1988-06

Document generator software design that supports Turkish alphabet

Akinci, Metin

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DOCUMENT GENERATOR SOFTWARE DESIGN
THAT SUPPORTS TURKISH ALPHABET

by

Metin Akinci

June 1988

Thesis Advisor: Daniel Davis

Approved for public release; distribution is unlimited

URITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

REPORT SECURITY CLASSIFICATION Unclassified	1b RESTRICTIVE MARKINGS
SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
DECLASSIFICATION / DOWNGRADING SCHEDULE	
PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
NAME OF PERFORMING ORGANIZATION aval Postgraduate School	6b OFFICE SYMBOL (If applicable) Code 52
7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School	
ADDRESS (City, State, and ZIP Code) onterey, California 93943-5000	7b ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000
NAME OF FUNDING / SPONSORING ORGANIZATION	8b OFFICE SYMBOL (If applicable)
9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO	PROJECT NO.	TASK NO	WORK UNIT ACCESSION NO.

TITLE (Include Security Classification)
OCUMENT GENERATOR SOFTWARE DESIGN THAT SUPPORTS TURKISH ALPHABET

PERSONAL AUTHOR(S)
kinci, Metin

a. TYPE OF REPORT	13b TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
laster's Thesis	FROM _____ TO _____	1988 June	

SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government

COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	EGA, CGA, TSR, ADT, Information hiding, Abstraction

ABSTRACT (Continue on reverse if necessary and identify by block number)

The objective of this study is to design and implement software for an automatic document generator supporting the Turkish alphabet. The implementation in this study in mainly based on IBM personal computers and dot matrix printers.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL Prof. Daniel Davis	22b TELEPHONE (Include Area Code) (408) 646-3091
	22c OFFICE SYMBOL Code 52Dv

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

GPO 1986—606-24

Approved for public release; distribution is unlimited.

Document Generator Software Design that Supports Turkish Alphabet

by

Metin Akinci
Lieutenant J.G. Turkish Navy
B.S., Turkish Naval Academy, 1982

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1988

ABSTRACT

The objective of this study is to design and implement software for an automatic document generator supporting the Turkish alphabet. The implementation in this study is mainly based on IBM personal computers and dot matrix printers.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

Technological advances in computer hardware and software of the past decade has been rapid enough to be called a revolution. The wider the use of computers has been spread, the more newer application areas have evolved. The need for the use of the computers for the Turkish Navy has grown rapidly in recent years. However, the absence of the computer technology has limited the variety of areas where computers can be used. One aspect of the absence of this technology is the inability to use some characters in the Turkish alphabet. Because of this, computers are not widely being used for text processing purposes.

Although the Turkish Navy has begun to use computers in a variety of areas in recent years, lower level organizations such as ships and administrative offices have been doing business without the use of computers. One significant example of these jobs is to generate and to process official documents.

Questions we plan to address are: Can we use microcomputers to generate and store official documents in computers and will this reduce the amount of paper work? Does current technology allow us to use our own alphabet to generate documents? If the answer is 'yes', what is the most suitable hardware for these purposes?

This research attempts to find answers to these questions and investigate the current microcomputer technology to determine the feasiblity of text processing on different character sets not available on the system. To apply this to a specific area, we will design and implement software that meets the requirements for the specific application, in order to generate and process official documents.

A. PURPOSE

We investigate the current microcomputer technology in order to be able to design and implement a software that allows us to display and print extra characters in the Turkish alphabet.

This study is a design and implementation of software which can be used in ship bureaus to generate and process unclassified documents which includes all characters in the Turkish alphabet. By using microcomputers on this particular area, the amount of paper work, the loss of manpower and the negative impact of the lack of personnel can be reduced. It also saves space by allowing us to get rid of files that are used to keep a record of correspondence.

B. SCOPE

The objective of this study is to design and implement software for automatic document generator supporting the Turkish alphabet. This software is to be for general purpose so that it can be used in different bureaus that require various form of documents. Since the needs for different bureaus are almost identical and the only thing that differs is the forms of documents , it should allow the user to define his own templates easily. The software should remain the same, but form definitions that meet user needs should be easy to prepare. The software should be compatible with the computer hardware and printers used in the Turkish Navy.

C. ORGANIZATION

This study consists of five main chapters followed by conclusions and recommendations. The first chapter provides a brief introduction by defining the research objectives and its associated scope of effort, outline and organization of this thesis. Chapter II presents the system overview and objectives. The user requirements, overall design considerations, hardware requirements, and program components are also presented in this chapter. Design and implementation of the program will be discussed in Chapter III. The implementation of extra characters on the screen and printer will be presented in Chapter IV. Using the document generator is presented in Chapter V. This thesis concludes in Chapter VI by stating the conclusions and recommendations inferred by this study. The Turkish alphabet is presented in Appendix A. A user manual is presented in Appendix B. Appendix C will contain the program listing. Example template definition and program output document will be presented in Appendix D.

II. SYSTEM OVERVIEW AND OBJECTIVES

A. INTRODUCTION

This chapter consists of three sections : system overview, system objectives, program considerations, hardware requirements and program components.

System design is the process of planning a new system or one to replace or complement the existing system. But before this can be done, the system must thoroughly be understood and the following things must be taken into considerations. Who will make use of the system? What will the system do? How is it operated? What are the user requirements? How portable will it be? Will it be suitable for existing hardware ?

B. SYSTEM OVERVIEW AND USER REQUIREMENTS

This section presents a system overview providing a more complete understanding of what will be required. This will help explain how the software to be developed can be designed to best satisfy the user reqirements.

Each destroyer has a bureau where correspondence is performed and documents are prepared. Generally one petty officer and one seaman are assigned to perform this job. Typewriters are the only equipments used to fulfill these tasks. Each ship has its own preprinted document forms and reports. The personnel assigned to ship administrative offices are in charge of filling out documents submitted by related department personnel and delivering them to the appropriate places. Both difficulty in supplying preprinted forms and ease of mistyping make the bureau's task harder. Additionally, one copy along with the records of incoming and outgoing documents must be retained. There are two reference values to access the document or find the document stored in notebooks: reference by date and reference by dōcument number (which is unique). The document number consists of three letters which indicates the class of document and the order number within the class of document and last two digit of the current year.

In order to answer the question 'who' , we must take the seamen into consideration. In other words we should assume that the system will be used by an illiterate user community. No computer knowledge should be required.

The system should respond to the user requirements. It should provide the following properties.

1. Users should be able to fill out documents as described in the template definition. They should not be allowed to change the form of the document.

2. The system should meet the requirements of different forms. It should be easy for users to define new forms.
3. The system should be user friendly therefore easy for training personnel.
4. It should also provide a 'Small Turkish Word Processor' property. In other words it should allow the user to edit something without format.
5. Records of incoming and outgoing documents must be provided within the system.
6. It should store the documents requested by the user.
7. It should provide an ability to browse documents. The documents stored in database can be browsed but not changed.

C. OVERALL DESIGN CONSIDERATIONS

The following assumptions have been made in the design of this project.

1. No user knowledge of computers is required. User will be responsible for filling predefined spaces on screen as in form of document. The program will prevent the user from overwriting on non-fillable fields on the form of document.
2. There will not be any limit for the number of document forms to be used. Templates should be easy to define.
3. Since no user knowledge is required, program should be user friendly as much as possible.
4. The most significant assumption is that all template definitions will be entered correctly. Program will not check the templates. It will assume they were entered correctly.
5. The key to database search is by the document number which is unique. It is assumed that keywords to the database will be entered correctly.

D. HARDWARE REQUIREMENTS

In order to make use of this program, the following hardware requirements should be met by the machine on which this program is running. I mainly focused on IBM personal computers and its compatibles. Following hardware components must exist on the system in addition to the system itself.

1. The machine on which this program is running has to have a fixed disk in order to store documents in the database.
2. To meet one of the user requirements which is usage of extra characters in the Turkish alphabet, the system has to have either a Color Graphics Adapter or Enhanced Graphics Adapter. These conditions will be checked by the program during the installation process. If these requirements are not met by the system, program will exit by prompting user.
3. The extra character set in the Turkish alphabet to be printed is designed with respect to dot matrix printer. Since there is no way to check printer type, program will assume that an appropriate printer is attached to the system.

No further hardware components are required.

E. PROGRAM COMPONENTS

The program consists of seven modules. Modules have been determined according to the meaning of the task performed. The program modules and the files related to them are listed below. The detailed explanation will be presented in Chapter III. The program consists of following modules.

1. Database Module
2. Editor Module
3. Template Module
4. User Interface Module
5. System Functions Module
6. Printer Routines Module
7. EGA Character Generator Module
8. CGA Character Generator Module
9. Main module
10. Linked List Abstract Data Type
11. Double Linked List Abstract Data Type

III. DESIGN AND IMPLEMENTATION

A. INTRODUCTION

There are several phases to software design. The purpose should not be merely to meet the user requirements. In the design of the software, software engineering concepts should be taken into consideration. These concepts are modularity, abstraction, reusabilty and information hiding. Especially in big projects, application of these concepts makes the software easy to construct, to maintain and test.

The choice of the programming language is also a major factor in achieving the goal. I chose the C language to implement this program. The programs in this thesis have been written in the C language by using a TURBO C compiler [Ref. 1]. The features offered by the C language made it easier to apply the concepts stated above. Although this is not a big project, it attempts to use these concepts by taking advantage of the features offered by the C language.

In this chapter, I will explain how this program was designed and implemented under the light of these concepts. This chapter will provide explanations on design and implementation of the program, reusabilty and portabilty issues, and the program modules.

B. OVERALL PROGRAM DESIGN CONSIDERATIONS

In the design of this program, I took a *top-down design* approach. The main tasks required to meet the user requirements led me the modules of the program. Once the program is modularized then it is easy to construct the entire program by stepwise refinement method. Stepwise refinement helps us to easily apply the process of abstraction [Ref. 2: p 1053-1058]. Well modularized program also helps us to easily apply the process of abstraction [Ref. 3: p 1-43], information hiding [Ref. 4: p 339-344].

I constructed the program modules according to the tasks to be performed. By using C language's feature, each module has been designed and compiled separately. Then, each module has been integrated to other modules after completion and testing of separate modules, by using structured programming and the stepwise refinement technique. Overall program construction, together with the module, is shown in Figure 1. This program consists of the following program modules.

1. TEMPLATE.C: Template generator routine.

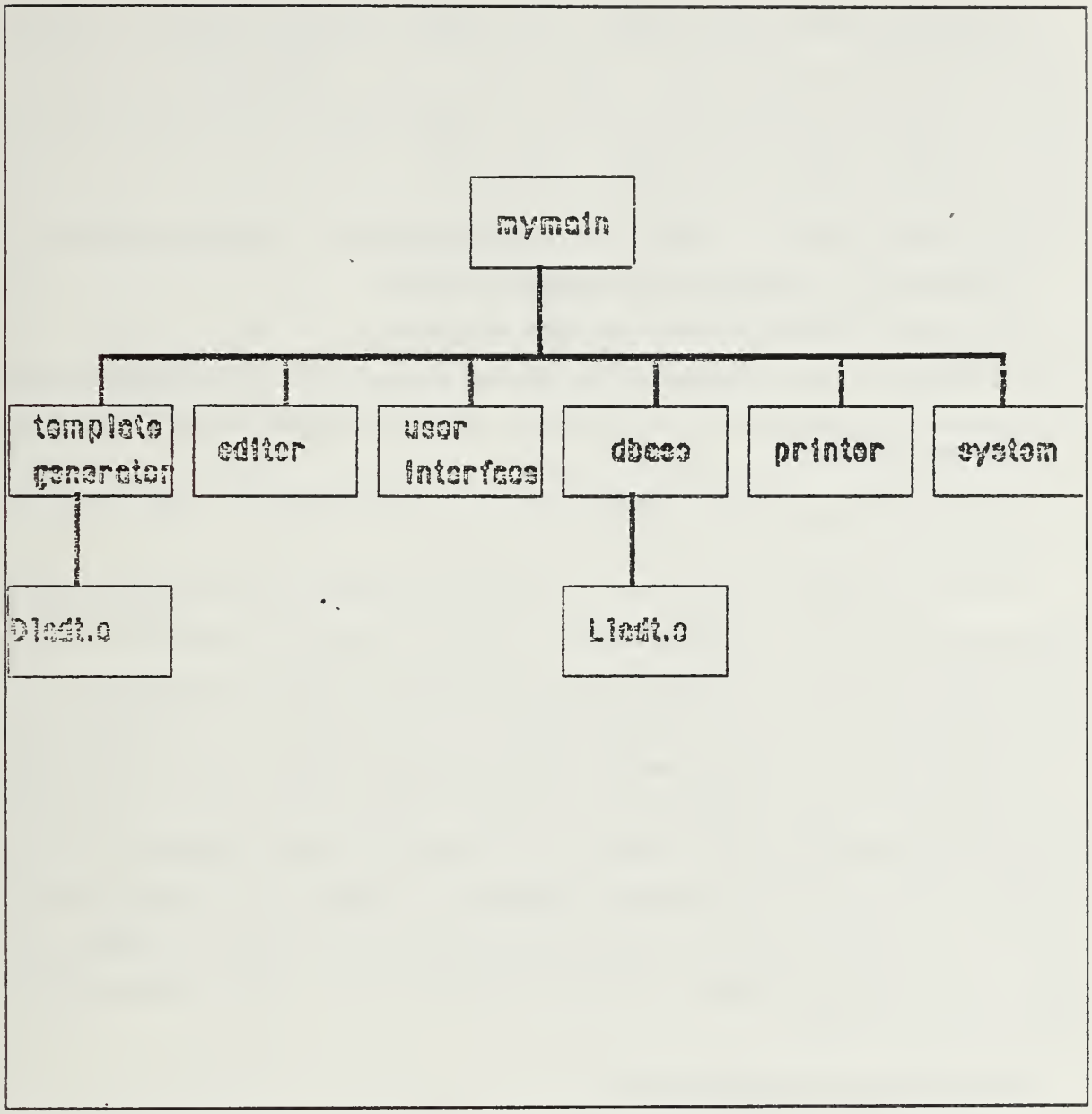

Figure 1. Program Components

2. EDITOR.C: Editor module.
3. USERINT.C: User interface module.
4. DBASE.C: Database module.
5. PRINTER.C: Printer module that contains printer related functions.
6. MYMAIN.C: Main module.

The main decision in the design of the modules is to provide functional interface among the modules. To hide the internal structure of each module, therefore to apply information hiding principle, modules are interfaced by means of predetermined function calls and overall design assumptions. Another design decision is the visibility of the program buffers among the modules.

In addition to program modules listed, there are four other modules. These are

1. SYSTEM.C: Contains system dependent functions
2. DLLADT.C: Doubly linked list abstract data type.
3. LLADT.C: Linked list abstract data type that is used by database program module.
4. EGA.C: Memory resident program that creates extra characters in the Turkish alphabet for EGA.
5. CGA.C: Memory resident program that creates extra characters in the Turkish alphabet for CGA.

From the portabilty point of view, I collected all hardware and operating system dependent functions in a separate module. When the program is ported, all functions in this module should be replaced with the appropriate ones. The second module, doubly linked list abstract data type, is the general purpose doubly linked list abstract data type. It is designed so that it is totally reusable. This is also an example of information hiding, abstraction and resuable program module. The portabilty and reusabilty issues will be presented later in this chapter. The last two modules are totally independent from the program. These modules are themselves independent programs which handles character generation on CGA and EGA. Since these programs contain TSR instructions, they have to be independent programs. These are compiled and run outside of integrated development environment.

C. PROGRAMS IN THIS PROJECT

In this section, program modules are explained separately. Each program module will be described functionally and together with its own structure. The complete source program and make file is presented in Appendix C.

1. TEMPLATE.C:

This module contains all function definitions that are related to template generation. It provides functional interface to other modules on the data structure chosen. Data structure to hold information is doubly linked list and all operations on data structure and needed by other modules are defined as a function. Therefore other modules are not dependent on the data structure used within this module. Since the

template definition for each document form is a set of fields, each node of linked list holds the information about one field on the form. This module performs all operations on data structure by using DLLADT.C module. This module is a general purpose doubly linked list abstract data type. Module TEMPLATE.C is not required to know the internal data structure of this module. It only sends a pointer to the data to be inserted into linked list. The implementation details for DLLADT.C will be explained later. Figure 2 shows the general data structure for the template generator module.

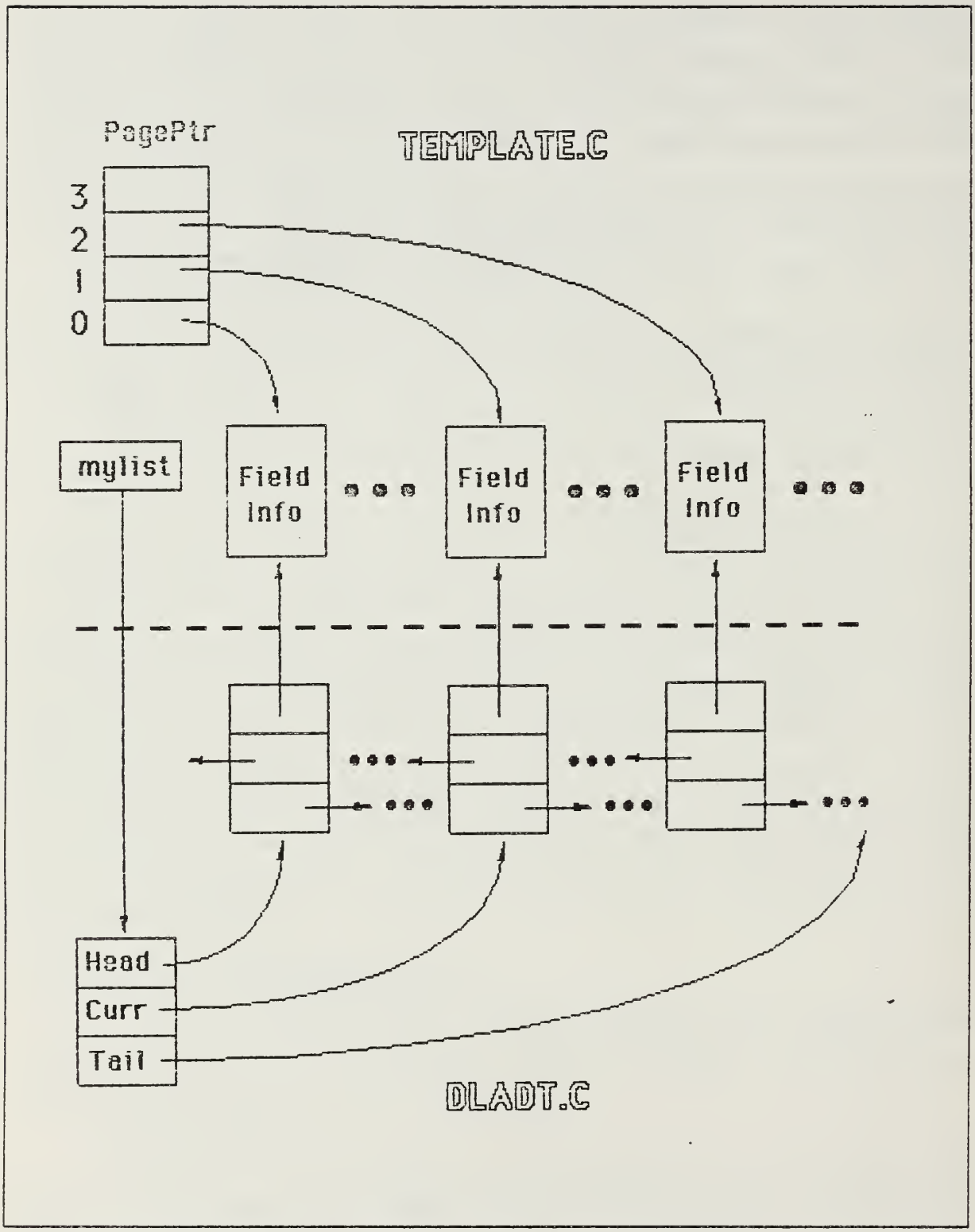

Figure 2. Data Structure for Template.C Program Module

Since creation of the extra characters in the Turkish alphabet requires graphics mode for color graphics adapter and text mode for enhanced graphics adapter, my design decision is to use only one video page for both video modes. Throughout the program the IBM PC default video page which is zero has been used. Instead of using scroll up and down functions on the screen, I used an IBM mainframe XEDIT-like editor with defined keys. When new page is requested by user, screen is cleared and a new page is written. This page swapping operation is handled by template generator routine. Since all template definitions for each document format are defined with respect to coordinates on the page, functions that responds to the requests to these coordinates from other module hide this fact and returns the coordinate values as if they are defined with respect to screen.

When requested by main module, this module converts all information in the data structure to ASCII text file format. text file format and holds them in a global buffer. This buffer is also visible to other related modules.

Which template to be loaded is determined by the user interface module and requested from this module by means of the order number of template in file. As will be explained in user manual, each template definition must start with a header line whose first character is '#'. When the user requests to fill out any document, all document definitions are displayed by reading the template file and seeking every line beginning with '#' character. Then the user is asked to enter the number of the document form and template definition for the requested form is read into the data structure with respect to this number. According to the number of the template definition entered, it displays the document form and then answers requests from other modules by providing a functional interface to the data structure.

2. EDITOR.C:

This module is a small editor. It consists of two functions. Function *getreply()* performs editing operations on each field of the form. This function gets the address of the message and reply field and the coordinates for those fields. It allows the user to edit each field by using defined keys. In case it is asked to edit an already edited field, first it copies everything from the address sent into its own buffer, clears everything in original address then performs editing operations on this buffer. Before exiting the current field, it copies everything into original buffer whose address is sent as a parameter to the function. It does not allow the user to overwrite to the uneditable part of the fields on the screen. When the user is trying to overwrite to undesirable field or to use undefined

keys, it warns the user by beeping. Defined keys and the explanation on how to use the editor will be presented in the user manual in Appendix B.

This function also performs one more task. It interprets the keys assigned for extra characters in the Turkish alphabet. The keyboard interpreter routine is embedded in this function. It interprets the defined keys for editor and combination of ALT keys assigned for extra characters.

The second function that takes place within this function is *edit_page()*. It performs operations for editing the entire page. It determines the next step according to a return code from function *getreply()*. This way it calls appropriate functions from template generator module. Return codes from function *getreply()* are the keys which cause an end to editing each field. The user is allowed to jump back and forth among the fields via up and down arrow keys or he may request a change to the video page. Function *edit_page()* determines where to go according to these return codes. It is invoked by the main module. It gets everything it needs by means of appropriate function calls from the template generator module.

3. **DBASE.C:**

This module performs operations in order to store documents requested by user. In order to access documents in database, an index sequential access method has been used.

The main assumption for this module is that the buffer that holds document in text form is visible to this module. In the implementation, each text to be stored is treated as a big string. It assumes that it terminates with null character which is the indication of the end of the string in C language. To store or retrieve any document from database, a global buffer is used. The files related to this module are shown below.

1. DBASE.FIL: Data base file. It is used to store documents.
2. INDEX.FIL: File that is used to keep keywords for database access.
3. TEMPLATE.FIL: File in which template definitions take place.

The key to database is a unique document number. When any document is saved, a keyword is entered by the user. The user interface module prompts the user to enter the keyword, gets the entry and stores it in a global buffer assigned for keyword. This buffer is known by the database module. The database module gets the keyword from the buffer and inserts it into a linked list which is data structure for holding keywords and index for each record. An index value for each record is the position in the linked list. When a document is requested, keyword is searched in the linked list, if

it is found according to position in the linked list, index sequential access is applied to the database file and the requested document is copied into a global buffer. The data structure is shown in Figure 3.

When the program is first run, keywords and implicit index values are read into the the data structure. During the execution of the program, all additions and deletions are performed on the linked list. When the program terminates, the last position of the linked list is written back into the index file in the same order in the linked list.

The appearance of the database file is a sequence of documents with a decimal number which indicates the size of each document and the document following it. The recently stored document is appended to the end of database file and the keyword associated with it is inserted at the end of the linked list.

4. USERINT.C:

All the user interface part of this program is performed by this module. It contains the function definitions for operations that require input or output. By using the field editing function, it allows the user to edit input and at the same time it permits the extra characters in the Turkish alphabet to be entered. All input operations that are limited to a certain size of characters are indicated by color and the input is checked by the field editor function. The simple error handling routine also takes place within this module. This is a general purpose error handling routine. When it is invoked by modules, this routine warns the user according to an error code determined by the calling module.

5. SYSTEM.C:

This module contains all hardware and operating systen dependent function definitions. From the portability point of view, all system dependent function calls take place within this module. BIOS video functions have been chosen so that they will operate both in video mode and text mode. All function and related service numbers for BIOS routines are defined in header files. Keyboard scan codes are defined within KEYDEF.H, BIOS functions and services are defined in BIOSLIB.H file. These two files are included in SYSTEM.H file.

6. PRINTER.C:

This module contains all function definitions for printer related functions. When printing any document is requested, it filters the text, seeks the ASCII codes assigned for extra characters in the Turkish alphabet. If any special character is trapped, it calls the appropriate function that prints the font associated with that characters.

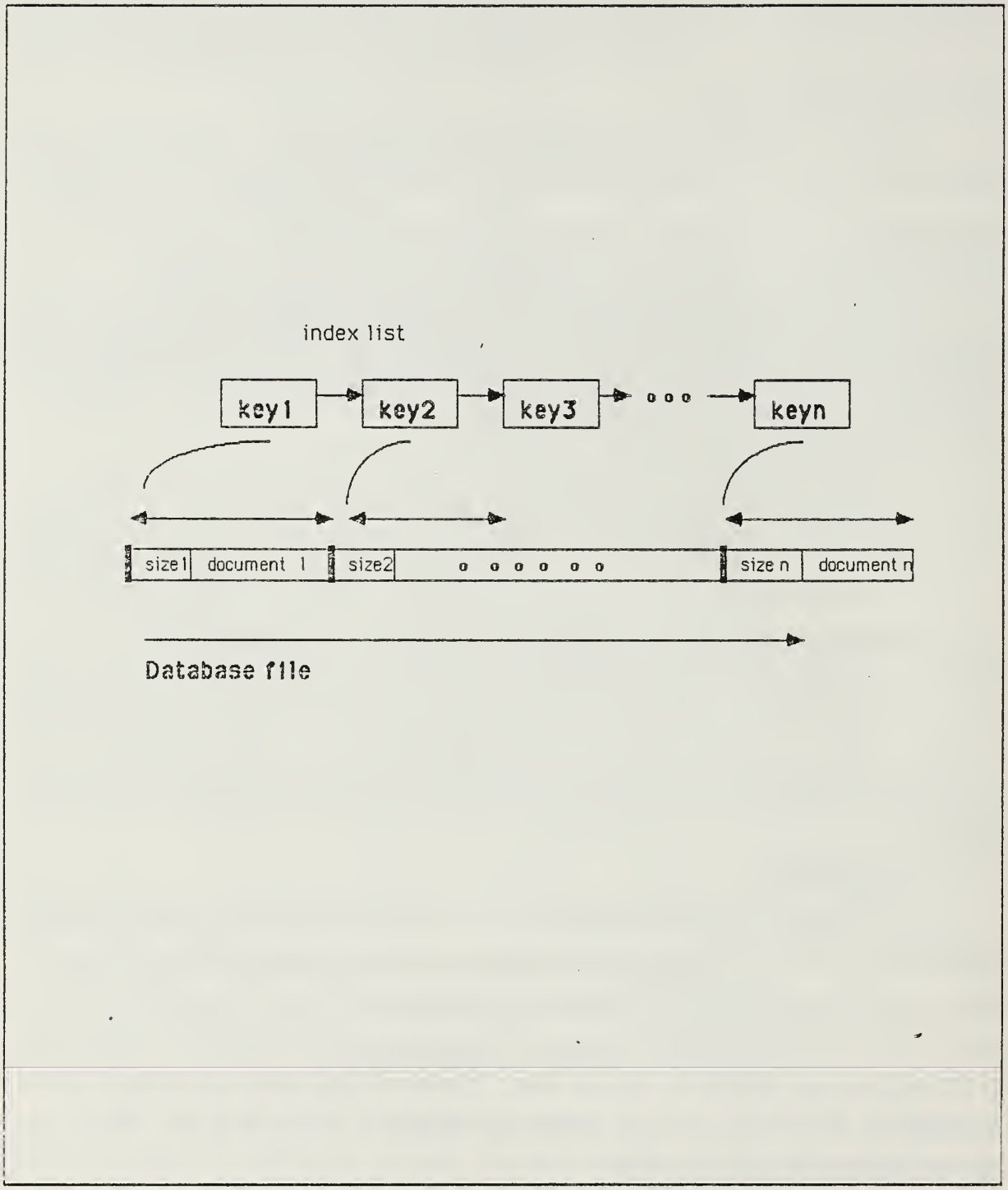

Figure 3. Data Structure for Dbase.C Program Module

There are two functions that are known by other modules. These are *print_page()* and *print_file()*. The design assumption for these module is that when any

document is requested to be printed, this module knows that it is stored in a global buffer. As in the database module, each document in the buffer is treated as a big string. So in order to print a page which is either already edited or retreived from the database to browse , function *print_page()* gets the characters from buffer, prints them out by filtering the ASCII codes for special characters.

Function *print_file()* is designed in order to be able to print the file which is used to keep record of incoming documents. This function performs the same operations by filtering and printing out special characters. The only difference is that this function gets the characters from file until the end of file rather than encountering the null character.

The user is allowed to print out any document up to eight copies. The number of copies requested is entered interactively by user. By considering that the user may want to fill out any document,wanting to store rather than printing it out, zero will imply no printer output. Printing a file is limited to only one copy. Design of the special characters will be explained in Chapter IV.

7. MYMAIN.C:

This module constitutes the top of the program structure. It is the mirror of how the entire program is structured. In top-down design approach, this is the first step I took.

This module contains only function calls rather than functions themselves. It knows the tasks performed by each module and interfaces modules according to logical order that is neccessary to perform the task.

8. DLADT.C:

This module is a general purpose double linked list implementation. Implementation details are hidden from application side. The user of the module has to provide some functions neccessary to apply abstract data type. The application side does not have to know the internal structure of the module. The functions that will be provided by application side are the comparison functions and appropriate requests from module.

The main idea in the design of this general purpose abstract data type is to have totally reusable module. It is implemented by taking advantage of the C language's features. It is written by using C language's generic pointer feature. The detailed explanation on the implementation of the module has been given along the source program as the comment. The data structure for double linked list is shown in Figure 4.

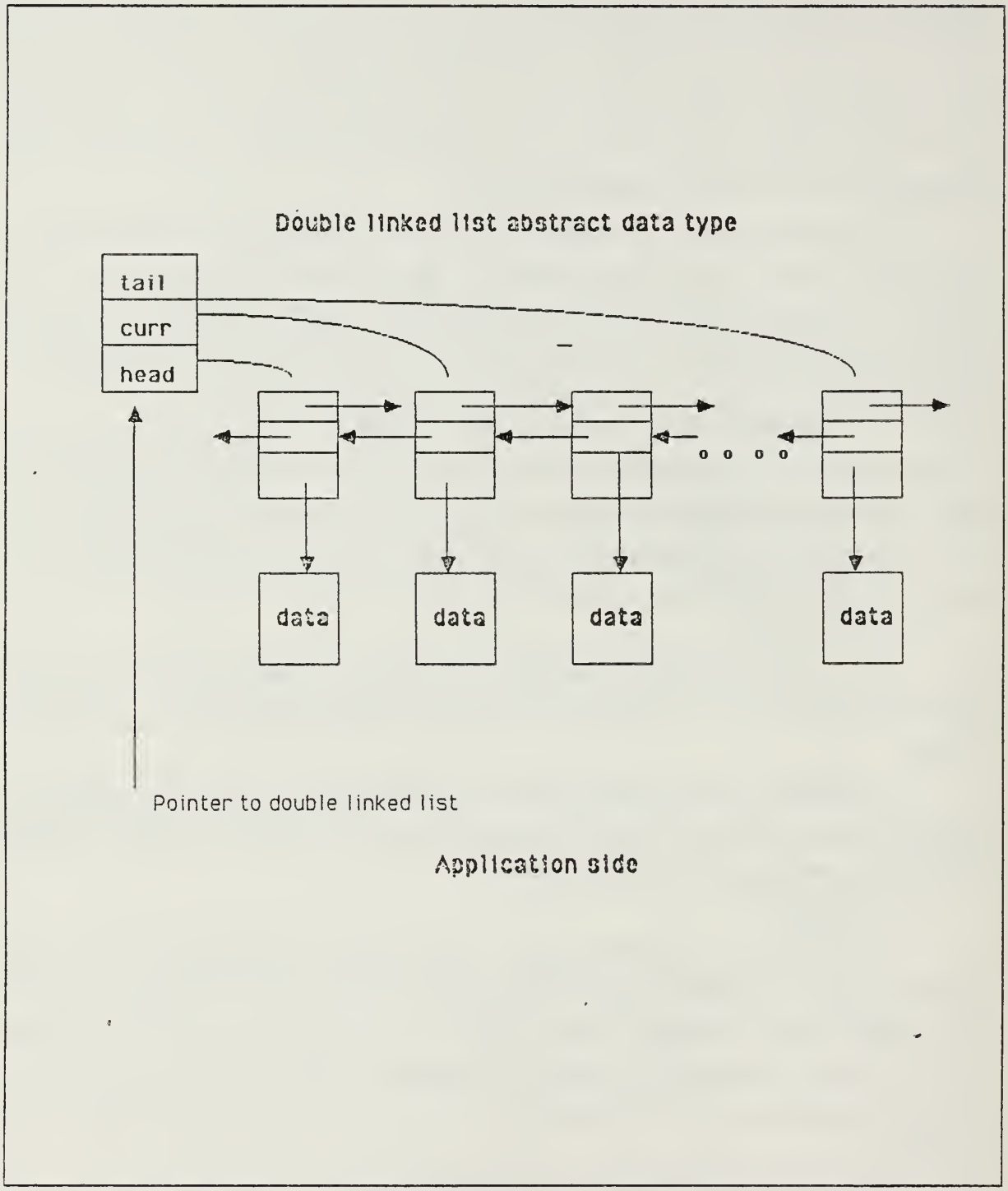

Figure 4. Data Structure for Double Linked List Abstract Data Type

9. LLADT.C:

This module is a general purpose linked list abstract data type implementation. It has been implemented by using C language's generic pointer feature. It is a totally reusable module. In this project, it is used by Dbase.C program module to hold keywords in order to implement index sequential access method for database access. The data structure for this module is shown in Figure 5.

10. EGACHR.C:

This program is separate from the program modules defined above. It is designed to create the extra characters in Turkish alphabet on the Enhanced Graphics Adapter. It is a memory resident program and it is run seperately from the main program. The detailed explanation on the implementation of extra characters will be presented in Chapter IV.

11. CGACHR.C:

This is also memory resident program and designed to create the extra characters on the Color Graphics Adapters. The detailed explanation on how to implement these extra characters will be presented in Chapter IV.

D. PORTABILITY AND REUSABILITY ISSUES

This particular application program is system dependent. In other words it is not portable to other systems. However, throughout this study, in order to make the program easy to port to other systems, in the design of the program these issues have been taken into consideration. PRPORT.H project portability header file is used for these purposes. Within this file, all portability issues considered throughout the project are defined. When this program is ported this file should be updated.

From portability point of view, designer must think about the following things. C compiler dependency and system dependency. This program has been written in TURBO C. All built-in language functions used within the program are defined using ANSI standards and provided by all other C language compilers. None of the functions is unique to TURBO C. However, some reserved words that are not provided by other compilers such as 'void' take place in the PRPORT.H file and are replaced with appropriate ones according to defined compiler in the same file.

The second thing considered for the portability is the system dependency of the program. This dependency may show up in two different stiuations. First the representation of the data types in the language differs from system to system. The second is the presence of the system dependent function calls within the program. To eliminate

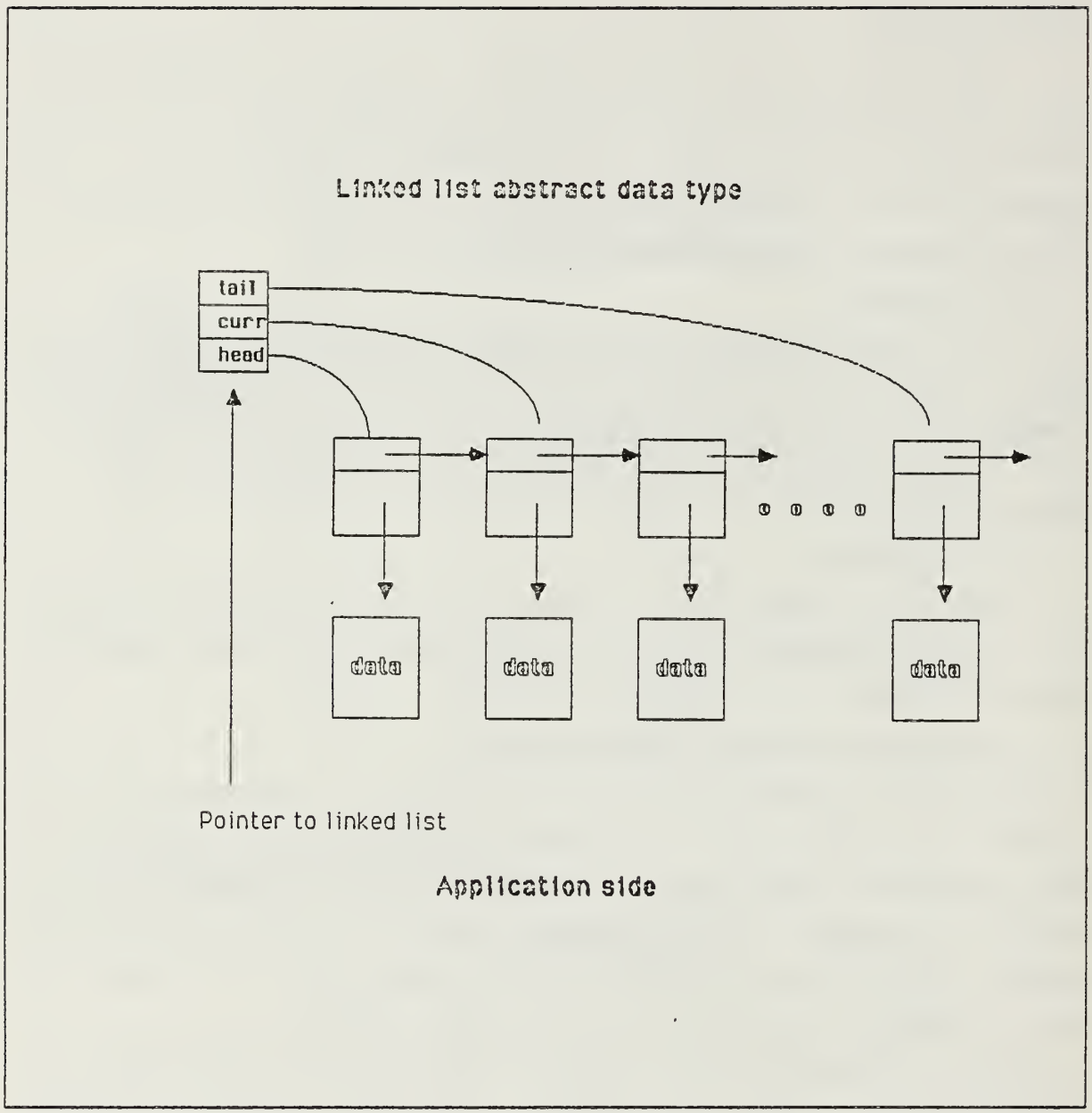

Figure 5. Data Structure for Linked List Abstract Data Type

the impact of the representation of data types, I defined my own data types within the PRPORT.H header file by taking advantage of C language's typedef feature. In the program, I used my own data type everywhere that is dependent upon the representation of data types on machine. And by forcing the compiler type casting on my own data type, I aimed to eliminate the impact of the representation of the data type on different machines. Before this program is ported to another system, user defined data types must

be redefined according to the new system. There is no need to make change within the program.

This program contains many BIOS routine calls. These calls are specific to IBM PC and compatibles. I collected all BIOS and OS dependent functions within a single program component namely SYSTEM.C. Before the program is ported, all functions that take place within the SYSTEM.C program component must be replaced with the appropriate ones so that function name and the order of the parameters will remain the same.

IV. IMPLEMENTATION OF SPECIAL CHARACTERS IN TURKISH APHABET

A. TO CREATE SPECIAL CHARACTERS ON SCREEN

Programmers writing for the monochrome display adapter(MDA) and the color graphics adapter (CGA) are stuck with the character sets provided in those board's ROMs. If you want a different character set, for instance as APL users do, you have to replace the ROM. But there are various commercially available adapters today. On the Enhanced Graphic Adapter things are different. Fonts are "soft", meaning that although the ROM character generator is used by default, it can be replaced by a character set of our choosing. In fact, EGA can support four different character sets.

In this section, we will examine how to design our own custom fonts. For different adapters creation of special characters will be discussed.

Only the monochrome adapter cannot display characters of programmer's own design. The color card allows 128 user defined characters. PC jr allows 256, EGA allows 1024, of which 512 may be on line at once. On the color graphic adapter, in text modes character sets in ROM are used by the system. There is no way to change this or replace by new fonts. But CGA allows user to define own custom character sets on graphics modes. In graphics mode, ROM contains data to draw first only 128 characters in the ASCII set (numbers 0-127). The second 128 characters can be redefined for our own purposes. System finds the table containing data drawing graphics characters via interrupt vector 1F [Ref. 5: p.1-91].

Characters on graphics card and PC jr's are designed within a box that is 8x8 pixels. Eight bytes hold the data for each character. Each byte holds the setting for a row of pixels, starting with the top row, and the high bit corresponds to the leftmost pixel of the row. When the bit equals 1, the pixel shows. To design a character, the bit patterns for eight bytes must be determined and placed in sequence in memory pointed by interrupt vector 1F. Figure 6 shows how extra characters in the Turkish alphabet are designed for CGA. To place user defined character table in memory, interrupt vector 1F must be redirected so that INT 1F will point to the user defined new character table. This can be achieved either by some built-in function calls provided by some high level language or writing assembly language routine. In case that this is done with assembly language, we also need to make our table memory resident. But if this is implemented

in any high level language (in this study the C language), the new table can be held in an array and INT vector can be set to point to the array that contains the user defined character set data.

In my approach, I implemented the character generator programs in the C language. These are memory resident programs and install our own characters. We need to define only 12 extra characters in the Turkish alphabet. Therefore, first I copied system defined graphics character table into array, then my own character table into an array. Before doing this we need to save the old interrupt vector value so that when program terminates we can have the system resume the original state. The next step is to change the interrupt vector so that it will point the array holding the character table data. But there is one significant point here. As stated above there is no way for us to use these characters in normal text mode. Therefore we need to change the video mode. In this particular example for IBM PC or compatibles, it should be set to video mode 6 which is black and white mode. In this program I utilized the built-in Turbo C language functions *setvect()*, *getvect()* to get old interrupt vector value and to change the interrupt vector to the user defined character table. Since the user defined character table will be held in an array and it will be in memory during the execution of program, I did not make this routine memory resident.

The Enhanced Graphics Adapter is much more complicated and much more versatile. When a text mode is initialized, one of the two character sets (8x8 or 8x14) is copied from EGA ROM onto bit map 2 of the video buffer. This part of the buffer is treated as if it were broken into blocks, and the standard character set is placed in block 0. Providing the EGA adequate memory, three more blocks of character data may be set up. The size of the block depends on the number of scan lines used in the character. Characters that are 8x8 need 8 times 256, or 2048 bytes. When more than one block of characters is enabled, bit 3 of the attribute byte determines which block will be used.

Enhanced Graphics Adapter gives the user the ability to define his own character set. New character set can be placed at whatever position user chose within any block. And even if it overwrites the standard character set, it can be replaced at any time from ROM data.

The most beautiful part of the things offered by EGA is that it gives the ability to replace the standard character set in text mode with one of chosen by the user. The EGA has BIOS support for the loading of an alternate character set through interrupt 10h, function 11h, subfunction 0 [Ref. 6: p 4-1). We can make a call to this function

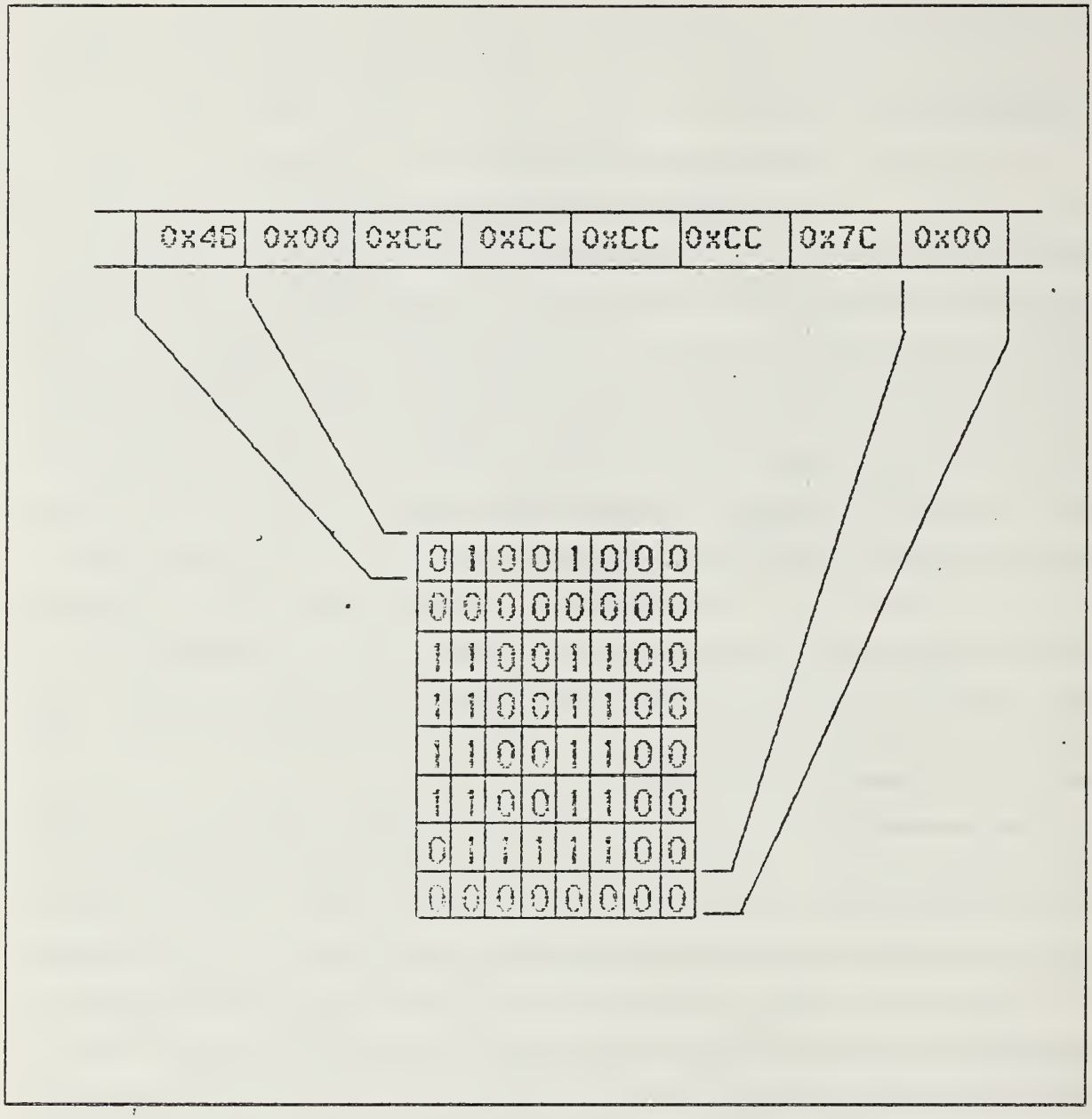

Figure 6. Memory and Screen Representation of Character for CGA

with ES:BP pointing to table containing our own font in a format that will be explained below, DX set to the ASCII ordinality of the first character of our character set, CX set to the number of the characters in user defined character set (maximum 256), BH set to the number of the bytes per character, and BL set to the block to load. These parameters provide the BIOS with sufficient information to load the new fonts defined by the user.

Figure 7 contains data and the way that extra characters in Turkish alphabet are plotted. This figures assumes an enhanced color display in 25-line mode, in which each character is 14 scan lines high and 8 pixels wide. For EGA character box is limited 8 pixels wide but can be defined up to 32 scan lines high.

B. PRINTING SPECIAL CHARACTERS ON PRINTER

In this section, I will explain how we can print out our own characters on the printer. First of all I would like to emphasize that by saying printers, I mean commercially available dot matrix printers. In this research I did not take bit map systems and, therefore, some printers called image writers into consideration.

There are hundreds of printers available today. Although most of the commercially available printers offer various features, for this specific application area, we will not be using most of the features offered. The need for a different character set is partially met by those features. Most of the dot matrix printers provide user some character set rather than standard character set. These characters can be utilized by sending appropriate printer commands. For this application, we could make use of some of these characters namely international character set, provided by printer manifacturers. But this would not meet our needs for extra characters in Turkish alphabet. Therefore we have to find a way to print out some user defined characters which are not available in the printers ROM. This goal can be achieved by exploiting one of the features provided by printers.

To print out characters which are not available in printers ROM, I took the following steps. First, dot matrix printers offer the ability to print graphics. This shows us anything can be printed as defined by user. Before starting to explain the way I print out my own characters, we need to take a closer look at where dots are printed and how we can control them. When any character is sent to printer, it prints that character using the dot pattern stored in its memory. In case that we want to print a pattern of dots that the printer does not have in its memory, we should control then the individual dots that are printed. Printer head usually consists of nine pins stacked one above the other. The print head therefore can print the columns of up to nine dots at a time. As plotted in creation of characters on screen section, if we plot the characters that will be printed we can print them column by column. Since print head will be printing column by column, we need to first send the byte that defines first column of character box. When we draw our own characters, each strike of pins on the print head will be represented by 1's. This technique is called *dot graphics*. In dot graphics, the line length and dot spacings are not fixed. We should tell the printer three things. First, which pins to print

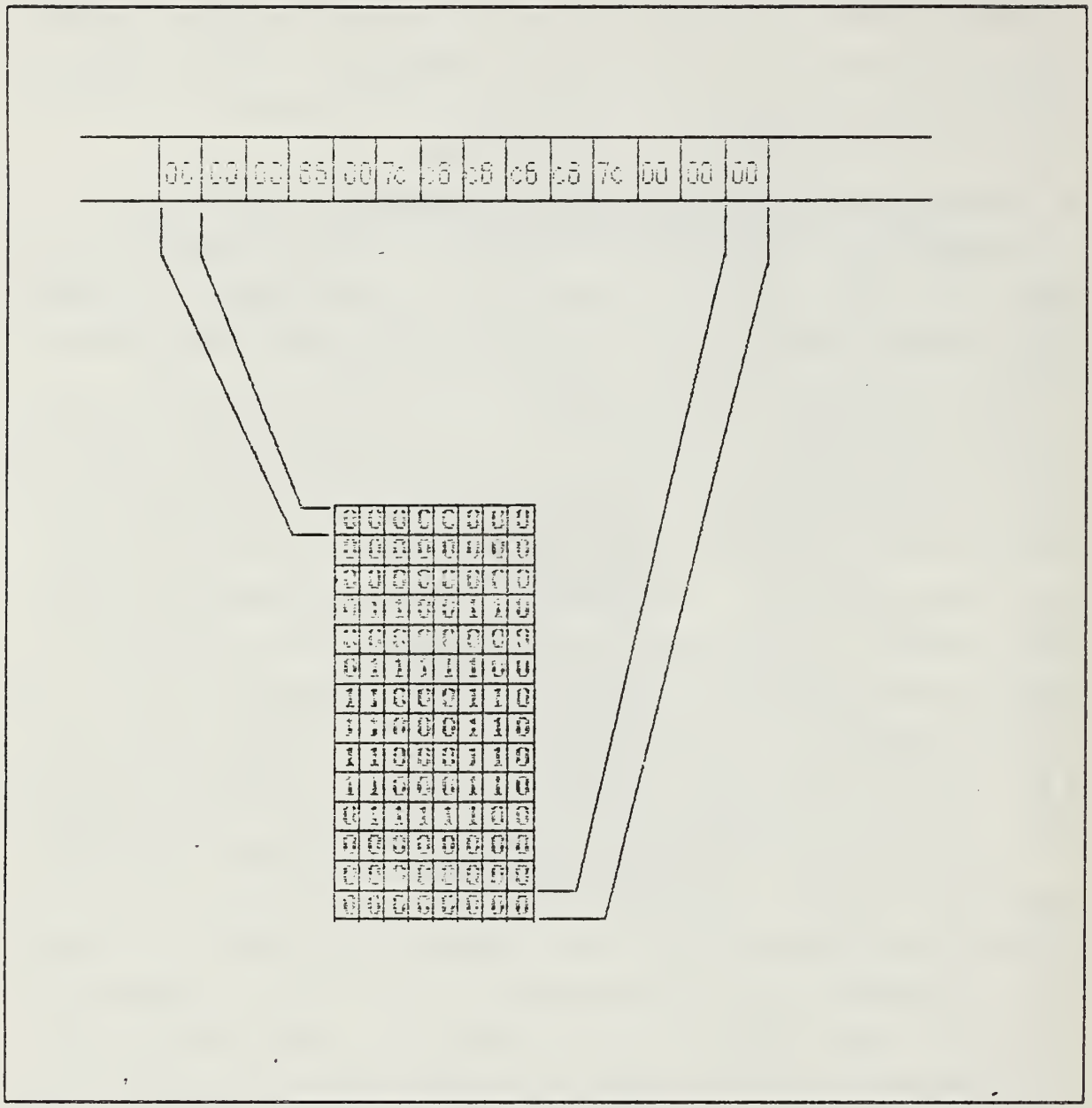

Figure 7. Memory and Screen Representation of Character for EGA

in each column, second, how closely to space the columns, and third how many columns there will be in the line. To tell all these things, appropriate printer commands, which are actually ESCAPE sequences, must be sent to the printer.

After having briefly explained how we can use printers to print out some characters that are not available in printers' memory, now I can explain the approach taken in this

program. Figure 8 shows how extra characters in Turkish alphabet are plotted for printer.

First step is to define characters on a 8x8 matrix. Second step is to calculate the values corresponding each column either in hex or decimal. There are two printer graphics mode: line and block graphics mode. I picked the line graphic mode. Since each character is 8x8 dot matrix (therefore it will be 8 dots wide), I defined line length as 8 in line graphics mode. ESC sequences to tell printer these should contain the following values, number of dots per inch, line length. By sending the following command to the printer, we can tell the printer that leading 8 bytes will be printed column by column, in other words they will not be interpreted as regular ASCII characters.

ESC 'K' n1 n2

If we interpret this, ESC 'K' means ' do not interpret the leading bytes as ASCII characters, and the line length is n1 + (256*n2) [Ref. 7: p.6-47]. This is a general purpose command. Since we are going to use it to print our own characters the line length should be eight. If the hexadecimal or decimal values which corresponds the column values of new characters are sent to printer after following escape sequence, the character that is defined by user will be printed. After line length bytes printer resumes its previous mode. Since characters are printed in fixed dot density, even though letter quality is set to anything, they will be printed in fixed dot density defined by user. For this application program I picked the dot density as sixty dots per inch and regular characters will be printed standard mode. To make sure that printer is operated in standard mode, program sends the ESC sequence that sets the printer to standard mode. Telling printer which character will be printed as special character is under program responsibility and the printer module handles everything.

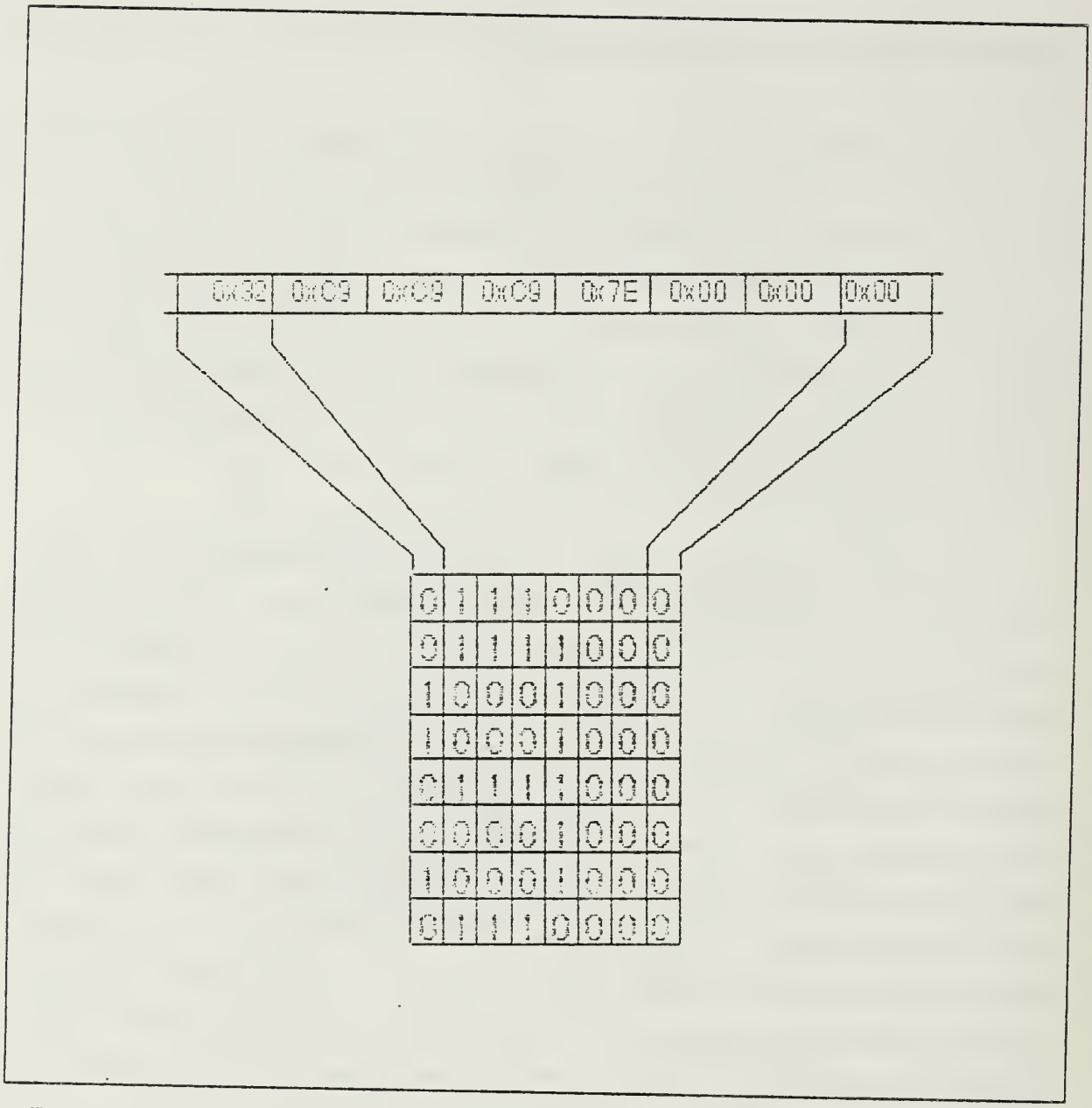

Figure 8. Character Representation for Printer

V. USING THE DOCUMENT GENERATOR

In Chapter III, the design and implementation details of this program have been discussed. In this chapter, what we have designed will be presented. This chapter briefly explains how to use the document generator software from user's point of view. The detailed explanation is presented in Appendix B as *User Manual.*

The entire project tends to use microcomputers on board for tasks performed by ship bureaus. It allows the user to fill out document forms and to store them in a database, including the ability of using extra characters in the Turkish alphabet. All operations performed by this software appear in the main menu. The user can select one of these operations. These operations are:

1. Preparing any document.
2. Browsing any document from database.
3. Entering incoming document log.
4. Printing incoming document log file.
5. Quit

This program tends to be user friendly. User is prompted for each step to be taken. All operations performed by this program take place within main menu. Main menu is the starting and ending point of each operation displayed in main menu. The program will be executed until the user enters the *Quit* option.

In order to prepare any document, *option one* should be selected. User will be prompted to enter appropriate form number after program displayed all templates that exist in TEMPLATE.FIL. If there is more than one screen full template definition, program automatically provides user to view one screen full at a time. The form that corresponds to the number entered by user will be displayed and the editing session will start. User will be allowed to edit each field displayed on screen and to go back and forth among the field. User will not be allowed to overwrite non fillable fields during the editing session. He will be warned by sound in any attempt at overwriting. After completion of editing session, user will be prompted if he wants to print it out. If he wants to print out, program will print out the currently edited document with as many copies as requested. Zero implies no printer output. Next step is to get user request if he wants to store it in database. If it is requested to store it in database, after getting

the document id for document (which is key to database access), it will be stored in database file and program will go back to main menu for next operations.

Another operation is to browse any document previously edited and stored in database. When this option is requested, user will be prompted to enter document id associated with request. Program will access database file according to keyword entered by user and retrieve the document into program buffer then display it one screen full at a time. User will be allowed to print the document browsed and he will be promted for print option as in option one. Zero will imply no printer output and program will go back to main menu for next operations.

There is no feature for retyping all incoming documents. Program will only provide a mechanism to keep records of all incoming documents. In option three, user can enter all information about incoming document. He is allowed to edit. The user input then will be appended to DOCLOG.FIL.

The fifth option provides user to print all incoming document information entered by option three. User will not be promtped for the number of copies. Program will print only one copy of the DOCLOG.FIL.

Program terminates after option five is selected. All document id's entered during the execution of program will be updated before program terminates.

The detailed explanation on operations performed by this program will be presented in Appendix B, User Manual.

VI. CONCLUSIONS AND RECOMMENDATIONS

The objective of this study is to design and implement a software for an automatic document generator supporting extra characters in the Turkish alphabet. In this study, I mainly focused on IBM personal computers to create our own characters. Being able to use the Turkish alphabet on computers was only one part of the project. There were several goals in the design and implementation of the program. The first part is to be able to use extra characters in the Turkish alphabet. This part includes two different hardware components. The system on which program is running and the printers to be used to print characters. This part of the problem is totally hardware dependent. Throughout this study, I focused on IBM personal computers to create extra characters in the Turkish alphabet on the screen. For printers I took the dot matrix printers. Since the dot matrix printers vary among themselves according to manifacturers, I considered three main types of dot matrix printers. These three types of printers are commercially available and widely used, respectively EPSON, IBM proprinter, OKIDATA dot matrix printers.

To expand this project, the hardware dependency of creating extra characters on screen can be reduced by expanding this study to other systems that are available today.

The second part of the problem is to meet user needs to generate and process documents. This program is designed to generate documents with only a single page length. This program can be developed so that there will be no limitation for the length of the document format. To process documents, to store them in database and to meet user needs, in database implementation of this project, the index sequential access method has been used to retrieve any document from database. To expand this project, database routine should be implemented by using a B-tree to achieve better disk access to the database.

APPENDIX A. TURKISH APHABET

a	b	c	ç	d	e	f	g	ğ	h	ı	i	j	k	l
A	B	C	Ç	D	E	F	G	Ğ	H	I	İ	J	K	L

m	n	o	ö	p	r	s	ş	t	u	ü	v	y	z
M	N	O	Ö	P	R	S	Ş	T	U	Ü	V	Y	Z

APPENDIX B. USER MANUAL

A. INTRODUCTION

This program attempts to provide all operations performed by ship administrative offices to generate and process documents. It is designed so that it will be user friendly and interactive program. Basic features provided by the program are displayed in the main menu. The main menu is the starting and ending point of each operation until *Quit option* is entered by user. User will be prompted during the execution of each option on the main menu according to logical sequence of the operation performed in real time.

B. REQUIREMENTS

This program is designed for IBM personal computers and DOS environment. The following conditions must be checked before the execution of program:

1. All program files must be in the same directory. Program files and their descriptions are as follows.
   a. YAZIBURO.EXE: Main executable program file. This is the file to be run.
   b. TEMPLATE.FIL: File in which all template and form definitions take place. This file must exist in order to be able to fill out some documents. The absence of this file implies that there is no template definition for program to display.
   c. INDEX.FIL: This file is used to keep keywords for database access. It does not have to exist when program is first run. However, it must exist along with DBASE.FIL if there is any document edited and stored in database. The presence of either INDEX.FIL or DBASE.FIL will imply that there was at least one document stored in database file and one of these files is missing.
   d. DBASE.FIL: This file is main database file and it is used to store all documents requested to be stored.
   e. EGACHR.EXE: Executable file that creates extra characters in the Turkish alphabet. It can be used independently in order to prepare new templates by using any word processor. It will be installed by program when it is used for document generator purposes.
   f. CGACHR.EXE: Executable program that creates extra characters in the Turkish alphabet. Its usage is the same with EGACHR.EXE.
2. Maintenance of the program files is under user responsibilty.
3. System should be equipped with either CGA or EGA in order to be able to display extra characters in the Turkish alphabet. Program will automatically terminate if these conditions are not met by the system.

4. All template definitions must be entered correctly. Detailed explanation on how to prepare template will be presented later.
5. Printer attached to the system should be a dot matrix printer.

C. GETTING STARTED

The complete list of the program files has been presented in section B. The executable program name (YAZIBURO) must be entered in DOS command line in order to run this program. Main menu will be displayed as soon as program is run. Main menu contains all operations performed by this program. User can choose the operation from main by simply entering the number associated with the request after the program prompt. The logical sequence of the events to perform any operation on main menu are as follows:

1. PREPARING ANY DOCUMENT
   a. Choose the document form and enter the choice after program prompt.
   b. The form of document will be displayed.
   c. Fill out document.
   d. Print it out. If the printer output is not requested, zero will imply no printer output.
   e. Do you want to save it? If the answer is 'yes', enter the keyword for the document to be saved after program prompt.
   f. Go back to main menu.
2. BROWSING ANY DOCUMENT FROM DATABASE
   a. Enter the keyword associated with the document.
   b. If attempt is succesful, browse the document.
   c. User may want to obtain print out. He will be prompted to enter the number of copies to be printed. Zero will imply no printer output as in option one.
   d. Go back to main menu.
3. ENTERING INCOMING DOCUMENT LOG.
   a. Prompts the user by displaying the fields in incoming document log.
   b. Enter the information about the incoming document.
   c. After hitting the RETURN key, go back to main menu.
4. PRINTING INCOMING DOCUMENT LOG
   a. When option four is selected, program will automatically generate one copy of printer output of incoming document file.
   b. Go back to main menu.

5. EXIT TO DOS

   a. This option is used to terminate the execution of the program.

User will be prompted according to logical sequence of the events presented above during the execution of the program. Each operation will be explained in detail below.

1. ENTERING EXTRA CHARACTERS IN THE TURKISH ALPHABET

There is a one to one relation between extra characters in the Turkish alphabet and the ones in English. In the design of this program, by taking advantage of this similarity, extra characters in the Turkish alphabet will be entered as ALT key and the similar letter in English combination. CAPS LOCK and the SHIFT keys on the regular keyboard layout will function same way.

1. ALT-c : will display the letter ç.
2. ALT-C : will display the letter Ç.
3. ALT-g : will display the letter ğ.
4. ALT-G : will display the letter Ğ.
5. ALT-i : will display the letter ı.
6. ALT-I : will display the letter İ.
7. ALT-o : will display the letter ö.
8. ALT-O : will display the letter Ö.
9. ALT-s : will display the letter ş.
10. ALT-S : will display the letter Ş.
11. ALT-u : will display the letter ü.
12. ALT-U : will display the letter Ü.

2. USING DOCUMENT GENERATOR

a. PREPARING DOCUMENT

Option 1 provides user to fill out any document whose definition takes place in TEMPLATE.FIL. All templates that take place in TEMPLATE.FIL will be displayed. User will be asked to enter the number of his choice. Program will automatically provide a mechanism for user to view all templates available. When there are more than one screen full template names, program will display one screen full template names at a time. The form of the document requested by user will be displayed. Program will automatically color the spaces to be filled out by user. Cursor will be on the first field to filled out. Program will prevent user from overwriting on non-fillable fields of document form. Cursor will automatically go to the next editable field after the user hits the

RETURN key. User is allowed to go back and forth among the fields by using defined keys in order to correct typos.

Program defined keys and the editing features provided by program are as follows.

1. ESC key: Will cancel all input for the current field.
2. HOME key: Will take the cursor to the beginning of the current field
3. CTRL-END: When the combination of CTRL-END keys is entered, it will terminate the editing session.
4. DEL key: allows deleting character at the cursor position.
5. LEFT ARROW: Moves the cursor to the left.
6. RIGHT ARROW: Moves the cursor to the right.
7. UP ARROW: When UP ARROW key is hit, editing session of the current field will terminate and this key will allow user to go one editable field back. If there is no editable field to go, user will be warned by sound and cursor will stay at the same field.
8. DOWN ARROW: will terminate the editing session for current field, and will move the cursor to next editable field. If there is no field to go, user will be warned by sound and cursor will stay at the same field.
9. SPACE BAR: Will either move cursor or make room for characters to be inserted.
10. PAGE UP: Will display the previous video page. If the current page is the first page, user will be warned by sound and current page will remain active.
11. PAGE DOWN: Will display the next video page. If the current video page is the last page, user will be warned by sound and current video page will remain active.

User can edit each field by using program defined keys listed above. Editor is always in insert mode. After having filled out all necessary spaces on the form of document, user can press CTRL-END to terminate editing session. User will be prompted for printer request as next step. Program prompt for printer request :

Howmany copy do you want = = > _
Zero implies no printer output !

User will enter the number of copies he wants to print out after the program prompt shown above. Program will execute the user request and then prompt user if he wants to store the currently edited document in database. If user wants to store the document in database, he should enter the keyword after program prompt. User is allowed to edit all entries by using program defined keys presented in section 2. Program will go to

main menu for next operation after saving the document in database. If user requests more than one copy of printer output,

program will print out up to *maximum eight copies* by automatically providing *form feed.*

b. BROWSING ANY DOCUMENT FROM DATABASE

Program allows user to view any document previously edited and stored in database. This can be done by selecting *Option 2* from main menu. Appropriate keyword must be entered in order to browse any document from database. User will be asked to enter keyword for the document to be viewed. User is allowed to edit his entry as in section two. All keys defined can be used. Program will get the document id and retrieve the document and then display it one screen full at a time. User will be prompted to see next page.

This operation will allow user to view rather than edit or change the contents of the document. However user can print it out as many copies as he wants. Program will prompt the user and print the document according to user request. Zero will imply no printer output as in *Option 1.* Program will go to main menu after performing request by user.

c. ENTERING INCOMING DOCUMENT LOG

Program provides a mechanism to keep record of all incoming documents. Since there is no use to retype all incoming documents, it basically allows user to enter all information about incoming documents. This can be done by *Option 3.* When this option is selected by user, program will display the following:

ENTER ALL INFORMATION ABOUT THE INCOMING DOCUMENT

DOCUMENT ID *FROM* *DATE*

_____ _____ _____

-

User will fill out the spaces as indicated above. All program defined keys can be used to enter the information about the incoming document. It is under user responsibility to format the entry. Program will save the entry as it was entered.

d. PRINTING INCOMING DOCUMENT LOG

Option 4 provides user to print out all informations about incoming documents. Program will print one copy of DOCLOG.FIL which holds all informations about incoming documents entered by *Option 3.* User will not be prompted for the

number of the copies. Program will print only single copy of this file and then go back to main menu.

e. EXIT TO DOS

Option 5 terminates the program and takes user back to DOS command line. Program updates the program files before it terminates.

3. HOW TO PREPARE NEW TEMPLATE

In this section, how to prepare new template will be explained. This program will run for the template definitions already existing in Template.Fil. User can easily add new templates to the system according to what they need. User should not forget that program always assumes template definitions have been entered correctly. All templates must be defined to the system as described below.

All template definitions take place in *TEMPLATE.FIL*. New templates can be added to already existing ones by using any word processor. Before starting to explain rules on defining new template to the system, it is better to give a brief explanation on terminology used here.

Each document form consists of fields. A field is the smallest unit on this form. A field consists of a field message and a field reply. *Field message is* the string appeared on the blank form of the document. Each entry that user will fill out constitutes a *field reply*. A field can be either editable or noneditable.

The noneditable fields are the ones the user is not allowed to edit. The location of each field is given by the coordinates with respect to page format. Each field has to have row, column and width values associated with itself.

Figure 9 shows the components that are necessary to define any field.

In order to prepare a template the following steps should be taken. Each form of document should be broken into fields. The components of each field must be determined. All components of a field must be appended to Template.fil according to the following rules.

1. Each template definition must have unique name and its name must be defined prior to field definitions. All template names must start with '#' character and in the first column. Program will display all template names by checking if their first characters are '#' or not.
2. Each field must be entered in a single line.
3. User is allowed to put comments anywhere in the file as long as it starts with '*' and it starts in the first column. All lines whose first character are '*' will be skipped by program.

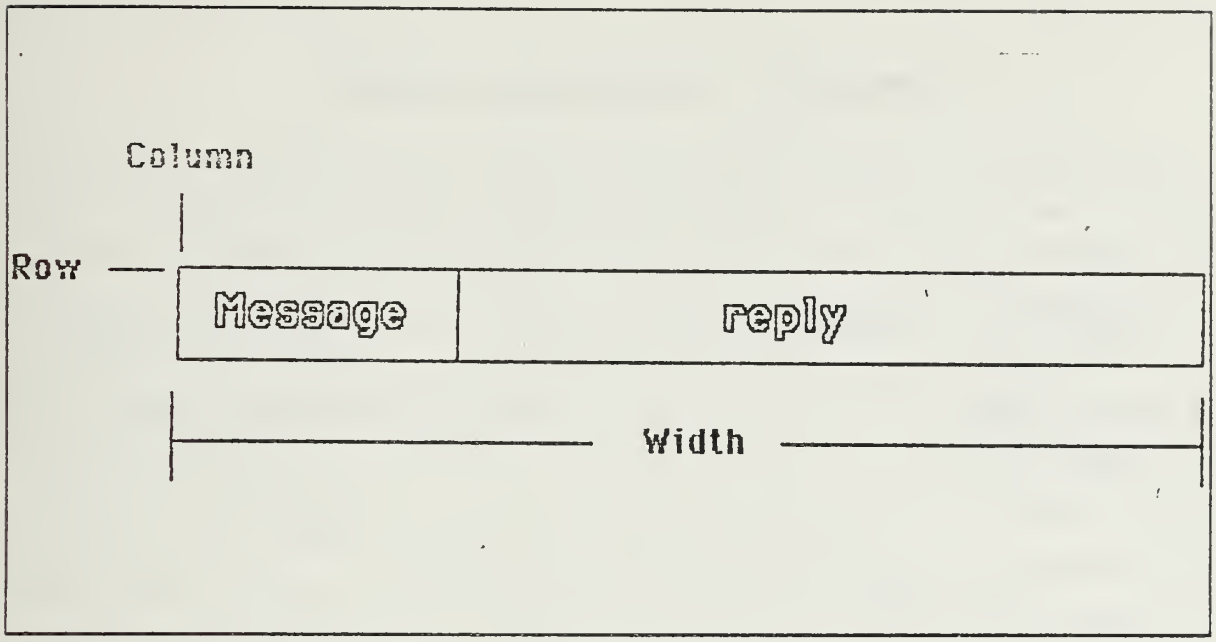

Figure 9. Field Components

4. The order of the field component values is very important. The order must be as follows:

   Field row column width editable @ field message @

5. Field messages must begin and end with the special character '@'.
6. Field width must be determined so that it will be large enough to hold both the field message and the field reply that will be entered by user when it is editable.
7. Field status can be either one or zero. One should be entered for the ones that user is allowed to edit. Otherwise it will be zero.
8. The order of each field is not important. It can be entered in any order. However the order of the field components must be entered as in the example.

```
* This is a comment line.
* row   column   width   editable   field message
  5      50       20       1        @Date: @
```

Example field definition defined according to rules above indicates a field starting at row 5, column 50, total width 20 characters and field message is Date: This shows that field message will be displayed on specified coordinates and user will fill out *width - message length= 14* characters length space. Example template definition and program output document are presented in Appendix D.

APPENDIX C. PROGRAM LISTINGS

Source listings of program modules are given in the following order. Relation among the program files can be found in *MAKEFILE*. The order of program listings is according to program structure. All program files are listed after program modules.

- MAKEFILE UTILITY
- MYMAIN.C
- TEMPLATE.H
- TEMPLATE.C
- EDITOR.H
- EDITOR.C
- DBASE.C
- PRINTER.C
- SYSTEM.H
- SYSTEM.C
- DLADT.C
- LLADT.C
- EGACHR.C
- PRPORT.H
- KEYDEF.H
- MYASCII.NUM
- EPSON.DAT
- EXTRA.FNT

```
#
#
#                   MAKEFILE  FOR PROJECT
#
#
#
#
#


# The order to search for rules and files is specified by .SUFFIXES
FFIXES : .exe .obj .c

# program files and their dependencies.

mymain.obj: prport.h

template.obj: template.h prport.h keydef.h bioslib.h

editor.obj: editor.h prport.h keydef.h bioslib.h myascii.num

userint.obj: prport.h

dbase.obj: prport.h

printer.obj: prport.h myascii.num extra.fnt epson.dat

system.obj: system.h prport.h keydef.h bioslib.h

dladt.obj: prport.h

lladt.obj: prport.h


#  Files
FILES=  system template editor mymain userint printer dbase dladt lladt

#  Object files
OBJS= system.obj template.obj editor.obj mymain.obj
userint.obj  printer.obj dbase.obj dladt.obj lladt.obj



# Libraries
LIBS= emu math$(MDL) c$(MDL)


# Model definition is SMALL for Turbo-C
MDL =  s

# make is :
# MAKE = ndmake

# plink is :
# PLINK = tlink
```

```
all: Yaziburo.exe

Yaziburo.exe: $(OBJS)
        tlink c0$(MDL) $(FILES),$*,$*, $(LIBS)

c.obj :
        tcc -c -m$(MDL) $*

clean:
        del mymain.obj
        del system.obj
        del template.obj
        del editor.obj
        del printer.obj
        del dbase.obj
        del userint.obj
        del dladt.obj
```

```
/*************************************************************************

                MODULE : MYMAIN.C

                VERSION: 1.0

                AUTHOR : Metin AKINCI

                DATE   : 31 MAY, 1988

                EXPLANATION:
                        Main project module.
                Interfaces modules and prepares program buffers.

                CHANGE LOG:



*************************************************************************/

#include "prport.h"
#include <stdio.h>
#include <mem.h>


/*         External Function Definitions        */

  extern Void editpage();
  extern Void start_up();
  extern Void print_page();
  extern Char *con_text();
  extern Int  getrequest();
  extern Bool ifsave();
  extern Void enterlog();
  extern Void deallocate();
  extern Void allocate();
  extern Bool loadtemplate(Int);
  extern Int  gettemplate();

/*  Buffer pointers which are visable to other program modules */
  Char *TEXTBUF;
  Char *KEYWORD;

/* Define  Size of Program Buffers      */
#define KEYSIZE 12
#define BUFSIZE 3500

/* Define function prototypes           */
 Void allocbuf();
 Void deallocbuf();
 Void flushbuffers();
```

```
main()
{
  Char ch;
  Bool done;

  done=FALSE;
                                  /* initialize the modules       */
  allocbuf();
  loadindex();
  while (!done)
  {
    switch (getrequest())
    {
     case 1:    tempinit();       /* Prepare document             */
                if (!loadtemplate(gettemplate()))
                {
                  error((Int) 2);
                  break;
                }
                start_up();
                editpage();
                con_text();
                print_page();
                if (ifsave())
                  savedoc();
                deallocate();
                break;
     case 2:                      /* Browsing a document          */
                getkey2db();
                if (browse())
                  print_page();
                break;
     case 3:                      /* Entering an incoming document*/
                enterlog();
                break;
     case 4:    print_file();     /* Printing incoming doc file   */
                break;

     case 5:                      /* Return to DOS                */
                done=TRUE;
                break;
     default: break;
    }
  flushbuffers();
  }
  updateindexf();
  deallocbuf();
}
```

```
/*****************************************************************
ALLOCBUF(): Allocates program buffers.
*****************************************************************/
   Void
allocbuf()
   {
     KEYWORD=(Char *)malloc(KEYSIZE);
     TEXTBUF=(Char *) malloc(BUFSIZE);
     if ((!KEYWORD) || (!TEXTBUF))
       {
         printf(" there is no enough memory. program exiting.");
         exit(0);
       }
     memset(KEYWORD,' 0',KEYSIZE);
     memset(TEXTBUF,' 0',BUFSIZE);
   }

/*****************************************************************
DEALLOCBUF(): Deallocates program buffers.
*****************************************************************/
   Void
deallocbuf()
   {
     free(KEYWORD);
     free(TEXTBUF);
   }
/*****************************************************************
FLUSHBUFFERS(): Clears program buffers.
*****************************************************************/
   Void
flushbuffers()
   {
     memset(KEYWORD,' 0',KEYSIZE);
     memset(TEXTBUF,' 0',BUFSIZE);
   }
```

```
/*****************************************************************************

                    MODULE : TEMPLATE.H

                    VERSION: 1.0

                    AUTHOR : Metin AKINCI

                    DATE   : 15 MAY, 1988

                    EXPLANATION:
                         Contains all definitions and declarations
                    for TEMPLATE.C Module.

                    CHANGE LOG:



*****************************************************************************/


#include "prport.h"              /* program defined header files*/
#include "keydef.h"
#include "bioslib.h"

#include <stdlib.h>              /* Compiler header files.      */
#include <alloc.h>
#include <conio.h>
#include <stdio.h>
#include <mem.h>
#include <string.h>


char * TEMPLATE_FILE="template.fil";
                                 /* define filename for templates */


typedef struct Field_record
  {
   Int   row;                    /* field start row number       */
   Int   column;                 /* field start column number    */
   Int   width;                  /* field width                  */
   Char  *msg;                   /* pointer to message buffer    */
   Char  *reply;                 /* pointer to reply buffer      */
   Bool   editable;              /* If 1 then  editable else not */
  } Field;



Field *PagePtr[3];               /* Video page pointer array     */
GLOBAL Int PG;                   /* Video page counter           */


typedef char dlist;              /* generic pointer to dllist    */
dlist *mylist;
```

```
GLOBAL extern Char *TEXTBUF ;    /* pointer to buffer            */


/*          Function prototypes             */

extern Void putcur(Int,Int);
extern Void clrscrn();
extern Void ring_bell();
extern Void  writea(Int,Int);

/*       Function prototypes for DLADT.C    */

extern   Field *dl_find(char *,Field *,int (* fieldcomp)());
extern   Bool  *dl_add(char *, Field *,int (* fieldcomp)());
extern   Bool  *dl_delete(char *,Field *,int (* fieldcomp)());
extern   char  *dl_alloc();
extern   Bool  *dl_free(char *);
extern   Field *dl_next(char *);
extern   Field *dl_prior(char *);
extern   Field *dl_first(char *);
extern   Field *dl_last(char *);
extern   Field *dl_curr(char *);
```

```
/*****************************************************************************

                MODULE : TEMPLATE.C

                VERSION: 1.0

                AUTHOR : Metin AKINCI

                DATE   : 15 FEB, 1988

                EXPLANATION:
                        This module contains the data structure
                holding template informations and provides the
                other modules functional interface by hiding
                the data structure. All operations on data
                structure are defined as a function within
                             this module.

                CHANGE LOG:



*****************************************************************************/

#include <template.h>

Void dispPage(Int);

/*****************************************************************************
TEMPINIT() :  Initiliaze the module variables.
*****************************************************************************/
   Void
tempinit()
{
  PagePtr[0]=NULL;              /* initialize page pointers     */
  PagePtr[1]=NULL;
  PagePtr[2]=NULL;
  PagePtr[3]=NULL;
  PG=0;                         /* initialize page counter      */

  mylist=dl_alloc();            /* create double linked list    */
  if (!mylist)
    exit(0);                    /* if fail to create dllist     */
}
```

```
/*****************************************************************************
FGETLINE() : Read line from file. Returns 0 if eof.
*****************************************************************************/
  Int
fgetline(fp,s,limit)
     FILE *fp;
     Char *s;
     Int limit;
   {
     int c,i;
     i=0;
     while (i<limit-1 || !feof(fp))
     {
         c=fgetc(fp);
         if (c==' n')
           return(i);
         s[i]=(Char) c;
         ++i;
     }
     if (feof(fp))
       return(0);

     return(i);
 }

/*****************************************************************************
FIELDCOMP():  Compares two field. Used by DLADT.C.
*****************************************************************************/
   int
fieldcomp(field1,field2)
   Field *field1,*field2;
   {
    if (field1->row!=field2->row)
      return(field1->row - field2->row);
    return(field1->column - field2->column);
   }
```

```
/*****************************************************************************
PARSELINE():
    This function gets input line and parses it in order to find
 field message string defined within two special characters (@..@).
 Returns the message string without @ character.
*****************************************************************************/
   Char *
parseline(line,fb)
   Char *line;
   Char *fb;
{
   Char *walkptr;

   walkptr=line;                  /*  set walkptr                    */

   while (*walkptr!='@')          /*  skip all characters until the*/
     ++walkptr;                   /*  beginning mark of msg string */
   ++walkptr;                     /*  skip @ mark                    */
   while (*walkptr!='@')          /*  get message characters until */
    {                          /*  the end mark of message field*/
       *fb=*walkptr ;
       ++fb;
       ++walkptr;
     }
   return(fb);
}
```

```
/*****************************************************************************
SHOWTEMPLATES():  Displays all template names in template.fil.
                  Assumes all lines whose first character is #
                  are template names. Displays string after #.
                  Provides --more-- facility if there is more than
                  one screen full template names.
*****************************************************************************/
   Void
showtemplates()
   {
     FILE *fptr;
     int  tnum;
     Char *line,*readline;

     fptr=fopen(TEMPLATE_FILE,"r");
     if (!fptr)
     {
       error((Int) 1);
       return;
     }
     readline=(Char *) malloc(80);
     memset(readline,' 0',80);
     line=readline;
     tnum=1;
     while (fgetline(fptr,line,(Int) 80))
     {                    /* while not EOF read lines     */
       if (line[0]=='#')        /* if new template definition   */
       {                  /* display it with order number */
         printf(" n%d  %s  n",tnum,(line+1));
         tnum=tnum+1;
        }
        if (!(tnum % 12))       /* control scrolling            */
        {
          printf(" n%s","---- to see more hit any key ----");
          getche();
        }
      memset(readline,' 0',80);
      line=readline;
      }
     printf(" n n%s n","---- to continue hit space bar ----");
     free(readline);
     getche();
     fclose(fptr);
}
```

```
/*****************************************************************************
LOADTEMPLATE():
    Gets the template information from file line by line,
all field information into data structure. Cooperates with
DLADT.C.  In case of failure returns false.
*****************************************************************************/
  Bool
loadtemplate(request)
  Int request;
  {
    FILE *fptr;                 /* pointer to template in file  */
    Char ch;
    Char *readline,*line;
    Int  l,r,c,w,e,i;
    Field *temp;
    Bool flag,done,neof;

        fptr=fopen(TEMPLATE_FILE,"r");
        if (!fptr)
          return(FALSE);
        if (request==0)
          return(FALSE);
                                /* prepare buffer for input line*/
        readline=(Char *) malloc(80);
        memset(readline,' 0',80);
        line=readline;

        i=0;
        do
        {
          flag= fgetline(fptr,line,(Int)80);
          if (!flag)            /* if EOF then return NULL      */
             return(FALSE);
          if (line[0]=='#')     /* if template definition       */
             i=i+1;
        }
        while (i<request);
    done=FALSE;
    while (!done)
      {
        if (!fgetline(fptr,line,(Int)80))
                                /* read line from template file */
          return(TRUE);
                                /* if not eof then              */
        switch (line[0])        /* evaluate the line            */
        {
          case  '*': break;     /* skip comment line            */

          case  '#': done=TRUE; /* new template definition      */
                     break;
          default  :            /* load the field definition    */
                    temp=(Field *)malloc(sizeof(Field));
                    if (!temp)
                      {
                         printf(" n out of memory  n");
                         printf(" n program exiting..");
                         exit(0);
```

```
                        /* return to DOS                */
                    }
                    sscanf(line,"%d%d%d%d",&r,&c,&w,&e);
                    temp->row=r;
                        /* row number of field          */
                    temp->column=c;
                        /* column number of the field   */
                    temp->width=w;
                        /* width of the field           */
                    temp->editable=e;
                        /* flag for if it is editable   */
                        /* dynamically allocate buffer  */
                        /* for both message and reply   */
                    temp->msg= (Char *)malloc(w+1);
                    memset(temp->msg,' 0',w+1);
                    parseline(line,temp->msg);
                    temp->reply= (Char *)malloc(w+1);
                    memset(temp->reply,' 0',w+1);
                        /* add the field record to list */
                    dl_add(mylist,temp,fieldcomp);
                        /* set video page pointer       */
                    if (PG==0)
                     {
                         PagePtr[0]=temp;
                         PG=PG+1;
                      }
                    else if (PG>3)
                      {
                      error((Int) 4);
                      return(FALSE);
                      }
                    else if (r/(PG*25))
                      {
                        PagePtr[PG]=temp;
                        PG=PG+1;
                      }
                    break;
      }
      memset(readline,' 0',80);
      line=readline;
    }
  free(readline);
  fclose(fptr);
  return(TRUE);
}
```

```
/*****************************************************************************
GO_PRIOR():
    Set the current field pointer to the prior field.
If there is no field to go, warns user maintains current field.
*****************************************************************************/
    Void
go_prior()
 {
    Field *curr,*temp;

    curr=dl_curr(mylist);        /* save the current field       */
    temp=dl_prior(mylist);
    while (temp)
    {
      if (PG>0  && temp<PagePtr[PG])
       {
         PG=PG-1;
         dispPage(PG);
         return;
       }
      if (temp->editable)        /* if there is no field to go   */
         return;
      temp=dl_prior(mylist);
     }                             /* do it until next editable field*/
     ring_bell();                /* warn the user and            */
     dl_find(mylist,curr,fieldcomp);
                                 /* resume the original position */
     return;
 }

/*****************************************************************************
GO_NEXT():
Set the current field pointer to the next fillable field.
If there is no field element to go, warns the user by sound.
*****************************************************************************/
   Void
go_next()

  {
    Field *curr,*temp;

    curr=dl_curr(mylist);        /* save the current field       */
    temp=dl_next(mylist);

    while (temp)
    {
      if (PG<3 && temp==PagePtr[PG+1])
       {
        PG=PG+1;
        dispPage(PG);
        return;
       }
      if (temp->editable)        /* seek for next editable field */
         return;
      temp=dl_next(mylist);
    }                              /* seek next editable field     */
    ring_bell();
    dl_find(mylist,curr,fieldcomp);
```

```
                                /* if there is no field to go   */
  return;                       /* resume the previous position */
}
```

```
/**************************************************************************
GFWIDTH():   Returns the field width ( range of field )
**************************************************************************/
    Int
gfwidth()
{
  Field *temp;
  temp=dl_curr(mylist);
  return(temp->width);
}
/**************************************************************************
GFCOL():  Returns current field column number.
**************************************************************************/
    Int
gfcol()

{
  Field *temp;
  temp=dl_curr(mylist);
  return(temp->column);
}
/**************************************************************************
GFROW():  Gets field starting row number and returns it.
**************************************************************************/
   Int
gfrow()
 {
   Field *temp;
   temp=dl_curr(mylist);
   return((Int) (temp->row) % 25);
 }
/**************************************************************************
GFREPLY():  This function returns the field entry edited by user
 It is going to be used when it is needed to be reedited.
**************************************************************************/
    Char *
gfreply()
 {
   Field *temp;
   temp=dl_curr(mylist);
   return(temp->reply);
 }
```

```
/**********************************************************************
GFMSG(): Returns the field prompt
**********************************************************************/
  Char *
gfmsg()
 {
   Field *temp;
   temp=dl_curr(mylist);
   return(temp->msg);
 }

/**********************************************************************
FILLABLE():
    Returns TRUE if current field is editable otherwise FALSE.
**********************************************************************/
    Bool
fillable()
{
 Field * temp;
 temp=dl_curr(mylist);

 if (temp->editable)
   return(TRUE);

 return(FALSE);
}
```

```
/**********************************************************************
DISPPAGE():
This function displays the document format on screen page by page.
It is invoked by sending appropriate pade number.
**********************************************************************/
    Void
dispPage(pagenum)
    Int pagenum;
 {
   Int r,c,w;
   Field *temp;

   clrscrn();
   dl_find(mylist,PagePtr[pagenum],fieldcomp);

   while ((dl_curr(mylist)!=NULL) &&
                        (dl_curr(mylist)!=PagePtr[pagenum+1]))
    {
     r=gfrow();
     c=gfcol();
     w=gfwidth();
     putcur(r,c);
     cputs(gfmsg());
     if (fillable())
       {
          writea((Char) ATTR,(Int) (w-strlen(gfmsg())));
          putcur(gfrow(),gfcol()+strlen(gfmsg()));
          putstr(gfreply());
       }
     dl_next(mylist);
    }
                                /* set current field pointer    */
                                /* to first editable field      */
    temp=dl_find(mylist,PagePtr[PG],fieldcomp);
    while (!(temp->editable))
      temp=dl_next(mylist);
 }
```

```
/*****************************************************************************
PAGE_DOWN():  Displays the next page.
*****************************************************************************/
   Void
page_down()
   {
     if ((PG>3) || (PagePtr[PG+1]==NULL))
       {
         ring_bell();
         return;
       }
     PG=PG+1;
     dispPage(PG);
   }

/*****************************************************************************
PAGE_DOWN():  Displays the previous page.
*****************************************************************************/
   Void
page_up()
   {
     if (PG==0)
       {
         ring_bell();
         return;
       }
     PG=PG-1;
     dispPage(PG);
   }

/*****************************************************************************
CON_TEXT(): This function converts the contents of data
structure into text format and stores it global buffer 'TEXTBUF'.
Returns the address of buffer.
*****************************************************************************/
   Char *
con_text()
{
   Int i ;
   Bool newline;
   Int lastcol,lastrow,lc;
   Int mfw,rfw,ccol,crow;
   Field *temp;
   Char *pp;                    /* pointer to global buffer     */

   lastrow=0;
   lastcol=0;

   temp=dl_first(mylist);       /* start from first element     */
   pp=TEXTBUF;                  /* pointer to global buffer     */

   for (i=lastrow;i< temp->row ;++i)
    {                             /* handle vertical tab          */
      *pp=(Char) CR;
      ++pp;
      *pp=(Char) LF;
      ++lastrow ;
      ++pp;
```

```
 }
while (temp)                    /* process each field          */
 {
  ccol=temp->column;
  crow=temp->row;
  mfw=strlen(temp->msg);
  rfw=strlen(temp->reply);

  if (crow > lastrow)
   {
    lastcol=0;
    newline=TRUE;
   }

  for (i=lastrow ; i<crow ;++i)
    {
      *pp=(Char) CR;
      ++pp;
      *pp=(Char) LF;
      ++pp;
      ++lastrow;
    }

  if (newline)
  {
    for (i=lastcol;i<ccol;++i)
       {
         *pp= (Char)BLANK;
         ++pp;
         ++lastcol;
       }
    newline=FALSE;
   }
  strcpy(pp,temp->msg);
  pp=pp+strlen(temp->msg);
  lastcol=lastcol+mfw;

  if (fillable()==TRUE)
    {
      strcpy(pp,temp->reply);
      pp=pp+strlen(temp->reply);
      lastcol= lastcol + rfw ;
    }

  for (i=mfw+rfw ;i<(temp->width);++i)
    {
      *pp= (Char)BLANK;
      ++pp;
      ++lastcol;
    }
  temp=dl_next(mylist);         /* advance to next field       */
}
*pp=(Char) CR;                  /* put end of document mark    */
++pp;
*pp=(Char) LF;
++pp;
*pp=' 0' ;                      /* treat each document as string*/
```

```
    return(pp);
}

/*****************************************************************************
DEALLOCATE(): This  function frees memory space already allocated
to doubly linked_list we should free memory when user is done
or he wants to make another page in order to be able to load
new template file and load it into linked list.
*****************************************************************************/
  Void
deallocate()

 {
    Field *temp;                /* first free data in the list  */
    temp=dl_first(mylist);
    while (!temp)
      {
        free(temp->msg);
        free(temp->reply);
        free(temp);
        temp=dl_next(mylist);
      }
 }

/*****************************************************************************
START_UP() :
In order to start filling out the form displayed on screen,
displays first page and sets the current field pointer to first
fillable field in the list. Initialize PG counter.
*****************************************************************************/
    Void
start_up()

 {
   PG=0;                        /* set page numbet to zero        */
   dl_first(mylist);            /* set the pointer to first field */
   dispPage((Int) 0);           /* start to display first videopage*/
   while (fillable()==FALSE)    /* proceed to first editable field*/
     dl_next(mylist);
 }

/*****************************************************************************
```

MODULE : EDITOR.H

VERSION: 1.0

AUTHOR : Metin AKINCI

DATE : 15 MAR, 1988

EXPLANATION:
Contains declarations and definitions for
EDITOR.C module

CHANGE LOG:

```
*****************************************************************************/

                                        /* Program Defined Header Files */
#include "prport.h"
#include "keydef.h"
#include "bioslib.h"
#include "myascii.num"

                                        /* Compiler Header Files        */
#include <ctype.h>
#include <mem.h>
#include <stdio.h>
#include <conio.h>


        /*          Function prototypes for system.c                    */

extern Void putcur(Int,Int);
extern Void readcur(Int *,Int *);
extern Void writec();
extern Void writea();
extern Void writeca(Char,Int,Int);
extern Bool shift_pressed(Void);
extern Int  putstr(Char *);
extern Void ring_bell();

        /*          Function prototypes for template.c                  */

extern Void go_next();
extern Int gfrow();
extern Int gfcol();
extern Int gfwidth();
extern Char *gfmsg();
extern Char *gfreply();
```

```
/*****************************************************************************

                    MODULE : EDITOR.C

                    VERSION: 1.0

                    AUTHOR : Metin AKINCI

                    DATE   : 15 MAR, 1988

                    EXPLANATION:
                         This module gets the user responds for
                    each field on the form specified by template
                    and also allows the user to edit his input.
                    Prevent user from overwriting on non fillable
                    fields and field message.

                    CHANGE LOG:



*****************************************************************************/


#include "editor.h"


/*****************************************************************************
GETREPLY():
       Gets user response for currently edited field ,returns it.
Allows user to edit field by defined keys. Interprets the key
combinations  for our own characters.
*****************************************************************************/
    Int
getreply(row,col,width,msg,reply)

    Int row,col,width;          /* window location and width  */
    Char *msg;                  /* field message              */
    Char *reply;                /* input buffer               */
{
    Int k;
    Int n,len;
    Int mfw;                    /* message field width        */
    Int rfw;                    /* response field width       */
    Int ccol;                   /* visible cursor column      */
    Char *cp  ;                 /* Character pointer to buffer*/
    Char *tmp;                  /* temporary character pointer*/
    Char *buffer;               /* Edit Buffer for response   */

   if (msg!=NULL)
      mfw=strlen(msg);          /* Get message field width   */
   else
```

```
   mfw=0;
putcur(row,col+mfw);          /* Place the cursor at the very
                                 first character location in
                                 the field to be filled out */
rfw=width-mfw;                /* Calculate reply field width*/
buffer=(Char *)malloc(rfw+1);
                              /* Allocate memory for buffer
                                 of size response field width*/
memset(buffer,' 0',rfw+1);    /* Clear  the buffer           */
memcpy(buffer,reply,strlen(reply));
                              /* copy reply into buffer      */
                              /* in case that it was previously
                                 edited                      */
memset(reply,' 0',strlen(reply));
                              /* clear previous string       */
cp=buffer;                    /* Set walk pointer on buffer  */

while ((k=get_key())!=K_RETURN)
                              /* Get key until RETURN key is hit*/

  {
     if (isascii(k) && (isprint(k)))
                              /* If it is not control key    */
       {
        len=strlen(cp);
        if (cp + len -buffer < rfw )
          {
            memcpy(cp+1,cp,len);
            *cp=(Char) k;
            ++cp;
          }
        else
            ring_bell();      /*    buffer full              */
       }
     else                     /* Else if it is control key */

        switch (k)
         {
           case K_LEFT :      /* move left one char          */
                         if (cp > buffer)
                            --cp;
                         break;

           case K_RIGHT :     /* move right one char         */
                         if (*cp !=' 0')
                            ++cp;
                         break;

           case K_UP    :     /* move prior field            */
                          memcpy(reply,buffer,strlen(buffer));
                            /* copy buffer back            */
                          free(buffer);
                          return(k);

           case K_DOWN  :     /* move next field             */
                          memcpy(reply,buffer,strlen(buffer));
                            /* copy buffer back            */
```

```
                        free(buffer);
                        return(k);

        case K_PGUP :    memcpy(reply,buffer,strlen(buffer));
                           /* copy buffer back          */
                        free(buffer);
                        return(k);

        case K_PGDN :   memcpy(reply,buffer,strlen(buffer));
                           /* copy buffer back          */
                        free(buffer);
                        return(k);
        case K_CTRLH :     /* destructive backspace     */
                        if (cp>buffer)
                        {
                          tmp=cp-1 ;
                          memcpy(tmp,cp+1,strlen(tmp));
                          --cp;
                        }
                        break;

        case K_HOME  :     /* go to the beginning of buffer*/
                        cp=buffer;
                        break;

        case  K_END  :     /* go to the end of buffer    */
                        while (*cp !=' 0')
                           ++cp;
                        break;
        case K_CEND    : memcpy(reply,buffer,strlen(buffer));
                             /* copy buffer back           */
                         free(buffer);
                         return(k);


        case K_DEL     :     /* delete character at cursor */
                         memcpy(cp,cp+1,strlen(cp));
                         break;
        case K_ESC     :     /* cancel current input       */
                         memset(buffer,' 0',rfw+1);
                         cp=buffer;
                         break;
/* Following keys are being used to convert to TURKISH chars  */

        case K_ALTC    : if (shift_pressed())
                            *cp= (Char) ascii_C;
                         else
                            *cp= (Char) ascii_c;
                            ++cp;
                         break;
        case K_ALTG    : if (shift_pressed())
                           *cp= (Char) ascii_G;
                         else
                           *cp= (Char) ascii_g;
                         ++cp;
                         break;
```

```
                case K_ALTI   : if (shift_pressed())
                                  *cp= (Char)  ascii_I;
                                else
                                  *cp= (Char)  ascii_i;
                                ++cp;
                                break;

                case K_ALTO   : if (shift_pressed())
                                  *cp= (Char)  ascii_O;
                                else
                                  *cp= (Char)  ascii_o;
                                ++cp;
                                break;
                case K_ALTS   : if (shift_pressed())
                                  *cp= (Char)  ascii_S;
                                else
                                  *cp= (Char)  ascii_s;
                                ++cp;
                                break;
                case K_ALTU   : if (shift_pressed())
                                  *cp= (Char) ascii_U;
                                else
                                  *cp= (Char) ascii_u;
                                ++cp;
                                break;

                default       :
                                ring_bell();
                                break;
            }
                                   /* display the reply window    */
        ccol=col+mfw;
        putcur(row,ccol);
        writec(' ',(Int) (width-mfw));
        putstr(buffer);
        putcur(row,ccol+(cp-buffer));
                                /* reposition the cursor       */
    }
    memcpy(reply,buffer,strlen(buffer));
                                /* copy buffer back            */
    free(buffer);
    return(k);                  /* return the key that cause
                                   to end function             */
}
```

```
/*****************************************************************************
EDITPAGE():
 Function that allows user to edit page form displayed on screen.
 Invokes function getreply that gets user reply for each field.
*****************************************************************************/
    Void
editpage()
{
    Int  row,col,width;
    Char *m ;
    Char *r ;
    Int flag;
    Bool done;

    done=FALSE;
    do
     {
       row=gfrow();
       col=gfcol();
       width=gfwidth();
       m=gfmsg();
       r=gfreply();

                                /* get user response and store it*/
       flag= getreply(row,col,width,m,r);
       switch (flag)
        {
          case   K_DOWN:          /* Go next fillable field        */
                            go_next();
                            break;
          case   K_UP  :          /* Go previous fillable field    */
                            go_prior();
                            break;
          case   K_PGUP:          /* scroll up                     */
                            page_up();
                            break;
          case   K_PGDN:          /* scroll down                   */
                            page_down();
                            break;
          case  K_CEND :          /* end of the editing session    */
                            done=TRUE;
                            break;
          default      : go_next();
                            break;
        }
     }
   while (!done);             /* do until end of page          */
}
```

```
/*****************************************************************************

                    MODULE : PRINTER.C

                    VERSION: 1.0

                    AUTHOR : Metin AKINCI

                    DATE   : 1 MAY,1988

                    EXPLANATION:
                            Performs two main functions.  Prints
                    content of program buffer and prints doclog.fil
                    file.  Allows printing Turkish characters .
                    For each character, separate function
                    is assigned.  Input is filtered and if any
                    special character is encountered, appropriate
                    function is invoked and that character is
                    printed.

                    CHANGE LOG:



*****************************************************************************/

#include  <stdio.h>

#include  "prport.h"
#include  "myascii.num"          /* file which contains  ascii    */
                                 /* numbers for our own char      */
#include  "extra.fnt"            /* Include file that contains extra*/
                                 /* characters fonts for printer  */

#include "epson.dat"             /* printer commands file         */


GLOBAL extern Char *TEXTBUF;     /* Pointer to text buffer        */


                                 /* External function declarations*/
extern Void putcur(Int,Int);
extern Void drawframe(Int,Int,Int,Int);


char *DOCLOGFILE="doclog.fil";   /* file name to be printed out   */
```

```
/*****************************************************************************
SETPRTGMODE():
Sets printer dot graphics mode by sending appropriate ESC sequence.
*****************************************************************************/
   Void
setprtgmode( pptr)
   FILE *pptr;                    /* Pointer to printer stream    */
  {
   int i;

    for (i=0; i<4; ++i)           /* Send ESC sequence to set     */
    fputc(p_grmode[ i] ,pptr);    /* printer to dot graphics mode */
   }

/*****************************************************************************
PRINT_C():
Prints the new lowercase 'c' letter by sending dot patterns to
the printer.
*****************************************************************************/
  Void
print_c(pptr)
  FILE *pptr;
  {
     int i;
     setprtgmode(pptr);           /* Send ESC sequence to set     */
                                  /* printer to dot graphics mode */
     for (i=0; i<8; ++i)
     fputc(c_pattern[ i] ,pptr);  /* Copy new c pattern to printer*/
   }

/*****************************************************************************
PRINT_CC():
Prints the new uppercase 'C' letter by sending dot patterns to
the printer.
*****************************************************************************/
 Void
print_CC(pptr)
  FILE *pptr;
  {
   int i;

   setprtgmode(pptr);             /* Send ESC sequence to set      */
                                  /* printer to dot graphics mode  */
   for (i=0; i<8; ++i)
     fputc(CC_pattern[ i] ,pptr); /* Copy new C pattern to printer */
  }
```

```
/*****************************************************************************
PRINT_I()
Prints the new lowercase 'i' letter by sending dot patterns to
the printer.
*****************************************************************************/
  Void
print_i(pptr)
  FILE *pptr;
  {
   int i;

    setprtgmode(pptr);           /* Send ESC sequence to set      */
                                 /* printer to dot graphics mode  */
    for (i=0; i<8; ++i)
      fputc(i_pattern[i],pptr); /* Copy new c pattern to printer */

  }
/*****************************************************************************
PRINT_CI():
Prints the new uppercase 'I' letter by sending dot patterns to
the printer.
*****************************************************************************/
  Void
print_CI(pptr)
  FILE *pptr;
  {
   int i;
   setprtgmode(pptr);            /* Send ESC sequence to set      */
                                 /* printerto dot graphics mode   */
   for (i=0; i<8; ++i)
     fputc(CI_pattern[i],pptr); /* Copy new c pattern to printer */

  }

/*****************************************************************************
PRINT_G():
Prints the new lowercase 'g' letter by sending dot patterns to
the printer.
*****************************************************************************/
  Void
print_g(pptr)
  FILE *pptr;

  {
   int i;

    setprtgmode(pptr);           /* Send ESC sequence to set      */
                                 /* printer to dot graphics mode  */
    for (i=0; i<8; ++i)
     fputc(g_pattern[i],pptr);   /* Copy new g pattern to printer */
  }
```

```
/*************************************************************************
 PRINT_CG():
 Prints the new uppercase 'G' letter by sending dot patterns to
 the printer.
*************************************************************************/
  Void
print_CG(pptr)
  FILE *pptr;

  {
   int i;
    setprtgmode(pptr);           /* Send ESC sequence to set      */
                                 /* printer to dot graphics mode  */
    for (i=0; i<8; ++i)
     fputc(CG_pattern[i],pptr); /* Copy new C pattern to printer */
  }

/*************************************************************************
PRINT_O():
Prints the new lowercase 'o' letter by sending dot patterns to
the printer.
*************************************************************************/
  Void
print_o(pptr)
  FILE *pptr;

  {
   int i;
    setprtgmode(pptr);           /* Send ESC sequence to set      */
                                 /* printer to dot graphics mode  */
    for (i=0; i<8; ++i)
      fputc(o_pattern[i],pptr); /* Copy new o pattern to printer */
  }
/*************************************************************************
 PRINT_CO():
 Prints the new capital 'O' letter by sending dot patterns to
 the printer.
*************************************************************************/
  Void
print_CO(pptr)
  FILE *pptr;

  {
   int i;

    setprtgmode(pptr);           /* Send ESC sequence to set      */
                                 /* printer to dot graphics mode  */
    for (i=0; i<8; ++i)
     fputc(CO_pattern[i],pptr); /* Copy new O pattern to printer */
  }
```

```
/*****************************************************************************
 PRINT_U():
 Prints the new lowercase 'u' letter by sending dot patterns to
 The printer.
*****************************************************************************/
  Void
print_u(pptr)
  FILE *pptr;
  {
    int i;
    setprtgmode(pptr);          /* Send ESC sequence to set      */
                                /* printer to dot graphics mode  */
    for (i=0;i<8;++i)
      fputc(u_pattern[i],pptr); /* Copy new u pattern to printer */
   }

/*****************************************************************************
 PRINT_CU():
 Prints the new uppercase 'U' letter by sending dot patterns to
 the printer.
*****************************************************************************/
  Void
print_CU(pptr)
  FILE *pptr;
  {
    int i;
    setprtgmode(pptr);          /* Send ESC sequence to set      */
                                /* printer to dot graphics mode  */
    for (i=0;i<8;++i)
      fputc(CU_pattern[i],pptr);/* Copy new U pattern to printer */
   }

/*****************************************************************************
  PRINT_S():
  Prints the new lowercase 's' letter by sending dot patterns to
  the printer.
*****************************************************************************/
  Void
print_s(pptr)
  FILE *pptr;
  {
    int i;
    setprtgmode(pptr);          /* Send ESC sequence to set      */
                                /* printer to dot graphics mode  */
    for (i=0;i<8;++i)
      fputc(s_pattern[i],pptr); /* Copy new s pattern to printer */
  }
```

```
/*****************************************************************************
 PRINT_CS() :
 Prints the new uppercase 'S' letter by sending dot patterns
 to printer.
*****************************************************************************/
  Void
print_CS(pptr)
  FILE *pptr;

  {
    int i;
    setprtgmode(pptr);           /* Send ESC sequence to set      */
    for (i=0;i<8;++i)            /* printer to dot graphics mode  */
     fputc(CS_pattern[i],pptr); /* Copy new C pattern to printer */
   }

/*****************************************************************************
  SETPRT(): Resets the printer. Cancels all possible modes .
*****************************************************************************/
   Void
setprt(pptr)
   FILE *pptr;
{
  int i;

  fputs(p_cbold,pptr);           /* Cancel bold mode            */
  fputs(p_cds,pptr);             /* Cancel double strike mode   */
  fputs(p_cital,pptr);           /* Cancel italic mode          */
  fputs(p_ccmp,pptr);            /* Cancel compressed mode      */
  for (i=0;i<3;++i)
     fputc(p_cul[i],pptr);       /* Cancel underline mode       */
  fputs(p_init,pptr);            /* Printer hardware initialize */
}
```

```
/*****************************************************************************
 PRINT_PAGE() :
 Gets the contents of TEXTBUF global buffer and by searching
 and printing extra  characters   sends them to printer.
*****************************************************************************/
   Void
print_page()
  {
    FILE *printer;
    Char ch;
    Char *bufptr;
    Int i,count;

    count=getnumber();

    printer= fopen("PRN","w");
    if (printer==NULL)
      {
        clrscrn();
        drawframe((Int)19,(Int)15,(Int) 23,(Int) 65);
        putcur((Int) 21,(Int) 17);
        printf(" n%s","PLEASE MAKE SURE PRINTER IS ON");
        printf("%s n","...AND HIT ANY KEY");
        getche();
        return;
      }
    clrscrn();
    setprt(printer);
    for (i=0;i<count;++i)
      {
         bufptr=TEXTBUF;
         while ((ch=*bufptr)!=' 0')
            {
              switch (ch)
                {
                   case ascii_c: print_c(printer);
                                 break;
                   case ascii_C: print_CC(printer);
                                 break;
                   case ascii_i: print_i(printer);
                                 break;
                   case ascii_I: print_CI(printer);
                                 break;

                   case ascii_o: print_o(printer);
                                 break;
                   case ascii_O: print_CO(printer);
                                 break;
                   case ascii_s: print_s(printer);
                                 break;
                   case ascii_S: print_CS(printer);
                                 break;
                   case ascii_u: print_u(printer);
                                 break;
                   case ascii_U: print_CU(printer);
                                 break;
                   case ascii_g: print_g(printer);
```

```
                                break;
                    case ascii_G: print_CG(printer);
                                  break;

                    default     : fputc(ch,printer);


                }
                putchar(ch);
                ++bufptr;
            }
            fputc(FF,printer);    /* Put form feed character      */
        }                           /* for next copy                */
}
```

```
/**************************************************************************
PRINT_FILE(): Prints program defined doclog.fil file.
**************************************************************************/
Void
print_file()
  {
    FILE *printer,*fptr;
    Char ch;
    Char *bufptr;
    Int i,count;

    fptr=fopen(DOCLOGFILE,"r");
    if (!fptr)
       {
         error((Int) 1);
         return;
       }
    printer= fopen("PRN","w");
    if (printer==NULL)
      {
        clrscrn();
        drawframe((Int)19,(Int)15,(Int) 23,(Int) 65);
        putcur((Int) 21,(Int) 17);
        printf(" n%s","PLEASE MAKE SURE PRINTER IS ON");
        printf("%s n","...AND HIT ANY KEY");
        getche();
        return;
      }
    clrscrn();
    setprt(printer);
    while (!feof(fptr))

      {
        ch=fgetc(fptr);
        switch (ch)
          {
              case ascii_c: print_c(printer);
                            break;
              case ascii_C: print_CC(printer);
                            break;
              case ascii_i: print_i(printer);
                            break;
              case ascii_I: print_CI(printer);
                            break;
              case ascii_o: print_o(printer);
                            break;
              case ascii_O: print_CO(printer);
                            break;
              case ascii_s: print_s(printer);
                            break;
              case ascii_S: print_CS(printer);
                            break;
              case ascii_u: print_u(printer);
                            break;
              case ascii_U: print_CU(printer);
                            break;
              case ascii_g: print_g(printer);
                            break;
```

```
            case ascii_G: print_CG(printer);
                          break;
            default     : fputc(ch,printer);
                          break;
        }
    }
}
```

```
/*************************************************************************

                    MODULE : DBASE.C

                    VERSION: 1.0

                    AUTHOR : Metin AKINCI

                    DATE   : 15 MAY, 1988

                    EXPLANATION:
                         Implements index sequential access to
                    database file which holds documents.
                    cooperates with LLADT.C module.

                    CHANGE LOG:



*************************************************************************/


#include "prport.h"

#include <stdio.h>
#include <string.h>
#include <mem.h>


GLOBAL extern Char *TEXTBUF;     /* pointer to global buffer     */

/*      definitions on the key to database                      */

extern Char *KEYWORD;            /* pointer to keyword buffer    */
#define KEYSIZE  12              /* define size of keyword       */
#define BUFSIZE 3500;

/*      function prototypes for userinterface module            */

extern Void  getdocinfo( Char *);
extern Char *getkey2db();

/*      File name definitions related to Dbase.C module         */

Char *INDEXFILE = "index.fil";   /* index file , contains kwords */
Char *DBASEFILE = "dbase.fil";   /* database file                */
Char *DOC_LOG = "doclog.fil";    /* file to keep record of       */
                                 /* incoming document            */
```

```
/*  definition of data structure for index sequential access    */
/*  implementation                                              */

typedef struct  INDEX
  {
    Char *key2db;
  } INDEX ;

typedef char  INDXLST;          /* generic pointer to llist     */
INDXLST *indexlist;

#define POSITIVE 1

/*********  Function prototypes for linked list module   **********/

extern char *ll_alloc();
extern INDEX *ll_find(INDXLST *,INDEX *,int (pfcomp)());
extern Bool ll_delete(INDXLST *,INDEX *);
extern INDEX *ll_next(INDXLST *);
extern INDEX *ll_first(INDXLST *);
extern INDEX *ll_last(INDXLST *);
extern Bool ll_add(INDXLST *,INDEX *,int (*pfcomp)());


/*****************************************************************************
MYSTRCMP(): Compares two string in llnode.
        Function like strcmp().
*****************************************************************************/
   Int
mystrcmp(node1,node2)
   INDEX *node1,*node2;
   {
    Int i;

    i=0;
    while ((node1->key2db)[i]==(node2->key2db)[i])
        if (node1->key2db[i++]==' 0')
           return(0);
    return((node1->key2db)[i] - (node2->key2db)[i]);
   }
```

```
/*****************************************************************************
INDEXCMP():
compares two node of linked list.  In order to put new item
at the end of the list, it always return positive.
*****************************************************************************/
   Int
indexcmp(s,t)
  INDEX * s,*t;
  {
    return(POSITIVE);
  }

/*****************************************************************************
LOADINDEX(): Loads keywords from index file into linked list.
*****************************************************************************/
  Void
loadindex()
  {
    FILE *fptr;
    Bool flag;
    Char line[80];
    INDEX *indexptr;

    indexlist=ll_alloc();
    fptr=fopen(INDEXFILE,"r");
    if (fptr==NULL)              /* if file is not existing,    */
      return;                    /* assume there is no document */
                                 /* in database.                */

     while (fgetline(fptr,line,(Int) KEYSIZE))
     {
        indexptr=(INDEX *)malloc(sizeof(INDEX));
        indexptr->key2db=(Char *) malloc(KEYSIZE+1);
        memset(indexptr->key2db,' 0',KEYSIZE+1);
        strcpy(indexptr->key2db,line);
        ll_add(indexlist,indexptr,indexcmp);
      }
    fclose(fptr);
  }

/*****************************************************************************
UPDATEINDEXF(): Writes all keywords back tto index file in the same
order in the linked list.
*****************************************************************************/
   Void
updateindexf()
    {
      FILE *fptr;
      Int i;
      INDEX *temp;

      fptr=fopen(INDEXFILE,"w+");/* open index file              */
                                 /* if file does not exist create*/
      temp=ll_first(indexlist);
      while (temp)               /* write everything in indexlist*/
        {                          /* back to index file .         */
          fprintf(fptr,"%s n",temp->key2db);
          temp=ll_next(indexlist);
```

```
    }
  fclose(fptr);
  temp=ll_first(indexlist);  /* first free the data in LLIST */
  while (temp)
  {
    free(temp);
    temp=ll_next(indexlist);
  }
  ll_free(indexlist);        /* then free the linked list     */
}
```

```
/*****************************************************************************
GETINDEX(): Gets the index value for a certain keyword.
The relative location in the list implies index for that keyword.
*****************************************************************************/
   Int
getindex()
   {
     Int i;
     INDEX *temp,*countptr;
        temp=(INDEX *) malloc(sizeof(INDEX));
        temp->key2db=(Char *) malloc(KEYSIZE);
        memset(temp->key2db,' 0',KEYSIZE);
        strcpy(temp->key2db,KEYWORD);
        countptr=ll_first(indexlist);
        i=1;
        while (countptr)
        {
          if (!(strcmp(countptr->key2db,temp->key2db)))
            return(i);
          countptr=ll_next(indexlist);
          i=i+1;
        }
        return(0);
    }
/*****************************************************************************
 SAVE_KEY():
 Saves the key to database and assigns an implicit index
 value to it.
*****************************************************************************/
  Bool
save_key()
  {
    INDEX *new;
    new=(INDEX*) malloc(sizeof(INDEX));
    new->key2db=(Char *)malloc(KEYSIZE);
    strcpy(new->key2db,KEYWORD);
    ll_add(indexlist,new,indexcmp);
  }
```

```
/*****************************************************************************
DISLPAYBUFFER(): Displays the program buffer.
It is invoked after document has been fetch in buffer.
Provides -- more -- facility.
*****************************************************************************/
  Void
displaybuffer()
 {
   Char *temp;                    /* temp pointer to buffer       */
   Int linecount,                 /* variables to control scroll  */
       pc;
   Char ch;

   temp=TEXTBUF;                  /* temp pointer to buffer       */
                                  /* intialize variables          */
   linecount=1;                   /* number of lines displayed    */
   pc=1;                          /* number of video pages        */
   clrscrn();
   temp=TEXTBUF;                  /* set pointer to the beginning */
                                  /* of buffer                    */
   while ((ch=*temp)!=' 0')       /* display document requested   */
     {
      putchar(ch);                /* control scrolling            */
      ++temp;
      if (ch==' n')
         linecount=linecount+1;
      if (linecount %(pc*23)==0)
         {
           pc=pc+1;
           fprintf(stdout," n%s","--- more ---    hit any key ... n");
           getche();
         }
     }
   fprintf(stdout," n%s","--- hit any key to continue --- n");
   getche();
 }
```

```
/**************************************************************************
BROWSE():  It is used to browse any document from database file.
It requests the keyword, gets index then fetches document into
buffer. Calls display buffer routine.
Returns FALSE in case of failure.
**************************************************************************/
  Bool
browse()
  {
   Int recnum;
   Int size;
   Int j;
   Char *temp;
   FILE *fptr;
   Int i;
   Int recsize;

     fptr=fopen(DBASEFILE,"rb");
     if (fptr==NULL)
     {
        error((Int) 1);
        return(FALSE);
      }
      recnum=getindex();          /*  Get the index                */
      if (! recnum)               /*  if not found                 */
      {                             /* return after error messages  */
        error((Int) 3);
        return(FALSE);
      }
      temp=TEXTBUF;               /* temp pointer to buffer        */
      for (i=1; i<recnum; ++i)    /* get the offset to document    */
      {
        fscanf(fptr,"%d",&recsize);
        fseek(fptr,(Long) recsize, 1);
      }
      fscanf(fptr,"%d",&size);    /*  get the size of document     */
      for (j=1; j<size; ++j)      /*  copy document into buffer    */
       {
        *temp=fgetc(fptr);
        ++temp;
       }
      *temp=' 0';                 /* attach string terminator      */
      displaybuffer();            /* display the buffer            */
                                  /* by controlling scrolling      */
   rewind(fptr);
   fclose(fptr);
   return(TRUE);
  }
```

```
/*****************************************************************************
 SAVEDOC(): Saves the document currently edited into DBASEFILE.
*****************************************************************************/
   Void
savedoc()
   {
     FILE *fptr;
     Char ch,*buffer;

     buffer=TEXTBUF;              /* set pointer to buffer          */
     getkey2db();                 /* get key to dbase from user     */
     save_key();                  /* save key in the linked list    */
     fptr=fopen(DBASEFILE,"ab+");
                                  /* append the document            */
                                  /* first write its size           */
     fprintf(fptr,"%d n", strlen(TEXTBUF));
     while ((ch=*buffer)!=' 0') /* then write buffer into file    */
      {
        fputc(ch,fptr);
        ++buffer;
      }
     fclose(fptr);
   }

/*****************************************************************************
 ENTERLOG(): Allows the user to enter the information about any
             incoming document.  Information about document is
             provided by the function getdocinfo() that is defined
             within userinterface module.
*****************************************************************************/
  Void
enterlog()
  {
    FILE *fptr;
    Char line[80];

    fptr=fopen(DOC_LOG,"a+");

    getdocinfo(line);
    fprintf(fptr,"%s n",line);
    fclose(fptr);
  }
```

```
/*************************************************************************

                MODULE : USERINT.C

                VERSION: 1.0

                AUTHOR : Metin AKINCI

                DATE   : 1 MAY, 1988

                EXPLANATION:
                     Implements user interface part of the project.
                Gets all input values from user returns
                to calling module.

                CHANGE LOG:



*************************************************************************/

#include "prport.h"
#include "bioslib.h"

#include <stdio.h>
#include <alloc.h>


/* DEFINE BOX CHARACTERS */

#define VBAR2 186
#define HBAR2 205
#define ULC22 201
#define URC22 187
#define LLC22 200
#define LRC22 188


/* Function prototypes   */

extern Void clrscrn();
extern putcur(Int,Int);
extern fgetline(FILE *, Char *,int);
extern putsbuf(Char *,Char *);
extern putcbuf(Char *,Char );


GLOBAL extern Char *KEYWORD;
```

```
/* Define pointers to messages to be  displayed in module */

static char MAINHEADER[]=
                          " <<<<<<    MAIN MENU    >>>>>>" ;
static char *MAINMENU[]=  {
                          "1. PREPARING DOCUMENT ...",
                          "2. BROWSING A DOCUMENT ..",
                          "3. ENTERING AN INCOMING DOCUMENT..",
                          "4. PRINTING INCOMING DOCUMENT LOG..",
                          "5. EXIT TO DOS.."
                         } ;

static Char  ASKENTER[]="Please enter your choice=>";

static Char  ASKSAVE[]=
                        " WILL THIS DOCUMENT BE SAVED? (Y/N)..";

static Char ASK_KEY[]=
            " PLEASE ENTER THE DOCUMENT ID IN CORRECT FORMAT..";

static Char HOWMANY[]=
                          "HOW MANY COPY DO YOU WANT ?   ==> ";

static Char HOWMANY_HELP[]=
                       "ZERO IMPLIES NO OUTPUT. MAXIMUM 8 COPIES..";

#define MENULX 5         /* Main menu left corner coordinates */
#define MENULY 15
#define MENURX 21        /* Main menu right corner coordinate */
#define MENURY 65
#define MSGX   23        /* Prompt area coordinates           */
#define MSGY   5

#define KEYSIZE 12
```

```
/*************************************************************************
PUTHEADER(): Displays any header to specified location.
*************************************************************************/
   Void
putheader(x,y,strptr)
   Int x,y;
   Char *strptr;
 {
   putcur((Int)x,(Int)y);
   puts(strptr);
 }

/*************************************************************************
DRAWFRAME(): Draws frame for the given coordinates.
*************************************************************************/
   Void
drawframe(leftupX,leftupY,rightdownX,rightdownY)
   Int leftupX,leftupY;
   Int rightdownX,rightdownY;
  {
    Int i;
    putcur((Int)leftupX,(Int)leftupY);
    putchar(ULC22);
    for (i=0;i< rightdownY-leftupY-1;++i)
      putchar(HBAR2);
    putchar(URC22);
    for (i=leftupX+1;i<rightdownX;++i)
      {
        putcur((Int)i,(Int)leftupY);
        putchar(VBAR2);
        putcur((Int)i,(Int)rightdownY);
        putchar(VBAR2);
      }
    putcur((Int)rightdownX,(Int)leftupY);
    putchar(LLC22);
    for (i=0;i<rightdownY-leftupY-1;++i)
      putchar(HBAR2);
    putchar(LRC22);
}

/*************************************************************************
 IFSAVE():
 Function that prompts the user if he wants to save the document.
*************************************************************************/
  Bool
 ifsave()
 {
   Char choice;

   clrscrn();
   putheader((Int)MSGX,(Int) MSGY,ASKSAVE);
   do
    {
     putcur((Int) MSGX,(Int) (strlen(ASKSAVE)+MSGY+2));
     scanf("%c",&choice);
    }
   while (choice!='y' && choice!='Y' && choice!='n' && choice!='N');
   if (choice=='y' || choice=='Y')
```

```
    return(TRUE);

  return(FALSE);
}
/*****************************************************************************
GETKEY2DB():
Gets the key to database. Prompts the user to enter the keyword
then gets it and stores in the global variable KEYWORD.
*****************************************************************************/
  Char *
getkey2db()
{
  clrscrn();                    /* Prompt the user to enter the */
                                /* keyword.                     */
  putcur((Int) MSGX,(Int)MSGY);
  printf("%s",ASK_KEY);
  putcur((Int) (MSGX+1),(Int) 15);
  writea((Char) ATTR,(Int) KEYSIZE);
                                /* color the input area         */
  getreply((Int) (MSGX+1),(Int) 15,(Int) KEYSIZE,NULL,KEYWORD);
                                /* get the keyword from the user*/
  strupr(KEYWORD);              /* convert it to uppercase      */
  return(KEYWORD);
}

/*****************************************************************************
GETDOCINFO(): Gets the incoming document information from user
allows user to edit his input.
*****************************************************************************/
  Void
getdocinfo(where)
  Char *where;
  {
    Char line[80];
    memset(line,' 0',80);
    clrscrn();
    drawframe((Int)16,(Int)15,(Int) 22,(Int) 75);
    putcur((Int) 17,(Int) 18);
    printf("PLEASE ENTER THE INFORMATION ABOUT INCOMING DOCUMENT");
    putcur((Int) 19,(Int) 18);
    printf("DOCUMENT ID          FROM              DATE        ");
    putcur((Int) 21,(Int) 17);
    writea((Char) ATTR,(Int) 58);
    getreply((Int)21,(Int) 17,(Int) 58, NULL,line);
    strcpy(where,line);
   }
```

```
/*****************************************************************************
MMENU(): Function that displays main menu.
*****************************************************************************/
   Void
 mmenu()
 {
   Int x,y,n,i;
   clrscrn();

   putheader((Int)3,(Int) 25,MAINHEADER);
   drawframe((Int) MENULX,(Int) MENULY,(Int) MENURX,(Int) MENURY);
   n= sizeof(MAINMENU)/sizeof(Char *);
   putcur((Int)(MENULX+2),(Int)(MENULY+2));

   for (i=0;i<n;++i)
     {
      printf("%s n",MAINMENU[i]);
      readcur(&x,&y);
      putcur((Int)(x+2),(Int)(MENULY+2));
     }
  }

/*****************************************************************************
GETREQUEST(): Gets the number of the request.
*****************************************************************************/
 Int
getrequest()
 {
   Char choice;
   mmenu();
   putheader((Int)MSGX,(Int)MSGY,ASKENTER);
   putcur((Int) MSGX,(Int)MSGY);
   printf("%s",ASKENTER);

   do
     {
       putcur((Int) MSGX,(Int)(strlen(ASKENTER)+2+MSGY));
       scanf("%c",&choice);
     }
   while (choice<'1' || choice>'5');

   return(choice-'0');
 }
```

```
/*************************************************************************
  GETNUMBER():
  Gets the count of the copies from user. 0 implies nothing
  will be printed.
*************************************************************************/
   Int
getnumber()
   {
     Char num;

     clrscrn();
     drawframe((Int)19,(Int)15,(Int) 23,(Int) 65);
     putcur((Int) 21,(Int) 20);
     printf("%s n",HOWMANY);
     putcur((Int) 22,(Int) 20);
     printf("%s n",HOWMANY_HELP);
     do
     {
       putcur((Int) 21,(Int) (20+strlen(HOWMANY)));
       scanf("%c",&num);
     }
     while (num<'0' || num> '8');
     return(num-'0');

   }
/*************************************************************************
GETTEMPLATE(): Gets the number of the template from user.
*************************************************************************/
   Int
gettemplate()
   {
     Int num;

     clrscrn();
     printf("%s n"," ALL TEMPLATES AVAILABLE HAS BEEN LISTED BELOW");
     printf(" Find the number for template that you need..");
     showtemplates();
     clrscrn();
     drawframe((Int)21,(Int)10,(Int) 23,(Int) 69);
     putcur((Int) 22,(Int) 12);
     printf("ENTER THE NUMBER OF TEMPLATE YOU WANT TO EDIT ..");
     writea((Char)ATTR,(Int) 3);
     scanf("%d",&num);
     return(num);
   }
```

```
/*****************************************************************************
 ERROR():
 Simple error handler function. Prompts user according to error_code
 which is determined by related modules.
*****************************************************************************/
    Void
 error(error_code)
    Int error_code;
 {
     char ch;

     clrscrn();
     ring_bell();
     drawframe((Int) 8,(Int) 20,(Int) 10,(Int) 60);
     putcur((Int) 9,(Int) 22);
     printf("%s","!!! ");

     switch (error_code)
     {
       case 1 :printf("%s"," Can not open file..");
               break;
       case 2 :printf("%s"," Error while loading template.. ");
               break;
       case 3 :printf("%s"," Error ! Possibly incorrect keyword");
               break;
       case 4: printf("%s"," Template too long  ");

       default:break;
     }
     putcur((Int) 24,(Int) 10);
     printf("%s","PLEASE HIT ANY KEY TO CONTINUE..");
     getche();
 }
```

```
/*****************************************************************************

                    MODULE : SYSTEM.H

                    VERSION: 1.0

                    AUTHOR : Metin AKINCI

                    DATE   : 25 FEB,1988

                    EXPLANATION: Header file for system.c module
                                 It contains all definitions and
                                 data structure used within system.c module

                    CHANGE LOG:


*****************************************************************************/


#include "prport.h"                 /* INCLUDE FILE FOR PROJECT GLOBALS */
#include "keydef.h"                 /* KEYBOARD  SCAN CODES             */
#include "bioslib.h"                /* BIOS FUNCTIONS                   */

#include <dos.h>                    /* Include neccesary TC header files*/
#include <bios.h>
#include <stdio.h>


/*         DEFINITIONS OF DATA STRUCTURE FOR REGISTERS                  */

struct WORD_REGS
  {
    WORD  ax,bx,cx,dx,si,di,cflag;
  } ;


struct BYTE_REGS
  {
    BYTE   al,ah,bl,bh,cl,ch,dl,dh;
  } ;


union REGISTER
 {
   struct WORD_REGS x ;
   struct BYTE_REGS h ;

 } REGS;
```

```
/*************************************************************************

                MODULE : SYSTEM.C

                VERSION: 1.0

                AUTHOR : Metin AKINCI

                DATE   : 14 JAN,1988

                EXPLANATION:
                        This module contains system dependent
                functions definitions.  All functions within
                this module are system dependent.  To port the
                program to the another system, this entire module
                must be changed with the one which contains
                appropriate system calls.

               CHANGE LOG:
                       Throughout this project IBM PC default video
               page number (which is zero) has been used.
               All functions in this modules work will under
               this assumption.
               To make them general purpose, page number
               must be added to function parameters.

*************************************************************************/
#include  "system.h"


/*************************************************************************
PUTCUR():
Gets the row and column  number of new cursor location and sets
the cursor to given coordinate.
*************************************************************************/
    Void
putcur(r,c)
    Int  r,c;
{
  union REGS inrg,outrg ;

  inrg.h.ah=CUR_POS;              /* cursor addressing func no  */
  inrg.h.dl=c;                    /* column coordinate          */
  inrg.h.dh=r;                    /* row coordinates            */
  inrg.h.bh=0;                    /* video page number          */

  int86(VIDEO,&inrg,&outrg);      /*  BIOS video routine call   */
}
```

```
/*****************************************************************************
CLRSCRN():  Clears screen by invoking BIOS VIDEO service 6.
*****************************************************************************/
    Void
clrscrn()
{
  union REGS rg;
  rg.h.ah=SCROLL_UP;            /* screen scroll code          */
  rg.h.al=0;                    /* clear screen code           */
  rg.h.ch=0;                    /* start row                   */
  rg.h.cl=0;                    /* start column                */
  rg.h.dh=24;                   /* end row                     */
  rg.h.dl=79;                   /* end column                  */
  rg.h.bh=0;                    /* blank line is black         */

  int86(VIDEO,&rg,&rg);         /* BIOS video routine call     */
  putcur((Int) 1,(Int) 1);      /* reposition the cursor       */
}

/*****************************************************************************
READCUR():
Reads the cursor position by calling  BIOS VIDEO service 3.
*****************************************************************************/
     Void
readcur(r,c)
     Int *r,*c;
{
  union REGS inrg,outrg ;

  inrg.h.ah=GET_CUR;            /* AH=function no 3            */
  inrg.h.bh=0;                  /* video page number           */

  int86(VIDEO,&inrg,&outrg);    /* DOS call                    */

  *r=outrg.h.dh;                /* returned row number         */
  *c=outrg.h.dl;                /* column number               */
}
/*****************************************************************************
GET_KEY():
Gets key from keyboard and if it is non ASCII key
returns scan code for it.
*****************************************************************************/
   Int
get_key()
 {
   Int ch;
                                /* if normal key codes         */
  if ((ch=bdos(KEYIN,0,0) & LOBYTE ) != ' 0')
       return(ch);
                                /*convert scan codes to unique*/
                                /* internal codes              */
  return((bdos(KEYIN,0,0) & LOBYTE) | XF );
 }
```

```
/*************************************************************************
WRITEC():
This function writes a character or string of identical characters,
starting at the current cursor position. It does not advance
the cursor.
*************************************************************************/
   Void
writec(ch,count)
   Char  ch;
   Int count;
 {
   union REGS inrg,outrg;

   inrg.h.ah=WRITE_CHAR;
   inrg.h.al=ch;
   inrg.h.bh=0;
   inrg.x.cx=count;
   int86(VIDEO,&inrg,&outrg);
 }
/*************************************************************************
READCA():  Reads the character with attribute
*************************************************************************/
  Void
readca(ch,attr)
  Char *ch;
  Int  *attr;
{
  union REGS inrg,outrg;
  inrg.h.ah=RE_C_ATT;
  inrg.h.bh=0 ;
  int86(VIDEO,&inrg,&inrg);
  *ch=outrg.h.al ;
  *attr=outrg.h.ah;
}
```

```
/*************************************************************************
WRITECA(): Writes character with attribute for the number of count
*************************************************************************/
  Void
writeca(ch,attr,count)
  Char ch;
  Char attr;
  Int count;
{
  union REGS inregs,outregs;

  inregs.h.ah=WR_C_ATT ;
  inregs.h.al= ch ;
  inregs.h.bh= 0;
  inregs.h.bl= attr ;
  inregs.x.cx= count;

  int86(VIDEO,&inregs,&outregs);
}

/*************************************************************************
WRITEA():
  Reads N characters from current cursor location on and
  writes them back with specified attribute
*************************************************************************/
  Void
writea(atr,n)
  Char atr;
  Int  n ;
 {
   Int i;
   Char attrx;
   Int chx;
   Int r,c;

   readcur(&r,&c);
   for (i=0;i<n;++i)
    {
      putcur(r,c+i);
      readca(&chx,&attrx);
      writeca(chx, atr,(Int) 1);
    }
   putcur(r,c);
 }

/*************************************************************************
RING_BELL():  Rings the bell by sending character defined as BELL
  to the output port.
*************************************************************************/
    Void
ring_bell()
 {
  putch(BELL);
 }
```

```
/*************************************************************************
PUTSTR():
  Displays the null terminated string on the screen from current cursor
  position on.
*************************************************************************/
  Int
putstr(s)
   Char *s;
 {
   Int r,c,c0;

   readcur(&r,&c);
   for (c0=c; *s!=' 0';++s,++c)
     {
      putcur((Int)r,(Int)c);
      writec(*s,(Int) 1);
     }

   putcur((Int)r,(Int)c);
   return(c-c0);
 }
/*************************************************************************
WRITEMSG():  Writes field message with specified attribute.
*************************************************************************/
  Int
writemsg(r,c,w,msg,attr)
  Int  r,c,w;
  Char *msg;
  Int  attr;
 {
   Int i;
   Char ch;
   putcur(r,c);
   i=0;
   while ((*msg!=' 0') && ( i<w ))
     {
       ch=*msg;
       writeca(ch,(Int) attr,(Int)1);
       putcur(r,++c);
       ++msg;
       ++i ;
     }
    return(i);
  }
/*************************************************************************
SVDPG():  Sets active video page
*************************************************************************/
   Void
svdpg(vp)
   Int vp;
{
  union REGS inregs,outregs;
  inregs.h.ah=SETVDPG ;          /* BIOS Video Service 5          */
  inregs.h.al= vp ;              /* Active display page to be set */
  int86(VIDEO,&inregs,&outregs);
}
```

```
/*****************************************************************************
ACVDPG():  Gets active video page
*****************************************************************************/
   Int
acvdpg()
 {
  union REGS inregs,outregs;

  inregs.h.ah=GETVDMOD ;          /* BIOS Video Service 15
                                     to get video information     */
  int86(VIDEO,&inregs,&outregs);
  return(outregs.h.bh);           /* return the active display page
                                     number in bh register        */
}

/*****************************************************************************
SHIFT_STATUS():   Gets shift status
*****************************************************************************/
   Bool
shift_pressed()
  {
    union REGS inregs,outregs;
    Char status_byte;

    inregs.h.ah=KBD_STATUS;
    int86(KBD_INT,&inregs,&outregs);

    status_byte=outregs.h.al;
    status_byte=status_byte & 0x43;
                                  /* mask sixth bit for 'CAPSLOCK'*/
                                  /* first and second bit for     */
                                  /* left and right 'SHIFT' keys  */
    switch (status_byte)
     {
      case 1: return(TRUE);       /* right shift key pressed      */

      case 2: return(TRUE);       /* left shift key pressed       */

      case 3: return(FALSE);      /* both right and left shift key*/
                                  /* pressed.                     */
      case 64: return(TRUE);      /* CAPS LOCK is on              */

      case 65: return(FALSE);     /* CAPS LOCK and right shift key*/

      case 66: return(FALSE);     /* CAPS LOCK and left shift key */

      case 67: return(TRUE);      /* three of them is active      */

      default: return(FALSE);
    }
  }
```

```
/*****************************************************************************

                    MODULE : DLADT.C

                    VERSION: 1.0

                    AUTHOR : Metin AKINCI

                    DATE   : 10 MAY, 1988

                    EXPLANATION:
                             General purpose Doubly Linked List
                    abstract data type.
                    Application side has to provide pointer to linked
                    list and pointer to data .
                    Appropriate compare function must be provided by
                    user of this module.
                             Implementation of this module is
                    independent from data structure.  It has been
                    implemented by using C language's generic pointer
                    feature.

                    CHANGE LOG:



*****************************************************************************/


#include "prport.h"

#include <mem.h>
#include <stdlib.h>
#include <stdio.h>
#include <alloc.h>

typedef char DATA;
                                    /* generic pointer to data      */
typedef struct DLNODE               /* Doubly linked list node      */
   {                                  /* data structure               */
      DATA * pdata;                 /* generic pointer to data      */
      struct DLNODE *left;          /* pointer to left node         */
      struct DLNODE *right;         /* pointer to right node        */
    } DLNODE;

typedef struct DLLIST               /* Doubly Linked List structure */
   {
     DLNODE  *head,*tail,*curr;
   } DLLIST;
```

```
/*****************************************************************************
DL_ALLOC(): Creates doubly linked list. Returns pointer to
            dllist.
*****************************************************************************/
   DLLIST *
dl_alloc()
   {
     DLLIST *pdl;
     pdl= (DLLIST *) malloc(sizeof(DLLIST));
     if (!pdl)
       return(NULL);
     pdl->head=NULL;            /* initialize to null          */
     pdl->tail=NULL;
     pdl->curr=NULL;
     return(pdl);               /* either null or succesful pdl  */
   }

/*****************************************************************************
DL_FIND(): Finds data in the doubly linked list and sets
current pointer then returns pointer to data found.
*****************************************************************************/
    DATA *
dl_find(pdl,pnode,pfcmp)
    DLLIST *pdl;
    DATA *pnode;
    int (* pfcmp)();
    {
       int comp;
       if (!pdl->head)
          return(NULL);         /* list is empty               */
       pdl->curr=pdl->head;
       while (pdl->curr)
       {
         comp=(*pfcmp)(pnode,pdl->curr->pdata);
         if (comp==0)
            return(pdl->curr->pdata);
         else if (comp<0)
            return(NULL);
         pdl->curr=pdl->curr->right;
       }
       return(NULL);
    }
```

```
/*****************************************************************************
DL_DELETE(): Deletes data provided by user from dllist
          Returns TRUE if attemp is succefull otherwise FALSE.
*****************************************************************************/
  Bool
dl_delete(pdl,pnode,pfcmp)
  DLLIST *pdl;
  DATA *pnode;
  int (*pfcmp)();
  {
    DATA * tempnode;

    if (!(tempnode=dl_find(pdl,pnode,pfcmp)))
        return(FALSE);          /* not exists                    */
    free(tempnode);

    if (pdl->head==pdl->curr)
       pdl->head=pdl->curr->right;
                                /* if it is first element        */

    if (pdl->tail==pdl->curr)
       pdl->tail=pdl->curr->left;
                                /* if it is last element         */

    if (pdl->curr->left)
       pdl->curr->left->right=pdl->curr->right;

    if (pdl->curr->right)
       pdl->curr->right->left=pdl->curr->left;

    free(pdl->curr);            /* delete the node               */
    pdl->curr=pdl->head;        /* reset the current pointer     */

    return(TRUE);

  }
```

```
/*****************************************************************************
DL_ADD(): Adds data provided by user into doublu linked list.
        Returns False in case of failure.
*****************************************************************************/
   Bool
dl_add(pdl,pnode,pfcmp)
  DLLIST *pdl;
  DATA *pnode;
  int (*pfcmp)();
  {
    DLNODE *pdlnode;

    if (dl_find(pdl,pnode,pfcmp))
       return(FALSE);           /* already exists in the list   */

    pdlnode=(DLNODE*) malloc (sizeof(DLNODE));
    pdlnode->pdata=pnode;

    if (pdl->head==NULL)
    {
      pdl->head=pdl->tail=pdlnode;
      pdlnode->right=pdlnode->left=NULL;
    }
    else if (pdl->curr==NULL)   /* my node is the greatest      */
    {
      pdl->tail->right=pdlnode;
      pdlnode->left=pdl->tail;
      pdl->tail=pdlnode;
      pdlnode->right=NULL;
    }
    else
    {                           /* my node is somewhere either in
                                   middle or at the beginning   */
      if (pdl->head==pdl->curr)
      {                         /* my node should be first      */
        pdlnode->right=pdl->head;
        pdlnode->left=NULL;
        pdl->head=pdlnode;
      }
      else
      {
        pdl->curr->left->right=pdlnode;
        pdlnode->right=pdl->curr;
        pdlnode->left=pdl->curr->left;
        pdl->curr->left=pdlnode;
       }
     }
   return(TRUE);
 }
```

```
/*****************************************************************************
DL_FREE(): Deallocates memory allocated for doublu linked list.
        Deallocation of memory allocated for user data is under
        user responsibilty.
*****************************************************************************/
  Bool
dl_free(pdl)
  DLLIST *pdl;
  {
     DLNODE *tempnode;

     if (!pdl)
       return(FALSE);           /* error                          */

     if (!pdl->head)
      {
         free(pdl);
         return(TRUE);
       }
     pdl->curr=pdl->head;
     while (pdl->curr)
       {
          tempnode=pdl->curr;
          pdl->curr=pdl->curr->right;
          free(tempnode);
        }

     free(pdl);
    return(TRUE);
   }

/*****************************************************************************
DL_NEXT(): Sets the current pointer to next node in the list.
        Returns pointer to next node data.
*****************************************************************************/
  DATA *
dl_next(pdl)
  DLLIST * pdl;
  {
        pdl->curr=pdl->curr->right;

        if (!pdl->curr)
           return(NULL);
        return(pdl->curr->pdata);
  }
```

```
/*****************************************************************************
DL_PRIOR(): Sets the current pointer to previous node.
        Returns the pointer to prior node data.
*****************************************************************************/
  DATA *
dl_prior(pdl)
  DLLIST *pdl;

  {     if (!pdl->curr->left)
          return(NULL);
        pdl->curr=pdl->curr->left;
        if (pdl->curr)
          return(pdl->curr->pdata);
        return(NULL);
   }

/*****************************************************************************
DL_FIRST(): Sets the current pointer to first element in the list.
        Returns the pointer to first data in the list.
*****************************************************************************/
  DATA *
dl_first(pdl)
  DLLIST * pdl;
  {
        pdl->curr=pdl->head;
        if (pdl->curr)
          return(pdl->curr->pdata);
       return(NULL);

  }

/*****************************************************************************
DL_LAST(): Sets the current pointer to last item  in the list
        Returns pointer to last data in the list.
*****************************************************************************/
   DATA *
dl_last(pdl)
   DLLIST *pdl;
   {
        pdl->curr=pdl->tail;
        if (pdl->curr)
          return(pdl->curr->pdata);
        return(NULL);
   }

/*****************************************************************************
DL_CURR(): Returns pointer to data pointed by current pointer.
*****************************************************************************/
  DATA *
dl_curr(pdl)
  DLLIST *pdl;
  {
  if (pdl->curr)
    return(pdl->curr->pdata);
  return(NULL);
  }
```

```
/*****************************************************************************

                    MODULE :   LLADT.C

                    VERSION:  1.0

                    AUTHOR :  Metin AKINCI

                    DATE    :  15 MAY, 1988

                    EXPLANATION:
                           General purpose linked list implementation.
                    It has been implemented by using  C language's
                    generic pointer feature.
                    Pointer to data and appropriate compare function
                    must be provided by application side.

                    CHANGE LOG:



*****************************************************************************/

#include "prport.h"
#include <stdio.h>

typedef char DATA;                  /* generic pointer to DATA      */

typedef struct LLNODE               /* define each node of the list */
{
  DATA *pdata;                      /* generic pointer to data      */
  struct LLNODE *next;              /* pointer to next node         */
} LLNODE;

typedef struct LLIST                /* linked list structure        */
{
   LLNODE *head,*tail,*curr;
} LLIST;
```

```
/*****************************************************************************
LL_ALLOC(); Creates an empty linked list.
        Returns pointer to newly created list.
*****************************************************************************/
   LLIST *
ll_alloc()
  {
    LLIST *pll;

    pll=(LLIST *)malloc(sizeof(LLIST));
    if (pll)
     {
      pll->head=NULL;
      pll->tail=NULL;
      pll->curr=NULL;
     }
    return(pll);
  }

/*****************************************************************************
LL_FIND(): Finds any data in the linked list and returns pointer
        to data. Appropriate compare function must be provided
        by user of this module.
*****************************************************************************/
   DATA *
ll_find(pll,pnode,pfcomp)
   LLIST *pll;
   DATA  *pnode;
   int  (*pfcomp)();
   {
     int comp;

     if (!pll->head)            /* linked list is empty         */
        return(NULL);

     pll->curr=pll->head;       /* start from beginning         */
     while (pll->curr)
     {
       comp=(*pfcomp)(pnode,pll->curr->pdata);
       if (comp==0)             /* data is found in the list    */
         return(pll->curr->pdata);
       else if (comp<0)
         return(NULL);          /* it is smaller than first     */
         pll->curr=pll->curr->next;
                                /* keep searching               */
     }
     return(NULL);
   }
```

```
/**************************************************************************
LL_DELETE(): Deletes data from  linked list.  Returns TRUE
        if attempt is succesful otherwise FALSE.
**************************************************************************/
    Bool
 ll_delete(pll,pnode,pfcomp)
    LLIST *pll;
    DATA  *pnode;
    int (*pfcomp)();
    {
      LLNODE *tempnode;
      DATA *temp;

      if (!(temp=ll_find(pll,pnode,pfcomp)))
         return(FALSE);          /* not exists in the list       */

      tempnode=pll->curr;        /* if exists,save the pointer   */
                                 /* to the data to be deleted    */
      if (pll->head==pll->curr) /* if it is first element in list*/
      {
         pll->head=pll->curr->next;
         free(pll->curr);
         return(TRUE);
      }

      if (pll->tail==pll->curr) /* if it is last element in the list*/
         {
          pll->curr=pll->head;  /* find the previous node        */
          while (pll->curr->next!=pll->tail)
            pll->curr=pll->curr->next;
          pll->tail=pll->curr;
          return(TRUE);         /* return the new last item     */
         }
                                /* otherwise it is somewhere
                                   in the middle                */
       pll->curr=pll->head;
       while  (pll->curr->next !=tempnode)
         pll->curr=pll->curr->next;
       tempnode=pll->curr;
       tempnode->next=tempnode->next->next;
       free(tempnode->next);
       return(TRUE);
  }
```

```
/*****************************************************************************
LL_NEXT(): Sets the current pointer to next node and
        returns pointer to data in the next node of list.
*****************************************************************************/
    DATA *
ll_next(pll)
   LLIST *pll;
   {
        pll->curr=pll->curr->next;
        if (pll->curr)
                return(pll->curr->pdata);
        return(NULL);
   }

/*****************************************************************************
LL_FIRST(): Sets the current pointer to first item in the list.
        Returns pointer to data in the first node of list.
*****************************************************************************/
    DATA *
ll_first(pll)
    LLIST * pll;
    {
     pll->curr=pll->head;
     if (pll->curr)
        return(pll->curr->pdata);
     return(NULL);
    }

/*****************************************************************************
LL_LAST(): Sets the current pointer to the last item in the list.
        Returns pointer to data in the last node.
*****************************************************************************/
     DATA *
ll_last(pll)
     LLIST *pll;
     {
       pll->curr=pll->tail;
       if (pll->curr)
          return(pll->curr->pdata);
       return(NULL);
     }
```

```
/*****************************************************************************
LL_ADD():  Adds new item into linked list.  Appropriate compare
        function must be passed by user.  Returns FALSE if item
        to be added is already in list.
*****************************************************************************/
     Bool
ll_add(pll,pnode,pfcomp)
     LLIST *pll;                /* pointer to linked list       */
     DATA  *pnode;
     int (*pfcomp)();
     {
        LLNODE *temp,*tempnode;

        if (ll_find(pll,pnode,pfcomp))
           return(FALSE);       /* already exists               */

                                /* prepare linked list node     */
        temp= (LLNODE *) malloc(sizeof(LLNODE));
        temp->pdata=pnode;

        if (pll->head==NULL)    /* if linked list is empty      */
        {
          pll->head=pll->tail=temp;
          temp->next=NULL;
        }
        else if (pll->curr==NULL)
                                /* mynode is the greatest       */
        {
          pll->tail->next=temp;
          pll->tail=temp;
          temp->next=NULL;
         }
         else                   /* else my node is somewhere in */
         {                        /* the middle or at the beginning*/
           if (pll->head==pll->curr)
           {
             pll->head=temp;
             temp->next=pll->curr;
           }
           else
           {
             tempnode=pll->curr;
                                /* save the current pointer     */
             pll->curr=pll->head;
                                /* start from beginning         */
                                /* find the previous node       */
             while (pll->curr->next!=tempnode)
                pll->curr=pll->curr->next;
             pll->curr->next=temp;
             temp->next=tempnode;
           }
         }
        return(TRUE);

    }
```

```
/*****************************************************************************
LL_FREE(): Deallocates the memory allocated for linked list.
        Deallocation of memory allocated for data must be freed
        before this function is invoked.
*****************************************************************************/
   Bool
ll_free(pll)
   LLIST *pll;
   {
      LLNODE *temp;

      if (!pll)                  /* error                          */
         return(FALSE);

      if (!pll->head)            /* if linked list is empty        */
      {
         free(pll);
         return(TRUE);
      }
      pll->curr=pll->head;
      while (pll->curr)
      {
         temp=pll->curr;
         pll->curr=pll->curr->next;
         free(temp);
      }

      free(pll);
     return(TRUE);
   }
```

```
/*************************************************************************************************

                    MODULE : EGACHR.C

                    VERSION: 1.0

                    AUTHOR : Metin AKINCI

                    DATE   : 15 APR, 1988

                    EXPLANATION:
                         This is a memory resident program.  Creates
                    extra characters in the Turkish alphabet for EGA
                    adapter. This program reads the system info,
                    if EGA is present, it is installed.
                    Otherwise terminates by prompting user.

                         This program should compile and run outside
                    of integrated environment.

                    CHANGE LOG:




*************************************************************************************************/


#include <stdio.h>
#include <dos.h>
#include <process.h>
#include <mem.h>
#include <stdlib.h>



#define TRUE 1
#define FALSE 0


/* Functions related to video operations                */

    void
loadegachr(char *fptr,int block,int bpc,int char_count,int spos);

    void
get_egafont(char *fptr, int font);

    int
get_video_info(void);
```

```
/* Global variables and #DEFINEs related to video operations    */

#include "egachr.asc"           /* egachr.asc contains our      */
                                /* own character font           */
#define VIDEO 0x10              /* BIOS video interrupt         */
char fontarray[3585];           /* buffer for font storage      */
char ega_color;


/* Global variables related to TSR operations */

unsigned save_bp1, save_bp2, old_ds, old_psp;
unsigned old_env;


/* Turbo C system variables                   */

extern unsigned __brklvl;
extern unsigned _psp;


    void
error(int errnum);


main()
{
        int i, j, k;
        union REGS regs;

        get_video_info();
        if (!ega_color) regs.x.ax = 0x7;
                                /* set the video mode           */
        else regs.x.ax = 0x3;
        int86(VIDEO,&regs,&regs);

/* system checks out -- go ahead and put own chars. in font     */

        get_egafont(fontarray,14);
                                /* store the ROM font in fontarray */

        for(i=14*128,j=0;  i< 14*140;j++)
        {
                for(k=0;k<14;k++)
                                /* overwrite our own characters  */
                    fontarray[i++] = egachr_array[j][k];
        }

        loadegachr(fontarray,0,14,256,0);
                                /* load our font                */

  /* terminate and stay resident. Program length is determined by*/
  /* subtracting the psp address (_psp) from __brkval which is   */
  /* dynamically set to the address of the end of DS.            */

        keep(FALSE,_DS + (__brklvl + 15)/16 - _psp);
}
```

```
/*****************************************************************************/
/* LOADEGACHR -- Load a user-defined font and reset page length.             */
/* Parms: ptr. to user table, block to load, bytes-per-char,                 */
/* number of chars to store, starting position in font table.                */
/*****************************************************************************/
    void
loadegachr(char *fptr,int block,int bpc,int char_count,int spos)
{
        unsigned byte_block;

        byte_block = (bpc << 8) | block;

        _ES = _DS;
        _AX = 0x1100;           /* call function 0x11                */
        _BX = byte_block;       /* block to load                     */
        _CX = char_count;       /* number of characters to load      */
        _DX = spos;             /* character offset into table       */
        save_bp2  = _BP;        /* save BP for stack addressing      */
        _BP = FP_OFF(fptr);     /* load address of user font         */
        geninterrupt(VIDEO);
        _BP = save_bp2;
}

/*****************************************************************************/
/* GET_EGAFONT: This routine grabs an EGA font from ROM                      */
/* and stores it in the global variable fontarray                            */
/*****************************************************************************/
    void
get_egafont(char *fptr, int font)
{
    struct REGPACK regs;

    regs.r_ax = 0x1130;         /* EGA BIOS call to return font */
    if (font == 8)
        regs.r_bx = 0x0300;
    else if (font == 14) regs.r_bx = 0x0200;
        intr(VIDEO,&regs);
    movedata(regs.r_es,regs.r_bp,_DS, (unsigned) fptr,14*256);
}

/*****************************************************************************/
/* GET__VIDEO_INFO: A VGA or an EGA must be installed for this               */
/* program to work. The monitor must be an Enahanced Color or                */
/* Monochrome display and the correct adaptor must be active.                */
/*****************************************************************************/
    int
get_video_info()
{
        union REGS regs;
        unsigned char e_byte;

        /* First check for the presence of an EGA */
        regs.h.ah = 0x12;       /* EGA BIOS alternate select    */
        regs.h.bl = 0x10;       /* return EGA information.      */
        int86(VIDEO, &regs, &regs);
        if (regs.h.bl == 0x10) error(1);       /* EGA not found */
```

```
        /* EGA is present -- is it active? */
        e_byte = peekb(0,0x487);
                                /* EGA info. byte               */
        if (e_byte & 8) error(2);
                                /* EGA not active               */

/* Does the present, active EGA drive a color or mono monitor?  */

        if (regs.h.bh) ega_color = FALSE;
                                /* EGA drives a mono monitor    */
        else ega_color = TRUE;  /* EGA drives a color monitor   */

/* See if EGA drives an Enhanced Color Display                  */

        if (ega_color)
                if (!(regs.h.cl == 3 || regs.h.cl == 9))
                        error(1);
        return (1);
}
/*****************************************************************************/
/* ERROR: A simple error handler.                                            */
   void
error(int errnum)
{
 switch (errnum)
 {
 case 1: printf(" An EGA and Enhanced Color or Monochrome Display");
         printf(" nmust be present to use this program.");
         break;

 case 2: printf(" Please make the EGA the active adapter");
         printf("in order to run this program.");
         break;

 default: break;
  }
 printf(" nProgram exiting. n");
 exit(0xf);             /* Return code for DOS errorlevel */
}
```

```
/*****************************************************************************/
/*          FILE NAME: EGACHR.ASC                                            */
/*                                                                           */
/* egachr.asc: This is an ASCII representation of the italic font            */
/* characters used in egachr.C. This file is #includeD.                      */
/* In the table below, each row corresponds to a character. The              */
/* 14 elements of each row correspond to the 14 scan lines of the            */
/* character.                                                                */
/*****************************************************************************/

char egachr_array[12][14] = {
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x78, 0xcc,
  0x60,0x38, 0xcc, 0x78, 0x00, 0x30,  0x00 } ,
/* s */
{ 0x00, 0x00, 0x78, 0xcc, 0xc0, 0xe0, 0x38,
  0x0c, 0x0c, 0xcc, 0x78, 0x00, 0x30, 0x00 } ,
/* S */
{ 0x00, 0x00, 0x00, 0x66, 0x00, 0x7c, 0xc6,
  0xc6, 0xc6, 0xc6, 0x7c, 0x00, 0x00, 0x00 } ,
/* o */
{ 0x6c, 0x00, 0x38, 0x44, 0xc6, 0xc6, 0xc6,
  0xc6, 0xc6, 0x44, 0x38, 0x00, 0x00, 0x00 } ,
/* O */
{ 0x00, 0x00, 0x00, 0xcc, 0x00, 0xcc, 0xcc,
  0xcc, 0xcc, 0xcc, 0x7c, 0x00, 0x00, 0x00 } ,
/* u */
{ 0x00, 0xc6, 0x00, 0xc6, 0xc6, 0xc6,0xc6,
  0xc6, 0xc6, 0xc6, 0x7e, 0x00, 0x00, 0x00 } ,
/* U */
{ 0x00, 0x00, 0x00, 0x7c, 0x00, 0x7e, 0xcc,
  0xcc, 0xcc, 0x7c, 0x0c, 0xcc, 0x78, 0x00 } ,
/* g */
{ 0x3c, 0x00, 0x3c, 0x66, 0xc0, 0xc0, 0xc0,
  0xde, 0xc6, 0x66, 0x3a, 0x00, 0x00, 0x00 } ,
/* G */
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x78, 0xcc,
  0xc0, 0xc0, 0xcc, 0x78, 0x00, 0x30, 0x00 } ,
/* c */
{ 0x00, 0x00, 0x3c, 0x66, 0xc2, 0xc0, 0xc0,
  0xc0, 0xc2, 0x66, 0x3c, 0x00, 0x18, 0x00 } ,
/* C */
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x38, 0x18,
  0x18, 0x18, 0x18, 0x3c, 0x00, 0x00,  0x00 } ,
/* i */
{ 0x18, 0x00, 0x3c, 0x18, 0x18, 0x18, 0x18,
  0x18, 0x18, 0x18, 0x3c, 0x00, 0x00, 0x00 } ,
/* I */
} ;
```

```
/*****************************************************************************
PRPORT.H : Project portabilty header file.
           Contains programmer defined data types, printers, systems
           and compilers.
*****************************************************************************/

/*   Programmer defined data types      */

typedef unsigned char  Char;

typedef int  Int;

typedef long int  Long;

typedef short int Bool;

typedef unsigned char BYTE;

typedef unsigned int WORD;

#define   Void    void

#define   EXTERN  /**/

#define   TRUE    (Bool) 1

#define   FALSE   (Bool) 0

#define   GLOBAL  /**/


                                        /* define compilers           */

#define   TC     1                      /*  Turbo C Compiler          */
#define   LC     0                      /*  Lattice C compiler        */
#define   MSC    0                      /*  Microsoft C Compiler      */
#define   IBMC   0                      /*  IBM C Compiler            */

                                        /* define OPERATING SYSTEMS   */

#define   DOS    1                      /* DOS                        */
#define   SYS5   0                      /* System-V  O/S              */
#define   CPM    0                      /* CPM  O/S                   */
#define   UNIX   0                      /* UNIX O/S                   */

                                        /* define PRINTERS            */
#define   EPSON          1
#define   IBMPROPRINTER  0
#define   OKIDATA        0
```

```
/*****************************************************************************

KEYDEF.H : Contains keyboard scan codes.

*****************************************************************************/
#define  XF        0x100
#define  K_PGDN    81 | XF
#define  K_LEFT    75 | XF
#define  K_RIGHT   77 | XF
#define  K_UP      72 | XF
#define  K_CTRLH   8
#define  K_DOWN    80 | XF
#define  K_ESC     27
#define  K_SPACE   32
#define  K_DEL     83 | XF
#define  K_BACKSP  15
#define  K_RETURN  13
#define  K_HOME    71  | XF
#define  K_END     79  | XF
#define  K_PGUP    73  | XF
#define  K_CTRLZ   26
#define  BELL     7
#define  K_CEND   117 | XF
#define  K_ALTC   46 | XF
#define  K_ALTG   34 | XF
#define  K_ALTI   23 | XF
#define  K_ALTO   24 | XF
#define  K_ALTS   31 | XF
#define  K_ALTU   22 | XF
#define LF  (Char) 10
#define BLANK  (Char) 32
#define CR  (Char) 13
```

```
/*****************************************************************************

BIOSLIB.H : This file contains all BIOS definitions used in program.

*****************************************************************************/


#define  VIDEO      0x10          /* BIOS VIDEO INT 10          */

#define  KEYIN      0x7           /* DOS function kbd input w/o echo */

                                  /* VIDEO routine service numbers
                                     placed in AH register before a
                                     BIOS interrupt 10h.          */
#define   CUR_POS       2
#define   GET_CUR       3
#define   SCROLL_UP     6
#define   SCROLL_DN     7
#define   WRITE_CHAR   10
#define   WR_C_ATT      9          /* write char with attribute     */
#define   RE_C_ATT      8          /* read char with attribute      */
#define   SETVDPG       5          /* BIOS Video service 5
                                     sets active video page        */
#define   GETVDMOD     15          /* BIOS Video service 15
                                     gets current video information*/

#define  KBD_INT     0x16          /* BIOS keyboard interrupt number*/

#define KBD_STATUS   2             /* kbd status function number    */

#define LOBYTE   0x00FF            /* Bit mask for low byte         */

#define HIBYTE   0xFF00            /* bit mask for high byte        */
                                   /*  define video attributes      */


#define ATTR    65                 /* BYTE Attribute is RED background*/
                                   /* BLUE foreground.             */
```

```
/*****************************************************************/
/*                FILE NAME: MYASCII.NUM                         */
/*   Definitions of ascii number assigned for extra              */
/*   characters in the Turkish alphabet.                         */
/*****************************************************************/

#define ascii_s  128

#define ascii_S  129

#define ascii_o  130

#define ascii_O  131

#define ascii_u  132

#define ascii_U  133

#define ascii_g  134

#define ascii_G  135

#define ascii_c  136

#define ascii_C  137

#define ascii_i  138

#define ascii_I  139
```

```
/*              FILE NAME: EPSON.DAT                                    */

/*  Definitions of Standard printer control commands                    */
/*    for EPSON and Compatible Printers                                 */


char p_init[]=" 033@";          /* hardware reset                        */

char p_bold[]=" 033E";          /* emphasized mode                       */

char p_ds[] = " 033G";          /* double strike mode                    */

char p_ital[]= " 0334";         /* italicized mode                       */

char p_cmp[]= " 017";           /* condensed mode                        */

char p_exp[]= " 016";           /* expanded mode                         */

char p_ul[]=" 033-1";           /* underlined mode                       */

char p_cbold[]= " 033F";        /* cancel emphasized mode                */

char p_cds[]= " 033H";          /* cancel double strike  mode            */

char p_cital[]=" 0335";         /* cancel italic mode                    */

char p_ccmp[]=" 022";           /* cancel condensed mode                 */

char p_cexp[]=" 024";           /* cancel expanded mode                  */

char p_cul[]= { 27,'-',0 } ;

char p_grmode[4]= { (char) 27,'K',(char)8,(char)0 } ;
                                /* set to dot graphics mode              */

char p_default[]=" 033P";

#define FF 12

#define LF 10
```

```
/*****************************************************************/
/*        FILE NAME:   EXTRA.FNT                                 */
/* Here extra character font pattern for printers are given.     */
/*****************************************************************/

char c_pattern [ 8 ]  = { 0x1c,0x22,0x23,0x23,0x22,0x00,0x00,0x00 } ;

char CC_pattern [ 8 ] = { 0x7c,0x82,0x83,0x83,0x82,0x44,0x00,0x00 } ;

char i_pattern [ 8 ]  = { 0x00,0x00,0x22,0x3e,0x02,0x00,0x00,0x00 } ;

char CI_pattern [ 8 ] = { 0x00,0x00,0x42,0xfe,0x42,0x00,0x00,0x00 } ;

char o_pattern [ 8 ]  = { 0x1c,0xa2,0x22,0x22,0xa2,0x1c,0x00,0x00 } ;

char CO_pattern [ 8 ] = { 0x3c,0xc2,0x42,0x42,0xc2,0x3c,0x00,0x00 } ;

char s_pattern [ 8 ]  = { 0x12,0x2a,0x2b,0x2b,0x2a,0x04,0x00,0x00 } ;

char CS_pattern [ 8 ] = { 0x64,0x92,0x93,0x93,0x92,0x4c,0x00,0x00 } ;

char u_pattern [ 8 ]  = { 0x3c,0x82,0x02,0x82,0x3c,0x02,0x00,0x00 } ;

char CU_pattern [ 8 ] = { 0x7c,0x02,0x82,0x82,0x02,0x7c,0x00,0x00 } ;

char g_pattern [ 8 ]  = { 0x32,0xc9,0xc9,0xc9,0x7e,0x00,0x00,0x00 } ;

char CG_pattern [ 8 ] = { 0x3c,0xc2,0xc2,0xca,0xca,0x2c,0x00,0x00 } ;
```

APPENDIX D. EXAMPLE TEMPLATE AND PROGRAM OUTPUT DOCUMENT

```
* EXAMPLE TEMPLATE DEFINITION
* This is a comment line
# DURUM RAPORU
* row column width  editable   message
3    30      30    0       @          T.C             @
4    30      30    0       @         Dz.K.K           @
5    30      30    0       @  TCG.PIYALEPASA K.ligi   @
8    55      20    1       @Tarih: @
10   30      30    0       @HAZIRLIK DURUM RAPORU     @
11   29      30    0       @-----------------------@
12   5       20    1       @IDARI:@
15   5       20    0       @PERSONEL  :@
16   5       20    0       @-------------@
18   30      30    0       @ TAM KADRO           MEVCUT @
19   25      40    0       @   --------------    ---------------@
21   10      30    1       @SUBAY       : @
21   41      20    1       @@
22   10      30    1       @ASTSUBAY    : @
22   41      20    1       @@
23   10      30    1       @ERAT        : @
23   41      20    1       @@
26   5       20    0       @MATERYAL  :@
27   5       20    0       @------------@
29   30      30    0       @ TAM MEVCUT         IHTIYAC   @
30   25      35    0       @   ---------------  -------------@
32   10      30    1       @A. GIDA     :@
32   41      20    1       @@
34   10      30    1       @B. SU       :@
34   41      20    1       @@
36   10      30    1       @C. YAKIT    :@
36   41      20    1       @@
38   10      30    1       @C. YAG      :@
38   41      20    1       @@
45   5       30    0       @ DAGITIM    :@
46   5       30    0       @-------------@
49   50      20    0       @ KOMUTAN    @
# NEXT TEMPLATE
```

T.C
Dz.K.K
TCG. PIYALEPASA K.ligi

Tarih:

HAZIRLIK DURUM RAPORU

IDARI:

PERSONEL :

	TAM KADRO	MEVCUT
SUBAY	:	
ASTSUBAY	:	
ERAT	:	

MATERYAL :

	TAM MEVCUT	IHTIYAC
A. GIDA	:	
B. SU	:	
C. YAKIT	:	
D. YAG	:	

DAGITIM :

KOMUTAN

LIST OF REFERENCES

1. Turbo C *Compiler Reference Manual* Version 1.0

2. Parnas, David L., *On the Criteria to be used in Decomposing Systems into Modules,* Communications of the ACM, Volume 15, Number 12, December 1972.

3. Liskov, Barbara. *Modular Program Construction Using Abstractions* MIT laboratory for Computer Science, Computation structures group memo 184, September 1979.

4. Parnas, David L., *Information Distribution Aspects of Design Methodolgy,* Proceedings of 1971 IFIC Congress, Amsterdam, The Netherlands North Island Publishing Company, 1971.

5. IBM *Technical Reference Manual,* Volume I.

6. IBM *Technical Reference Manual,* Options and Adapters, Volume II, Chapter 4.

7. IBM *Proprinter Reference Manual*

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Deniz Kuvvetleri Komutanligi Bakanliklar Ankara, Turkey	4
4.	Deniz Harp Okulu Komutanligi Kutuphanesi Tuzla Istanbul, Turkey	1
5.	Department Chairman, 52Mz Computer Science Naval Postgraduate School Monterey, CA 93943	1
6.	Daniel Davis MBARI 160 Central Pacific Grove, CA 93950	1
7.	John Yurchak, 52Yu Computer Science Department Naval Postgraduate School Monterey, CA 93943	1
8.	Lt.JG Metin AKINCI Bakirkoy Akatlar Sok No 9/6 Istanbul, Turkey	2
9.	Metin G. Ozisik 427 West Str. Salinas, CA 93901	1

Thesis
A333805 Akinci
c.1 Document generator software design that supports Turkish alphabet.