4-1980

Student Information Management

Randy Rohrbach
Western Michigan University

STUDENT INFORMATION MANAGEMENT

by

Randy Rohrbach

A Thesis submitted
in partial fulfillment of
the course requirements for
The Bachelor of Science Degree

Western Michigan University
Kalamazoo, Michigan

April, 1980

ABSTRACT

This project is the design and implementation of a special data-base for the Paper Science Department. The data-base allows for sorting and searching of student information based on 1.)name, 2.) age, 3.) number of credits, 4.) sex, and 5.) home state. The program was implemented in Pascal and therefore can be modified at a later date if additional features are described.

TABLE OF CONTENTS

This project involves the creation of a data-base which will contain the student information of all students in the Paper Department. Such information will include:

1. personal information
   a. name
   b. age
   c. sex
2. classes taken
3. grades

This information is all maintained on paper by the secretarial personnel of the Paper Department. Much time and effort is required to maintain this information. With the establishment of an organized data-base, the work required to update the student information will decrease significantly. This will leave the secretarial staff and the faculty free to do more important work.

Another important application of the data-base will be in the area of summer and/or permanent employment. The data-base will be designed in such a way that a cross-reference of the student information will produce a list of names that fit the requirements of the employers.

The second part of this project deals with another important aspect of the Paper Department. The Department is known as one of the hardest departments on campus. Therefore, the dropout rate for the Paper Department is high. However, the freshmen enrollment is increasing. Because of the limited number of faculty members the Paper Department will soon be forced to limit freshmen enrollment. The question of who to enroll and who to

deny enrollment to is a difficult one. Thus, the second part of this project is an attempt to find a correlation between student information such as 1.) GPA, 2.) PSAT scores, 3.) SAT scores, etc. and a student's likelyhood of making it through the program. The correlation search will involve a stepwise linear regression upon student information from the last four or five years. If any positive results are obtained then perhaps this will help the Paper Department in establishing an enrollment policy for incoming freshmen.

There are many questions and areas of concern which must be considered in the design of this data-base and the program that will manipulate it. Among these concerns are

1. information to be maintained and accessed.
2. how the information will be obtained, held, and manipulated within the program itself.
3. forms of output from the program
4. the language most useful in writing the program and in allowing the program to be readily understood and modified at a later date.
5. security concerning the student information and the program itself
6. backup files and failsafes in order to ensure that the data-base and program are safe from intentional or unintentional tampering.

All of the student information will be maintained in two data files. The first data file will be the file from which the student information will be obtained by the program. The second file will then, upon successful completion of inputing the data from the first file, be written into with the information that has just been input. Upon completion of execution of the program the first data file will then be written into with the now current (including any changes, deletions, and insertions) student information. In this way the Paper Department will have a current informa-

tion file and a backup file which will contain the student information from the last usage of the program.

Concerning the matter of security for these files and for the program a file will be created and permanently stored on the Paper Science computer number. This file which is called an ACCESS.USR file will serve two purposes.[1,2] The first function is to only allow those people whom the Paper Department wishes to have access to the student information and programs. The second function will be to create a data file containing a list of all people who attempt to access our information but do not have our permission. Such a list will be put in a file called ACCESS.LOG.

Having the ACCESS.USR file therefore guarantees that no outside users will be able to access any of the student information. Only the Computer Center will have access. If the Paper Department feels that even the Computer Center should not have access, then the data file can be easily altered so that even the Computer Center cannot look at it without a secret password.

In accordance with the above security precautions it is recommended that the Paper Department should buy a magnetic tape. This magnetic tape could be used to hold the program itself and once a semester the current student information file could be put on this tape. A large magnetic tape can hold almost 60,000 blocks of data.[3] The student information files will take up between 250 and 500 blocks. The Paper Department could then store at least 30 years worth of student records on one magnetic tape. This would eliminate a great deal of wasted filing cabinet space. To go back to a particular set of student data one need simply restore the information to the Paper Department number and then access that information with the pro-

gram.

Concerning the language which will be used for this program the DEC-10 system has several languages available.

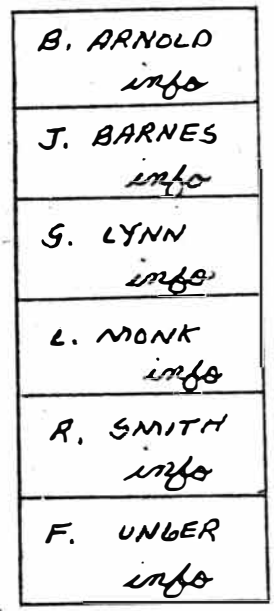
1. Fortran
2. Cobol
3. Algol
4. Pascal
5. Simula
6. Lisp
7. Snobol
8. Basic

Of these languages, Fortran and Cobol are the most well known. However neither of these languages will be used for this project. First of all, Fortran is a language primarily used with engineering and calculation problems and is not designed for data processing.[4,5,6,7] Cobal certainly has the file handling capabilities, however the language is difficult to analyze and follow.[8,9,10] Also, programs cannot be written in COBOL in a structured manner. All of the other languages except PASCAL are fairly specialized and could only be applied to this particular problem with a great deal of difficulty.[11,12,13,14,15] Therefore, the language which will be used to implement the program will be PASCAL. PASCAL is both versatile enough to handle this type of data processing project and can be written in a forthright and structured manner which can be easily modified at a later date.

Finally let us turn to the description of the data structure that the program will be based on. There are basically two forms of memory access

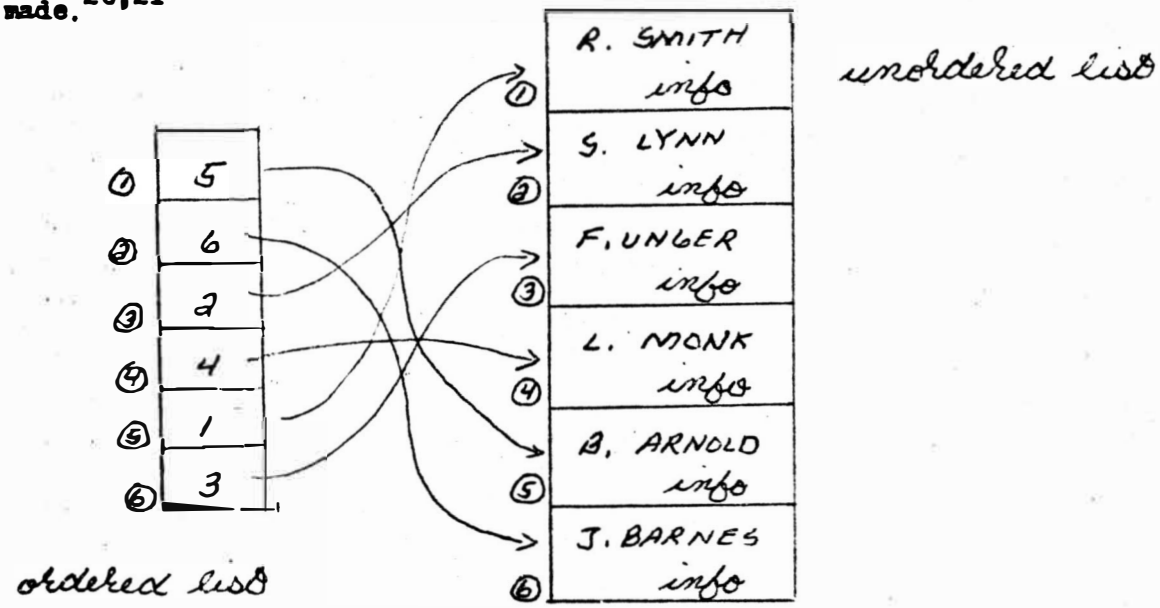
and manipulation. These forms of storage are called 1) sequential and 2,) linked (non-sequential).[16,17,18]

Sequential storage is as it sounds, a consecutive logical ordering of information. The information may be ordered according to numerical or alphabetical information.



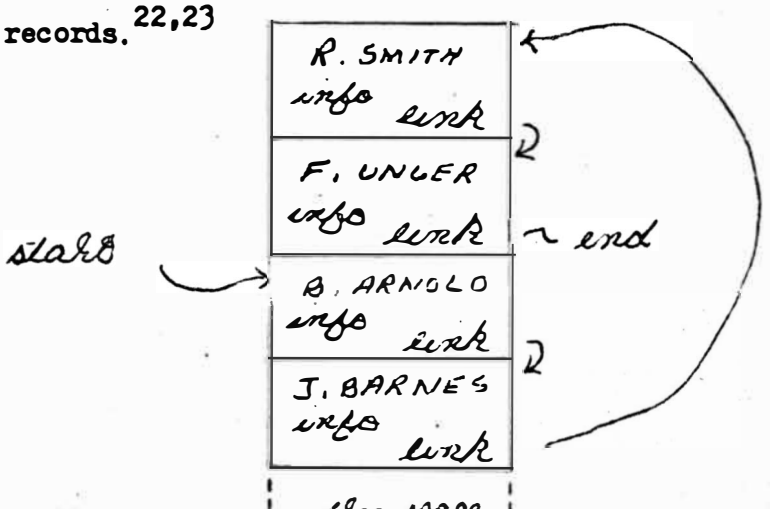
There are two basic disadvantages of a sequential data-base.[19] First of all the program must necessarily reserve a large amount of space in case the data-base is large. But if it is not then a lot of computer memory will be wasted. The other major disadvantage with sequential storage and access is in the manipulation of the data. Suppose for example, there are 200 student records taking up 4000 memory locations. If we have to insert a student record at the start of the list then the program will have to transfer at most 4000 entries. So, depending upon where in the list a record is to be inserted or deleted many memory transfers will have to be made. This can be quite inefficient and costly.

To minimize the problem of record transfer one could use a variation of the sequential storage theme. The student records themselves are no longer ordered but instead an ordered list of pointers to the student records is made.[20,21]

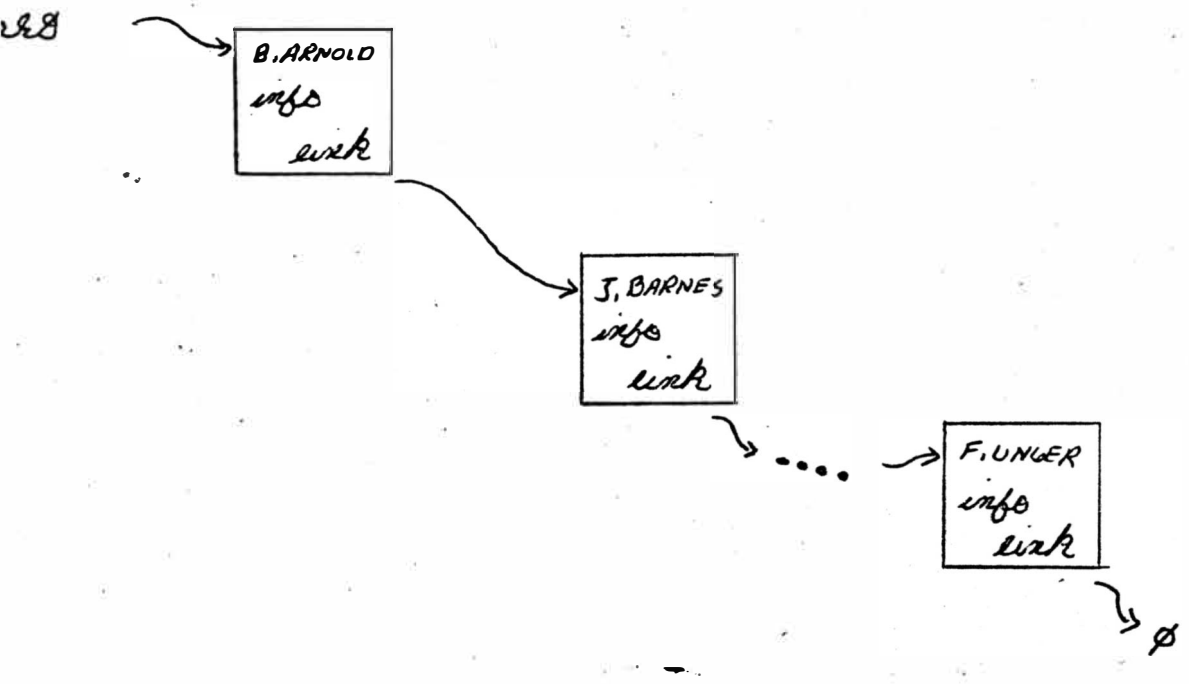


So using our previous values to insert a student record the program will simply add the record to the bottom of the record list and insert the pointer to this record in the ordered list. Thus at most the program will have to shift 200 memory locations. However, the unordered list will have to be operated upon for garbage collection of the unused space.

A way to reduce memory transfer even furthur is by the use of a data structure called a linked list. The list is kept in order by link fields in each of the student records.[22,23]



This is a much more efficient scheme. To insert a name and record the program need only add the information to the free space at the bottom and change the two appropriate links. Garbage collection is still needed and an initial memory block must be allocated. The solution to this overhead problem lies in the use of PASCAL as the language the program will be written in. PASCAL has the ability to allocate and deallocate memory space upon request. This will allow the program to use a minimum amount of memory while retaining all of the student information. Therefore, the data structure will now appear something like the following diagram.[24]



The linked list data structure will allow for easy insertions or deletions of student information. The sorting and comparison routines will then simply traverse the linked list and create new linked lists of answers.

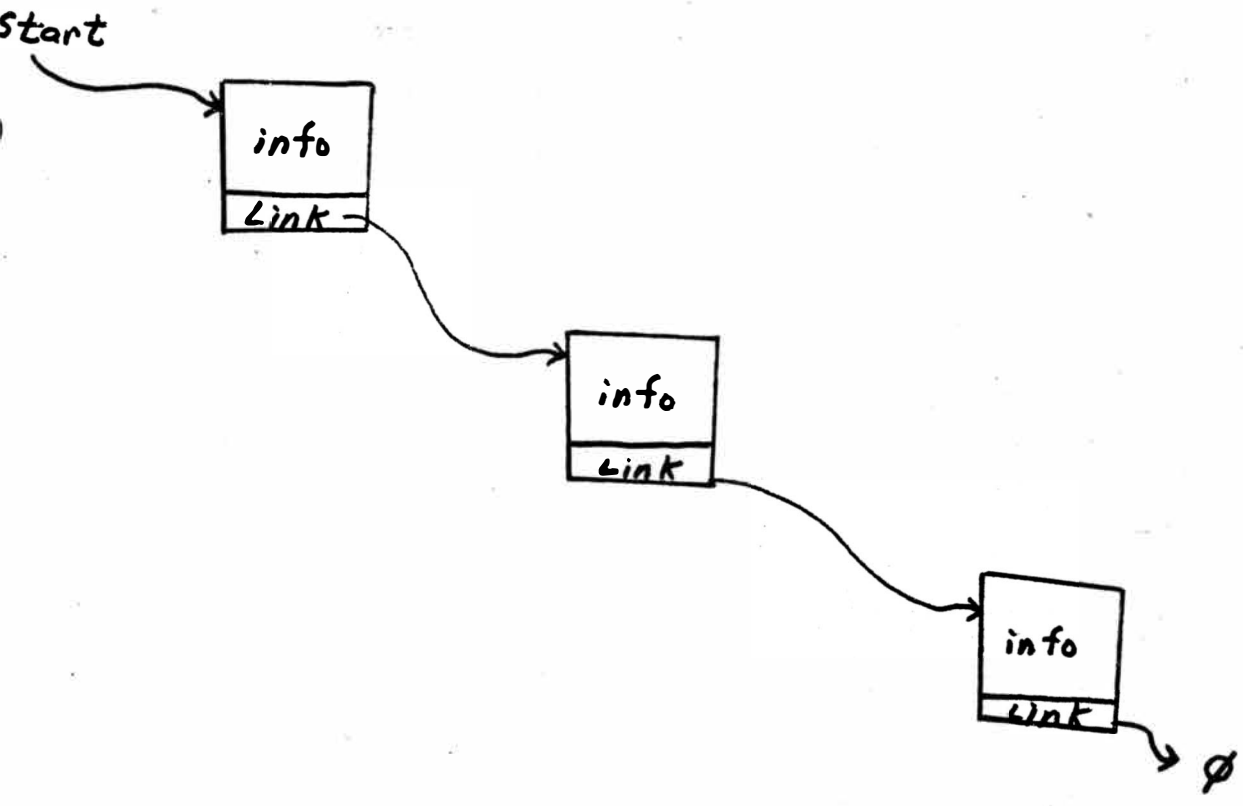
To sum up, the above considerations indicate that a linked list type of data structure implemented in the PASCAL language will suit the needs

of this project. Because it will be written in PASCAL it will be readable, understandable, and most importantly it can be modified at a later date.

EXPERIMENTAL WORK

In the design and writing of the program the modular approach was taken. This approach leads to a more logical and understandable program. Along with the modular emphasis an I/o sectioning approach was taken. This means that the I/o routines are segregated from the rest of the program so that if a general change needs, at some later time, to be made to the I/o then these seperate routines will be the only ones which will need to be changed.

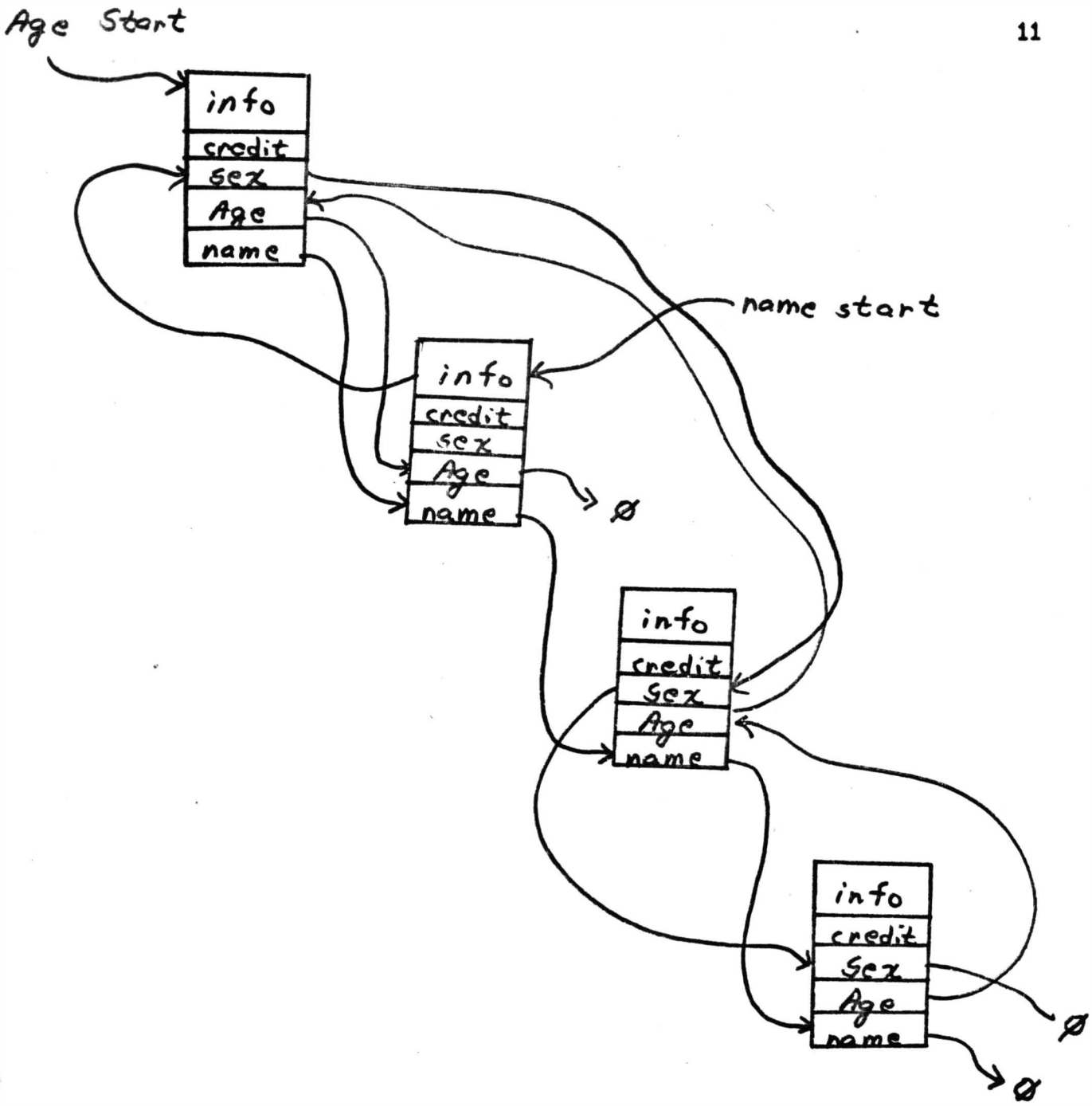
Only one major design change was made before the actually writing and coding of the program was undertaken. This change involved the redesigning of the original linked list structure. The original linked list was a simple non-circular linked list containing only one link.



However, upon further consideration this structure proved to be inadequate in terms of sorting efficiency. If the list had to be sorted then the program would have to traverse the list several times to obtain the proper list of names. This is not in itself inefficient, however, if that same sort was called again the program would have to traverse the list in the same way to find the correct order of names. To eliminate this unnecessary searching and traversing a new data structure called a multilinked list was implemented. This structure is exactly like the original with the exception of the type and number of linkages. Instead of one link, there are five links in each node of the list. These links refer to:

1. alphabetic order
2. sex
3. age
4. home state
5. number of credits taken

In this way when a traverse and sort of the list takes place, the appropriate links will be connected. Then, upon a second request for that sort, the program simply follows the particular link for that sort and only has to make one pass through the student list. This results in a considerable increase in efficiency, roughly on the order of 250%. The individual sorting links will hold their values until a deletion or insertion of a student into or from the data base occurs. In the event of a deletion the links may be quickly and easily reordered. Therefore the efficiency of sorting is still maintained upon deletion of a student. If however, a student is being added to the data base then the sorting links will have to be reordered as they were originally.



During the implementation of the program only two major problems developed. The first problem involved the interactive teletype I/o package of the Pascal language. Pascal allows the user to very easily send alphanumeric information to the user's teletype. However, to input information from the user's teletype the program must manually take con-

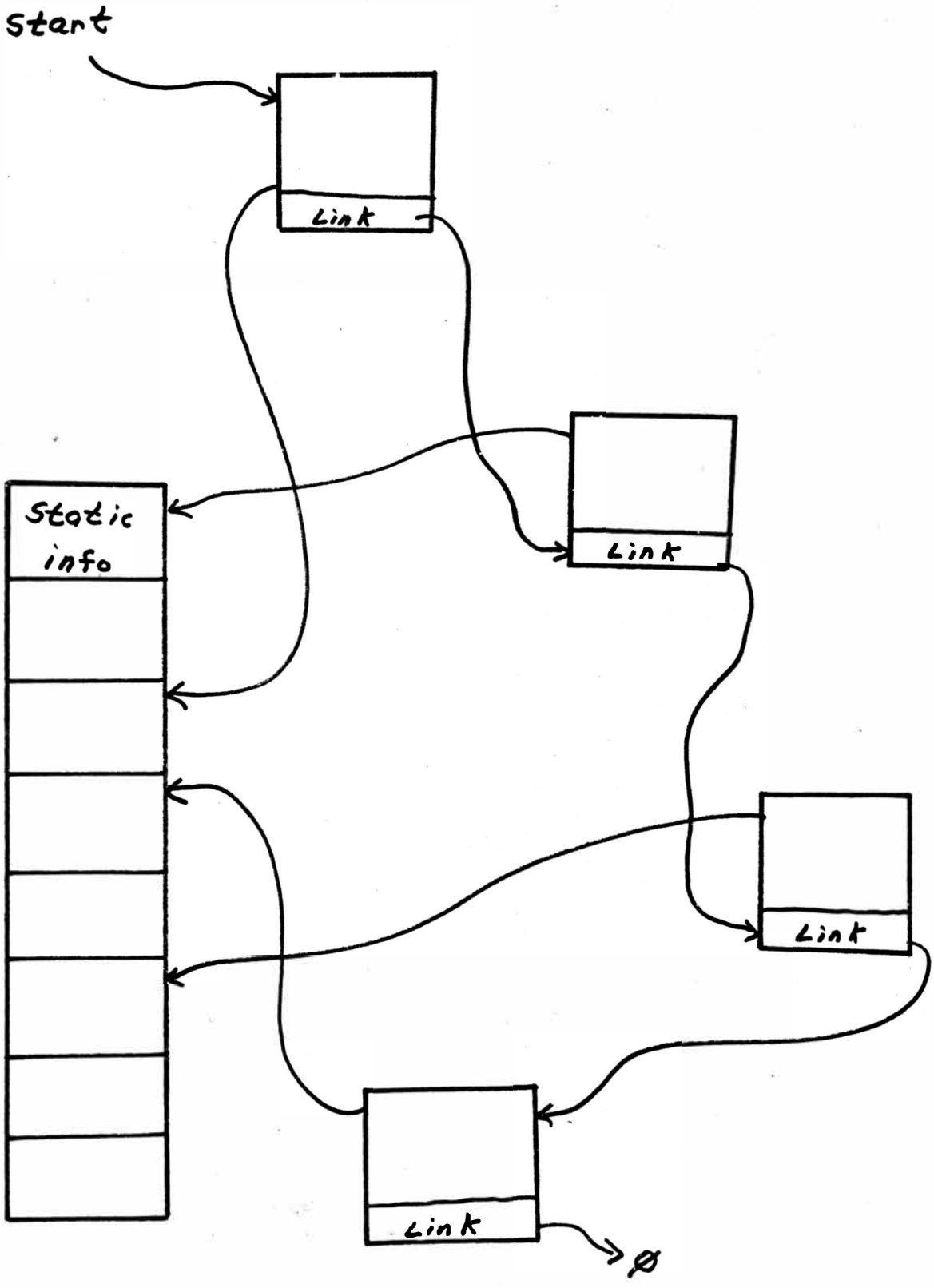
trol of the teletype. This takeover of the teletype requires more memory from the computer. If enough interactive teletype I/o is performed the amount of memory that the individual user is allowed will be exceeded and the program will fail.

One approach to this problem is to limit the user to a certain amount of teletype I/o before ceasing execution of the program. This approach is too restrictive. Instead, a special type of I/o controller was implemented. This controller allowed for the reuse of previously allocated computer memory. This way there is no physical limit to the amount of interactive teletype I/o the user may perform.

The second major problem also involved the amount of memory that the computer could make available to the program. In an attempt to create single and self contained nodes (records) for the multilinked list the memory allocation of the program was exceeded. This was due to the failure of the Pascal compiler to efficiently allocate space for the program. The easiest and most understandable approach to solving this problem was to create parallel and distinct memory storage. The parallel memory would contain certain relatively fixed portions of the student information. This memory is associated with the nodes of the multilinked list by the use of pointers contained in the nodes of the linked list. See Figure 1.

After overcoming these implementation problems the program was thoroughly tested by several programmers. Unless one inadvertantly changes the program, operation of the program will be relatively easy and error free. If however changes to the program are necessary, then they should be made with care and the guidelines of the User's Manual, provided with the program, should be followed.

Figure 1.



LITERATURE CITED

1. DecSystem-10 Monitor Calls, 1979, Digital Equipment Corp., Sect 8.6-8.8.

2. DecSystem-10 Hardware Reference Manual, 1978, Digital Equipment Corp., Sect. 7.4-7.7.

3. Magnetic Tape Usage, WMU Comp. Cen. Documentation.

4. Rich Didday, Rex Page, Fortran For Humans (St. Paul: West Publishing Co.), 1977, p. 326-345.

5. Mary McCammon, Understanding Fortran (New York: Thomas Crowell Co.), 1968, p. 97-102, 232-244.

6. Laura Cooper, Marilyn Smith, Standard Fortran: A Problem Solving Approach, (Boston: Houghton Mifflin Co.), 1978, p.25-27, 145-171.

7. Bob Thomas, Structured Fortran, (New York: Prentice Hall), 1978, p. 217-227.

8. H. Cashman, ANSI COBOL - Introduction to Computer Programming (Fullerton: Anaheim Publ. Co.), 1977.

9. Asad Khailony, Claude Duplissey, Cobol for Small and Medium Size Computers (Boston: Houghton Mifflin Co.), 1976, p. 36-44, 243-274.

10. Carl Feigold, Fundamentals of Structured Cobol Programming (Dubuque: WMC Brown Co. Pub.), 1978, p.328-506.

11. Grahoun Birtwistle, Lans Enderin, Mats Ohlin, Jacob Palme, DecSystem-10 Simula Vol 1, (Westerm Michigan University: Kalamazoo) 1978.

12. Lynn H Quam, LISP 1,6 (Stanford: Stanford Artificial Intelligence Language), 1969.

13. R.E. Griswold, J.f. Poage, I.P. Polonsky, The Snobol-4 Programming Language (Englewood Cliffs: Prentice Hall), 1971.

14. Daniel D. McCracken, A Guide to ALGOL Programming (New York: John Wiley & Sons), 1969, p. 23-34.

15. William F. Sharpe, Nancy L. Jacob, Basic (New York: The Free Press) 1971 p. 95-173.

16. Donald Knuth, Fundamental Algorithms (Reading: Addison-Wesley Pull. Co.), 1973, p. 228-295.

17. James Martin, Computer Data-Base Organization (Englewood: Prentice-Hall Inc), 1977.

18. Vivian C. Prothro, Information Management Systems: Data-Base Primer (New York: Petrocell: Pross), 1976, p. 24-76.

19. John L. Pfaultz, Computer Data Structures (New York: Mc Graw Hill Col.), 1977, 102-157.

20. Ibid. p 203-273.

21. William D. Haseman, Andrew B. Winston, Introduction to Data Management (Homewood: Richard Inwin Inc.), 1977, 116-139.

22. Opcit. Donald Knuth, p. 317-349.

23. Opcit. John Pfaultz, 182-207.

24. Opcit. Donald Knuth, p. 317-349.