FORM EDITOR SYSTEM

by

JONY CHANG

B.S., National Taiwan University, 1978

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1984

Approved by :

[illegible]

Major Professor

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.0 Background

Office automation is becoming a very important subject of computer science research. The office is the part of business which handles information dealing with such operations as accounting, payroll, and billing. Office work involves information-handling activities such as text editing, form editing, filing documents, performing simple computations, verifying information, and communicating within the office and between offices. An automated office attempts to perform or assist office workers in performing some of the functions of an ordinary office by means of a computer system [2]. One goal of office automation is the design of application systems and facilities to help improve personnel efficiency.

Forms play a central role in the technologically advanced business world. Day-to-day business activities generally involve the use of forms. Most business organizations use forms as communication tools. Forms can be thought of as texts which require certain values to be filled in certain slots called fields. A filled form can be thought of as a document, and is called a form instance (i.e., a form instance is a single filled form that is one of a group of forms of the same style, all having the same field names). Blank forms are usually called form structures. Forms arise because there are many documents which are similar in that they have basically the same form structure. They differ from each other only in terms of the filled entries. These entries are called field data.

Computerization has led to the development of so-called 'intelligent' electronic forms which are preferred over the ordinary paper form due to the following advantages : An intelligent form can automatically retrieve data from data base (a data base is a collection of many data files), which may reside in other work stations, for certain fields on the form; it can make calculations for the extension or total item field; and it can have routing information associated with it such that they will be delivered automatically to the next person on the routing list. Electronic forms provide a very good interface for the clerical facet of office automation system such as handling personnel information, purchase orders and airline tickets.

Forms can serve as a high-level protocol for information communication. Data can be extracted from a form for transmission. Transferring information from one or more forms to another is a common activity in an office environment. The extraction of information from forms for various purposes such as report generation is another common activity. Scrutinizing data given in forms to ensure that they fit certain criteria is an exercise practiced in almost all organizations [11]. Therefore, an important part of an office automation is a system for processing forms as a natural and effective interface between an office worker and information. To carry this out, one requires the means to specify the processing of data in the forms. Basically each form processing specification contains the following : (1) the name of the input form, the operation to be performed (e.g., enter, create, copy, send, etc.), and the name of the output form; (2) a description of the field data constituting the form (called field types) and the constraints to be imposed (such as allowance of data ranges, signature, and unchangeable fields); and (3) qualifications for the intended process which may include the source of data, the conditions to be applied in selecting form instances, etc. Once the form specification is defined, the flow of forms within the system and the functional relationships among them can be defined.

To implement form processing in a system, one has to design and create form structures and store form templates in the system. A form template is a definition of the form structure, describing the types, lengths, and the contents of fields in a form structure. A facility which does this is called a form editor system. A form editor system also allows a user to modify an old form structure, print a hard copy of the form structure, display it on the screen, or delete the form template and form structure from the system.

1.1 Review of Related Research

Forms have been the subject in several completed research projects. These studies gave rise to the following systems : Business Definition Language (BDL) [5], System for Business Automation (SBA) [12,13,14], Officetalk-Zero [2], Office Forms System (OFS) [7,9,10], Forms Programming System (FPS) [3], Structured Message System (SMS) [11].

1.1.1 Business Definition Language (BDL)

IBM's Business Definition Language (BDL) was one of the first to make use of forms as information processing specification. In BDL the basic data structure is a form. Forms are the templates with which specific documents are produced. They consist of preprinted information plus specifications about the data used to fill them out. Using BDL, an applications programmer defines a form by specifying its name and its width and height in character positions. This is followed by specifying its preprinted information : what a blank document would look like. Preprinted information consists of column headings, boxes, and the like. This information is represented directly by drawing it on a graphic device over an array of rectangles used to show character positions.

The remainder of a form's definition consists of several definitions of the various properties of the form. Each property has its own method of definition, but all are done by adding information to the form shown on the screen. A description of these form definition properties of the form such as fields, data type, and data formats as follows : (1) Field definition : the form is filled out with the names of the fields. Field names may not be longer than the field they name; if they are shorter, they are extended with a special extender character. Thus a field name specifies not only the location of the field but also its size. (2) Data type definition : the form is filled out with data type names for each field. Specific data types such as address, date, inches, and dollars are available. This permits better error detection and automatic conversion between fields of like dimension but different units. (3) Data format definition : the form is filled out with sample data generated by the system to show an example format for each field. The format of any field can be changed by the programmer by selecting an alternative from a list of samples generated by the system. Once forms are defined, their flow within the system and the precise functional relationships among them can be defined.

1.1.2 System for Business Automation (SBA)

IBM's System for Business Automation (SBA) also makes use of forms as information processing specification. An SBA user views and manipulates a two-dimensional picture of tables, forms, and reports on a display terminal. The SBA project produced Query-By-Example (QBE), a query language based on two-dimensional tables or forms [12]. A recent extension of QBE is the Office-By-Example (OBE) [13]. OBE is a two-dimensional system that is an attempt to mimic manual procedures of business and office system. OBE is a superset and natural extension of the QBE data base management system, and contains features from IBM's earlier work on a SBA. Although the fundamental data object in QBE is the table, in OBE the objects are more general, and include letters, forms, reports, charts and graphs.

A form in OBE is a generalization of a table and is viewed as a data object. A form skeleton is a derived object and must first be constructed by the users before defining its various fields. Once the skeleton form is constructed, the user links its fields to fields in columns of an associated table(s), and defines these fields almost as though they were table fields. A form is considered to be a data base object and thus can capture data, and its data fields can be modified. Thus, The user first graphically structures the image of the form on the screen, mimicking manual construction, then enters example elements in the fields to be defined.

1.1.3 Officetalk-Zero

At the Xerox Palo Alto Research Center, a group of researchers have developed Officetalk-Zero, a prototype office information system based on forms. Officetalk-Zero, referred to hereinafter as officetalk, is implemented in an environment of multiple minicomputers interconnected by a high-speed communication network. Officetalk is based on the data object of single page forms and files of forms. Intercommunication is accomplished by electronically passing forms among the work stations. The user is shown an image of a desktop (a desktop resembles the top of an office desk) containing parts of forms and employs a mouse (a mouse is a pointing device) to manipulate the forms on the desktop.

An officetalk desktop contains the following four forms, which are called file indexes : (1) the in-basket, an index of incoming mail; (2) the out-basket, an index of mail to be sent and mail that has been recently sent; (3) the forms index, forms that the user has saved; and (4) the blank stock index, an index of the set of available forms. Each file index entry contains several fields : one field names the file, an action field specifies a command which can be applied to that file entry, while other fields list other information. A file index form is special in the sense that it contains a field on the form itself which allows command invocation. Ordinary forms do not contain an action field (instead, all commands are invoked from the window menu).

To implement a particular officetalk application, a tailored set of blank forms must be designed and entered into the database. Officetalk provides a forms editor which allows one to specify the graphical design of a form and the style of each field on the form. The forms editor requires that the newly designed forms satisfy certain conditions, such as no overlapping fields. It also permits certain fields to be designated as signature fields.

1.1.4 Office Form System (OFS)

At the University of Toronto, a group of researchers whose primary interest is database management have begun to look at office automation and have chosen to base their approach on forms. Forms can be viewed, at two different extremes, as text and formatted data. By handling forms there is a natural way to incorporate limited text capabilities, together with ways to structure data for further retrieval and processing. A correspondence can be established between a form instance and a record of values appearing in the form. In this way, we can integrate office form procedures with regular data processing and data management dealing with records.

Forms are manipulated by users via stations. A station can be a personal computer or a terminal running on a computer. Form processing in the system is provided by OFS. Forms consist of form fields and have dispersed printed text. Form fields take values from particular domains. Form fields can not constitute repeating groups. Form fields are declared to be one of three distinct types. The first type insists that the value in the field is entered at form instance creation time and cannot change afterward. For instance, a person's name in an expense account is of the first type. The second type allows delayed entry of the field value but once the value is entered it cannot be changed. For instance, a manager's authorization on a form is of the second type. Finally, the third type allows the field to be updated. For instance, a field recording the status of a form as it passes through stations can be of the third type.

1.1.5 Forms Programming System (FPS)

The Forms Programming System is based on forms and the flow of forms. A form consists of blanks to be filled in, keywords or keyword phrases that indicate what information should be filled in the blanks, and explicit or implicit relationships among the various items of information on the form. An applications programmer working with FPS accomplishes a programming task by sketching an applicable form on a display screen, by specifying conditions under which the form is activated and processed, and by indicating for each item of information on the form the source and destination of its data. The programmer may also need to interact with the system to assist in the determination of entry types for blanks and the formulation of relationships among items on the form. Once forms are designed, information may flow to them from a user, from the corporate database, and from other defined forms. Such information may be combined to compute additional entries and may then be displayed for the user, may invoke database updates, and may be transmitted to other forms or work stations in the system.

In general, a form can be cast into a Pascal record definition : keywords are field names and are associated with types defined appropriately for the domain of expected entries. For each entry whose values is not obtained from a computational relationship, an appropriate input or database retrieval statement is defined; output can be considered to simply be a filled out form.

1.1.6 Structured Message System (SMS)

The system is composed of a number of logical units called stations which may be grouped together on physical units called nodes. Each user of the system operates a single station. Each station contains a station database that is used to store information associated with the station. For each message type (such as a meeting announcement or a referee report) that is known by the system there is a relation (A relation is any subset of Cartesian product of one or more domains. A Domains is simply a set of values.) in each station database. Tuples (tuples are the members of a relations) within such a relation correspond to instances of the message type. Attributes (attributes are the components of a relation) of this relation correspond to the fields of the message type.

A user defines a message type by creating a display template. Once the type has been defined users may create message instances by 'filling in' the template. During message instance creation the system generates a globally unique identifier for the new message instance. This identifier (known as the message key) is permanently attached to the message instance and cannot be modified. There are constraints upon the modification of particular message fields. There are four types of message fields. The first three types are all user supplied. Fields of the first type must be entered during message creation and then cannot be modified. Fields of the second type cannot be modified once entered. Fields of the third type have no restrictions in terms of operations. The fourth type is used for automatic fields (such as dates, signatures, message keys, and other functionally determined fields) which are generated by the system when required and cannot be modified by the user.

1.2 The Problem

The form structure in BDL is represented directly by drawing it with a graphic device, the terminal for user with this system must have modest graphic capability. The field length is fixed by the field name, and the data field type is not flexible. In SBA, forms are syntactically more restrictive than other form based system and also require the user to have some understanding of the underlying data base model.

Although Officetalk has the form editor, it still use the graphical design of a form structure, and it does not support flexible operations on data base, and the fields on a form does not permit to be designated as flexible type. In OFS, they plan to implement a facility for outlaying forms and designing form blanks via a terminal. At this point, the formats of a form blank for both display and print outs are not very flexible. In SMS system stations are based on PDP 11/23's running UNIX. Each PDP 11/23 has up to 64 K bytes of memory and a hard disk. In contrast with SMS, the Form Editor System is an experimental system based on a small microcomputer without hard disk storage capabilities. It tries to use a small microcomputer with less memory and floppy disk to implement the form based concept in an application system.

With FPS system, a programmer would sketch forms, indicate the source and destination for each data item, and give activation condition. FPS would formulate expressions for computed entries and access paths for data base queries and would interact with the programmer to resolve any ambiguities and incomplete specifications. In this kind of situation, it is conceivable that an application specialist who is knowledgeable about the system is needed.

Most of the previously described systems have a complicated system structure.

The user has to spend much more time to understand their operation before the system can be used. Therefore, it is not understandable to a non-technician like a secretary. Both BDL and SBA are IBM products; they require a mainframe computer which has a large memory. The corresponding cost for a system is high. Officetalk, OFS, and SMS use multiple minicomputers interconnected by a communication network. Both the initial and maintenance cost are thus too high for a small business to afford these systems.

With the rapid decline in hardware cost, microcomputers are now easily affordable. The microcomputer is bringing the advantages of computerization to all levels of business. Computerization provides a convenient way of handling business forms. However, although the small company businesses have the resources to purchase small computers, they are often faced with the use of several non-integrated software systems. Few are based on the form concept. Since the form is an important user interface in the business world, there is a need for a form editor system for the small company using a small microcomputer which does not have hard disk storage capability.

1.3 The Solution

This paper describes a form editor system which can be effectively used on a small microcomputer without hard disk storage. Due to the small disk storage, and no facility available to connect other work stations, the system described is a semi-intelligent form, which is capable of retrieving data from data bases and make some very simple calculations but is not capable of manipulating itself among other work stations.

The Form Editor System allows design of a form structure by use of the create form function in the system without use of a graphic's device. The Form Editor

System is a user-friendly system which is menu-driven for easy operation. When the menu is displayed on the screen, the user just strikes the command key for the function he wants. There is a help menu for the user who is not familiar with the system. A person who has the computer programming background possibly will not need the user's manual. Therefore, it is understandable to a non programmer.

There were two kinds of microcomputers available for use with this research. These were the Zenith-100 and the Columbia Data Products (CDP) Multi-Personal Computer (MPC). The CDP is an IBM compatible microcomputer, and having more compilers, such as CP/M-86 Pascal, IBM MS/DOS Pascal, and TURBO Pascal. It was therefore chosen for this project.

In this research, we had the option of using the following languages : Ada, BASIC, and Pascal. Pascal, a high level and general purpose language, was chosen for the implementation of this research. Ada language is a new language with stronger type checking and extensive data abstraction capabilities, but Ada/Janus is only a subset of compiler of Ada language available at KSU. BASIC is not a structured programming language. If complicated programs are written in BASIC, they are difficult to read and maintain by others. Pascal, on the other hand, allows easy modification of the system to a more intelligent system. The language Pascal embodies the constructs for use in structured programming so that it is easy for programmers to understand a program written by others. It also has the following advantages : (1) better control structures, e.g., CASE, IF-THEN-ELSE etc.; (2) strong type checking; (3) a variety of data structures, including RECORD and SET; and (4) user defined types. TURBO Pascal was used for the implementation because it has its own editor and it takes less compilation time than other Pascal compilers available to this research (e.g., CP/M-86 Pascal, and IBM MS/DOS Pascal) [15].

1.4 Organization of the Report

Chapter 2 introduces the system capabilities, and how the user interfaces to the system. The overall system design with data structure and the detail design of each module is presented in Chapter 3. Chapter 4 explains the programming language, coding, testing, and the extent of implementation. The limitations of the system and the directions for future work are discussed in Chapter 5.

Finally, appendices are provided for those who intend to use the system or modify the program. Appendix A contains the user's manual. Appendix B contains the listing of the program. Appendix C contains the listing of screen display files.

CHAPTER 2

SYSTEM CHARACTERISTICS

2.0 System Capabilities

The Form Editor System was designed to allow the user to create the form structure, modify the form structure which was previously created, store the data field type to create a form template, print the form structure on the printer or display it on the screen, delete the form template (the accompanying form structure will also be deleted) from the system. From hereon the word 'form' shall be taken to mean a 'form structure' unless otherwise specified.

To start the system, one has to follow three steps : (1) Put the system disk on 'A' disk drive and load MS-DOS by entering the date and time. (2) Put the working disk on 'B' disk drive. (3) Type 'form' (the name of program) and hit the 'return' key. The system first displays "WELCOME TO FORM EDITOR SYSTEM" on the screen (see Figure 2.0.1) and then displays the main menu on the screen (see Figure 2.0.2). On the main menu, the user has six choices of commands: 'C', 'M', 'S', 'P', 'D', and 'E'. CREATE(C) allows the user to create a form within the limits of 75 characters by 16 lines. MODIFY(M) allows the user to retrieve a previously created form and allows the user to add, change, insert or delete any field on it. STORE(S) allows the user to store the field types of a form to create the form template. PRINT(P) allows the user to print out the form on the printer or display it on the screen. DELETE(D) allows the user to delete the form and the form template from the system. EXIT(E) allows the user to exit the system. The command 'E' is the only command which can exit and stop the system.

2.1 User Interface

What makes a good user interface ? There are four criteria for the design of

```
******************************************************************************
*                                                                            *
*                                                                            *
*  *       * ******** *           *******     *****    *      **  ********   *
*  *   *   * *        *         *           *       *  * *   * *  *          *
*  *  * *  * ******** *         *          *         * *  * *  *  ********   *
*  * *   * * *        *         *           *       *  *   *   *  *          *
*   *     *  ******** ********    *******     *****    *       *  ********   *
*                                                                            *
*                                                                            *
*                         *********     *****                                *
*                             *       *       *                              *
*                             *      *         *                             *
*                             *       *       *                              *
*                             *         *****                                *
*                                                                            *
*                                                                            *
*                                                                            *
*                                                                            *
*                    F O R M   E D I T O R   S Y S T E M                     *
*                                                                            *
*                                                                            *
******************************************************************************
```

Figure 2.0.1 WELCOME Display

```
 ______________________________________________________________________________
|                                                                              |
|                   ****************************                               |
|                   **      MAIN    MENU      **                               |
|                   ****************************                               |
|     THERE ARE FIVE FUNCTIONS IN THE SYSTEM :                                 |
|                                                                              |
|     FORM CREATE - ALLOWS YOU TO CREATE A FORM WITHIN THE LIMITS OF           |
|                   75 CHARACTERS X 16 LINES.                                  |
|                                                                              |
|     FORM MODIFY - ALLOWS RETRIEVAL OF A PREVIOUSLY CREATED FORM AND          |
|                   ALLOWS YOU TO ADD, CHANGE OR DELETE ANY FIELDS ON IT.      |
|                                                                              |
|     FORM STORE - STORES THE FIELD DEFINITION AND THE FORM FOR LATER USE.     |
|                                                                              |
|     FORM PRINT - PRINTS A COPY OF THE FORM ON THE PRINTER OR DISPLAYS IT.    |
|                                                                              |
|     FORM DELETE - DELETES THE NAMED FORM FROM THE SYSTEM FOREVER.            |
|                                                                              |
|     EXIT THE SYSTEM - ALLOWS YOU GET OUT OF THE SYSTEM.                      |
|                                                                              |
|------------------------------------------------------------------------------|
|     COMMAND :  (C-CREATE  M-MODIFY  S-STORE  P-PRINT  D-DELETE  E-EXIT)      |
|______________________________________________________________________________|
```

Figure 2.0.2 Main Menu

any office automation system user interface [6] : (1) It should be natural; that is, the user should be able to communicate with the system in a vocabulary and in a style that is comfortable for him and that reflects the context in which he works. (2) It should be easy to learn, in order to minimize frustration, resistance, and training costs. The user should feel that he is in control, that the system will react in predictable ways to his commands, and that help is available when he needs it. Learning should be system-assisted and incremental. (3) It should be easy to use. The user should not have to struggle to accomplish what he wants to do, and should not feel that the system gets in the way of his work. (4) Most important, it should be consistent.

The system meets the criteria and is a 'user friendly' system. The following subsections explain how forms are created, modified, and stored. Other functions can be referred to in the User's Manual (Appendix A).

2.1.1 Creating a Form

The FORM CREATE function allows the user to create a form within the limits of 75 characters by 16 lines. To design the form required, one simply needs to duplicate the image of the paper form on the screen. After issuing a command 'C', one fills in the name of the form being created. The form name must be a unique for all other form names in the system. A ':' must follow the field name, and the field length counted by the underscores length which must follow a blank. To exit the CREATE function, one types '*' on a new line of the screen. On exiting, the form is automatically saved as a form mapping file (a form mapping file is a file that stores the form structure in the system).

Figure 2.1.1 illustrates how a form named 'Address' is created. The form name 'Address' is typed first. When the form name is unique to the system, the

```
--------------------------------------------------------------------------
|                        **  FORM CREATE  **                             |
| FORM-NAME : Address                                                    |
|------------------------------------------------------------------------|
| Name:________________                                                  |
| Address:____________________________________                          |
| City:_____________    State:__________    Zip:_________               |
| *                                                                      |
|                                                                        |
|                                                                        |
|                                                                        |
|                                                                        |
|                                                                        |
|                                                                        |
|                                                                        |
|                                                                        |
|                                                                        |
|                                                                        |
|                                                                        |
|                                                                        |
|------------------------------------------------------------------------|
|a. Field name must be followed by ':'. b. Field length counted by underscores|
|, followed by a space. c. Type '*' on the new line to terminate the creation.|
|------------------------------------------------------------------------|
```

Figuer 2.1.1 Address Form

```
--------------------------------------------------------------------------
|                        **  FORM CREATE  **                             |
| FORM-NAME : Payroll                                                    |
|------------------------------------------------------------------------|
|                                                                        |
|       ____________________________________________________             |
|       |                                                   |            |
|       | Name:____________________                         |            |
|       | Address:_____________________________             |            |
|       | Pay-Rate:______________     Deduction:______________|            |
|       | Net-Pay:__________________                        |            |
|       |                                                   |            |
|       |___________________________________________________|            |
| *                                                                      |
|                                                                        |
|                                                                        |
|                                                                        |
|                                                                        |
|                                                                        |
|------------------------------------------------------------------------|
|a. Field name must be followed by ':'. b. Field length counted by underscores|
|, followed by a space. c. Type '*' on the new line to terminate the creation.|
--------------------------------------------------------------------------
```

Figuer 2.1.2 Payroll Form

cursor moves to the first line of the create form area (the create form area is 75 characters by 16 lines area). One then types the first field name, e.g., 'Name', followed by a colon and then as many underscores (in this case, that is 20 underscores) that are needed to fill the length of the field, followed by a blank for the delimiter. The return key is pressed and the same routine is performed for the 'Address' field. The process is repeated until the form is created. To terminate the process, the user types a '*' on a new line on the screen to denote the end of the creating form. In Figure 2.1.2, another example, the 'Payroll' form, was created using a box line for the border of the form.

2.1.2 Modifying a Form

The FORM MODIFY function allows the user to retrieve a previously created form, and to add, insert, change or delete any field on it. First the name of the form which is to be modified must be typed. It must be a form name which is already in the system. If the form name does not exist in the system, the system will display an error message for the user to try again. The form can be modified line by line on the screen by using the following symbols :

Symbols	Meaning
'D'	deleting the current line
'I'	inserting a new line or lines
'A'	adding a new line or lines to the end of form
'Y'	changing the current line
'N' or <CR>	no changing to the current line
'E'	the end of modifying form

Figure 2.1.3 illustrates how the form 'Address' created earlier is modified. The form name 'Address' is typed first. The system then retrieves the form, displays it on the screen for modification. As the cursor moves from line to line, the user will be asked about the status of the current line. In this example, the user has typed 'A' so the cursor goes directly to the line following the

```
------------------------------------------------------------------------------
|                        **  FORM  MODIFY   **                               |
| FORM-NAME : Address                                                  STATUS |
|----------------------------------------------------------------------------|
|  Name:__________________                                               (A) |
|  Address:____________________________                                      |
|  City:__________________  State:__________________  Zip:____________       |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
|----------------------------------------------------------------------------|
|STATUS : E - end of modify.  D - delete line.  I - insert new line(s) before |
|the line. A - add new line(s).  Y - change line.  N or 'RETURN' - no change. |
|----------------------------------------------------------------------------|
```

Figuer 2.1.3 Modify Form Example (1)

```
------------------------------------------------------------------------------
|                        **  FORM  MODIFY   **                               |
| FORM-NAME : Address                                                  STATUS |
|----------------------------------------------------------------------------|
|  Name:__________________                                                   |
|  Address:____________________________                                      |
|  City:__________________  State:__________________  Zip:____________       |
|  Phone:__________________                                                  |
|  Remarks:___________________                                           (E) |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
|  ** Are you finished with the modification (y/n) ? N                       |
|----------------------------------------------------------------------------|
|STATUS : E - end of modify.  D - delete line.  I - insert new line(s) before |
|the line. A - add new line(s).  Y - change line.  N or 'RETURN' - no change. |
|----------------------------------------------------------------------------|
```

Figure 2.1.4 Modify Form Example (2)

```
--------------------------------------------------------------------------------
|                         **  FORM  MODIFY   **                                 |
| FORM-NAME : Address                                                    STATUS |
|------------------------------------------------------------------------------|
|  Name:_____________________                                              (D) |
|  Address:____________________________                                    (I) |
|  City:_______________   State:____________________   Zip:____________    (N) |
|  Phone:_______________                                                   (Y) |
|  Remarks:____________________                                            (E) |
|                                                                              |
|                                                                              |
|                                                                              |
|                                                                              |
|                                                                              |
|                                                                              |
|                                                                              |
|                                                                              |
|                                                                              |
|------------------------------------------------------------------------------|
|STATUS : E - end of modify.  D - delete line.  I - insert new line(s) before |
|the line. A - add new line(s).  Y - change line.  N or 'RETURN' - no change. |
|------------------------------------------------------------------------------|
```

Figure 2.1.5 Modify Form Example (3)

```
--------------------------------------------------------------------------------
|                         **  FORM  MODIFY   **                                 |
| FORM-NAME : Address                                                    STATUS |
|------------------------------------------------------------------------------|
|  Last Name:____________________   First Name:____________________            |
|  Address:____________________________                                        |
|  City:_______________   State:____________________   Zip:____________        |
|  Office Phone:_______________   Home Phone:_________________                 |
|  Remarks:____________________                                                |
|                                                                              |
|                                                                              |
|                                                                              |
|                                                                              |
|                                                                              |
|                                                                              |
|                                                                              |
|                                                                              |
|                                                                              |
|  ** Are you finished with the modification (y/n) ? Y                       |
|------------------------------------------------------------------------------|
|STATUS : E - end of modify.  D - delete line.  I - insert new line(s) before |
|the line. A - add new line(s).  Y - change line.  N or 'RETURN' - no change. |
|------------------------------------------------------------------------------|
```

Figure 2.1.6 Modify Form Example (4)

last line of the form. The user then types in the new lines as shown in Figure 2.1.4, and also types 'E' when finished with the addition. The user is then asked the question, "Are you finished with the modification (y/n) ?". An answer of 'N', the system returns to the routine modifying form for further modification.

Figure 2.1.5 illustrates the operation of modification using other symbols. When the cursor moves to the first line of the modifying form, the user types 'D' (delete line) for status, then the system will delete this current line. The cursor goes on to the next line, the user types 'I' (insert line) for the status, and the cursor then moves to the beginning of the current line to let the user type in the new line, which will be overwritten on the original line (the original line will not be deleted). In this example, the user types a new line shown on Figure 2.1.6. To specify that the 'City' field line does not change, the user just types 'N' (no change). As the cursor moves on to the 'Phone' field line, the user types 'Y' (yes change). The cursor then moves to the beginning of the current line for the user to type the whole line to rewrite the original line, which is shown in Figure 2.1.6. Finally, the user types 'E' (end of modification) for the end of modification. After finishing the whole modifications for the 'Address' form (it is shown in Figure 2.1.6), the user is then asked the question, "Are you finished with the modification (y/n) ?". An answer of 'Y', the system returns to the main menu.

2.1.3 Storing a Form

The FORM STORE function allows the user to store the field types on a form to create the form template. First, the system will display the message on the screen, " Is the current form [Address] to be stored (y/n) ?" If the answer is 'N', the system will ask for the user to type in the storing form name, and then check the form name to see whether it exists in the system. If it does not exist in the system, the system will display an error message and allow the user to try again. Otherwise, the system retrieves the named form or the current form (if the answer is 'Y') and displays it on the screen. For each field, the field type is then inserted by typing the following field type symbols for each field.

Symbols	Meaning
'A'	Alphanumeric
'N'	Numeric
'U'	Unchangeable
'R'	Retrieved
'C'	Calculated
'S'	Signature

If the field type is 'R', the user will be asked for the retrieve file name for retrieving. If the field type is 'C', the user will be asked for a formula to calculate the value for the field. After the field types are stored, the system will ask the user whether if he/she wants to check the form template. An answer of 'N', the system returns to the main menu; an answer of 'Y' causes the form template to be displayed on the screen for the user to check. After the user strikes any key on the keybroad, the system returns to the main menu.

The storing of the 'Address' form is illustrated in Figure 2.1.7. An 'A' (for alphanumeric field) was inserted into the fields 'Last Name', 'First Name', 'Address', 'City', 'State', 'Office Phone', 'Home Phone', and 'Remarks'. A 'N' (for numeric field) was inserted into the field 'Zip'. When all field types have been inserted, the system creates a form template for the form and displays the message, "Do you want to check the form template (y/n) ?". An answer of 'N' returns the system to the main menu. An answer of 'Y' causes the form template to be displayed on the screen (See Figure 2.1.8) which displays the field name, field type, and field length. The system then displays the message, "If the form template is not correct, please store the form again !!", and then returns to the main menu.

```
--------------------------------------------------------------------------------
|                          **  FORM  STORE   **                                  |
| FORM-NAME : Address                                                           |
|-------------------------------------------------------------------------------|
|  Last Name:A____________________      First Name:A____________________        |
|  Address:A______________________________                                      |
|  City:A_____________________     State:A________________     Zip:N___________ |
|  Office Phone:A___________       Home Phone:A__________                       |
|  Remarks:A_______________________________                                     |
|                                                                               |
|                                                                               |
|                                                                               |
|                                                                               |
|                                                                               |
|                                                                               |
|                                                                               |
|                                                                               |
|                                                                               |
|                                                                               |
|  ** What is the field type ?                                                  |
|-------------------------------------------------------------------------------|
|FIELD TYPE:  A = alphanumeric  N = numeric          U = unchangeable           |
|             R = retrieved     C = calculated       S = signature              |
|-------------------------------------------------------------------------------|
```

Figure 2.1.7 Store Form Example

```
--------------------------------------------------------------------------------
|                          **  FORM TEMPLATE CHECK  **                          |
| FORM-NAME : Address                                                           |
|-------------------------------------------------------------------------------|
|  Last Name:Alphanumeric[20]   First Name:Alphanumeric[20]                     |
|  Address:Alphanumeric[40]                                                     |
|  City:Alphanumeric[15]         State:Alphanumeric[10]   Zip:Numeric[10]       |
|  Office Phone:Alphanumeric[12]    Home Phone:Alphanumeric[12]                 |
|  Remarks:Alphanumeric[30]                                                     |
|                                                                               |
|                                                                               |
|                                                                               |
|                                                                               |
|                                                                               |
|                                                                               |
|                                                                               |
|                                                                               |
|                                                                               |
|                                                                               |
|-------------------------------------------------------------------------------|
|  ** If the form template is not correct, please store the form again !!       |
|     Please hit any key to return to the main menu !!                          |
|-------------------------------------------------------------------------------|
```

Figure 2.1.8 Form Template Display

CHAPTER 3

SYSTEM DESIGN

3.0 Introduction

Design is the first step in the development phase for a system. It may be defined as the process of applying various techniques and principles for the purpose of defining a system in sufficient detail to permit its physical realization [8]. The central phase in the software life cycle is development which is comprised of four distinct steps : preliminary design, detail design, coding, and testing.

The functional requirements, information flow and structure feed the preliminary design step to develop the software structure. The software structure is a hierarchical representation that indicates the relationship between elements (called modules) of the program. Detailed design transforms structural elements into a procedural description of the software called software procedure which focuses on the processing details of each module individually.

"A picture is worth a thousand words," but as Carl Machover (a leading computer graphics expert) says, "it's rather important to know which picture and which 1000 words." [8]. There is no question that graphical tools, such as the hierarchical block diagram or the flowchart, provide excellent pictorial patterns that readily depict procedural detail. The hierarchical block diagram depicts information as a series of multilevel blocks organized as a tree structure. At the top level of the structure, a single block is used to represent the entire hierarchy. Succeeding levels contain blocks that represent various categories of information that may be viewed as a subset of blocks further up the tree. The flowchart is the most widely used method for design representation of software. In the following sections, we use the hierarchical block diagram and the flowchart to explain the structure of the Form Editor System.

3.1 Overall System Structure

The system allows the design of a form and the insertion of field types and semantic information into a form to create a form template for later use. The system is structured by a main module and six subsidiary modules as shown below :

```
                          ----------
                          | MAIN   |
                          | MODULE |
                          ----------
                               |
         ----------------------------------------------
         |         |         |         |         |          |
      --------  --------  --------  --------  --------  --------
      |CREATE|  |MODIFY|  |STORE |  |PRINT |  |DELETE|  | EXIT |
      |MODULE|  |MODULE|  |MODULE|  |MODULE|  |MODULE|  |MODULE|
      --------  --------  --------  --------  --------  --------
```

Figure 3.1.1 Main Module

3.1.1 Data Structure

All the screen displays in the system are declared as the text files. The file is read line by line which is a string of 80 characters. There are nine text files in different modules of the system. They are described as follows :

File Name	Purpose	Module
'init.doc'	Displays the WELCOME screen	Main
'menu.doc'	Displays the main menu of the system	Main
'create.doc'	Displays the create form and the create help menu	Create
'modify.doc'	Displays the modify form and the modify help menu	Modify
'store.doc'	Displays the store form and the store help menu	Store
'template.doc'	Displays the form template	Store
'print.doc'	Displays the print form	Print
'delete.doc'	Displays the delete form	Delete
'exit.doc'	Displays the exit form and the BYE screen	Exit

When the user creates a form in the Create module, a form mapping file is created. This file consists of records of 75 characters fixed length string (maximum records is 16). The file name is the form name with an extension '.map' and is a temporary file until the user creates the form template for it. In the Modify module, the user retrieves the form mapping file and displays it on the screen to modify and update this file. When the user stores the field types into a form to create a form template in the Store module, a form template file is created. This file consists of 24 fixed length records consisting of the field number (2 digits), the field location (2 digits for X axis, 2 digits for Y axis), the field type (1 character), the field length (2 digits), and the field content (a string of 15 characters). The file name is the form name with an extension '.tmp'. Thus, after storing the form, the system has two files for each form : the form mapping file and the form template file. In the Print module, the user displays the form mapping file on the screen or prints this file (i.e., the form) on the printer. When the user deletes a form in the Delete module, the form mapping file and the form template file are deleted at the same time.

3.1.2 Main Module

The main module consists of six submodules : Create(C), Modify(M), Store(S), Print(P), Delete(D), and Exit(E). The user is given the choice of six different functions, each of which causes a different module to be called to perform a function. Except for the exit function which allows an exit of the system, other functions can be performed as many times as desired by entering the appropriate keystroke.

The CREATE function allows the user to create a new form. The MODIFY function allows the user to modify an existing form. The STORE function allows the user to store field type of each field on a form to create a form template. The

PRINT function allows the user to print out a hard copy of a form or display it on the screen. The DELETE function allows the user to delete the form and the form template from the system. The EXIT function allows the user to exit and stop the system.

Figure 3.1.2 depicts the flow chart of the main module. Once the system starts, the main module displays the main menu on the screen (See Figure 2.0.2) and waits for the command to continue the system. These commands are : 'C' (go to the Create Module), 'M' (go to the Modify Module), 'S' (go to the Store Module), 'P' (go to the Print Module), 'D' (go to the Delete Module), and 'E' (go to the Exit Module). Each module performs different functions as was mentioned in the previous paragraph. Upon finishing any one of modules : create, modify, store, print, and delete, the system returns to the main menu for another command. Only the exit module can exit and stop the system.

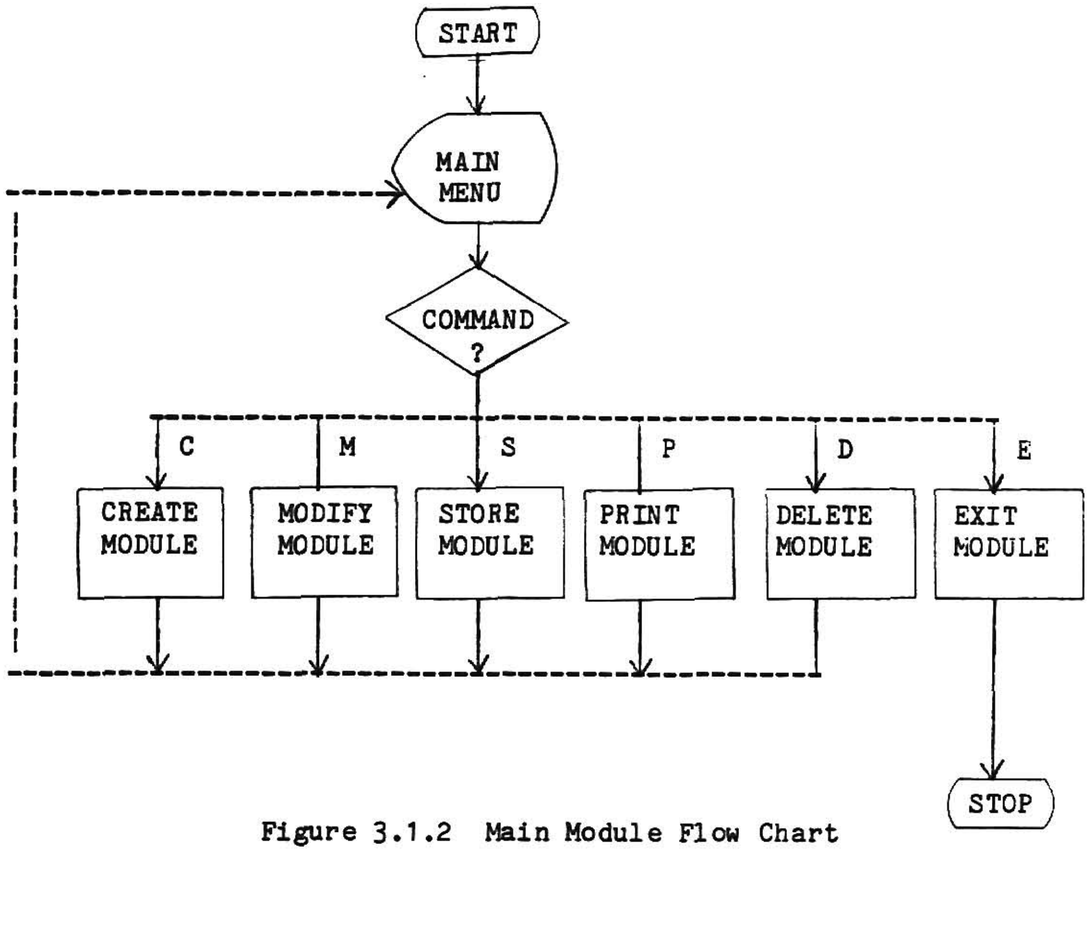


Figure 3.1.2 Main Module Flow Chart

3.2 Create Module

If the user selects command 'C' on the main menu, the create module (See Figure 3.2.1) is entered. The create module which consists of four procedures: screen_display, help_menu, check_form_name, and create_form.

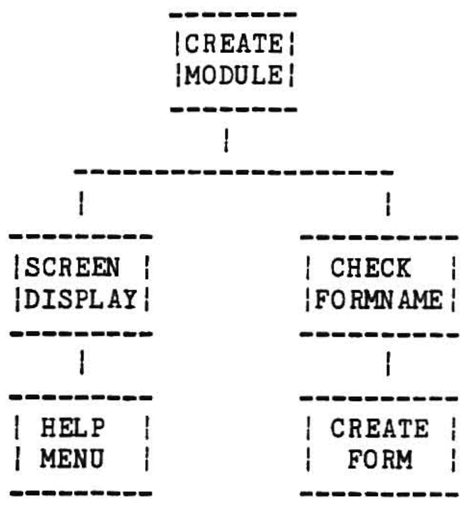


Figure 3.2.1 Create Module

The screen_display procedure displays the create form area on the screen, and asks the user whether the help menu is needed. If the user needs the help menu, the system executes the help_menu procedure. The check_form_name procedure checks the form name that the user types to see whether it is unique form to the system. If the form name does not exist, then the create_form procedure lets the user create the form. The Pseudo code for this module is as follows :

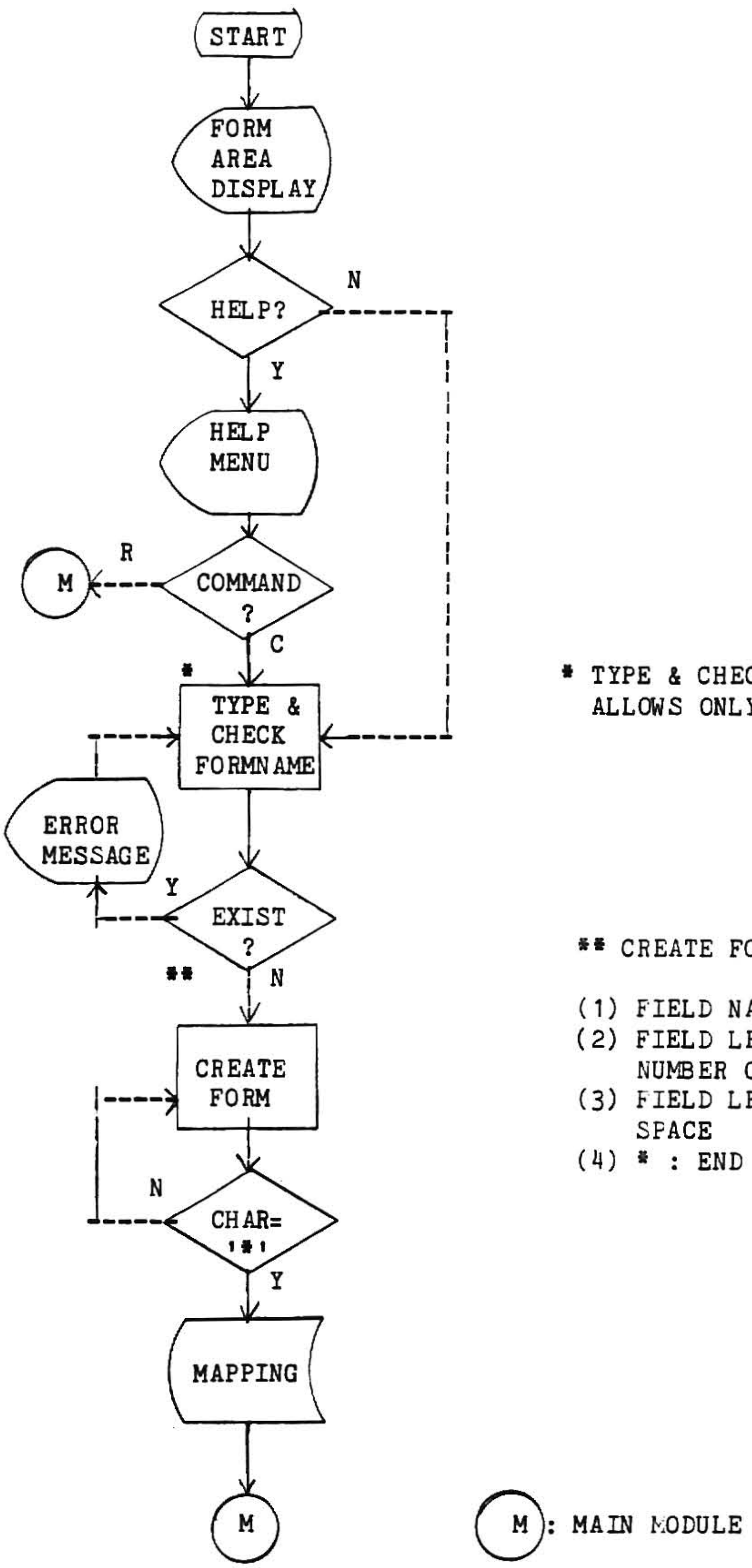
```
START

screen display

IF help needed THEN
   help menu
   command
   IF command = return to main menu THEN
      go to END
   ENDIF
ENDIF

REPEAT
```

```
  type formname
  check formname
UNTIL unique formname

REPEAT
  create form
UNTIL char = '*'

mapping form

END
```

Figure 3.2.2 depicts the flow chart of the create module. When the system enters the create module, the create form area (See Figure 3.2.3) is first displayed on the screen. The system waits for the request for help menu. An answer of 'N' continues the create form process; an answer of 'Y' displays the help menu (See Figure 3.2.4). The create form area and the create help menu are declared as the text file 'create.doc' which is read and displayed on the screen by this module. If the help menu is selected, the user will be given two choices of commands, 'R' to return to the main menu or 'C' to continue the create module.

The create form area is constrained to 75 characters by 16 lines, due to one display screen for a form. The user has to type in the form name (maximum 15 characters) at the cursor position for the form name. A function 'exist' checks whether the form name exists in the system. If it does (i.e., the form name is not unique), the system displays an error message and lets the user type another form name. The process is repeated until an acceptable form name is found, or until three attempts have been made. When the form name is acceptable the cursor moves to the first line of the create form.

Since the system needs a delimiter to identify the right boundary of a field name, the user has to type a ':' after each field name. The field length is specified by the number of underscores typed after the delimiter. A space must be typed after the desired number of underscores, otherwise a default length of 10 is



* TYPE & CHECK FORM NAME
ALLOWS ONLY 3 CHANCES

** CREATE FORM :

(1) FIELD NAME FOLLOW ':'
(2) FIELD LENGTH COUNTED BY NUMBER OF UNDERSCORES
(3) FIELD LENGTH FOLLOW A SPACE
(4) * : END OF CREATION

Figure 3.2.2 Create Module Flow Chart

```
-----------------------------------------------------------------------------
|                           **  FORM CREATE  **                             |
| FORM-NAME :                                                               |
|---------------------------------------------------------------------------|
|                                                                           |
|                                                                           |
|  Do you need help (y/n) ? __                                              |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|---------------------------------------------------------------------------|
|a. Field name must be followed by ':'. b. Field length counted by underscores|
|and followed by a space. c. To terminate the creation,type '*' on a new line.|
|---------------------------------------------------------------------------|
```

Figure 3.2.3 Create Form Display

```
-----------------------------------------------------------------------------
|                                                                           |
|                    ****************************                           |
|                    **   FORM  CREATE  HELP   **                           |
|                    ****************************                           |
|                                                                           |
|     (1) First fill in the name of the form you are creating.  It must be  |
|         a unique form name to all other forms in the system.              |
|                                                                           |
|     (2) You can create any kind of form on the FORM CREATE AREA, that is, |
|         75 characters X 16 lines area.                                    |
|                                                                           |
|     (3) A ':' must follow all FIELD NAMEs.  The FIELD LENGTH counted by   |
|         underscores length, and it must be followed a space.              |
|                                                                           |
|     (4) When you finish the form and want to exit the CREATE MODE, type   |
|         '*' on a new line of the screen.                                  |
|                                                                           |
|     (5) After you exit the create form module, you can print the form out |
|         or you can store the form into system or both.                    |
|                                                                           |
|---------------------------------------------------------------------------|
|      COMMAND :   (C-CONTINUE CREATE FORM   R-RETURN TO MAIN MENU)         |
|---------------------------------------------------------------------------|
```

Figure 3.2.4 Create Form Help Menu

given. To terminate the creation of a form, the user types a '*' on a new line of the screen. The system assigns a file as the form mapping file for the created form. The file name is the form name that the user types in with an extension '.map'. This file is a temporary file until the user stores the form to the system.

3.3 Modify Module

If the 'M' command is selected on the main menu, the modify module (See Figure 3.3.1) is entered. The modify module which consists of five procedures: screen_display, help_menu, check_form_name, retrieve_form, and modify_form.

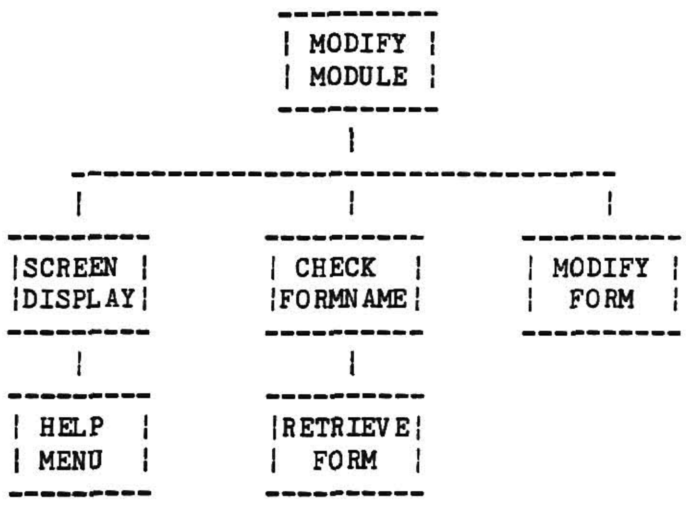
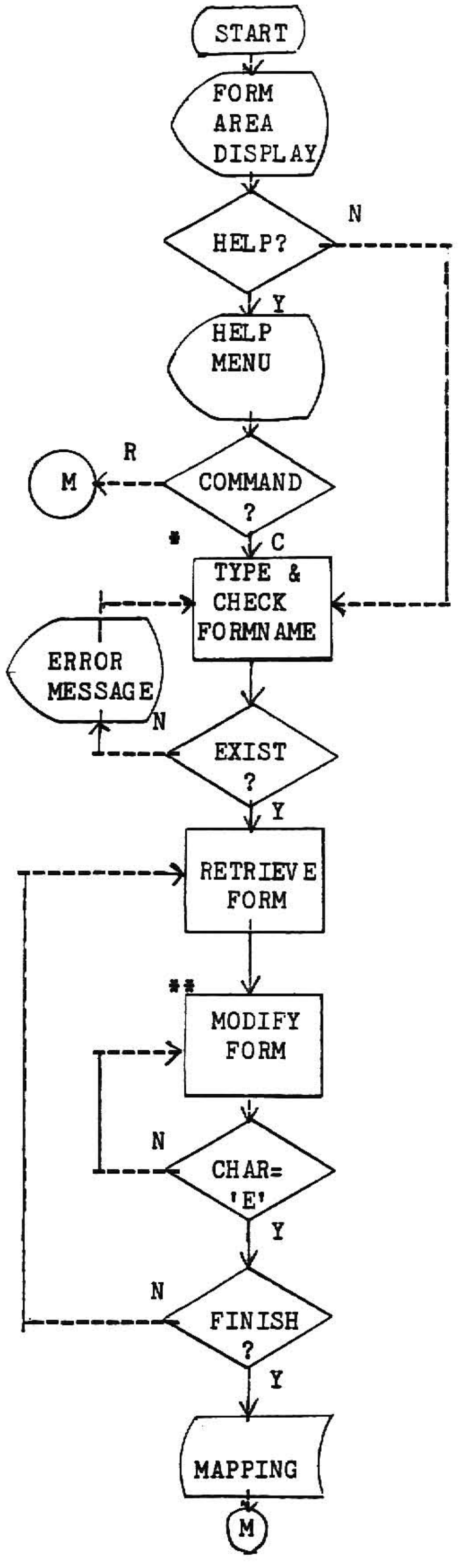


Figure 3.3.1 Modify Module

The screen_display procedure displays the form area on the screen, and asks the user whether the help menu is needed. If the user needs the help menu, the system executes the help_menu procedure. The check_form_name procedure checks the form name that the user types to see whether it exists in the system. The retrieve_form procedure retrieves and displays the acceptable form. The modify_form procedure allows the user to modify the form.

Figure 3.3.2 depicts the flow chart of the modify module. As the system enters the modify module, the form area (See Figure 3.3.3) is displayed on the screen. The system waits for the request for help menu. An answer of 'N' continues the modify form process; an answer of 'Y' displays the help menu (See Figure 3.3.4). The form area and the modify help menu are declared as the text file 'modify.doc' which is read and displayed on the screen by this module. If the help menu is selected, the user is given two choices of commands: 'R' to return to the main menu or 'C' to continue the modify module. First, the user has to type in the existing form name at the cursor position for the form name. The function 'exist' checks whether the form name exists in the system. If the form name is not in the system, the system displays an error message and allows the user to type another form name. The process is repeated until an acceptable form name is found or until three attempts have been made. When the form name is acceptable, the form is retrieved and displayed on the screen.

In the modify form process, a line editor is provided to allow modification of the form line by line. At the end of each line, the system asks for the status. The following symbols are used : 'E' to specify the end of modification; 'A' to add a new line or lines at the end of the form; 'I' to insert a new line or lines before the current line (the new line will overwrite on the original line, but the original line will not be deleted); 'D' to delete the current line; 'Y' to change the current line (the user has to retype the whole line); and 'N' or 'return' to leave the current line unchanged. When 'E' is typed, the system asks the user whether the modification is finished. An answer of 'N' causes the modified form displayed on the screen again to allow the user to verify the modification. An answer of 'Y' updates the form mapping file for the modified form.



* TYPE & CHECK FORMNAME
ALLOWS ONLY 3 CHANCES

** MODIFY FORM USE SYMBOLS :

(1) E : END OF MODIFICATION
(2) D : DELETE CURRENT LINE
(3) I : INSERT LINE(S)
(4) A : ADD LINE(S)
(5) Y : CHANGE CURRENT LINE
(6) N OR RETURN : NO CHANGE

(M): MAIN MODULE

Figure 3.3.2 Modify Module Flow Chart

```
-----------------------------------------------------------------------------
|                       **  FORM  MODIFY    **                              |
| FORM-NAME :                                                       STATUS |
|---------------------------------------------------------------------------|
|                                                                           |
|                                                                           |
|  Do you need help (y/n) ? __                                              |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|---------------------------------------------------------------------------|
|STATUS : E - end of modify.  D - delete line.  I - insert new line(s) before |
|the line. A - add new line(s).  Y - change line.  N or 'RETURN' - no change. |
|---------------------------------------------------------------------------|
```

Figure 3.3.3 Modify Form Display

```
-----------------------------------------------------------------------------
|                                                                           |
|                     ****************************                          |
|                     **   FORM  MODIFY  HELP   **                          |
|                     ****************************                          |
|                                                                           |
|   (1) First fill in the name of the form you are modifying.  It must be  |
|       a form name which is already in the system.                        |
|   (2) The system will retrieve the form on the screen.                   |
|   (3) The form will be modified line by line on the screen.  Use symbols:|
|           'E' : no more change to the end of the form.                   |
|           'D' : delete the current line of the form.                     |
|           'I' : insert new line(s) after the line of the form.           |
|           'A' : add new line(s) after the the last line of the form.     |
|           'N' or `return' : no change on the current line of the form.   |
|           'Y' : change the current line of the form.  Please type one    |
|                 character by one character or press the 'return' for     |
|                 each character to the whole line.                        |
|   (4) After you get out of the modify form module, you can print the form|
|       out or you can store the form in the system or both.               |
|                                                                           |
|---------------------------------------------------------------------------|
|       COMMAND:   (C-CONTINUE MODIFY FORM  R-RETURN TO MAIN MENU)          |
|___________________________________________________________________________|
```

Figure 3.3.4 Modify Form Help Menu

3.4 Store Module

If the user selects command 'S' on the main menu, the store module (See Figure 3.4.1) is entered. The store module which consists of six procedures: screen_display, help_menu, check_form, retrieve_form, store_form, and form_template.

```
                      ---------
                      | STORE |
                      | MODULE|
                      ---------
                          |
          -------------------------------
          |               |              |
      ---------      ----------     ----------
      |SCREEN |      | CHECK  |     | STORE  |
      |DISPLAY|      |  FORM  |     | FORM   |
      ---------      ----------     ----------
          |              |              |
      ---------      ----------     ----------
      | HELP  |      |RETRIEVE|     |  FORM  |
      | MENU  |      |  FORM  |     |TEMPLATE|
      ---------      ----------     ----------
```

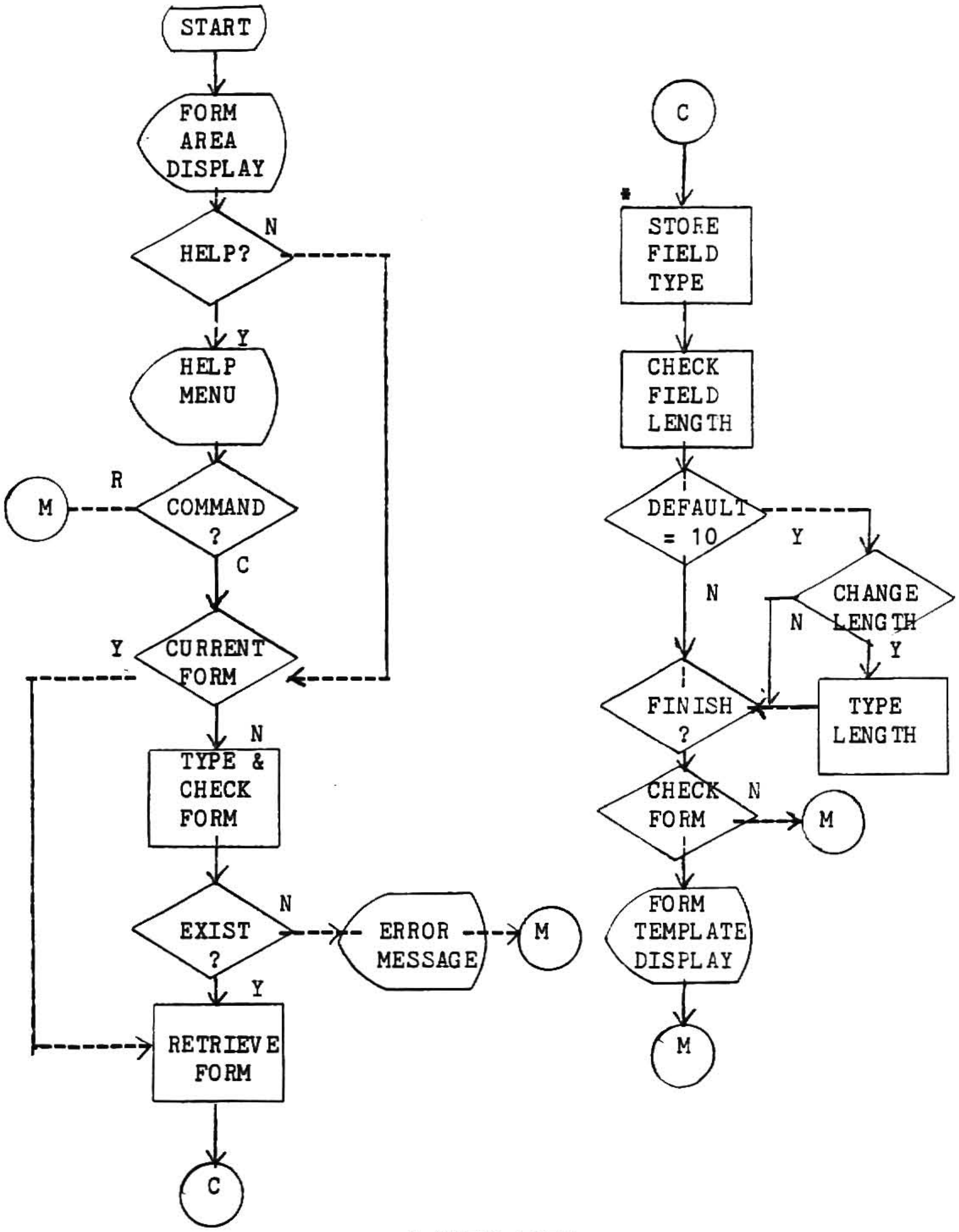
Figure 3.4.1 Store Module

The screen_display procedure displays the form area on the screen, and asks the user whether the help menu is needed. If the user needs the help menu, the system executes the help_menu procedure. The check_form procedure checks whether the form exists in the system. If the form exists then the retrieve_form procedure retrieves and displays the form on the screen. The store_form procedure stores each field type of the form in the system. If the user asks for visually checking the form template then the form_template procedure displays the form template on the screen.

Figure 3.4.2 depicts the flow chart of the store module. When the system enters the store module, the form area (See Figure 3.4.3) is displayed on the screen. It then waits for the request for help menu. An answer of 'N' continues the store form process; an answer of 'Y' displays the help menu (See Figure 3.4.4). The form area and the store help menu are declared as the text file 'store.doc' which is read and displayed on the screen by this module. If the help menu is selected, the user is given two choices of commands, 'R' to return to the main menu or 'C' to continue the store form module.

In the store form process, the user first is asked whether the current form is going to be stored. An answer of 'Y' causes the current form to be displayed on the screen. Otherwise the user is asked to type in the name of the form. The function 'exist' checks whether the form exists in the system. If it does not exist, the system displays an error message on the screen and returns to the main menu. If it does exist, the form is displayed on the screen and the system asks for the field type for each field on the form. The user may use one of the following symbols : A, N, U, R, C, and S. 'A' specifies an alphanumeric field; 'N' specifies a numeric field; 'U' specifies an unchangeable field; 'R' specifies a retrieved field; 'C' specifies a calculated field; and 'S' specifies a signature field.

In the process of storing a field type, the system checks the field length. If the field length has the default length of 10 (due to the underscores was not followed by a space for delimiter use when the form was created), the user is asked whether he wants to change the field length. If the user wants to change it then gives the new field length, otherwise the length is set to 10. When the user finishes storing a form, a form template file is created. The file consists of 24 fixed length records consisting of the field number (2 digits), the field location (2 digits for X axis, 2 digits for Y axis), the field type (1 character), the field length (2 digits), and the field content (a string of 15 characters). The file name is the form name with an extension '.tmp'. After storing the form, the



* FIELD TYPE :

(1) A: ALPHANUMERIC
(2) N: NUMERIC
(3) U: UNCHANGEABLE
(4) R: RETRIEVE FROM DB
(5) C: CALCULATED FIELD
(6) S: SIGNATURE FIELD

Ⓜ : MAIN MODULE
Ⓒ : CONTINUE

Figure 3.4.2 Store Module Flow Chart

```
---------------------------------------------------------------------------
!                        **  FORM  STORE   **                              !
! FORM-NAME :                                                              !
!--------------------------------------------------------------------------!
!                                                                          !
!                                                                          !
!  Do you need help (y/n) ? __                                             !
!                                                                          !
!                                                                          !
!                                                                          !
!                                                                          !
!                                                                          !
!                                                                          !
!                                                                          !
!                                                                          !
!                                                                          !
!                                                                          !
!                                                                          !
!                                                                          !
!--------------------------------------------------------------------------!
!FIELD TYPE:  A = alphanumeric  N = numeric          U = unchangeable      !
!             R = retrieved     C = calculated       S = signature         !
!--------------------------------------------------------------------------!
```

Figure 3.4.3 Store Form Display

```
---------------------------------------------------------------------------
!                                                                          !
!                        ***************************                       !
!                        **   FORM  STORE  HELP   **                       !
!                        ***************************                       !
!    (1) The system will retrieve the form and display it on the screen.   !
!    (2) The form will be stored field by field, and the field type has to !
!        be assigned by the user, using the symbols as belows :            !
!          'A' : alphanumeric variable.                                    !
!          'N' : numeric variable.                                         !
!          'U' : unchangeable field.                                       !
!          'R' : retrieve from other data base.  Requires the file name for!
!                retrieving.                                               !
!          'C' : calculated from others fields.  Requires the formula for  !
!                this field to calculate.                                  !
!          'S' : signature field.                                          !
!    (3) The field length will be calculated automatically by underscores  !
!        length.                                                           !
!    (4) After you exit the store form module, you can check the field types
!        of the stored form.                                               !
!                                                                          !
!--------------------------------------------------------------------------!
!        COMMAND:   (C-CONTINUE STORE FORM   R-RETURN TO MAIN MENU)        !
!__________________________________________________________________________!
```

Figure 3.4.4 Store Form Help Menu

system has two files for each form : the form mapping file and the form template file. At last, the user is asked whether he wants to check the form template. An answer of 'N' returns the system to the main menu; an answer of 'Y' causes the form template which including each field name, field type, field length, and field content (only 'R' and 'C' field type have filled in for retrieving file name and formula) to be displayed. The user may then hit any key to return to the main menu.

3.5 Print Module

If the user selects the command 'P' on the main menu, the print module (See Figure 3.5.1) is entered. The print module which consists of four procedures: screen_display, check_form, hard_copy, and display_form. The screen_display procedure displays all of the needed messages on the screen. The check_form procedure checks whether the named form exists in the system. If the form exists then the hard_copy procedure prints out the form on the printer or the display_form procedure displays the form on the screen.

```
                 ---------
                 | PRINT |
                 | MODULE|
                 ---------
                     |
            --------------------
            |                  |
        ---------         ----------
        |SCREEN |         | CHECK  |
        |DISPLAY|         |  FORM  |
        ---------         ----------
                              |
                     ----------------
                     |              |
                 ---------      ---------
                 | HARD  |      |DISPLAY|
                 | COPY  |      | FORM  |
                 ---------      ---------
```

Figure 3.5.1 Print Module

Figure 3.5.2 depicts the flow chart of the print module. At the start of the print module, *Figure 3.5.3* is displayed on the screen and asks the user whether the current form is to be printed or displayed. If the current form is not to be printed or displayed, the system proceeds to ask for the name of form to be printed or displayed. If the named form cannot be found, the system displays the message "The form can not be found !", and returns to the main menu. Otherwise the system asks the user, "Do you want it to be printed or displayed (P/D) ?" An answer of 'P' cause a hard copy of the form to be printed. An answer of 'D' causes the form to be displayed on the screen (See Figure 3.5.4). When the system exits the print module, a return to the main menu occurs.

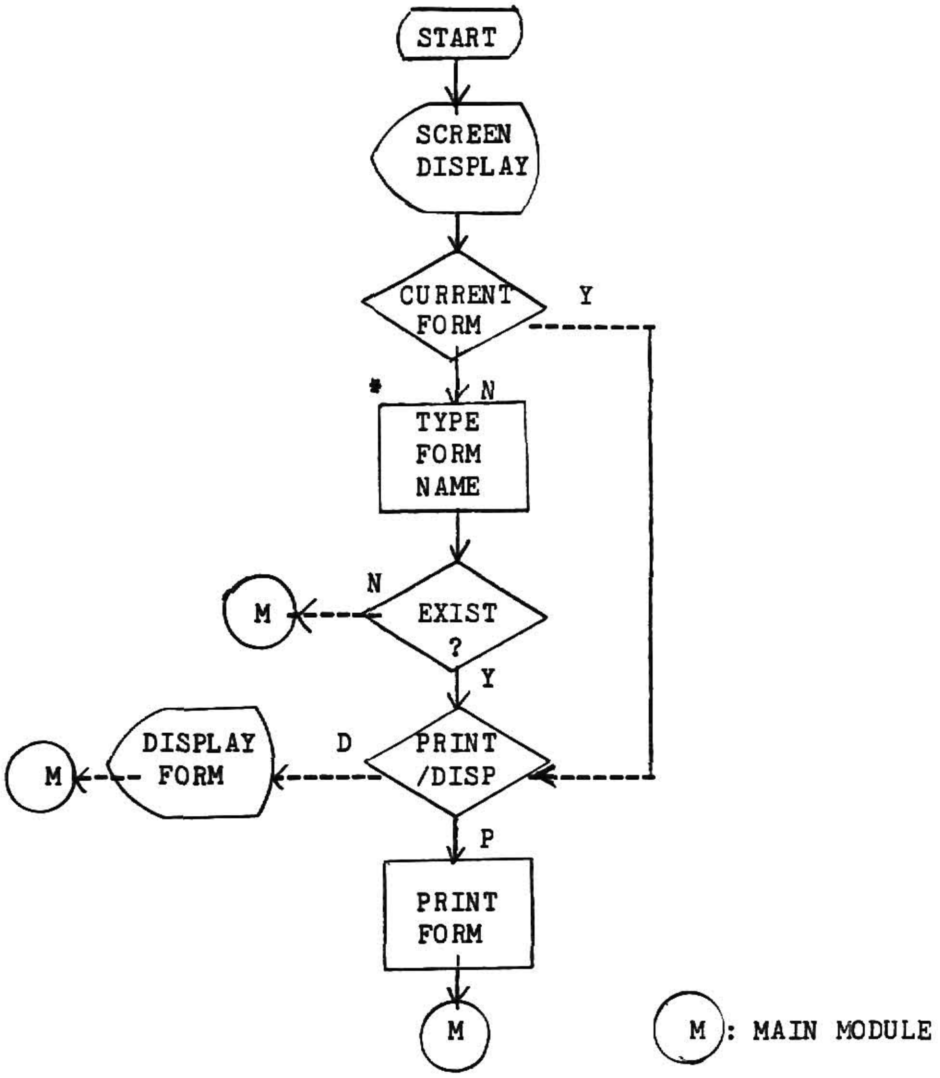
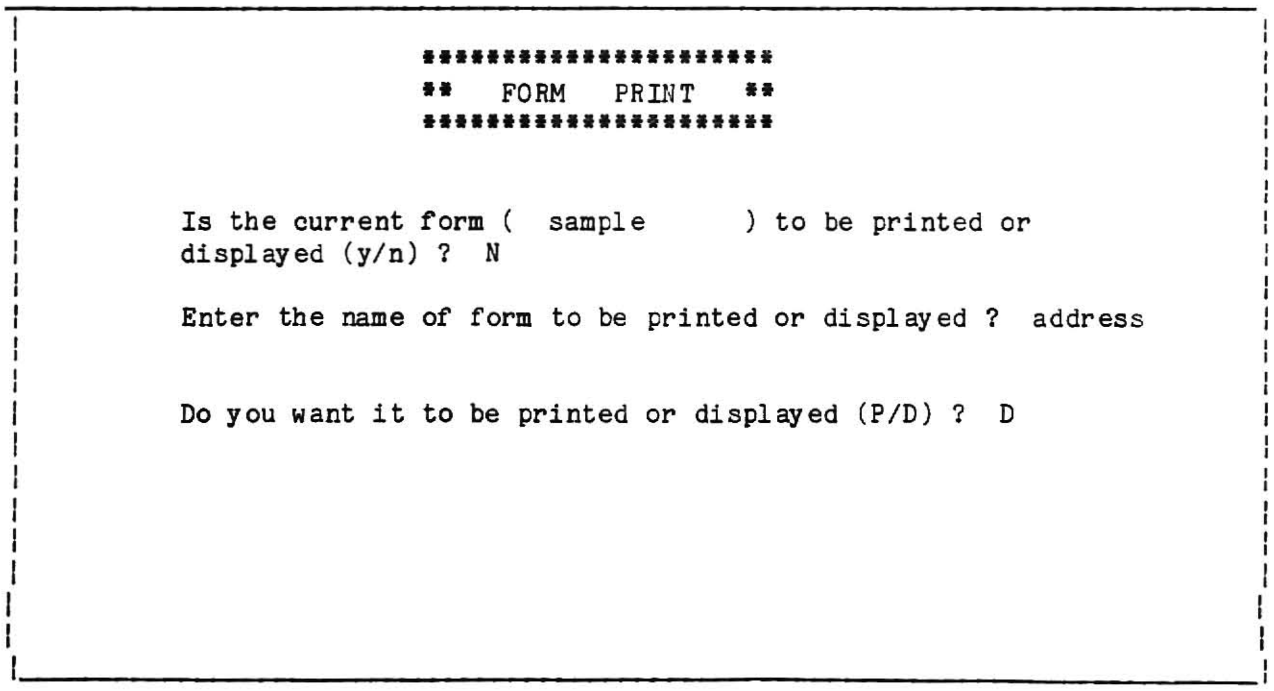
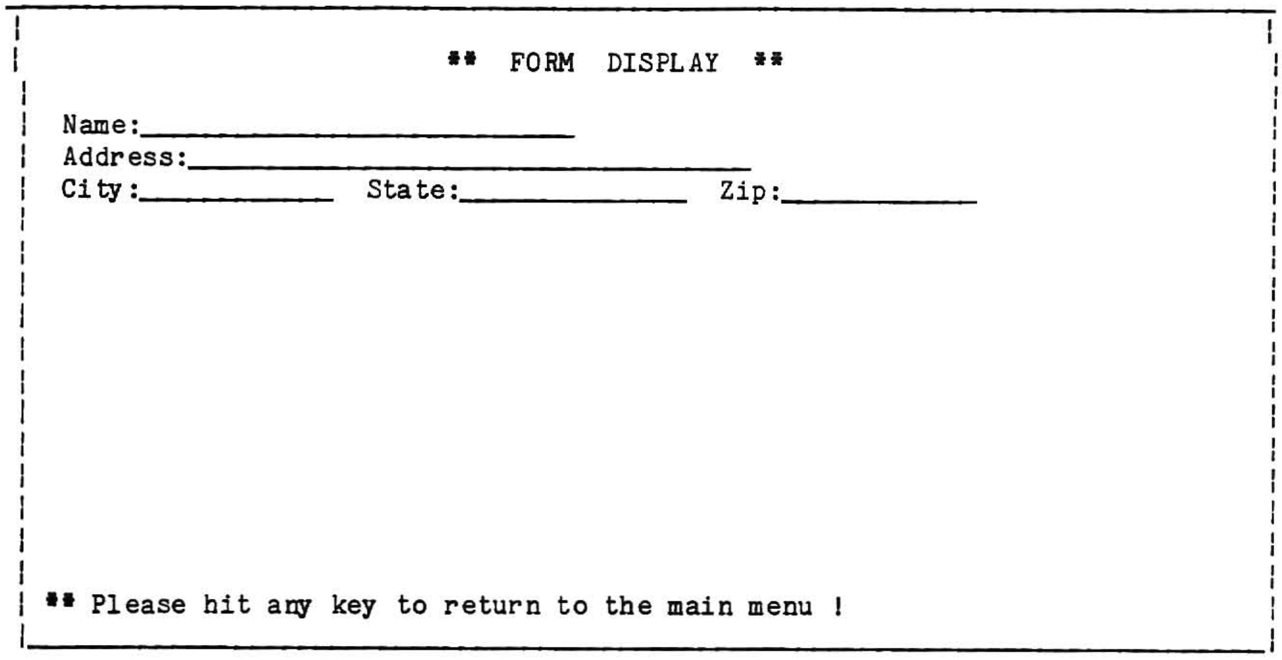


Figure 3.5.2 Print Module Flow Chart



```
                    ***********************
                    **   FORM   PRINT   **
                    ***********************


        Is the current form (  sample      ) to be printed or
        displayed (y/n) ?  N

        Enter the name of form to be printed or displayed ?  address


        Do you want it to be printed or displayed (P/D) ?  D
```

Figure 3.5.3 Print Display



```
                    **  FORM  DISPLAY  **

  Name:____________________________
  Address:____________________________________
  City:____________  State:______________  Zip:____________












** Please hit any key to return to the main menu !
```

Figure 3.5.4 Form Display

3.6 Delete Module

If the command 'D' is selected on the main menu, the delete module (See Figure 3.6.1) is entered. The delete module, which consists of three procedures: screen_display, warning_display, and delete_form.

```
                ----------
                | DELETE |
                | MODULE |
                ----------
                    |
            -----------------
            |               |
        ---------       ----------
        |SCREEN |       |WARNING |
        |DISPLAY|       |DISPLAY |
        ---------       ----------
                            |
                        ----------
                        | DELETE |
                        |  FORM  |
                        ----------
```

Figure 3.6.1 Delete Module

The screen_display procedure displays the needed message for the delete module. The warning_display procedure displays the confirmation of the decision. If the user confirms to delete the form then the delete_form procedure deletes the form and the form template from the system.

Figure 3.6.4 depicts the flow chart of the delete module. At the start of the delete module, Figure 3.6.2 is displayed on the screen. Before a form being deleted, the user is given an opportunity to confirm the decision (See Figure 3.6.3). If the user answers 'N' then the system returns to the main menu. If the user answers 'Y' then the delete form command erases the form and the form template from the system forever. After deleting the form, a return to the main menu occurs.

```
                **********************
                **   FORM  DELETE   **
                **********************


      Enter the name of form to be deleted ?   testform


      ** The form can not be found ! **
```

Figure 3.6.2 Delete Display (1)

```
                **********************
                **   FORM  DELETE   **
                **********************


         The form will be erased PERMANENTLY.

        Enter Y to confirm you wish to DELETE, or

        enter N if you changed your mind.

         The answer : __
```

Figure 3.6.3 Delete Display (2)

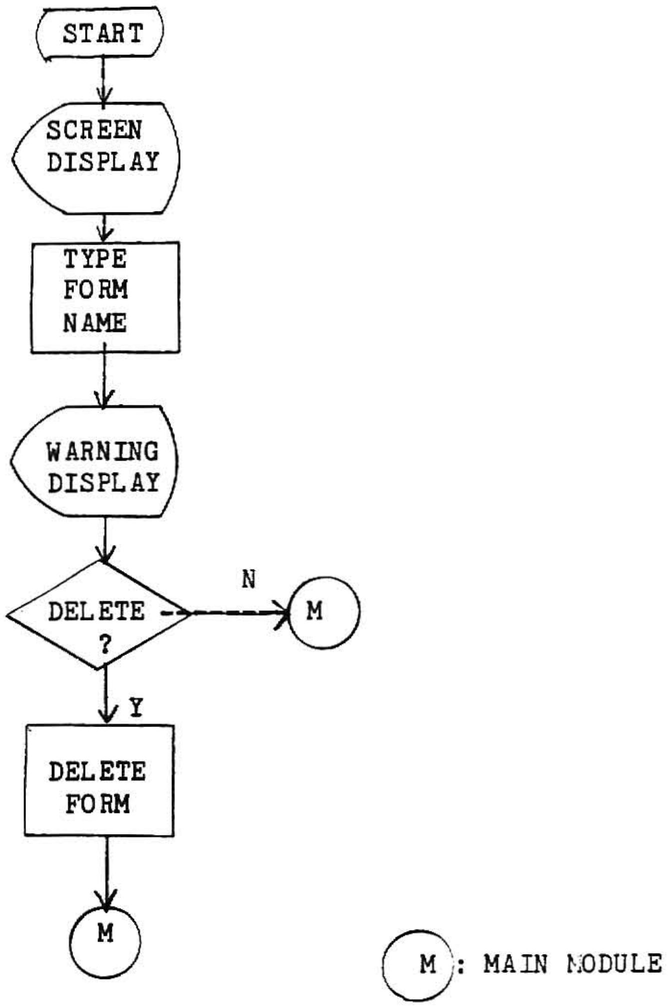


Figure 3.6.4 Delete Module Flow Chart

3.7 Exit Module

If the command 'E' is selected on the main menu, the exit module (See Figure 3.7.1) is entered. The exit module which consists of two procedures: check_store and BYE_display. The check_store procedure checks whether the current form has been stored in the system. If it has not been stored, a warning message to the user. The Bye_display procedure displays 'BYE' on the screen.

Figure 3.7.2 depicts the flow chart of the exit module. The exit module contains codes to check whether the form has been stored. If it has not been stored yet, the system provides the user an opportunity to store the form before termination (See Figure 3.7.3). If the user answers 'N' then the system returns to the main menu. If the user answers 'Y' (or the current form has been stored before exiting) then the system displays "BYE, FORM EDITOR SYSTEM" (See Figure 3.7.4) and exits the system.

```
                 --------
                 | EXIT |
                 |MODULE|
                 --------
                     |
           -------------------
           |                 |
       ---------         ---------
       | CHECK |         |  BYE  |
       | STORE |         |DISPLAY|
       ---------         ---------
```

Figure 3.7.1 Exit Module

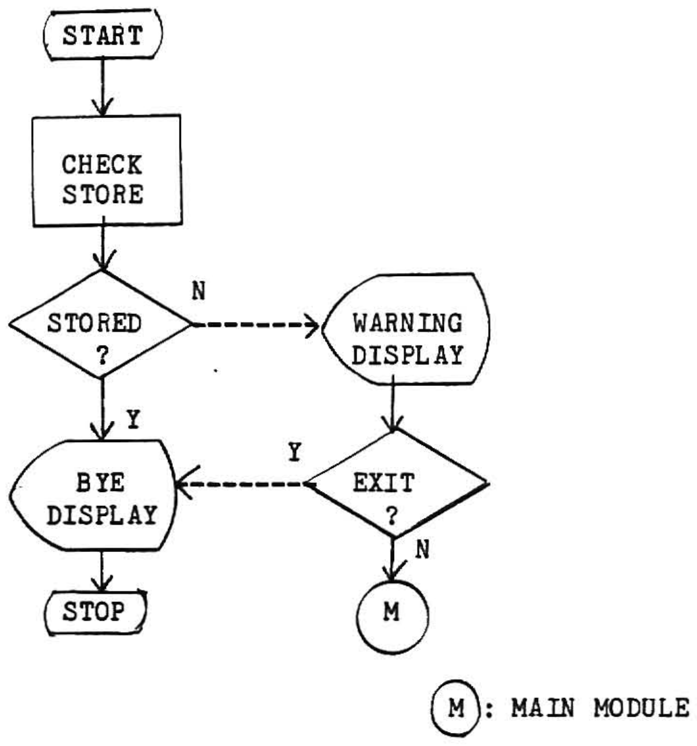
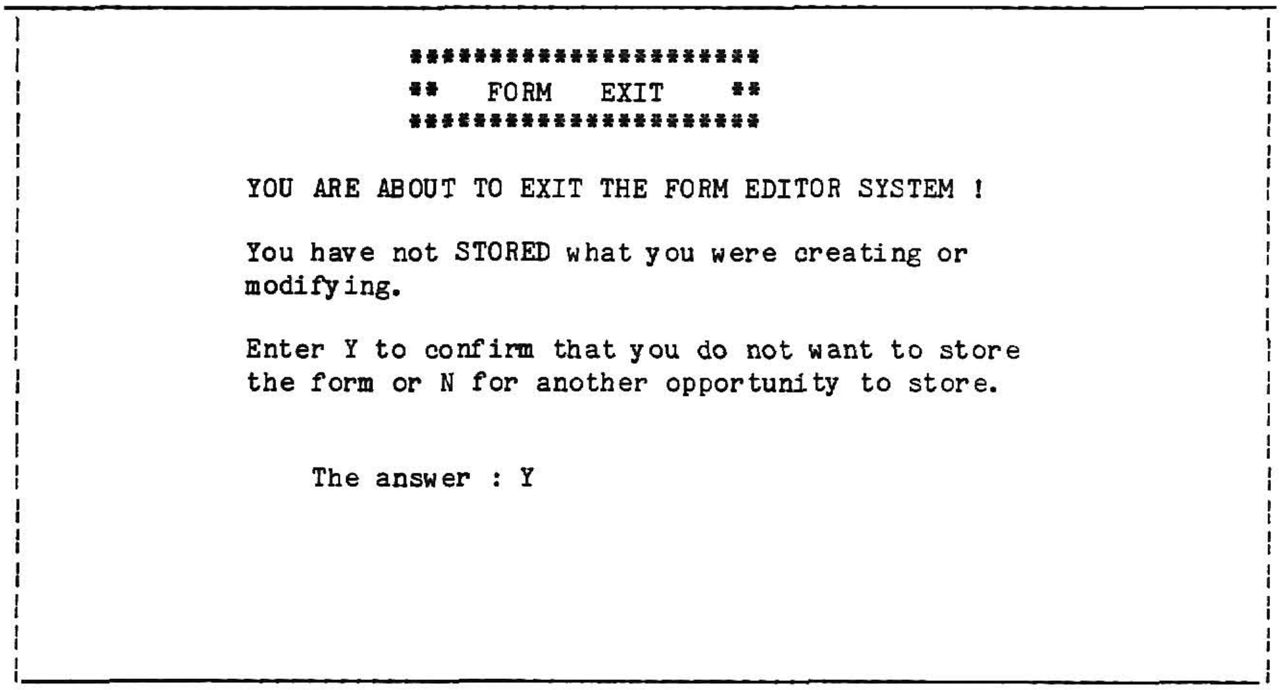
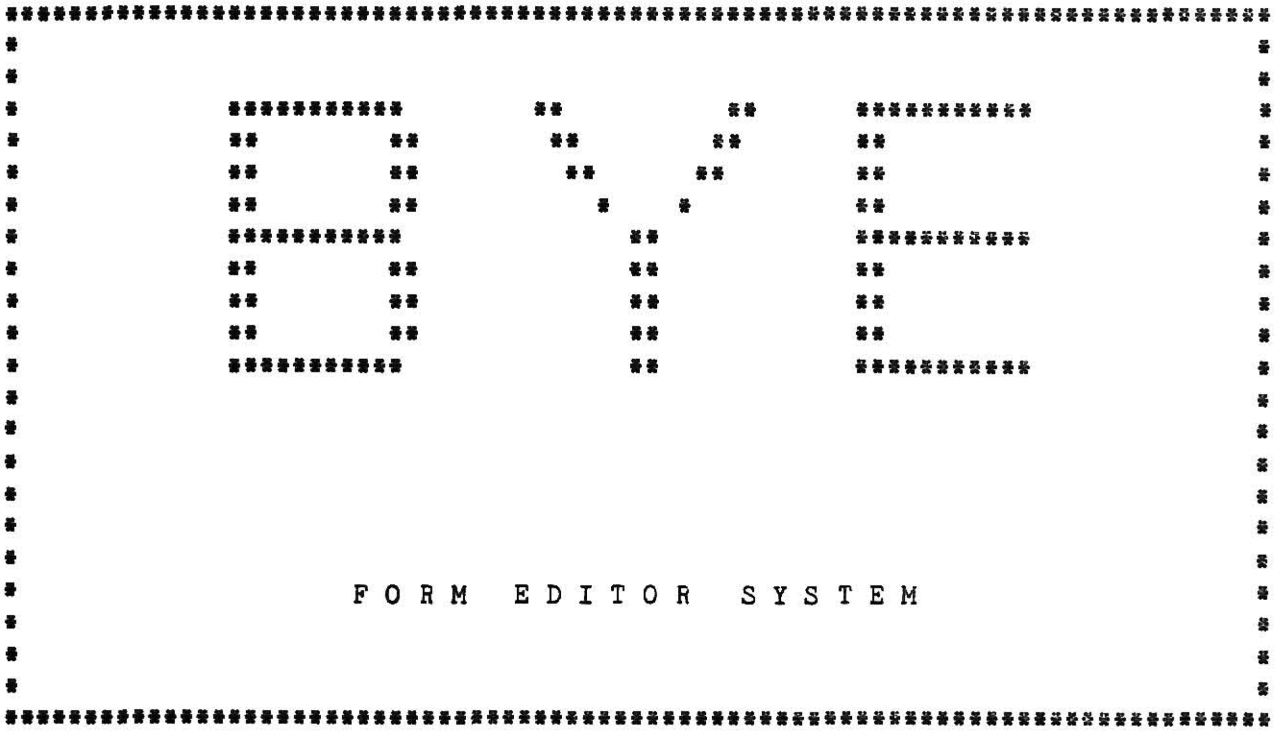


Figure 3.7.2 Exit Module Flow Chart



```
                  **********************
                  **   FORM   EXIT    **
                  **********************

YOU ARE ABOUT TO EXIT THE FORM EDITOR SYSTEM !

You have not STORED what you were creating or
modifying.

Enter Y to confirm that you do not want to store
the form or N for another opportunity to store.


     The answer : Y
```

Figure 3.7.3 Exit Display



```
********************************************************************************
*                                                                              *
*                                                                              *
*            ***********       **          **       ***********               *
*            **        **       **        **        **                        *
*            **        **        **      **         **                        *
*            **        **          *    *           **                        *
*            ***********             **             ***********               *
*            **        **            **             **                        *
*            **        **            **             **                        *
*            **        **            **             **                        *
*            ***********             **             ***********               *
*                                                                              *
*                                                                              *
*                                                                              *
*                                                                              *
*                                                                              *
*                                                                              *
*                    F O R M   E D I T O R   S Y S T E M                      *
*                                                                              *
*                                                                              *
*                                                                              *
********************************************************************************
```

Figure 3.7.4 BYE Display

CHAPTER 4

IMPLEMENTATION

4.0 Introduction

This Form Editor System was implemented on the Columbia Data Products (CDP) Multi-Personal Computer (MPC) using the TURBO Pascal language. The standard MPC system is comprised of double sided (40 data track per side), double density, 320K disk drives (5 1/4-Inch) to handle soft-sectored diskettes. The disk operating system(s) provided by CDP are compatible with the IBM Personal Computer. The MPC can execute a variety of high-level languages under CP/M, MS/DOS, or MP/M. The System was implemented using Pascal, a general purpose and high level language, because it would be easier to modify the system in the future and to move the system from its present form to a more intelligent system if it is written in a high level, and presumably, more portable language.

4.1 Programming Language

Pascal is a structured programming language developed in the early 1970s as a language for teaching modern techniques (e.g., structured programming) in software development. It is characterized by strong procedural and data structuring capabilities. It has been implemented on computers of all sizes and shows particular promise as a development language for microcomputer application.

TURBO Pascal closely follows the definition of the Standard Pascal as defined by K. Jensen and N. Wirth in "The Pascal User Manual and Report". There are two additional advantages of using TURBO Pascal for this System. One is that the compilation time is faster than other Pascal Compilers for microcomputer; the other is that some built-in procedures and functions, such as screen related procedures and string functions which do not exist in Standard Pascal, were used to increase the versatility of the System.

The following is to list the built-in procedures or functions of TURBO Pascal that do not exist in standard Pascal and were used in the Form Editor System.

1. ASSIGN (Filevar, Filename)

   The Filename is assigned to the file variable Filevar, and all further operation on Filevar will operate on the disk file Filename.

2. CLRSCR

   The function clears the screen and places the cursor in the upper-left-hand corner. It is used before the next screen is displayed.

3. CONCAT (Str1, Str2 {Strn})

   The result is a string which is the concatenation of the arguments in the same order as they are specified. This function is used to have an extension connected with a form name in the Form Editor System, and the user just types in the form name.

4. COPY (Str, Pos, Num)

   The function returns a string containing Num character from Str starting at position Pos. It is used to read nine text files for screen display. Read the text file until it returns a mark '.PA' which means one whole screen display.

5. DELAY (Time)

   The procedure creates a loop which runs for approximately as many milliseconds as defined by its argument which must be an integer. This procedure is used for displaying message on the screen so that they remained a short period of time before being cleared.

6. ERASE (Filename)

The procedure is used in 'Delete Function' of the System to erase the existing disk file associated with Filevar.

7. GOTOXY (Xpos, Ypos)

The procedure moves the cursor to any addressable point on the screen. It is used for displaying message, for input routines etc.

8. HALT

The procedure is used to exit the Form Editor System from the screen display to MS-DOS system.

9. KEYPRESS

The function is used to check if a key has been pressed at the console.

10. LOWVIDEO

The function sets the screen to the video attribute defined as 'end of highlighting' in the installation procedure, i.e., 'dim' characters. Meanwhile, highvideo is the 'normal' screen mode. In the System, highvideo is the yellow color, and lowvideo is the blue color.

11. POS (Obj, Target)

The function scans the string target to find the first occurrence of Obj within Target. If the pattern is not found, it returns 0. This function is used to define the field length, field location etc.

12. RENAME (Filevar, Str)

The disk file associated with Filevar is renamed to a new name given by the string expression Str. This procedure is used in 'Modify Function' to have a new form instead of the old form.

13. UPCASE (Char)

The function returns a capital character. Whatever the user inputs the character for command and answer in the Form Editor System which will be transformed into capital character.

4.2 Coding

The System consisted of 869 lines of the main program of source code (See Appendix B) which included a few statements of internal documentation, and occupied 20,780 bytes of disk file memory. All the screen displays in the System are declared as the text files. The file is read line by line each line of which is a string of 80 characters. There are nine text files in the System as follows :

File Name	Line	Byte
'init.doc'	49	3858
'menu.doc'	26	1990
'create.doc'	77	5892
'modify.doc'	77	5892
'store.doc'	77	5892
'template.doc'	27	1996
'print.doc'	50	3778
'delete.doc'	48	3619
'exit.doc'	49	3698
TOTAL	480	36615

TURBO Pascal supports the separation of screen display from the main program. They are linked with the main program prior to execution. This is useful when creating a program which utilizes many screen displays for the menu display.

4.3 Extent of Implementation

In a form based system, a user defines a form by creating a form template. Once the form has been defined users may create form instances by 'filling in' the form template. The form instances will be stored into a data base, which will be used for transmission. To interface to a form based system, the Form Editor System is an important supporting component as it provides a facility for creating a form template. The form template is a definition of the form, describing the field type, field length, and the field content of a form. Once the form template has been stored, the user may input the data or information to create many form instances for the form processing that is not included in this implementation.

The Form Editor System was designed to allow the user to create a form, modify a form, store the field types to create a form template, print or display a form, delete the form and form template in the system. Therefore, the implementation is centered on creating a form and a form template for later use.

4.4 Testing

The Form Editor System is highly dependent on the form based system which is currently not implemented. This prevents the complete testing of the System, however, a test program has been utilized to verify its consistency, completeness, and correctness and to examine its behavior during execution.

First, we test the form editing. Actual examples of form editing in Create, Modify, and Store functions have been presented in Chapter 2. When the user creates a form, a form mapping file created in the system. We use the print form function to display the form or the modify form function to retrieve the form on the screen to test the created form. When the user modifies a form, a new form mapping file is created and replaces the old form mapping file. Using 'E' symbol for modification in this module, the user can verify the modify form process to prove its correctness. When the user stores a form to create the form template for later use, the testing of form template is discussed in the next paragraph. When the user prints or displays a form, the hard copy is the same as the screen display. When the user deletes a form (both the form and the form template), it can no longer be retrieved from the System. Therefore, the form editing can supply electronic forms instead of paper forms.

Second, the test program was used to check the form template. It exercises checks on the field length, and then the field type. When the user types in the field data is too long for the defined field length. It displays an error message and lets the user input the data again. When the field type is alphanumeric or numeric, it checks the field type after the user typing the value for this field. If the input data is wrong field type, it displays an error message and lets the user input data again. When the field type is unchangeable or signature, it shows the field type message and lets the user fill in the field. Their results should be implemented in a form processing system. When the field type is retrieved or calculated, the field content contains the name of the retrieving data file or the formula for calculation. The result of calculation will be displayed on this field automatically. The retrieved data from the named data file (which is designed for testing too) also will be displayed automatically. Therefore, the form template in this system is feasible for other form based systems.

CHAPTER 5

CONCLUSION

The preceding chapters describe a form editor system which allows the design of a form and the insertion of field types and semantic information into a form to create a form template for later use. The development phase for a system consists of preliminary design, detail design, coding, and testing. The first two steps have been presented in Chapter 3; the last two steps have been presented in Chapter 4.

The System has been tested and actual examples have been presented in Chapter 2. This Form Editor System has proved that implementation of form-based concepts on a microcomputer without hard disk storage capability is quite feasible. The System appears to exhibit : (1) Simplicity - Functions are basically based on just a few operations : create, modify, store, print, and delete. (2) Independence - Each function in the system is independent. (3) Uniformity - The form in the System is uniform to any function. (4) Understandability - The system is user-friendly, and is understandable to a non-technician to operate it. (5) Feasibility - The form template in the System is feasible for form-based systems.

This Form Editor System is an important supporting component to a form-based system as it provides a facility for creation and storage of electronic forms and form templates, but it does have some limitations which should be the subject of future research work. Among these limitations are:

(1) In the create function, the users can design a form with a limited form area. The form can not be scrolled, it is a single-paged form within an area of 75 characters by 16 lines.

(2) One field name corresponds to one data item only. The main reason for this

restriction was to avoid difficulty of displaying arbitrarily repeating groups under the form template.

(3) The field name must be specified by a colon, otherwise it will be skipped.

(4) The field length is specified by the number of underscores and follows a space for the delimiter use. If the user forgets the space, the System provides a default length for the field length.

(5) The modify function uses line editor to modify the form. To modify a form line by line is inconvenient for the user. Also the insert and add functions for the line overwrites the original line (the original line will not be deleted). The change line has to be retyped the whole line.

(6) The store function is rudimentary. There are only six field types allowed; they are Alphanumeric(A), Numeric(N), Unchangeable(U), Retrieved(R), Calculated(C), and Signature(S). And only one field type is allowed for each field on a form.

(7) Simple calculation involves only extensive or total field items. The formula must be specified for each field for which it is used.

(8) The System does not let the user input data to create the form instance.

(9) The form template file in the System should be included the form mapping file.

In summary, the future work needs to be done on the following :

. Expansion of the form size.

. Scrolling the form as up-down, left-right, and the window function.

. More flexibility for allowable forms.

. Fields should be able to be compound valued or repeating group values.

. More field types and combined field types can be allowed.

. More complex calculation could be carried out.

. The formula can be specified once and applied to several fields.

. The form mapping file and the form template should be able to be combined into one file after store function.

. Connect to other work stations to make an intelligent form.

The broad objects of the Form Editor System have guided the ongoing development. Actual user experience and feedback with the current system have been an invaluable means of validating both the implementation and presentation techniques. More experience with the System will uncover new areas for potential improvement.

BIBLIOGRAPHY

1. Barcomb D. "Office Automation : A Survey of Tools and Technology," Digital Equipment Corporation, 1981.

2. Ellis, C. A., and Nutt, G. J. "Office Information Systems and Computer Science," ACM Computing Surveys, Vol. 12, No. 1, March 1980, PP. 27-60.

3. Embley, D. W. "A Form-Based Programming System," Department of Computer Science, University of Nebraska, October 1980, PP. 1-37.

4. Gehani, N. H. "The Potential of Forms in Office Automation," IEEE Transactions on Communications, Vol. COM-30, No. 1, January 1982, PP. 120-125.

5. Hammer, M., Howe, W. G., Kruskal, V. J., and Wladawsky, I. " A Very High Level Programming Language for Data Processing Applications," Communications of the ACM, Vol. 12, No. 11, November 1977, PP. 832-840.

6. Hammer, M., Kunin J. S., Schoichet S. "What Makes a Good User Interface ?" The Fourth Annual Office Automation Conference, February 1983, PP. 121-130.

7. Ladd I., and Tsichritzis, D. "An Office Form Flow Model," National Computer Conference, 1980, PP. 533-540.

8. Pressman, Roger S. "Software Engineering : A Practitioner's Approach," McGraw-Hill, Inc. 1982.

9. Tsichritzis, D. "OFS : An Integrated Form Management System," Computer Systems Research Group, University of Toronto, 1980, PP. 1-17.

10. Tsichritzis, D. "Form Management," Communications of the ACM, Vol. 25, No. 7, July 1982, PP. 453-478.

11. Tsichritzis, D., Rabitti F. A., Gibbs S., Nierstrasz O., and Hogg J. "A System for Managing Structured Messages," IEEE Transactions on Communications, Vol. COM-30, No. 1, January 1982, PP. 66-73.

12. Zloof, M. M., and de Jong, S. P. "The System for Business Automation (SBA) : Programming Language," Communications of the ACM, Vol. 20, No. 6, June 1977, PP. 385-396.

13. Zloof, M. M. "Query-By-Example : A Data Base Language," IBM System Journal, Vol. 16, No. 4, 1977, PP. 324-343.

14. Zloof, M. M. "Office-By-Example : A Business Language that Unifies Data and World Processing and Electronic Mail," IBM System Journal, Vol. 21, No. 3, 1982, PP. 272-304.

15. IBM Disk Operating System Reference Manual, Microsoft, Inc., Boca Raton, Florida 33432, 1983.

16. TURBO Pascal Reference Manual, Boralnd International Inc., 4808 Scotts Valley Drive, Suite 1, Scotts Valley, CA 95066, 1983.

APPENDIX A

THE USER'S MANUAL

Table of Contents

1. Introduction

Forms play a central role in the technologically advanced business world. Day-to-day business activities generally involve the use of forms. This is a reference manual for the Form Editor System, a facility designed to help you with your form processing needs.

Forms can be thought of as texts which require certain values to be filled in certain slots called fields. A blank form as we normally see it is called a form structure and shall be referred to hereinafter simply as 'form'. To store a form in this System, however, would require you to create what is called a form template. This is done by specifying the field types on the form. The field type is simply the entry of data type that goes into the field. Therefore, a form template is a definition of the form, describing the field type, field length, and the field content (field content specifies the retrieving data file name for field type 'R' or the formula for field type 'C') of a form.

This System allows you to create a form, modify a form which was previously created, store the field types to create a form template, print a form on the printer, display a form on the screen, and delete the form and the form template from the system. The System is 'user-friendly' and menu-driven, you should rarely find yourself in need of going through the details in this manual. Refer to this manual for starting procedures and for interpretation of command or message for which you might not be sure of the meaning. An error message directory is provided in section 10.

2. Starting the System

To start the system, one has to follow four steps : (1) Connect a printer with the computer, and turn on the power for both. (2) Put the system disk on 'A' disk drive and load MS-DOS by entering the date and the time. (3) Put the working disk on 'B' disk drive. (4) Type 'form' (the name of program) and hit the 'return' key. This will display the message, "WELCOME TO FORM EDITOR SYSTEM". After the WELCOME display, the main menu will appear on the screen. From hereon, the computer will be guiding you with questions. Answer the questions by typing the letter corresponding to one of the answer options given. Capital and lower case letters are both acceptable. Then, hit the return key to proceed. The following sections describe the computer message and what they mean.

3. The Main Menu

The main menu allows you to specify what you want to do. You are given access to one of six functions by simply typing the following letters. If any other character is typed, the System will sound the bell twice as a warning, and an error message " ** Sorry ! It is a wrong command, Please try again !! **" will be displayed on the screen. After a correct command is given, the main menu will be erased and the menu for the function you chose will appear.

Command	Function
'C'	Create a form within 75 X 16 area
'M'	Modify an existing form
'S'	Store the field types to create a form template
'P'	Print a hard copy of form on the printer or display it on the screen
'D'	Delete the form and form template
'E'	Exit the system (the only command for quitting)

4. CREATE FORM Function

The System first displays the create form area on the screen; it then sounds the bell and asks, "Do you need help (y/n) ?". If you answer 'Y', the help menu will be displayed on the screen. At the same time, the following is displayed at the bottom of the screen :

COMMAND :__ (C-CONTINUE CREATE FORM R-RETURN TO MAIN MENU)

This means you are to type 'C' or 'R' if you are through with the help menu. If you type any other character, the system will sound the bell twice as a warning and give the error message, " ** Sorry! It is a wrong command, Please try again !! **". Type 'R' if you want to return to the main menu. If you type 'C' (or answered 'N' to the first question), you may proceed with the create form process.

In the form creation process, the cursor first moves to the 'FORM-NAME' position and waits for the name of the form that is to be created. After you type the form name (maximum of 8 characters), the system checks whether it is unique to the system. If it is not, the system will sound the bell twice as a warning and display error message "** Duplicate form name, please use another form name !! **" on the screen. After the message, the cursor then goes back to the 'FORM-NAME' position for another name to be entered. The system allows you three attempts to give an acceptable form name. If you do not succeed after three attempts, the system will return to the main menu.

It would be a good idea to sketch a draft of the form before starting. All you will need to do then would be to duplicate the image of the form on the screen. In creating the form, be aware of the following : (1) A number of fields can be put on the same line. (2) To get to the next line, hit the 'return' key. (3) The field name must be immediately followed by a ':'. (4) After the ':' type the desired number of the underscores and a space(otherwise the length will be set to only 10 underscores). (5) the maximum allowable field length is 60. When you are done, type a '*' on a new line and the system will then return to the main menu.

5. MODIFY FORM Function

Like the create form function, this function starts with a help menu. Please refer to section 4 for details.

Once you are ready to proceed with the modification process, the cursor moves to the 'FORM-NAME' position and waits for you to give the name of the form you want to modify. After the form name has been typed, the system checks whether the name exists in the system. If it does not exist, the system will sound the bell twice as a warning and display the error message "** Form is not found, please try again !! **" on the screen. After the message is displayed, the cursor will go back to the 'FORM-NAME' position for you to enter another name. The system give you three chances to give an acceptable form name. If you do not succeed after three attempts, the system will return to the main menu.

After an acceptable form name has been entered, the system retrieves the named form and displays it on the screen. The form can be modified line by line only. At the end of each line, the system sounds the bell and waits for a command. You are given the following options :

Command	Meaning
'A'	Add a new line or lines to the end of the form
'D'	Delete the current line
'I'	Insert a new line or lines before the current line (the new line will overwrite on original line, but the original line will not be deleted)
'N' or <return>	Make no change on the current line
'Y'	Change the current line (type the whole line)
'E'	End of the modification

If you type the command 'E', the system will display the message "** Are you finished the form modification (y/n) ? ". If you answer 'N', the modified form will be displayed on the screen for verification. You may then repeat the modifying procedure until you are satisfied. If you answer 'Y' then the system returns to the main menu.

6. STORE FORM Function

This function starts with a help menu. Please refer to section 4 for detail.

Once you are ready to proceed with the store-form process, the system sounds the bell and displays the message "Is the current form to be stored (y/n) ?". If you answer 'Y' then you may proceed to store the current form. Otherwise the system sounds the bell and displays the message "Enter the name of form to be stored ?" and waits for you to type in the form name. The system checks whether it exists in the system. If it does not exist, the system sounds the bell twice as a warning and displays the message "** The form is not found !! **", and returns to the main menu.

If the form exists, the system retrieves the form and displays it on the screen. As the cursor moves to each field, the system will sound the bell and display the message "** What is the field type ?". Use the following symbols to specify the field type :

Symbol	Meaning
'A'	Alphanumeric
'N'	Numeric
'U'	Unchangeable
'R'	Retrieved
'C'	Calculated
'S'	Signature

If the field type is 'R', the system will ask the question "** Retrieve from which file ? **" and wait for you to type in the name of the file. If the field type is 'C', the system will ask "** Please give the formula for this field = " and wait for you to type the formula for this field. The formula is expressed in term of the other fields represented by ordinal number of fields on the form. The only mathematical operators allowed are +, -, *, and /.

Before storing a field type, the system checks the field length. If the underscores were not followed by a space when the form was created, the message "** Field length not specified, length set = 10. Do you want to change the field length (y/n) ?" is displayed. If you answer 'N', this field length is set to the default field length of 10. If you answer 'Y', the system will ask you "What is the length ?".

When you have specified all the field types, the system will create a form template. The system will sound the bell and display the message, "Do you want to check the form template (y/n) ?" If you answer 'N', the system returns to the main menu. If you answer 'Y', the system displays the form template which includes the field name, field type, field length, and field content of each field. The system will then sound the bell and display the message "** If the form template is not correct, please store the form again !!", and then return to the main menu.

7. PRINT FORM Function

The system first displays the message "Is the current form to be printed or displayed (y/n) ?". If you answer 'N' the system will display the message "Enter the name of form to be printed or displayed ?" Type in the form name. The system will check whether it exists in the system. If it does not exist, the system sounds the bell and displays the message "The form is not found !" and returns to the main menu. If it does, the system will display "Do you want it to be printed or displayed (P/D) ?" If you want a hard copy, type 'P'. After printing the form on the printer, the system returns to the main menu. If you want the form to be displayed on the screen, type 'D'. To return to the main menu, hit any key.

8. DELETE FORM Function

The system first displays the message "Enter the name of form to be deleted ?" Once you have typed in the form name. The system will display a warning message (in case you change your mind). Type 'Y' if you want to proceed with the deletion. Otherwise, type 'N' the system then return to the main menu.

9. EXIT Function

Before exiting the system, the system first checks to determine if current form has been stored. If the form has not been stored, the system displays a warning message to provide you an opportunity to store the form before exiting. Type 'N' if you want to store the current form. The system then returns to the main. Type 'Y' if you want to exit the system. The system then displays the message, "BYE FORM EDITOR SYSTEM" on the screen.

10. Error Message Directory

(1) ** Duplicate form name, please use another form name !! **

: When the user creates a form which exists in the system.

(2) ** Field length not specified, length set = 10 !! **

: When the user stores a form, the field length is not followed by a space for delimiter, the system gives a default length.

(3) ** Form is not found, please try again !! **

: When the user retrieves a form for Modify and Store functions, the system can not find the form.

(4) ** Sorry ! It is a wrong command, please try again !! **

: When the user requests a command which is wrong.

(5) ** The ending line of the form !! **

: When the user creates or modifies a form excess 16 lines.

(6) ** The form is not found !! **

: When the user retrieves a form for Print function.

(7) ** The name is not acceptable, please type again !! **

: When the user types in the form name which is up to 8 characters and must be letters.

```
(******************************************)
(*   FORM EDITOR SYSTEM MAIN PROGRAM    *)
(******************************************)
program formeditor;
const
  blankline = '                                                            ';
  blank = '                              ';
type
  ss = string[2];
  nn = string[15];
  mm = string[40];
  ms = string[60];
  ll = string[75];
  ls = string[80];
var
  ch, bell : char;
  q : file of ll;
  h : text;
  line : ll;  line2: ls;1
  count : integer;
  name, filename, flname : nn;
(* THE PROCEDURE FOR READING THE SCREEN DISPLAY FILE *)
procedure screendisplay;
begin
  readln(h,line2);
  repeat
    writeln(con,line2);
    readln(h,line2);
  until (copy(line2,1,3) = '.PA') or eof(h);
end;
(* THE PROCEDURE ACCEPTS THE COMMAND POSITION *)
procedure position(x,y:integer; var ch:char);
begin
  gotoxy(x,y);  read(kbd,ch);1
  ch := upcase(ch);
end;
(* THE PROCEDURE DISPLAYS THE WRONG COMMAND MSSAGE *)
procedure errorcommand;
begin
  if ch in [' '..'~'] then
     begin
       gotoxy(10,21);
       write(bell,bell);
       write('** Sorry ! It is a wrong command, please try again !! **');
     end;
end;
(* THE PROCEDURE DISPLAYS THE WELCOME SCREEN DISPLAY *)
procedure initial;
begin
  assign(h,'b:init.doc');
  reset(h);
  clrscr; screendisplay;
  delay(2000);
  clrscr; screendisplay;
  close(h);
end;
(* THE PROCEDURE DISPLAYS THE MAIN MENU *)
procedure menu;
begin
  assign(h,'b:menu.doc');
  reset(h);
  clrscr; screendisplay;
  close(h);
end;
(* THE PROCEDURE DISPLAYS THE HELP MENU FOR CREATE, MODIFY, AND STORE MODULES *)
procedure helpmenu(name:nn);
begin
  assign(h,name);
  reset(h);
  clrscr; screendisplay;
  gotoxy(10,10);
  write(bell,'Do you need help (y/n) ? ');
  position(35,10,ch);
  if ch = 'Y' then
     begin
       clrscr; screendisplay;
       repeat
         position(18,23,ch);
         case ch of
           'C': begin clrscr; screendisplay; end;
           'R': begin clrscr; close(h); end;
         else errorcommand; end;
       until (ch = 'C') or (ch = 'R');
     end  else
     begin
       gotoxy(10,10);
       write(blankline); close(h);
     end;
end;
(* THE FUNCTION FOR CHECKING WHETHER THE FORM EXISTS IN THE SYSTEM *)
function exist(filen:nn): boolean;
var  f:file;
begin
  {$I-}  assign(f,filen);
  reset(f);  {$I+}
  if ioresult <> 0 then
       exist := false else
       exist := true;
end;
(* THE PROCEDURE CHECKS THE FORM NAME FOR CREATE, MODIFY, AND STORE MODULES *)
procedure checkformname(ttype:char; var filename:nn; var count:integer);
var
  flag : boolean;
  message: ms;
  i,j,k,l : integer;
begin
  if ttype = 'c' then
       message := '** Duplicate form name, please use another form name !! **'
```

```
        else message := '** Form is not found, please try again !! **';
   if ttype = 'p' then
      begin  i := 10; j := 14; k := 48; l := 9;
      end  else
      begin  i := 2; j := 10;  k := 14;  l := 3; end;
   if ttype = 'c' then
        flag := exist(filename)
        ielse flag := not exist(filename);
   while flag and (count < 3) do
     begin
       count := count + 1;
       write(bell,bell);
       gotoxy(i,j);
       write(message);
       gotoxy(k,l);
       writeln(blank);
       gotoxy(k,l);
       readln(con,filename);
       filename := concat(filename,'.map');
       gotoxy(i,j);
       writeln(blankline,blank);
       if ttype = 'c' then
            flag := exist(filename)
          else flag := not exist(filename);
     end;
end;
(* THE PROCEDURE FOR CREATING A FORM *)
procedure createform(filename:nn);
var
  x,y : integer;
begin
  assign(q,filename);
  rewrite(q);
  x := 3;  y := 5;
  gotoxy(x,y);
  readln(con,line);
  while(copy(line,1,1)<> '*') and (y<20) do
  begin
    write(q,line);
    y := y+1;
    gotoxy(x,y);
    readln(con,line);
  end;  close)q);
end;
(* THIS IS CREATE MODULE *)
procedure createmodule;
var
  i,l,p : integer;
  filename,ffname:nn;
  namechar:string[1];
  filelist:ms;
  ok:boolean;
begin
  filelist := 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';
  i := 0;  ok := false;
  name := 'b:create.doc';
  helpmenu(name);
  if ch <> 'R' then
     begin
       repeat
         gotoxy(14,3);
         readln(flname);
         ffname := flname;
         l := length(ffname);
         i := i + 1;
         namechar := copy(ffname,i,1);
         p := pos(namechar,filelist);
         if p = 0 then begin
              gotoxy(10,10);
              write('The name is not acceptable, please type again !!');
              delay(2000);
              gotoxy(10,10);
              write(blankline);
              ok := false;
            end else
              ok := true;
       until (l < 16) and (ok);
       filename := concat(flname,'.map');
       count := 0;
       checkformname('c',filename,count);
       if count < 3 then
          begin
            l := pos('.',filename);
            l := l -1;
            flname := copy(filename,1,l);
            createform(filename);
          end;
     end;
  menu;
end;
(* THE PROCEDURE FOR RETRIEVING A FORM AND DISPLAYING IT ON THE SCREEN *)
procedure retrieveform(filename:nn);
var
  x,y : integer;
  r : file of ll;
begin
  filename := concat(filename,'.map');
  assign(r,filename);
  reset(r);
  x := 3;  y := 4;
  while not eof(r) do
    begin
      y := y+1;
      gotoxy(x,y);
      read(r,line);
      write(line);
    end;
  close(r);
```

```
end;
(* THE PROCEDURE FOR MODIFYING A FORM *)
procedure modifyform(filename:nn; var ok:boolean);
var
  i,j,x,y : integer;
  f: file of ll;
  fname : nn;
  oldline, newline : ll;
  finish : boolean;
begin
  assign(q,filename);
  reset(q);
  fname := concat(flname,'.mmap');
  assign(f,fname);
  rewrite(f);
  ch := ' '; x := 3;  y := 4;  j := 0;
  while (ch <> 'E') and (y < 20) do
    begin
      for i := 1 to 75 do
          newline[i] := ' ';
      y := y + 1;
      if not eof(q) then read(q,line)
         else line := newline;
      repeat
        write(bell);
        position(78,y,ch);
        case ch of
        'E' : begin
                write(f,line);
                while not eof(q) do
                begin  read(q,line);  write(f,line);  end;
        'A' : begin
                write(f,line);
                y := y + 1;
                while not eof(q) do
                  begin read(q,line); write(f,line); y := y+1; end;
                finish := false;
                while ( y < 20) and ( finish = false ) do
                  begin
                    gotoxy(x,y);
                    read(line);
                    write(f,line);
                    write(bell);
                    position(78,y,ch);
                    if ch = 'E'
                       then finish := true
                       else  finish := false;
                    y := y + 1;
                  end;
                if y = 20 then
                  begin
                    gotoxy(x,y);
                    lowvideo;
                    write(bell,'** The ending line of the form **');
                    highvideo;
                    ch := 'E';
                  end;
              end;
        'I' : begin
                oldline := line;
                finish:= false; j:=y;
                while ( j < 19 ) and ( finish = false ) do
                  begin
                    gotoxy(x,y);
                    read(line);
                    write(f,line);
                    j := j + 1;
                    position(78,y,ch);
                    if ch = 'I'
                       then finish := false
                       else begin
                           ch := 'I';
                           finish := true; end;
                  end;
                write(f,oldline);
              end;
        'Y' : begin
                gotoxy(x,y);
                readln(con,line);
                write(f,line);
              end;
        ^M,'N' : write(f,line);
        'D' :    ;
        end;
      until ch in ['E','Y','N','D',^M,'A','I'];
    end;
  close(q); erase(q); close(f);
  rename(f,filename);
  gotoxy(3,20);
  write(blankline,blank);
  gotoxy(3,20);
  lowvideo;
  write(bell,'** Are you finished the form modification (y/n) ? ');
  highvideo;
  position(56,20,ch);
  if ch = 'Y' then  ok := true
  else ok := false;
end;
(* THIS IS THE MODIFY MODULE *)
procedure modifymodule;
var
  filename:nn;
  l,i,x,y : integer;
  ok : boolean;
begin
  ok := false;
  name := 'b:modify.doc';
  helpmenu(name);
  if ch <> 'R' then
    begin
      gotoxy(14,3);
      readln(con,flname);
      filename := concat(flname,'.map');
      count := 0;
      checkformname('m',filename,count);
      if count < 3 then
         begin
           while not ok do
             begin
               for i := 5 to 20 do
                    begin
                      gotoxy(3,i);
                      write(blankline,blank);
                    end;
               l := pos('.',filename);
               l := l - 1;
               flname := copy(filename,1,l);
               retrieveform(flname);
               modifyform(filename,ok);
             end;
         end;
      close(h);
    end;
  menu;
end;
(* THE PROCEDURE FOR THE FIELD CONTENT OF THE FORM TEMPLATE *)
procedure fieldcontent(typef :char; content :nn);
var
  message : ll;
begin
  if typef = 'R'
    then message := concat('** Retrieved from [',content,'] **');
  if typef = 'C'
    then message := concat('** The formula is [',content,'] **');
  gotoxy(3,22);
  write(bell,message);
  gotoxy(3,23);
  write('** Please hit any key to continue !!');
  repeat
    gotoxy(70,23);
  until keypressed;
end;
(* THE PROCEDURE DISPLAY THE FORM TEMPLATE FOR CHECKING *)
procedure formtemplate(filename: nn);
type
  fields = record
              fno : integer;
              fx : integer;
              fy : integer;
              ftype : char;
              flength : integer;
              finfo : nn;
```

```
          end;
var
  content : fields;
  g : file of fields;
  i,s : integer;
  ff, fieldtype : nn;
begin
  assign(h,'b:template.doc');
  reset(h);
  clrscr; screendisplay;
  gotoxy(14,3);
  write(filename);
  retrieveform(filename);
  ff := concat(filename,'.tmp');
  assign(g,ff);
  reset(g);
  while not eof(g) do
    begin
      delay(100);
      gotoxy(10,10);
      read(g,content);
      lowvideo;
      with content do
          begin
            case ftype of
              'A' : fieldtype := 'alphanumeric';
              'N' : fieldtype := 'numeric';
              'U' : fieldtype := 'unchangeable';
              'R' : fieldtype := 'retrieved';
              'C' : fieldtype := 'calculated';
              'S' : fieldtype := 'signature';
            end;
            delay(100);
            s := fx -1;
            for i := 1 to flength do
              begin
                s := s + 1;
                gotoxy(s,fy);
                write(' ');
              end;
            gotoxy(fx,fy);
            write(fieldtype,'[',flength,']');
            if ftype = 'R'
               then fieldcontent('R',finfc);
            if ftype = 'C'
               then fieldcontent('C',finfo);
        end;
      delay(100);
    end;
  gotoxy(3,22);
  write(bell,'** If it is not correct, please store the form again !!');
  gotoxy(6,23);
  write('Please hit any key to return to the main menu !! ');
  repeat
```

```
    gotoxy(70,23);
  until keypressed;
  highvideo;
end;
(* THE PROCEDURE FOR STORING A FORM *)
procedure storeform(filename:nn);
type
  fields = record
             fno : integer;
             fx : integer;
             fy : integer;
             ftype : char;
             flength : integer;
             finfo : nn;
           end;
var
  message : mm;
  scnt, lcnt, fxx,fyy,i : integer;
  sfilename, finfo : nn;
  ftype : char;
  fnum : ss;
  content : fields;
  g : file of fields;
  ok : boolean;
begin
  i := 0; scnt := 0;  lcnt := 0;
  sfilename := concat(filename,'.tmp');
  assign(g,sfilename);
  rewrite(g);
  filename := concat(filename,'.map');
  assign(q,filename);
  reset(q);
  fyy := 4;
  while not eof(q) do
    begin
      read(q,line);
      scnt := pos(':',line);
      lcnt := 0; fxx := 2;
      fyy := fyy +1;
      while scnt <> 0 do  begin
          ok := false;
          delay(1000);
          i := i + 1;
          str(i:2,fnum);
          content.fno := i;
          fxx := fxx + scnt + lcnt + 1;
          content.fx := fxx;
          content.fy := fyy;
          message := concat('** What is the [',fnum,'] field type ? ');
          gotoxy(3,20);
          write(bell,bell);
          write(blankline,blank);
          gotoxy(3,20);
          lowvideo;
  write(message);
  highvideo;
  repeat
    position(fxx,fyy,ftype);
    if ftype in ['A','N','U','R','C','S'] then
       ok := true;
  until ok;
  case ftype of
    'R' : begin
            gotoxy(3,20);
            lowvideo;
            write(bell,'** Retrieve from which file ? **');
            highvideo;
            gotoxy(40,20);
            read(finfo);
          end;
    'C' : begin
            gotoxy(3,20);
            lowvideo;
            write(bell,'** Please give the formula for this field = ');
            highvideo;
            gotoxy(47,20);
            read(finfo);
          end;
    'A','N','U','S' : finfo := '                    ';
  end;
  gotoxy(3,20);
  write(blankline,blank);
  content.ftype := ftype;
  content.finfo := finfo;
  delete(line,1,scnt);
  lcnt := pos(' ',line);
  if lcnt = 0 then
     begin
       lowvideo;
       write(bell,bell);
       gotoxy(3,20);
       write('** Field length not specified, length set = 10');
       delay(1000);
       gotoxy(3,20);
       write('Do you want to change the field length (y/n) ?');
       position(51,20,ch);
       case ch of
         'Y' : begin gotoxy(51,20);
                 write('What is the length ?');
                 read(lcnt);  end;
         'N' : lcnt := 10;
       end;
     end;
  content.flength := lcnt;
  delete(line,1,lcnt);
  scnt := pos(':',line);
  write(g,content);
end;
```

```
      delay(1000);
    end;
  delay(1000);
  close(q); close(g); close(h); clrscr;
  gotoxy(10,10);
  write(bell);
  write('Do you want to check the form template (y/n) ?');
  gotoxy(10,12);
  write('The answer : ');
  position(23,12,ch);
  if ch = 'Y' then
     formtemplate(flname);
end;
(* THE PROCEDURE FOR CHECK THE FORM IN THE SYSTEM *)
procedure checkformexist(filename:nn);
var  ff:nn;
begin
  ff := concatfilename,'.map');
  if exist(ff) then
     begin
       flname := filename;
       gotoxy(14,3);
       write(flname);
       gotoxy(3,10);
       write(blankline,blank);
       gotoxy(3,12);
       write(blankline,blank);
       retrieveform(flname);
       storeform(flname);
     end  else
     begin
       gotoxy(14,13);
       write(bell,bell);
       lowvideo;
       write('** The form is not found !! **');
       highvideo;
       delay(1000);
     end;
end;
(* THIS IS THE STORE MODULE *)
procedure storemodule;
var  fname :nn;
begin
  name := 'b:store.doc';
  helpmenu(name);
  if ch <> 'R' then
     begin
       gotoxy(6,10);
       lowvideo;
       write(bell,'Is the current form [ ',flname,' ] to be stored (y/n) ? ');
       highvideo;
       repeat
         position(70,10,ch);
         case ch of
```

```
          'Y' : checkformexist(flname);
          'N' : begin
                  gotoxy(14,12);
                  lowvideo;
                  write(bell,'Enter the name of form to be stored ?');
                  highvideo;
                  gotoxy(55,12);
                  read(fname);
                  checkformexist(fname);
                end;
          else errorcommand; end;
        until ch in ['N','Y'];
        close(h);
      end;
  menu;
end;
(* THE PROCEDURE FOR PRINTING A HARD COPY OF FORM *)
procedure hardcopy;
begin
  assign(q,filename);
  reset(q);
  while not eof(q) do
    begin
      read(q,line);
      writeln(lst,line);
    end;
  close(q);
end;
(* THIS IS THE PRINT MODULE *)
procedure printmodule;
var  ff:nn;
begin
  assign(h,'b:print.doc');
  reset(h);
  clrscr; screendisplay;
  gotoxy(35,10);
  write(con,flname);
  position(30,12,ch);
  if ch = 'N' then
     begin
       gotoxy(12,14);
       write('Enter the name of form to be printed or displayed ?');
       gotoxy(65,14);
       readln(con,ff);
     end  else
       ff := flname;
  filename := concat(ff,'.map');
  if exist(filename) then
    begin
      gotoxy(12,16);
      write('Do you want it to be printed or displayed (P/D) ?');
      repeat
        position(65,16,ch);
        case ch of
```

```
          'P' : hardcopy;
          'D' : begin
                  clrscr; screendisplay;
                  retrieveform(ff);
                  gotoxy(3,22);
                 write('** Please hit any key to return to main menu !');
                  repeat
                    gotoxy(70,22);
                  until keypressed;
                end;
        else errorcommand; end;
      until (ch = 'P') or (ch = 'D')
    end else
    begin
      gotoxy(12,16);
      write(bell,'** The form is not found !!');
    end;
  close(h);
  menu;
end;
(* THIS IS THE DELETE MODULE *)
procedure deletemodule;
var  ff:nn;
begin
  assign(h,'b:delete.doc');
  reset(h);
  clrscr; screendisplay;
  gotoxy(53,9);
  readln(con,ff);
  filename := concat(ff,'.map');
  ff := concat(filename,'.tmp');
  clrscr; screendisplay;
  close(h);
  position(33,16,ch);
  if (ch = 'Y') and (exist(filename)) then
     begin
       assign(q,filename);
       erase(q);
     end;
  if (ch = 'Y') and (exist(ff)) then
     begin
       assign(q,ff);
       erase(q);
    end;
  menu;
end;
(* THE PROCEDURE FOR CHECKING STORE FORM BEFORE EXIT THE SYSTEM *)
procedure checkstore(ff : nn; var out : char);
begin
  if exist(ff) then
     begin
       repeat
         readln(h,line2);
       until copy(line2,1,3) = '.PA';
      end else
      begin
        screendisplay;
        position(33,16,out);
      end;
end;
(* THIS IS THE EXIT MODULE *)
procedure exitmodule;
var
  out : char;
  fname,ffname : nn;
begin
  out := ' ';
  fname := concat(flname,'.tmp');
  clrscr;
  assign(h,'b:exit.doc');
  reset(h);
  checksore(fname,out);
  if out = 'N' then
     begin close(h); menu;
     end  else
     begin
       clrscr; screendisplay;
       close(h);
       ffname := concat(flname,'.tmp');
       if not exist(ffname) then
          begin
            ffname := concat(flname,'.map');
            assign(q,ffname);
            erase(q);
          end;
       halt;
     end;
end;
(* THIS IS THE MAIN MODULE, AND THE PROGRAM STOPS IN EXITMODULE PROCEDURE *)
begin
  initial;
  bell := ^G;
  flname := 'dummyform';
  repeat
    position(18,23,ch);
    case ch of
    'C' : createmodule;
    'M' : modifymodule;
    'S' : storemodule;
    'P' : printmodule;
    'D' : deletemodule;
    'E' : exitmodule;
    else
    errorcommand;
    end;
  until true = false;
  end.
```

FORM EDITOR SYSTEM

by

JONY CHANG

B.S., National Taiwan University, 1978

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1984

ABSTRACT

Forms have been used extensively in offices. In addition, they have been proposed and used as the basic user interface data structure in many data entry and office information systems. Therefore. electronic forms are becoming part of our every life in the modern, technological society. Generally anyone who deals in the business world knows how to fill out forms, understands their processing and most people can also design a form of their own for a given situation. Forms play a central role in several current business automation projects, e.g., Business Definition Language (BDL), System for Business Automation (SBA), Officetalk-Zero, Office Forms System (OFS), and Forms Programming System (FPS), Structured Message System (SMS).

This report describes a Form Editor System which allows the design of a form and the insertion of field types and semantic information into a form to create a form template for later use. The System is implemented on the Columbia Data Products (CDP) Multi-Personal Computer (MPC) using the TURBO Pascal language. This Form Editor System an implementation of form-based concepts on a microcomputer without hard disk storage capability has shown the feasibility of the use of such concepts on contempory personal computer systems.