK BUFFER EQ $LTRIM(TEMP DATA(LINE)).
IF $LEN($TRIM(CHECK_BUFFER)) NE 8 THEN
   LET AKCEPT_IT EQ FALS.
   TY "ERROR - INCORRECT PRIORITY NUMBER LENGTH:" FMT A $ END.
ELSEIF $SUBSTR(CHECK_BUFFER,3,4) NE (’10’, ’20’, ’30’, ’40’) THEN
   LET AKCEPT_IT EQ FALS.
   TY "ERROR - INCORRECT PRIORITY NUMBER FORMAT:" FMT A $ END.
ENDIF.
IF AKCEPT_IT EQ TRU THEN
   CALL CHECK_INTEGER.
   IF AKCEPT_IT EQ FALS THEN
      TY "ERROR - INCORRECT PRIORITY NUMBER MUST BE AN INTEGER:" FMT A $ END.
   ENDIF.
ENDIF.
IF AKCEPT_IT EQ FALS THEN
   TY BEEP FMT A $ END.
   TY $STRIM(TEMP DATA(LINE)) FMT A END.
   TY "REMEMBER IT MUST BE AN 8 CHARACTER INTEGER.".
   TY "IN THIS FORMAT ‘YYQONNNN’." FMT 8X A END.
   TY "PLEASE RE-ENTER." FMT A / END.
ELSE
   LET FEELD_DATA(LINE) EQ CHECK_BUFFER.
ENDIF.
RETURN.
CHECK_PHONE:
!
! CHECK PHONE NUMBERS - P
!
LET CHECK_BUFFER EQ $LTRIM(TEMP_DATA(LINE)).
LET CHECK_BUFFER EQ $REPALL(\(,\$NULL,CHECK_BUFFER).
LET CHECK_BUFFER EQ $REPALL(\(-,\$NULL,CHECK_BUFFER).
IF $LEN($TRIM(CHECK_BUFFER)) NE (7,10) THEN
   LET AKCEPT_IT EQ FALS.
   TY "ERROR - PHONE NUMBER IS THE WRONG LENGTH:" FMT A $ END.
ELSE
   LET I EQ 0.
   WHILE I LT $LEN($TRIM(CHECK_BUFFER)) DO
      LET I EQ I + 1.
      IF $ICHAR(CHECK_BUFFER,I) NE-
         (48,49,50,51,52,53,54,55,56,57) THEN
         LET AKCEPT_IT EQ FALS.
      ENDIF.
   ENDSLWHILE.
   IF AKCEPT_IT EQ FALS THEN
      TY "ERROR - INCORRECT CHARACTER IN PHONE NUMBER:" FMT A $ END.
   ENDIF.
ENDIF.
IF AKCEPT_IT EQ TRU THEN
   LET FEELD_DATA(LINE) EQ CHECK_BUFFER.
ELSE
   TY BEEP FMT A $ END.
   TY TRIM(TEMP_DATA(LINE)) FMT A END.
   TY "PLEASE RE-ENTER." FMT A / END.
ENDIF.
RETURN.
CHECK_QUARTER:
  !
  !  CHECK QUARTER APPLIED FOR - Q
  !
  LET CHECK_BUFFER EQ $CAPS($LTRIM(TEMP_DATA(LINE)));
  LET CHECK_BUFFER EQ $REPALL( '  ',NULL,CHECK_BUFFER).
  IF $LEN($TRIM(CHECK_BUFFER)) NE FEELD_LEN(LINE) THEN
    LET AKCEPT_IT EQ FALS.
  ENDIF.
  IF $LEFT(CHECK_BUFFER,1 ) NE ('F','W','S') THEN
    LET AKCEPT_IT EQ FALS.
  ENDIF.
  IF AKCEPT_IT EQ TRU THEN
    LET I EQ 1. ! START AT THE SECOND CHARACTER
    WHILE I LT 3 DO ! IS IT IN 0..9 (ASCII VALUE)
      LET I EQ I + 1.
      IF $ICHAR(CHECK_BUFFER,I) NE -
         (48,49,50,51,52,53,54,55,56,57) THEN
        LET AKCEPT_IT EQ FALS.
      ENDIF.
  ENDDO.
  IF AKCEPT_IT EQ TRU THEN
    LET FEELD_DATA(LINE) EQ CHECK_BUFFER.
  ELSE
    TY "ERROR - INCORRECT ENTRY:" FMT A $ END.
    TY TEMP_DATA(LINE) FMT A END.
    TY "CORRECT FORMAT IS QUARTER (F, W, or S) YEAR (YY)" FMT A END.
    TY BEEP FMT A $ END.
    TY "PLEASE RE-ENTER." FMT A / END.
  ENDIF.
RETURN.

CHECK_ROOM:
  !
  !  CHECK ROOM - R
  !
  LET CHECK_BUFFER EQ $CAPS($LTRIM(TEMP_DATA(LINE)));
  IF $TRIM(CHECK_BUFFER) NE ('D','S') THEN
    LET AKCEPT_IT EQ FALS.
    TY "ERROR - INCORRECT ROOM TYPE:" FMT A $ END.
    TY TEMP_DATA(LINE) FMT A END.
    TY "PLEASE RE-ENTER." FMT A / END.
  ELSE
    LET FEELD_DATA(LINE) EQ CHECK_BUFFER.
  ENDIF.
RETURN.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
CHECK_SSN:

! CHECK SOCIAL SECURITY NUMBER - S 
!
LET CHECK BUFFER EQ $CAPS($LTRIM(TEMP_DATA(SSN_POS))).
LET CHECK BUFFER EQ $REPALL( '-', $NULL, CHECK_BUFFER).
LET TEMP BUFFER EQ $REPALL( ' ', $NULL, CHECK_BUFFER).
IF $LEN($TRIM(TEMP BUFFER)) NE $LEN($TRIM(CHECK BUFFER)) THEN
  LET ACCEPT IT EQ FALS.
  TY "ERROR - 'SPACES' NOT ALLOWED IN SOC. SEC. NUM:" FMT A X $. END.
ENDIF.
IF (ACCEPT IT EQ TRU) THEN
LET I EQ 0.
WHILE (I LT 20) AND (ACCEPT IT EQ TRU) DO
  LET I EQ I + 1.
  IF $ICHAR(CHECK BUFFER, I) NE -
    ! THE ASCII VALUES FOR DIGITS OR TRAILING BLANKS UP TO
    ! NINE PLACES. IF THE NUMBER IS TOO SHORT, THE NEXT TEST
    ! WILL CATCH IT.
    (32, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57) THEN
    LET ACCEPT IT EQ FALS.
    TY "ERROR - INCORRECT CHARACTER IN SOC. SEC. NUM.:
      FMT A X $. END.
  ENDIF.
ENDWHILE.
ENDIF.
IF (ACCEPT IT EQ TRU) AND ($LEN($TRIM(CHECK BUFFER)) NE 9) THEN
  LET ACCEPT IT EQ FALS.
  TY "ERROR - INCORRECT NUMBER OF DIGITS IN SOC. SEC. NUM.:
    FMT A X $. END.
ENDIF.
IF ACCEPT IT EQ TRU THEN
  LET FIELD_DATA(SSN_POS) EQ CHECK BUFFER.
ELSE
  TY BEEP FMT A $. END.
  TY $TRIM(TEMP DATA(SSN_POS)) FMT A END.
  TY ESCAPE_CHAR -
    FMT "PLEASE RE-ENTER OR TYPE ' " A ' ' TO ESCAPE" / END.
ENDIF.
RETURN.
CHECK_TEXT:

! CHECK TEXT AGAINST LENGTH IN FEELD_LEN ARRAY!

LET CHECK_BUFFER EQ $CAPS($LTRIM(TEMP_DATA(LINE))).
IF $LEN($TRIM(CHECK_BUFFER)) GT FEELD_LEN(LINE) THEN
  LET AKCEPT_IT EQ FALS.
  TY "ERROR - ENTRY TOO LONG:" FMT A $ END.
  TY $TRIM(TEMP_DATA(LINE)) FMT A END.
  TY FEELD_LEN(LINE) FMT "THIS FIELD CAN ONLY BE " A " CHARACTERS LONG" END.
  TY "PLEASE RE-ENTER." FMT A / END.
ELSE
  LET FEELD_DATA(LINE) EQ CHECK_BUFFER.
ENDIF.
RETURN.

CHECK_STU_STATUS:

! CHECK STUDENT STATUS - U!

LET CHECK_BUFFER EQ $CAPS($LTRIM(TEMP_DATA(LINE))).
IF $TRIM(CHECK_BUFFER) NE ( 'F', 'N', 'T' ) THEN
  LET AKCEPT_IT EQ FALS.
  TY "ERROR - INCORRECT STATUS:" FMT A $ END.
  TY BEEP FMT A $ END.
  TY $TRIM(TEMP_DATA(LINE)) FMT A _D.
  TY "PLEASE RE-ENTER." FMT A / END.
ELSE
  LET FEELD_DATA(LINE) EQ CHECK_BUFFER.
ENDIF.
RETURN.

CHECK_VISIT:

! CHECK VISITATION CODE - V!

LET CHECK_BUFFER EQ $CAPS($LTRIM(TEMP_DATA(LINE))).
IF $TRIM(CHECK_BUFFER) NE ( 'CU', 'NU', 'NL' ) THEN
  LET AKCEPT_IT EQ FALS.
  TY "ERROR - INCORRECT VISITATION CODE:" FMT A $ END.
  TY BEEP FMT A $ END.
  TY $TRIM(TEMP_DATA(LINE)) FMT A END.
  TY "PLEASE RE-ENTER." FMT A / END.
ELSE
  LET FEELD_DATA(LINE) EQ CHECK_BUFFER.
ENDIF.
RETURN..

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
CHECK_SEX:

! CHECK SEX - X
!
LET CHECK_BUFFER EQ $CAPS($LTRIM(TMP_DATA(LINE))).
IF $TRIM(CHECK_BUFFER) NE ('M', 'F') THEN
    LET ACCEPT_IT EQ FALSE.
    TY "ERROR - INCORRECT SEX CODE:" FMT A $ END.
    TY BEEP FMT A $ END.
    TY $TRIM(TMP_DATA(LINE)) FMT A END.
    TY "PLEASE RE-ENTER." FMT A / END.
ELSE
    LET FEELD_DATA(LINE) EQ CHECK_BUFFER.
ENDIF.
RETURN.

CHECK_Y_ OR_N:

! CHECK FOR YES OR NO. - Y
!
LET CHECK_BUFFER EQ $CAPS($LTRIM(TMP_DATA(LINE))).
IF $TRIM(CHECK_BUFFER) NE ('Y', 'N') THEN
    LET ACCEPT_IT EQ FALSE.
    TY "ERROR - INCORRECT ANSWER:" FMT A $ END.
    TY BEEP FMT A $ END.
    TY $TRIM(TMP_DATA(LINE)) FMT A END.
    TY "PLEASE RE-ENTER." FMT A / END.
ELSE
    LET FEELD_DATA(LINE) EQ CHECKBUFFER.
ENDIF.
RETURN.
CHECK_ZIP:
  !
  ! CHECK THE ZIP CODE - Z
  !
  LET CHECK_BUFFER EQ $STRIM(TEMP_DATA(LINE)).
  LET CHECK_BUFFER EQ $REPALL('-', $NULL, CHECK_BUFFER).
  IF $LEN($STRIM(CHECK_BUFFER)) NE (5, 9) THEN
    LET AKCEPT_IT EQ FALS.
    TY "ERROR - INCORRECT ZIP ENTRY:" FMT A $ END.
  ELSE
    LET I EQ 0.
    WHILE I LT $LEN($STRIM(CHECK_BUFFER)) DO
      LET I EQ I + 1.
      IF $ICHAR(CHECK_BUFFER, I) NE -
        (48, 49, 50, 51, 52, 53, 54, 55, 56, 57) THEN
        LET AKCEPT_IT EQ FALS.
      ENDIF.
    ENDWHILE.
    IF AKCEPT_IT EQ FALS THEN
      TY "ERROR - INCORRECT CHARACTER IN ZIP:" FMT A $ END.
    ENDIF.
  ENDIF.
  IF AKCEPT_IT EQ TRU THEN
    LET FIELD_DATA(LINE) EQ CHECK_BUFFER.
  ELSE
    TY BEEP FMT A $ END.
    TY $STRIM(TEMP_DATA(LINE)) FMT A END.
    TY "PLEASE RE-ENTER." FMT A / END.
  ENDIF.
RETURN.

CHECK_DOLLARS:
  !
  ! CHECK DOLLARS - $
  !
  LET SYSCVTERR EQ 0.
  LET SYSCVTMSG EQ 1.
  LET TEMP_REAL EQ $REAL(TEMP_DATA(LINE)).
  IF SYSCVTERR EQ 1 THEN
    LET AKCEPT_IT EQ FALS.
    TY "ERROR - INCORRECT ENTRY:" FMT A $ END.
    TY BEEP FMT A $ END.
    TY $STRIM(TEMP_DATA(LINE)) FMT A END.
    TY "PLEASE RE-ENTER." FMT A / END.
    LET SYSCVTERR EQ 0.
  ELSE
    LET FIELD_DATA(LINE) EQ TEMP_DATA(LINE).
  ENDIF.
  LET SYSCVTMSG EQ 0.
RETURN.
INIT_VARIABLES:
!
!  INITIALIZE INPUT VARIABLES
!
LET LINE EQ 0.
WHILE LINE LT NUM_FEELDS DO
  LET LINE EQ LINE + 1.
  LET FEELD_DATA(LINE) EQ $NULL.
  LET TEMP_DATA(LINE) EQ $NULL.
ENDWHILE.
RETURN.

INIT_TEMP_INFO:
!
!  FILLS THE TEMP_INFO ARRAY WITH BLANKS
!
LET I EQ 1.
WHILE I LE MAX_TEMP_INFO DO
  LET TEMP_INFO(I) EQ $BLANKS(50).
  LET I EQ I + 1.
ENDWHILE.
RETURN.
VALIDATE_SOC_SEC_NUM:

! CHECKS FOR CORRECTNESS AND UNIQUENESS OF THE SOCIAL
! SECURITY NUMBER. REPROMPTS FOR NEW NUMBER IF PROBLEM.
! SETS 'ABORTED' TO 'TRU' IF NONE RETURNED.
!
REPEAT
  LET ABORTED EQ FALS.
  REPEAT
    LET AKCEPT_IT EQ TRU.
    CALL CHECK_SSN.
    IF AKCEPT_IT EQ FALS THEN
      TY "SOCIAL SECURITY NUMBER:" FMT / A $ END.
      CALL GET_IN_BUFFER.
      LET TEMP_DATA(SSN_POS) EQ $LEFT(IN_BUFFER, 20).
      IF $TRIM(TEMP_DATA(SSN_POS)) EQ ESCAPE_CHAR THEN
        LET ABORTED EQ TRU.
        ENDIF.
      ENDIF.
    ENDIF.
  UNTIL (AKCEPT_IT EQ TRU) OR (ABORTED EQ TRU).
  IF AKCEPT_IT EQ TRU THEN
    SET FMSG OFF.
    FIND SOCIAL_SECURITY_NUMBER EQ $#INT(FIELD_DATA(SSN_POS)).
    SET FMSG ON.
    IF SYSNREC GT 0 THEN
      TY BEEP FMT / A $ END.
      TY "DUPLICATE SOCIAL SECURITY NUMBER FOUND IN THE DATA BASE." .
      TY "ALREADY USED BY:" FMT / A $ END.
      TY $TRIM(STUDENT_LAST_NAME) FMT / A ", " $ END.
      TY $TRIM(STUDENT_FIRST_NAME) FMT / A END.
      LET FIELD_DATA(SSN_POS) EQ $NULL.
      LET AKCEPT_IT EQ FALS.
      TY ESCAPE_CHAR -
      FMT "RE-ENTER OR TYPE ' ' A ' ' TO ESCAPE" END.
      TY "SOCIAL SECURITY NUMBER:" FMT / A $ END.
      CALL GET_IN_BUFFER.
      LET TEMP_DATA(SSN_POS) EQ $LEFT(IN_BUFFER, 20).
      IF $TRIM(TEMP_DATA(SSN_POS)) EQ ESCAPE_CHAR THEN
        LET ABORTED EQ TRU.
        ENDIF.
      ENDIF.
    ENDIF.
  ENDIF.
UNTIL (AKCEPT_IT EQ TRU) OR (ABORTED EQ TRU).
RETURN.
CHECK_TEMP_INFO:

! CHECKS FOR DUPLICATE SSN IN TEMP_INFO AND THEREFORE
! IN TEMP FILES.
! ALSO WILL FORCE TRANSACTION OF TEMP FILES INTO THE DATA BASE
! IF THE ARRAY IS FILLED. THIS SHOULD BE TRANSPARENT TO THE USER.
! TEMP_INFO(MAX_TEMP_INFO) SHOULD ALWAYS REMAIN BLANKS(50).
! RETURNS 'DUPLICATE' EQ 'TRU' IF A DUPLICATE IS FOUND.
!
LET I EQ 1.
LET STOP EQ FALS.
LET DUPLICATE EQ FALS.
WHILE (DUPLICATE EQ FALS) AND (STOP EQ FALS) DO
  IF $TRIM(LEFT_DATA(SSN_POS)) EQ $LEFT(TEMP_INFO(I), 9) THEN
    LET DUPLICATE EQ TRU.
    TY "DUPLICATE SOCIAL SECURITY NUMBER FOUND IN THIS SESSION.".
    TY "ALREADY USED BY:" FMT / A X $ END.
    TY $SUBSTR(TEMP_INFO(I), 10, 26) FMT A ", " $ END.
    TY $SUBSTR(TEMP_INFO(I), 27, 41) FMT A / END.
  ELSE
    LET I EQ I + 1.
    IF TEMP_INFO(I) EQ $BLANKS(50) THEN
      LET STOP EQ TRU.
    ENDIF.
  ENDIF.
ENDWHILE.
IF I EQ MAX_TEMP_INFO THEN ! ARRAY FULL, TRANSACT + RESET
  TY "TEMPORARY STORAGE FULL, HOUSECLEANING IS ABOUT TO OCCUR.".
  RELEASE 2.
  RELEASE 3.
  CALL TRANSACT INTO_DATA_BASE.
  INIT 2 GLOBAL_TMP.
  INIT 3 LOCAL_TMP.
  LET VALID_TMP_FILE EQ FALS.
  CALL INIT_TEMP_INFO.
  TY "HOUSECLEANING COMPLETE, OK TO PROCEED." FMT / A / END.
ENDIF.
RETURN.
MAIN_MENU:

! PRINT THE MAIN MENU

TY " **************** MAIN MENU ****************" FMT / A END.
TY " SELECT DESIRED FUNCTION " FMT A / END.
TY "U: Update data base: (options apply to A STUDENT RECORD)".
TY " Add, Complete, Modify, Delete, Undelete, Withdraw/Cancel.".
TY "A: Assign rooms:".
TY " Check student records, Record assignments.".
TY "R: Reports:".
TY "I: Review on screen, Print paper copy.".
TY "I: Inquiry mode:".
TY " Interactive mode, Canned queries.".
TY "T: Tools:".
TY " Files, Print queue, Etc.".
TY "Q: Quit:".
TY " End 1022 session.".
RETURN.

UPDATE_MENU:

! PRINT THE MENU FOR THE UPDATE FUNCTION

TY " **************** UPDATE MENU ****************" FMT / A END.
TY "ADD TO OR MODIFY THE STUDENT RECORDS IN THE DATA BASE" FMT A / END.
TY "A: Add:".
TY " another student record in the data base.".
TY "C: Complete:".
TY " put additional data in an existing student record.".
TY "M: Modify:".
TY " change data in an existing student record.".
TY "D: Delete:".
TY " remove a student record from the active data base.".
TY "U: Undelete:".
TY " return a student record to the active data base.".
TY "W: Withdraw or Cancel:".
TY " enter or view withdrawal or cancellation information.".
TY "B: Back to the main menu.".
RETURN.
MODIFY_RECORD:
  !
  ! MODIFY RECORD SECTION
  !
  TY CLEAR_SCRN.
  TY "********** ENTERING MODIFY RECORDS **********" FMT / A / END.
  TY "DO YOU WISH TO CONTINUE? ([Y] or N):" FMT A X $ END.
  LET VALID_TMP_FILE EQ FALS. ! SET TRANSACTION FLAG
  CALL GET_IN_BUFFER.
  LET ANSWER EQ $LEFT(IN_BUFFER, 20).
  IF ($CAPS($TRIM(ANSWER)) EQ "Y") OR -
     ($LEN($TRIM(ANSWER)) EQ 0) THEN
    LET CONTINUE EQ TRU.
  INIT 2 GLOBAL_TMP.
  INIT 3 LOCAL_TMP.
  CALL INIT_TEMP_INFO.
  ELSE
    LET CONTINUE EQ FALS.
  ENDIF.
  WHILE CONTINUE EQ TRU DO
    CALL INIT_VARIABLES.
    CALL GET_RECCMD.
    IF (SYSNREC EQ 1) AND (ABORTED EQ FALS) THEN
      LET LINE EQ 1.
      WHILE LINE LE NLM_FEELDS DO
        LET FEELD_DATA(LINE) EQ TEMP_DATA(LINE).
        LET LINE EQ LINE + 1.
      ENDWHILE.
      TY "DO YOU WANT TO MODIFY THE SOC.SEC.NUM.? (Y or [N]):" -
        FMT / A X $ END.
      CALL GET_IN_BUFFER.
      LET ANSWER EQ $LEFT(IN_BUFFER, 20).
      IF ($CAPS($TRIM(ANSWER)) EQ "Y") THEN
        LET SAVE_SSN EQ FEELD_DATA(SSN_POS).
        CALL REVISE_SOC_SEC_NUM.
        IF SAVE_SSN NE FEELD_DATA(SSN_POS) THEN
          DBSET GLOBAL_DMS.
          ! THIS 'FIND' BELOW IS NECESSARY TO RESELECT THE RECORD,
          ! SINCE VALIDATE_SOC_SEC_NUM HAS USED 'FIND' IN THE INTERIM
          SET FMSG OFF.
          FIND SOCIAL_SECURITY_NUMBER EQ $INT(SAVE_SSN).
          SET FMSG ON.
          CHANGE SSN $INT(FEELD_DATA(SSN_POS)).
          DBSET LOCAL_DMS.
          ! THIS 'FIND' BELOW IS NECESSARY TO RESELECT THE RECORD,
          ! SINCE VALIDATE_SOC_SEC_NUM HAS USED 'FIND' IN THE INTERIM
          SET FMSG OFF.
          FIND SOCIAL_SECURITY_NUMBER EQ $INT(SAVE_SSN).
          SET FMSG ON.
          CHANGE SSN $INT(FEELD_DATA(SSN_POS)).
          !
          ! NOTE - ALSO MUST CHECK BEDS DATA SET WHEN IMPLEMENTED
! DBSET GLOBAL, DMS. ! RESET DATA BASE
TY "SOC_SEC_NUM. MODIFIED SUCCESSFULLY" FMT / A / END.
ENDIF.

ENDIF.
TY "DO YOU WANT TO MODIFY SOME OTHER DATA? ([Y] or N):" -
FMT / A X $ END.
CALL GET IN BUFFER.
LET ANSWER EQ SLEFT(IN BUFFER, 20).
IF ($CAPS($TRIM(ANSWER)) NE "N") THEN
  CALL REVIEW DATA.
  TY "RETURN THIS MODIFIED DATA TO THE DATA BASE ([Y] or N):" -
  FMT / A X $ END.
  CALL GET IN BUFFER.
  LET ANSWER EQ SLEFT(IN BUFFER, 20).
  IF ($CAPS($TRIM(ANSWER)) EQ "Y") OR -
    ($LEN($TRIM(ANSWER)) EQ 0) THEN
    CALL MANAGE_TMP_FILE.
    CALL MANAGE TEMP INFO.
    LET VALID_TMP FILE EQ TRU.
  ENDIF.
ENDIF.

TY "MODIFY ANOTHER STUDENT'S DATA? ([Y] or N):" FMT / A X $ END.
CALL GET IN BUFFER.
LET ANSWER EQ SLEFT(IN BUFFER, 20).
IF $CAPS($TRIM(ANSWER)) EQ "N" THEN
  LET CONTINUE EQ FALS.
ENDIF.
ELSEIF ABORTED EQ TRU THEN
  LET CONTINUE EQ FALS.
ENDIF.
ENDWHILE.
RELEASE 2.
RELEASE 3.
IF VALID TMP FILE EQ TRU THEN
  CALL TRANSACT INTO DATA BASE.
ENDIF.
RETURN.
DELETE_RECORD:
  !
  !
  !
  TY "NOT IMPLEMENTED" FMT / A / END.
  TY "TYPE <RETURN> TO CONTINUE".
  ACCEPT ANSWER.
RETURN.

UNDELETE_RECORD:
  !
  !
  !
  TY "NOT IMPLEMENTED" FMT / A / END.
  TY "TYPE <RETURN> TO CONTINUE".
  ACCEPT ANSWER.
RETURN.
CANCEL_WITHDRAW:

! CANCEL/Withdraw Section
!
TY CLEAR_SCRN.
TY "**** ENTERING CANCEL/Withdraw Section ****" FMT / A / END.
TY "DO YOU WISH TO CONTINUE? ([Y] or N):" FMT A X $ END.
LET VALID_TMP_FILE EQ FALS.  ! SET TRANSACTION FLAG
CALL GET_IN_BUFFER.
LET ANSWER EQ $LEFT(IN_BUFFER, 20).
IF ($CAPS($TRIM(ANSWER)) EQ "Y") OR -
   ($LEN($TRIM(ANSWER)) EQ 0 ) THEN
   ! THIS INITIALIZATION MUST BE HERE TO INSURE THAT THE TEMP ARRAY
   ! IS BLANK WHEN GET_RECORD IS CALL
   CALL INIT_TEMP_INFO.
   CALL INIT_VARIABLES.
   CALL GET_RECORD.
IF (SYSNREC EQ 1) AND (ABORTED EQ FALS) THEN
   INIT 2 GLOBAL_TMP.
   INIT 3 LOCAL_TMP.
   LET LINE EQ 1.
   WHILE LINE LE NUM_FEELDS DO
      LET FEELD_DATA(LINE) EQ TEMP_DATA(LINE).
      LET LINE EQ LINE + 1.
   ENDWHILE.
   LET DONE EQ FALS.
   WHILE DONE EQ FALS DO
   TY CLEAR_SCRN.
   TY "CANCEL/Withdraw Information For:" FMT / A X $ END.
   TY $SUBSTR(FEELD_DATA(SSN_POS), 1, 3) FMT A "-" $ END.
   TY $SUBSTR(FEELD_DATA(SSN_POS), 4, 5) FMT A "-" $ END.
   TY $SUBSTR(FEELD_DATA(SSN_POS), 6, 9) FMT A $ END.
   LET LINE EQ END_OFFICIAL_DATA + 1.
   WHILE LINE LE NUM_FEELDS DO
      TY $RTRIM(FEELD(LINE)) FMT / A " = " A END.
      TY $RTRIM(TEMP_DATA(LINE)) FMT A END.
      LET LINE EQ LINE + 1.
   ENDWHILE.
   TY "DO YOU WISH TO MODIFY? ([Y] or N):" FMT A X $ END.
   CALL GET_IN_BUFFER.
   LET ANSWER EQ $LEFT(IN_BUFFER, 20).
   IF ($CAPS($TRIM(ANSWER)) EQ "Y") OR -
      ($LEN($TRIM(ANSWER)) EQ 0 ) THEN
      LET LINE EQ END_OFFICIAL_DATA + 1.
      LET STOP EQ FALS.
      REPEAT
         TY FEELD(LINE) TEMP_DATA(LINE) FMT / A " = " A END.
         TY "REPLACEMENT?:" FMT A X $ END.
         CALL GET_IN_BUFFER.
         LET TEMP_ANSWER EQ $LEFT(IN_BUFFER, 20).
         IF $TRIM(TEMP_ANSWER) EQ ESCAPE_CHAR THEN
            LET STOP EQ TRU.
         ELSE
            LET LINE EQ LINE + 1.
         ENDIF.
      ENDFOR.
   ELSE
      IF STOP EQ FALS.
      ELSE
         LET STOP EQ TRU.
      ENDIF.
   ENDIF.
ELSE
   IF STOP EQ FALS.
   ELSE
      LET STOP EQ TRU.
   ENDIF.
ENDIF.

ELSEIF $LEN($STRIM($TRIM($TEMP_ANSWER))) NE 0 THEN
    LET TEMP_DATA(LINE) EQ TEMP_ANSWER.
    LET ACCEPT IT EQ TRU.
    CALL ACCEPT_DATA.

    IF ACCEPT IT EQ FALS THEN
        LET TEMP_DATA(LINE) EQ FEELD_DATA(LINE).
        LET LINE EQ LINE - 1.
    ELSE
        LET VALID_TMP_FILE EQ TRU. ! SET TRANSACTION FLAG
    ENDIF.

ELSE
    TY "ITEM UNCHANGED".
ENDIF.

LET LINE EQ LINE + 1.
UNTIL (STOP EQ TRU) OR (LINE GT NUM_FEELDS).

ELSE
    LET DONE EQ TRU.
ENDIF.
ENDWHILE.

IF VALID_TMP_FILE EQ TRU THEN
    TY "RETURN THIS MODIFIED DATA TO THE DATA BASE ([Y] or N):"-
    FMT / A X $ END.
CALL GET_IN_BUFFER.
LET ANSWER EQ $LEFT(IN_BUFFER, 20).
IF ($CAPS($TRIM(ANSWER)) EQ "Y") OR-
   ($LEN($TRIM(ANSWER)) EQ 0) THEN
    CALL MANAGE_TMP_FILE.
    RELEASE 2.
    RELEASE 3.
    CALL TRANSACT_INTO_DATA_BASE.
ELSE
    RELEASE 2.
    RELEASE 3.
    FILE DELETE GLOBAL.TMP.
    FILE DELETE LOCAL.TMP.
ENDIF.
ENDIF.

ENDIF.

ENDF.

RETURN.

REPORTS:
!
!
!
TY "NOT IMPLEMENTED" FMT / A / END.
TY "TYPE <RETURN> TO CONTINUE".
ACCEPT ANSWER.
RETURN.
REPORT REVIEW:

!  TY "NOT IMPLEMENTED" FMT / A / END.
  TY "TYPE <RETURN> TO CONTINUE".
  ACCEPT ANSWER.
RETURN.

REPORT_PRINT:

!  TY "NOT IMPLEMENTED" FMT / A / END.
  TY "TYPE <RETURN> TO CONTINUE".
  ACCEPT ANSWER.
RETURN.

INQUIRY:

!  TY "NOT IMPLEMENTED" FMT / A / END.
  TY "TYPE <RETURN> TO CONTINUE".
  ACCEPT ANSWER.
RETURN.

INTERACTIVE:

!  TY "NOT IMPLEMENTED" FMT / A / END.
  TY "TYPE <RETURN> TO CONTINUE".
  ACCEPT ANSWER.
RETURN.

CANNED_INQUIRE:

!  TY "NOT IMPLEMENTED" FMT / A / END.
  TY "TYPE <RETURN> TO CONTINUE".
  ACCEPT ANSWER.
RETURN.
TOOLS:

! 
!

TY "NOT IMPLEMENTED" FMT / A / END.
TY "TYPE <RETURN> TO CONTINUE".
ACCEPT ANSWER.
RETURN.

FILE_TOOL:

!
!
!

TY "NOT IMPLEMENTED" FMT / A / END.
TY "TYPE <RETURN> TO CONTINUE".
ACCEPT ANSWER.
RETURN.

PRINT_TOOL:

!
!
!

TY "NOT IMPLEMENTED" FMT / A / END.
TY "TYPE <RETURN> TO CONTINUE".
ACCEPT ANSWER.
RETURN.

GET_IN_BUFFER:
REPEAT
    LET IN_BUFF_OK EQ TRU.
    ACCEPT IN_BUFFER.
    IF $LEN($TRIM(IN_BUFFER)) GT 20 THEN
        LET IN_BUFF_OK EQ FALS.
        TY "ERROR - INPUT LINE MUST BE LESS THAN 21 CHARACTERS.".
    ENDIF.
UNTIL IN_BUFF_OK EQ TRU.
RETURN.

CLEAN_UP:

!
!
!

TY "CLEANING UP AND QUITTING -".
TY "GOODBYE" FMT / A / END.
RETURN.
ASSIGN_ROOMS:

! 
!
!
TY "ASSIGN ROOMS - NOT IMPLEMENTED".
TY "TYPE <RETURN> TO CONTINUE".
ACCEPT ANSWER.
RETURN.

!
!
!
!
PL1022 END.