PROJECT REPORT (VOL I)

GENERAL PURPOSE DATABASE IN 'C'

SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENT
FOR THE DEGREE
OF

MASTER OF COMPUTER APPLICATIONS

BY

BHAVDEEP
(3/90)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
THAPAR INSTITUTE OF ENGINEERING & TECHNOLOGY
PATIALA

(DEEMED UNIVERSITY)

MAY 1993

FAX : 0172-571492

PUNJAB COMMUNICATIONS LIMITED

(A Government Undertaking)

Ref. No. PCL/VT-28/1120/93

Date : 24-5-1993

PERFORMANCE CERTIFICATE

Sh Bhavdeep, Final Year student of M.C.A of Thapar Institute of Engineering & Technology, Patiala underwent four months Practical Training at PUNJAB COMMUNICATIONS LIMITED (A Govt.Undertaking) S.A.S. Nagar.

He was associated with Systems Department. He took keen interest in the work and has successfully completed his training in the software field.

Punjab Communications Limited S.A.S Nagar

(COL S S BEDI)
Addl.G.M. (Q.A)

CC:- Dr D P Goyal
Placement & Training Officer
T.I.E.T.,
Patiala

ACKNOWLEDGEMENTS

First of all my thanks are due to Mr. Chehal, Mr. Ramnik, Mr Prabhinder, Mr Ajay Bansl, Mr Davinder for learning advance feature of DOS and C from them.

I thank my friends Sukhjit, Deeraj, Rakesh, Mohit, Bedi & Gurinder for their nice & humorous company, for their guidance & for the conveyance services they offered to me.

All credit goes to project manager Mr. G.B.S. Bindra (Information Systems Manager, P.C.L.) He personally discusses with me the a lot regarding the project besides his tight shedule all over the day. He makes me aware about the problems I may, get in designing the various modules. He allowed me to work late in evenings and holidays and to use all resources & tools of department without any demurance.

A word of praise for Mr. Parveen for his all time available valuable friendly guidance.

Acknowledgements would not be completed without having thanks to all the employees of systems deptt. for their cooperation and help in many ways.

I would like to express sincere thanks to my teachers at the institute who were my source of inspiration through out my academic period in the institute.

Last but not least I owe my thanks to the organisation for providing me a conducive environment for the development of project.

Bhavdeep

(BHAVDEEP)

CONTENTS

VOLUME - 1

PUNJAB COMMUNICATIONS LIMITED, MOHALI.

INFRASTRUCTURE & OTHER RESOURCES

Punjab Communications Limited was incorporated in June, 1981 as a wholly owned subsidiary of Punjab State Electronics Development and Production Corporation Limited, Chandigarh .It has an independent board of directors.

PCL has been manufacturing Telecom Products like large Telephone Exchanges, Switching Equipment, Digital Radio and Multiplexers, and diversified into Information Management couple of years ago. PCL has three plants with covered area of 15700 square metres, located at the Electronics Park at Mohali, outside Chandigarh, and offices in various cities of India. The Electronics Park at Mohali is also to get a satellite hook-up shortly for Software Exports.

With a short span of twelve years, PCL has achieved a national status and has been recognised as a competent andreliable manufacturer of sophisticated professional grade telecommunication equipment . PCL is today in the forefront for producing indigeneously designed and developed products.

PRODUCT MIX

PCL has now established a full fledged capacity for offering to its customers a total system packages consisting of one or more of the following equipments :

-- Direct to line multiplexing equipment (DTL-MUX).
-- Group modemequipment, through Group filter and super through group filter, 3/4 way branching network.
-- 30 Channel Pulse code Modulated Multiplexing equipment (PCM-MUX).
-- Trans-Multiplexer (TRANS-MUX).
-- Voice frequency Telegraph Equipment (VFT).
-- Electronic Private Automatic Branch Exchange (EPABX).
-- 128P Rural Automatic Exchange (RAX).
-- Medium Size Digital Switching System.
-- Digital UHF Radio .
-- Digital Microwave radio for 8/34 Mbit capacity.
-- Subscriber Carrier System .

The various divisions of the company are:

1. Marketing
2. Production
3. Quality Assuarance
4. Information Systems Group

5. Materials Management

6. Finance and Administration

The Information Systems Group (ISG) of PCL manufactures packaged software, develops customised application software, takes up turnkey projects (including procurement of hardware etc.) and also offers facilities management (equip, staff, run, maintain and upgrade the computer centers/EDP cells of customers).

Another very relevant point to be noted is that, PCL besides offering the best financial remuneration to its employees, also creates and maintains the best possible working conditions. The facility, needless to say is completely centrally airconditioned and aesthetically landscaped. There is even a day care centre for the infants of the women employees.Thus a congenial work enviroment is created and maintained, leading to high productivity low manpower turnover.

ISG has, besides around 70 IBM PC compatibles, a twin-CPU 68030 based minicomputer with more than 40 terminals. ISG also has a 80486 based minicomputer with about 20 terminals. As for RISC based minis, ISG has one HP 9000/817 and is in the process of purchasing a SUN SPARC

Business Server.

ISG is also purchasing an IBM RS/6000 or AS/400 shortly. Further hardware can be purchased at very short notice , depending on the size and potential of the business ISG's expertise lies in the field of DOS based, UNIX based, networking and other communication software, because of the overwhelming presence of DOS and UNIX operating systems. As for RDBMS, ORACLE and SYBASE are already being used by ISG, with access to INGRES and UNIFY. INFORMIX can also be installed, if required. CASE Tools like Turbo Analyst are also available, while HP Soft Bench is being purchased. Open View and Motif enviroments are also available.

ISG is also developing packages for the Financial Institutions. Besides banking software, we are also developing other loans & advances accounting systems.

The Versatile Multiplexer (VMUX) is a hardware and software based product developed by ISG. The VMUX is a very critical component in the telecom sector. Similarly, ISG has developed AMCOS, a PC based Data Acquisition System useful typically in the medium scale process industries.

ANALYSIS

PROBLEM

DEFINITION

PROBLEM DEFINITION

Project assigned to me was to develop a

GENERAL PURPOSE DATABASE IN C

NEED FOR THIS PROJECT : Whenever any organisation approaches to PCL for developing a project similar to databse such as **PAYROLL, ACCOUNTANCY, BILLING SYSTEM** etc. Then PCL shifts to **DBASE, FoxBase, FoxPro** or other such package to develop that project for user. The projects developed using these packages run only under the environment of these packages. So PCL has to provide a legal copy of any of these packages with the project developed to user. The legal copy of any of these software causes thousands of rupees which is a big expenditure.

To avoid this expenditure I was asked to develop a " **GENERAL PURPOSE DATABASE IN C** ".

The database should be flexible so that in short time interwal it could be converted into a very specific product for a particular organisation. It means with few modifications it will work as a complete **PAYROLL SYSTEM, ACCOUNTANCY SYSTEM, BILLING SYSTEM** or like any other database related system

Being database designed in "C" so PCL will provide only the execution file of product to user and not the "C" or "TURBO-C" compiler. Thus the datbase will save the extra expenditure incurred in purchasing the legal copy of package in which it has to be developed otherwise.

EXISTING SYSTEM

EXISTING SYSTEM

I may say that there was no existing system as such. I studied some general database packages such as Dbase, FoxBase, FoxPro, and some sql facilities of R.D.B.M.S.

PROPOSED SYSTEM

THE PROPOSED SYSTEM SHOULD HAVE THE FOLLOWING FACILITIES :-

a.) **AUTOMATIC CREATION AND APPENDATION** : It could create the structure of record then records can be entered in it when_ever required. If dbf file already exists then starts appending it.

b.) **MULTI KEY FILE HANDLING** : Database should support the multiple indexing, ie. there can be multiple indexes associated with one file with no limits on the number of record in the dbf file. File can be retreived or appended according to dbf or indexed file.

c.) **EDITTING AND MODIFICATIONS** : Any record in dbf file can be altered or changed according to dbf file or indexed file.

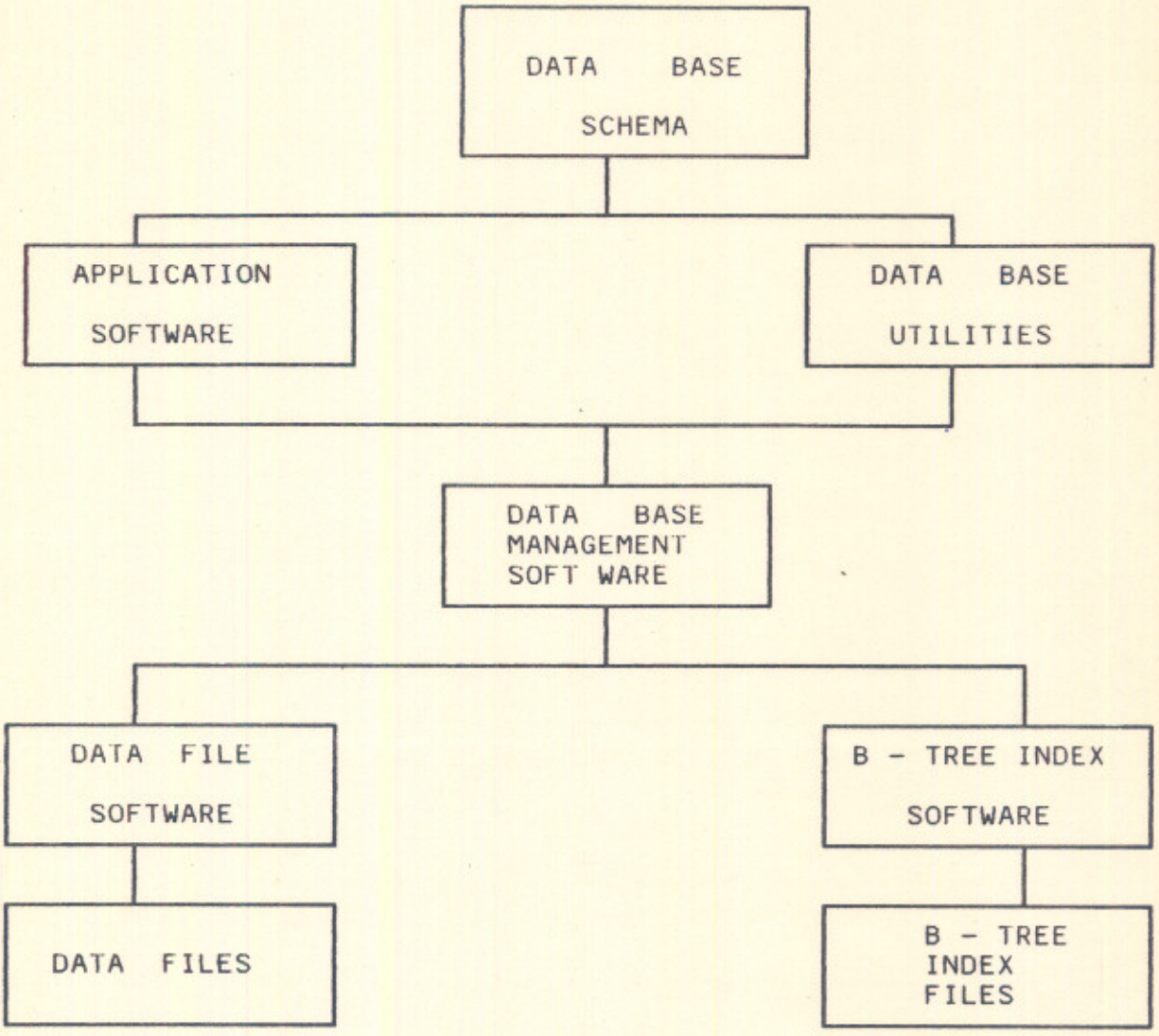
d.) **INSERTION OR DELETION** : The database would be capable of inserting deleting and searching the record either in dbf file or in index file.

e.) **SORTING** : The datbase should be such that it can sort out the dbf file of any size with minimum possible time.

f.) **UPDATION** : Database should have facility to seek the records and then replace it with the new value.

g.) **QUERY PROCESSOR** : A query processor should be provided to answer the queries asked by user.

DETAILED DESIGN



DATABASE SYSTEM ARCHITECTURE

FIG - 1

At the bottom of the fig 1 data and index files are shown. Data files are managed by a set of software functions that are dedicated to file management. Index files are managed by a different software function set. the two set of functions are unrelated, and the file themselves are unconnected. These functions are unseen by the programs that are written in support of application.

Above the data file and index file software is a library of functions whose purpose is to manage the database. Since the database is a collection of data and index files, the data base maintains the relationship that logically exists among many files that constiute the database. This set of functions represents the Data Manipulation Language (D M L).

Above the data base management software are two sets of programs application software and data base utility programs. the programs are concerned with files in the data base. it would seem that they deal primarily with data files, but they use index file as well

Above everything is the schema that describes the actual database.

PROCEDURE NAME	:	MAIN()
PART OF	:	CREATE, APPEND, EDIT, MODIFY, INDEX
CALLED BY	:	NONE
CALLS TO	:	SIGNATURE(), REC_HEADER()
PURPOSE	:	PROGRAM STARTS FROM THIS FUNCTION IT DECIDES THAT RETREIVALOF RECORDS SHOULD BE ACCORDING TO DBF FILES OR ACCORDING TO INDEX FILES
PARAMETERS	:	ARGC - COUNTS THE NUMBER OF ARGUMENTS PASSED TO PROGRAM AT RUN TIME ARGV - STORE THE ARGUMENTS PASSED AT RUN TIME
GLOBAL	:	NONE

PROCEDURE NAME : SIGNATURE()

PART OF : CREATE MODULE

CALLED BY : MAIN()

CALLS TO : NONE

PURPOSE : IT CREATES DBF FILE BUT IF DBF FILE IS ALREADY EXISTING THEN IT INSTRUCT TO APPEND IT

PARAMETERS : NONE

FUNCTION RETURNS : FILE HANDLER

GLOBALS : FILE_FLAG - IT IDENTIFY THAT FILE IS OPENED IN APPEND OR CREATE MODULE

****

PROCEDURE NAME : FILE_COPY()

PART OF : CREATE MODULE

CALLED BY : REC_HEADER()

CALLS TO : NONE

PURPOSE : IT WRITES THE FIELD DESCRIPTION SUCH AS (FIELD NAME,LENGTH,TYPE & DECIMAL WIDTH)

PARAMETERS : IT IS OF TYPE STRUCT VAR_REC & STORE FIELD DESCRIPTION

PARAMETERS : NUM_OF_FIELDS - GVES THE TOTAL NUMBER OF

PROCEDURE NAME : REC_DES_SIZE()

PART OF : CREATE MODULE

CALLED BY : REC_HEADER()

CALLS TO : NONE

PURPOSE : IT WRITES THE SIZE OF STRUCT VAR_REC ie. (FIELD DESCRIPTION) AND NUMBER OF FIELDS IN THE RECORD

PARAMETERS : NUM_OF_FIELDS - GVES THE TOTAL NUMBER OF FIELDS IN THE RECORD

GLOBALS : NONE

****

PROCEDURE NAME : REC_SIZE()

PART OF : CREATE MODULE

CALLED BY : REC_HEADER()

CALLS TO : NONE

PURPOSE : WRITE SIZE OF RECORD IN THE DBF FILE

PARAMETER : REC_SIZE - GIVES SIZE OF RECORD

GLOBAL : NONE

PROCEDURE NAME	:	FIELD_NAME()
PART OF	:	CREATE MODULE
CALLED BY	:	REC_HEADER()
PROCEDURES CALLED	:	FORMAT() , MOVE_LR()
PURPOSE	:	THIS FUNCTION READS THE FIELD NAME OF RECORDS. THE FIELD NAME MUST BE START WITH AN ALPHABET AND MAY CONTAIN ALPHANUMERIC CHARACTER OR HYPEN ('_').
PARAMETERS	:	VAR_NAME :-IT IS AN ARRAY OF 15 CHARACTERS TO STORE FIELD NAMES. V_COL_START , V_ROW_START , V_COL_END :-REPRESENTS THE SCREEN POSITION WHERE THE FIELD NAME IS TO BE ENTERED. IT EXITS FROM ENTERING FIELDS IN THE RECORD.
GLOBALS	:	NONE

PROCEDURE NAME	:	MOVE_LR()
PART OF	:	CREATE MODULE
CALLED BY	:	FIELD_NAME()
PROCEDURES CALLED	:	NONE
PURPOSE	:	THIS FUNCTION IS USED TO CHANGE THE NAME OF THE FIELD.
PARAMETERS	:	I - IT IDS THE SUBSCRIPT FOR ARRAY OF CHARACTERS TO STORE FIELD NAME V_COL_START ,V_ROW_START - ARE THE POSITIONS ON THE SCREEN WHERE FIELD NAME IS WRITTEN. I_DUPL - TELLS THAT FIELD NAME CONSISTS OF HOW MANY CHARACRES. VAR_NAME - IS AN ARRAY TO STORE THE NAME OF FIELD.
GLOBAL	:	NONE

PROCEDURE NAME	:	FIELD_TYPE()
PART OF	:	CREATE MODULE
CALLED BY	:	REC_HEADER()
PROCEDURES CALLED	:	FORMAT()
PURPOSE	:	THIS PROCEDURE IDENTIFY THE TYPE OF FIELD.
		ie. 'INTEGER','CHARACTER','FLOAT','DATE'.
PARAMETERS	:	VAR_TYPE - TO STORE THE TYPE OF FIELD.
		V_COL_START , V_COL_END , V_ROW_START
		:- SIGNAL THE POSITION ON SCREEN WHERE
		THE TYPE OF FIELD IS SELECTED.
GLOBALS	:	NONE

PROCEDURE NAME	:	FIELD_LENGTH()
PART OF	:	CREATE MODULE
CALLED BY	:	REC_HEADER()
PROCEDURES CALLED	:	FORMAT()
PURPOSE	:	IT'S FUNCTION IS TO GET THE FIELD WIDTH OF INTEGER AND CHARACTER TYPE FIELDS.
PARAMETERS	:	VAR_LEN - IT STORE THE FIELD WIDTH IN CHARACTERS . FIELD_LEN - IT STORE THE FIELD WIDTH IN INTEGER FORM . V_COL_START , V_COL_END , V_ROW_START ARE CURRENT POSITION ON THE SCREEN WHERE FIELD WIDTH IS ENTEREED .
GLOBAL	:	NONE

PROCEDURE NAME : FLOAT_LENGTH()

PART OF : CREATF MODULE

CALLED BY : REC_HEADER()

PROCEDURES CALLED : FORMAT()

PURPOSE : IT STORES THE DECIMAL WIDTH OF FLOAT TYPE VARIABLES .

PARAMETERS : FLOAT_LEN2 - TO STORE THE DECIMAL WIDTH IN FORM OF CHARACTERS .

FLT_LEN - TO STORE THE DECIMAL WIDTH IN FORM OF INTEGERS .

VAR_LEN - IS A CHECK TO MEASSURE THAT DECIMAL WIDTH DOES NOT EXCEED THE FIELD WIDTH .

V_ROW_START , V_COL_START , V_COL_END : - GIVES THE SCREEN POSITION FOR ENTERING THE DECINMAL LENGTH .

GLOBAL : NONE

PROCEDURE NAME	:	SCREEN()
PART OF	:	CREATE & APPEND MODULE
CALLED BY	:	REC_HEADER()
PROCEDURES CALLED	:	TYPES()
PURPOSE	:	THIS FUNCTION GENERATE THE SCREEN FOR ENTERING DATA IN THE RECORDS .
PARAMETERS	:	VAR_NAME - PRINTS THE FIELD NAME ON THE SCREEN VAR_TYPE - GENERATE THE SCREEN ACCORDING TO THE TYPE OF FIELD VAR_LEN - TELLS THE SIZE OF SCREEN TO BE GENERATED K - GIVES THE ROW POSITION TO ENTER THE DATA
GLOBAL	:	NONE

PROCEDURE NAME : TYPES()

PART OF : CREATE & APPEND MODULE

CALLED BY : SCREEN()

PROCEDURES CALLED : NONE

PURPOSE : TO GENERATE THAT SECTION OF SCREEN WHERE DATA IS TO BE ENTERED FOR DIFFERENT FIELDS

PARAMETERS : VAR_TYPE - IDENTIFIES THE TYPE OF FIELD
VAR_LEN - GIVES THE FIELD LEN
VAR_DEC - GIVES THE DECIMAL WIDTH

GLOBALS : NONE

****

PROCEDURE NAME : READ_VALUES()

PART OF : CREATE & APPEND MODULE

CALLED BY : REC_HEADER()

PROCEDURES CALLED : FLOAT_READ ,CHAR_READ() ,DATE()

PURPOSE : IT READS THE RECORD FROM THE SCREEN

PARAMETERS : VALUE_TYPE - GIVES THE TYPE OF FIELD
INC - GIVES THE CURRENT ROW POSITION
I,I_DUPL,VALUE_LEN,VALUE_DEC - ARE PASSED TO FUNCTION FLOAT_READ(), CHAR_READ()

GLOBALS : NEXT

PROCEDURE NAME	:	PAD_CHAR()
PART OF	:	CREAT , EDIT , APPEND ,MODIFY
CALLED BY	:	MOVEMENT()
PROCEDURES CALLED	:	NONE
PURPOSE	:	IT REMOVES THE SPACES AT THE END OF STRING AND INSERTS THE GIVEN CHARACTER AT THE BEGINNING OF THE STRING .
PARAMETERS	:	STR - IS THE STRING TO BE PADDED. CH - IS CHARACTER TO BE INSERTED FIELD_LENGTH - GIVES THE SIZE OF STRING
GLOBAL	:	NONE

PROCEDURE NAME	:	TRIM_CHAR()
PART OF	:	CREAT , EDIT, APPEND , MODIFY
CALLED BY	:	MOVE_MENT()
PROCEDURES CALLED	:	NONE
PURPOSE	:	IT DELETES THE CHARACTER (CH) BEGINNING FROM THE START OF STRING AND INSERT CHARACTER CH2 AT THE END OF THE STRING .
PARAMETERS	:	STR - STRING TO BE TRIMMED CH - CHARACTER TO BE DELETED CH2 - CHARACTER TO BE INSERTED LEN - GIVES THE LENGTH OF STRING.
GLOBAL	:	NONE

DETAILED DESIGN

PROCEDURE NAME	:	DATE()
PART OF	:	CREAT , APPEND
CALLED BY	:	READ_VALUES()
CALLS TO	:	YEAR() , MONTH() , DATE_OF_MONTH()
PURPOSE	:	GET DATE FROM THE SCREEN AND STORE ONLY THE CORRECT DATE.
PARMETERS	:	INC - TO POSITION THE CURSOR AT A PARTICULAR ROW ON THE SCREEN RECORD - STORE THE DATA ENTERED ON SCREEN
GLOBALS	:	NONE

****

PROCEDURE NAME	:	YEAR()
PART OF	:	CREAT , APPEND
CALLED BY	:	DATE
CALLS TO	:	NONE
PURPOSE	:	CHECK WEATHER THE ENTERED YEAR IS ORDINARY YEAR OR LEAP YEAR
PARMETERS	:	Y - GIVES THE YEAR TO BE CHECKED
GLOBALS	:	NONE

PROCEDURE NAME : DATE_OF_MONTH()

PART OF : CREAT , APPEND MODULE

CALLED BY : DATE

CALLS TO : NONE

PURPOSE : CHECK WEATHER THE ENTERED DATE IS VALID OR NOT

PARMETERS : FLAG_Y - TELLS WEATHER LEAP YEAR OR NOT

: M - REPRESENTS THE MONTH OF YEAR

: D - REPRESENTS THE DATE OF MONTH

****

PROCEDURE NAME : FORMAT()

PART OF : CREAT MODULE

CALLED BY : FIELD_NAME(), FIELD_TYPE(), FIELD_LENGTH() FLOAT_LENGTH()

CALLS TO : NONE

PURPOSE : PRINTS SPACES AT SCREEN AT A SPECIFIED LOCATION

PARAMETERS : V_COL_START - REPRESENTS THE STARTING POSITION OF COLUMN

V_ROW_START - GIVES STARTING ROW POSITION

V_COL_END - ENDING COLUMN NUMBER

GLOBALS : NONE

DETAILED DESIGN

PROCEDURE NAME	:	FLOAT_READ()
PART OF	:	CREAT , APPEND MODULE
CALLED BY	:	READ_VALUES()
CALLS TO	:	NONE
PURPOSE	:	IT READS FLOAT AND INTEGER TYPE OF DATA
PARAMETERS	:	RECORD - STORED THE VALUE OF FIELD ENTERED BY USER
		FIELD_TYPE - TO IDENTIFY THAT FIELD TYPE IS INTEGER OR FLOAT
		I - CURRENT CURSOR POSITION ON THE SCREEN
		I_DUPL - NO OF DIGITS ENTERED OF FLOAT AND INTEGER TYPE DATA
		LEN - REPRESENTS THE FIELD LENGTH
		DEC - REPRESENTS THE DECIMAL LENGTH
		NUM - GIVES THE MAXIMUM POSSIBLE NON DECIMAL DIGITS
		COL - POSITION CURSOR TO A SPECIFIC COLUMN
		INC - POSITION CURSOR AT A PARTICULAR ROW
		C - PASSES CHARACTER TO CALLING FUNCTION
GLOBALS	:	NEXT - GIVES STANDARED COLUMN POSITION
	:	ROW - GIVES STANDARED ROW POSITION

PROCEDURE NAME	:	CHAR_READ()
PART OF	:	CREAT ,APPEND MODULE
CALLED BY	:	READ_VALUES
CALLS TO	:	NONE
PURPOSE	:	READ CHARACTER TYPE OF DATA
PARMETERS	:	RECORD - STORED THE VALUE OF FIELD ENTERED BY USER
		FIELD_TYPE - TO IDENTIFY THAT FIELD TYPE IS INTEGER OR FLOAT
		I - CURRENT CURSOR POSITION ON THE SCREEN
		I_DUPL - NO OF DIGITS ENTERED OF FLOAT AND INTEGER TYPE DATA
		LEN - REPRESENTS THE FIELD LENGTH
		DEC - REPRESENTS THE DECIMAL LENGTH
		NUM - GIVES THE MAXIMUM POSSIBLE NON DECIMAL DIGITS
		COL - POSITION CURSOR TO A SPECIFIC COLUMN
		INC - POSITION CURSOR AT A PARTICULAR ROW
		C - PASSES CHARACTER TO CALLING FUNCTION
GLOBALS	:	NONE

DETAILED DESIGN

PROCEDURE NAME	:	MODIFY()
PART OF	:	MODIFY MODULE
CALLED BY	:	RETRIEVE() , REC_HEADER()
CALLS TO	:	NONE
PURPOSE	:	IT DISPLAYS THE RECORD ENTERED BY USER IN DBF FILE
PARMETERS	:	FIELD_LEN - LENGTH OF THE FIELD DEFINED BY USER
		FLT_LEN - DECIMAL LENGTH OF FLOAT TYPE DATA
		VAR_TYP - TYPE OF THE VARIABLE (FIELD)
		RECORD - STORE THE VALUE OF FIELD TO BE DISPLAYED ON THE SCREEN
		INC - POSITION THE CURSOR AT A PARTICULAR
GLOBALS	:	"ROW" - GIVES THE STANDARED ROW POSITION
		"NEXT" - GIVES THE STANDARED ROW POSITION

PROCEDURE NAME	:	MOVE_MENT()
PART OF	:	EDIT , MODIFY MODULES
CALLED BY	:	REC_HEADER()
CALLS TO	:	READ_VALUES() , TRIM_CHAR() , PAD_CHAR()
PURPOSE	:	1. TO MOVE THE CURSOR IN A FIELD. 2. TO MOVE CURSOR FROM ONE FIELD TO OTHER IN A RECORD 3. TO MOVE TO PRECEDING RECORD, PROCEDING RECORD OR TO LAST RECORD.
PARMETERS	:	RECORD - STORED THE VALUE OF FIELD ENTERED BY USER VAR_TYPE - TO IDENTIFY THAT FIELD TYPE IS INTEGER OR FLOAT I - CURRENT CURSOR POSITION ON THE SCREEN I_DUPL - NO OF DIGITS ENTERED OF FLOAT AND INTEGER TYPE DATA FIELD_LEN - REPRESENTS THE FIELD LENGTH FLT_LEN - REPRESENTS THE DECIMAL LENGTH INC - POSITION CURSOR AT A PARTICULAR ROW C - PASSES CHARACTER TO CALLING FUNCTION
GLOBALS	:	"ROW" - GIVES THE STANDARED ROW POSITION "NEXT" - GIVES THE STANDARED ROW POSITION

PROCEDURE NAME : IS_RECORD_EMPTY()

PART OF : CREAT, APPEND MODULES

CALLED BY : FILE_COPY2()

CALLS TO : NONE

PURPOSE : IT TELLS THAT RECORD IS EMPTY OR NOT

PARAMETERS : RECORD - CONTAIN THE SIZE OF RECORD

BUF_SIZE - GIVES SIZE OF RECORD

VARIABLE RETURN : RETURN 'Y' IF EMPTY RECORD ELSE 'N'

GLOBALS : NONE

****

PROCEDURE NAME : KEY_NAME()

PART OF : INDEX, APPEND MODULES

CALLED BY : INDEX_CHECK()

CALLS TO : NONE

PURPOSE : INITIALIZE THE FIELD NAME TO SPACES

PARAMETERS : NAME - ARRAY OF CHARACTERS TO STORE FIELD NAME

GLOBALS : NONE

DETAILED DESIGN

PROCEDURE NAME	:	INDEX_CHECK()
PART OF	:	INDEX, APPEND MODULES
CALLED BY	:	REC_HEADER()
CALLS TO	:	KEY_NAME()
PURPOSE	:	IT LINKS INDEX FILE WITH DBF FILE TO PERFORM EDITTING
PARAMETERS	:	HAVE ARGUMENTS ENTERED AT THE RUN TIME INDEX_FLAG - IDENTIFY THAT INDEX FILE ON KEY FIELD EXIST OR NOT T_F_L - GIVES RECORD SIZE D - GIVES TOTAL NUMBER OF BYTES IN INDEX FILE
GLOBALS	:	T_F_L2 : GIVES RECORD SIZE OF DBF OR INDEX FILE DEPENDING UPON THAT RETREIVAL SHOULD BE PERFORMED ON INDEX OR DBF FILE
	:	FIL_POS2 - GIVES STARTING BYTE NUMBER OF FIRST RECORD OF FILE ON WHICH RETREIVAL IS TO BE PERFORMED

PROCEDURE NAME : INDEX_CHECK2()

PART OF : INDEX, APPEND MODULES

CALLED BY : REC_HEADER()

CALLS TO : NONE

PURPOSE : IT READS ADDRESSES OF RECORDS OF DBF FILES FROM ACTIVE INDEX FILE

PARAMETERS : INDEX_FLAG - INFORMS THAT INDEX FILE IS ACTIVE OR NOT

GLOBALS : F_DESC2 - FILE HANDLER FOR INDEX FILE

: F_DESC - FILE HANDLER FOR DBF FILE

****

PROCEDURE NAME : TEXT_SETTER()

PART OF : CREAT MODULE

CALLED BY : FIELD_NAME(), FIELD_TYPE(), FIELD_LENGTH() FLOAT_LENGTH()

CALLS TO : NONE

PURPOSE : SETS SCREEN IN REVERSE VIDEO MODE

PARAMETERS : NONE

GLOBALS : NONE

DETAILED DESIGN

PROCEDURE NAME	:	RECORD_RETREIVAL()
PART OF	:	MODIFY MODULE
CALLED BY	:	REC_HEADER()
CALLS TO	:	MODIFY()
PURPOSE	:	PRINT RECORDS ON SCREEN
PARAMETERS	:	RECORD_S, RECORD_C : - STORES RECORD OF DBF FILE
		TOT_FIELDS_LEN GIVES RECORD SIZE
		T_F_L - REPRESENTS RECORD SIZE
		J - TOTAL NUMBER OF FIELDS
		CHECK - STATUS FOR SHOWING RECORDS
		FIELD_LEN, FLT_LEN, VAR_BUF ARE PASSES TO FUNCTION MODIFY()
GLOBALS	:	F_DESC - GIVES FILE HANDLER OF DBF FILE

While explaining the detailed design of index file a different approach was applied . The global variables used in a procedure are not explained in the description of that procedures instead they are explained at the end of description of all the procedures

This approach is applied because of large number of global variables and very few parameters in the index file. While the situation is exactly reverse in other files.

This approach is effective because it reduces redundancy and space storage that would otherwise occur in explianing the same global variables in diferent procedures.

PROCEDURE NAME : MAIN()

PART OF : INDEX MODULE

CALLED BY : NONE

PROCEDURES CALLED : CREAT(), INSERT(), WRITE_HEADING(), RETREIVE(), FREE_NODE()

PURPOSE : IT IS THE MAIN FUNCTION OF INDEX FILE
IT CREATS INDEX FILE ON DBF FILE

PARAMETERS : ARGC - COUNTS NUMBER OF ARGUMENTS PASSED AT RUN TIME
ARGV - CONTAINS THE PARAMETER PASSED AT RUN TIME

GLOBALS : FD, FD2, KEY_NAME, FLAG2, ROOT, CUR

****

PROCEDURE NAME : CREAT()

PART OF : INDEX MODULE

CALLED BY : MAIN

CALLS TO : KEY_BUFF(), READ_KEY_FIELD()

PURPOSE : READ FIRST RECORD FROM DBF FILE AND STORE IN ROOT NODE OF TREE

PARAMETERS : NONE

GLOBALS : ROOT, KEY_LEN

PROCEDURE NAME : KEY_BUFF()

PART OF : INDEX MODULE

CALLED BY : CREAT()

CALLS TO : REC_COUNT(), APPEND()

PURPOSE : POSITION THE CURSOR TO THE STARTING LOCATION OF FIRST RECORD IN FILE

PARAMETERS : NONE

GLOBALS : REC_ADDR, REC_START_LOC

****

PROCEDURE NAME : APPEND()

PART OF : INDEX MODULE

CALLED BY : KEY_BUFF()

CALLS TO : NONE

PURPOSE : IDENTIFY THE KEY FIELD IN DBF FILE AND RETURN THE STARTING LOCATION OF FIRST RECORD

PARAMETERS : RECORD - CONTAINS THE FIELD DESCRIPTION OF ALL THE FIELDS
FIELD_LEN - CONTAIN FIELD WIDTH OF ALL THE FIELDS

GLOBALS : KEY_LEN, TOT_FIELDS_LEN, NO_OF_FIELDS, "MAX_VAR_LEN"

PROCEDURE NAME : INSERT()

PART OF : INDEX MODULE

CALLED BY : MAIN()

CALLS TO : READ_KEY_FIELD()

PURPOSE : IT CREATES BINARY TREE OF KEY FIELDS READ FROM DBF FILE

PARAMETERS : NONE

GLOBALS : CUR, ROOT, "FIELD_SPECIF_SIZE", "WHILE", "new_cur"

****

PROCEDURE NAME : WRITE_HEADING()

PART OF : INDEX MODULE

CALLED BY : MAIN()

CALLS TO : NONE

PURPOSE : WRITE KEY_FIELD AND FIELD_WIDTH AT THE TOP INDEX FILE

PARAMETERS : NONE

GLOBALS : FD2, KEY_NAME, KEY_LEN

PROCEDURE NAME : RETREIVE()

PART OF : INDEX MODULE

CALLED BY : MAIN() & ITSELF

CALLS TO : ITSELF ie. (RETREIVE())

PURPOSE : TO WRITE DOWN THE BINARY TREE ON INDEX FILE IN SORTED ORDER

PARAMETERS : CUR - CONTAINS RECORD TO BE WRITTEN ON INDEX FILE

GLOBALS : FD2,REC_COUNTER, KEY_LEN

****

PROCEDURE NAME : FREE_NODE()

PART OF : INDEX MODULE

CALLED BY : MAIN(), ITSELF

CALLS TO : ITSELF ie. (FREE_NODE())

PURPOSE : IT FREES THE NODE OF BINARY TREE

PARAMETERS : CUR - CONTAINS RECORD ie.(NODES OF BINARY TREE)

GLOBALS : NONE

DESCRIPTION OF GLOBAL VARIABLES

FD	:	REPRESENTS FILE HANDLER OF DBF FILE
FD2	:	REPRESENTS FILE HANDLER OF INDEX FILE
KEY_NAME	:	STORE FIELD NAME ON WHICH INDEXING IS TO BE DONE
FLAG2	:	CHECKS THAT INDEX FIELD NAME IS VALID OR NOT.
ROOT	:	REPRESENTS ROOT NODE OF BINARY TREE
CUR	:	REPRESENTS CURRENTLY ACTIVE NODE OF BINARY TREE
REC_COUNTER	:	GIVES NUMBER OF RECORDS RETREIVED FROM DBF FILE
NO_OF_RECORDS	:	GIVES TOTAL NUMBER OF RECORDS IN DBF FILE
REC_START_LOC	:	GIVES NUMBER OF BYTES BEFORE FIRST RECORD IN DBF FILE
REC_ADDR	:	GIVES STARTING LOCATION OF FIRST RECORD
KEY_LEN	:	CONTAINS THE LENGTH OF KEY FIELD
TOT_FIELDS_LEN	:	REPRESENTS RECORD SIZE
"MAX_VAR_LEN"	:	IS A DEFINE CONSTANT WHOSE VALUE IS 15
"FIELD_SPECIF_SIZE"	:	IS A DEFINE CONSTANT FOR REPRESENTING FIELD DESCRIPTION SIZE ITS VALUE IS 24

```
"new_cur"                 :  is defined as "(strcmp( new->c,cur->c))
"cur_pre"                 :  is defined as "(strcmp(cur->c,pre->c))
"WHILE"                   :  is defined as " while (((( new_cur  >=
0  ) && ( cur->rch != NULL )) || (( new_cur <= 0 ) && (  cur->lch
!= NULL )))
```

I/O Formats

FIELD NAME	FIELD TYPE	LEN	DEC
NAME	character	12	
ROLL_NO	integer	04	
MARKS	float	05	02
CLASS	character	15	
ORGANISATION	character	15	
DATE_OF_JOINING	date		
AGE	character	18	
DESIGNATION	character	15	
SALARY	float	07	02
MARITAL_STATUS	character	01	
HOBBIES	character	9	

FIELD NAME	FIELD TYPE	LEN	DEC
NAME	character	12	
ROLL_NO	integer	04	
MARKS	float	05	02
CLASS	character	15	
ORGANISATION	character	15	
DATE_OF_JOINING	date		
AGE	character	18	
DESIGNATION	character	15	
SALARY	float	07	02
MARITAL STATUS	character	01	
HOBBIES	character	09	
SEX	character	1	

NAME :
ROLL NO :
MARKS :
CLASS :
ORGANISATION :
DATE_OF_JOINING: / /
AGE :
DESIGNATION :
SALARY :
MARITAL_STATUS :
HOBBIES :
SEX :

```
NAME              :     BHAVDEEP
ROLL_NO           :     0003
MARKS             :     800.1
CLASS             :     MCA3rdYear
ORGANISATION      :     TIET
DATE_OF_JOINING:        01/09/90
AGE               :     22
DESIGNATION       :     STUDENT
SALARY            :     00000.0
MARITAL_STATUS    :     U
HOBBIES           :     CRICKET
SEX               :     M
```

IMPLEMENTATION

PSEUDOCODE

In pseudo code like detailed design all **variables** are shown in capital letters all though they are used in small letters in actual code and the **define constants** are shown surrounded by aphostrophies (").

PROCESSING LOGIC FOR FIELD_NAME()

```
INITIALIZE THE FIELD NAME OF 15 CHARACTERS TO SPACES (' ').
CALL FUNCTION FORMAT TO GENERATE SCREEN IN REVERSE VIDEO MODE .
GET FIRST CHARACTER OF FIELD NAME
 IF ( FIRST CHARACTER IDS NOT ENTER KEY )
    THEN
       DO WHILE ( NOT ENTER KEY AND LENGTH OF FIELD NAME <= 15 )
          BEGIN
              GET CHARACTER OF FIELD NAME
               IF ( CHARACTER IS ALPHANUMERIC OR HYPEN ('_')
                  THEN
                      STORE IT
               ELSE IF (KEY PRESSED IS L-ARROW, R-ARROW OR DEL )
                     THEN
                         CALL TO FUNCTION MOVE_LR()
               END IF
         END DO
 END IF.
```

PROCESSING LOGIC FOR MOVE_LR()

```
DO
  GET KEY
  IF ( PRESSED KEY IS L-ARROW )
      IF ( CURSOR IS NOT AT THE STARTING POSITION )
          SHIFT THE CURSOR ONE POSITION LEFT .
  ENDIF
  IF ( PRESSED KEY IS R-ARROW )
       IF ( CURSOR IS NOT AT THE END POSITION )
         SHIFT THE CURSOR ONE POSITION RIGHT .
  END IF
  ELSE IF ( KEY PRESSED IS DEL KEY )
       IF ( AT CURRENT POSITION THERE IS NOT A SPACE CHARACTER )
              ERASE THE CHARACTER AT CURRENT POSITION.
              SHIFT ALL THE CHARACTERS OF RIGHT SIDE OF CURSOR TO
              ONE POSITION LEFT.
       END IF
  END IF
  ELSE IF ( KEY PRESSED IS ALPHANUMERIC OR HYPEN ('_') )
      STORE IT AT THE CURRENT POSITION .
  END IF
END DO WHILE ( PRESSED CHARACTER MORE THAN ONE ASCII VALUE )
```

PROCESSING LOGIC FOR FIELD_TYPE()

```
INITIALIZE  AN ARRAY OF 4 STRINGS (ie. VARIABLE FIELD_TYPE  )  TO
FOUR DIFFIRENT TYPES ie. "INTEGER ,CHAR, FLOAT AND DATE " .
GET KEY
DO WHILE ( NOT ENTER KEY )
   BEGIN
      IF ( SPACE KEY IS PRESSED )
         I = ( ( I++ ) % 4 )
      DISPLAY  THE  TYPE STORED AT THE I  SUBSCRIPT  OF  VARIABLE
      FIELD TYPE
      ELSE NO ACTION
END DO
STORE THE FIELD_TYPE[I] IN THE VARIABLE VAR_TYPE.
```

PROCESSING LOGIC FOR FIELD_LENGTH()

```
DO
   BEGIN
        GET KEY
        IF ISDIGIT (KEY)
             IF ( CURSOR IS AT THE START POSITION )
                  STORE IT AND MOVE ONE PLACE RIGHT .
             ELSE IF ( CURSOR IS AT THE END POSITION )
                  STORE IT AND EXIT FROM FUNCTION .
             END IF
        ELSE IF ( START POSITION IS NON EMPTY AND KEY PRESSED KEY
                                              IS R-ARROW (' -> ') )
           THEN
               SHIFT ONE POSITION RIGHT .
        ELSE  IF ( CURSOR IS AT END POSITION AND PRESSED  KEY  IS
                                                 L-ARROW (' <- ') )
               SHIFT ONE POSITION LEFT .
        END IF.
END DO WHILE ( KEY PRESSED IS NOT REURN KEY )
```

PROCESSING LOGIC FOR FLOAT_LENGTH()

```
DO
     FIELD_LENGTH()
WHILE  ( DECIMAL WIDTH > FIELD WIDTH ) .
```

PROCESSING LOGIC FOR DATE()

```
INITIALIZE COUNT2 TO ZERO
DO WHILE ( COUNT2 < 7 )
   BEGIN
       INITIALIZE COUNT TO ZERO
       DO WHILE ( COUNT < 2 )
          GET KEY
          IF ( KEY PRESSED S A DIGIT KEY )
              BEGIN
                   STORE IT IN VALUE COUNT
                   INCREMENT COUNT BY ONE
              END
          END IF
       END DO
       IF ( COUNT2 != 6 )
          BEHIN
              INCREMENT COUNT2 BY ONE
              STORE BACK SLASH "/" IN VALUE [ COUNT2 ]
          END
       ELSE INCREMENT COUNT2 BY ONE
       END IF
END DO
RUN FUNCTION YEAR
RUN FUNCTION MONTH
IF ( MONTH IS CORRECTLY ENTERED IN THE DATE )
   RUN FUNCTION DATE_OF_MONTH()
```

PROCESSING LOGIC FOR YEAR()

```
IF ( (YEAR MOD 4) == 0 )
   RETURN STATUS LEAP YEAR
ELSE RETURN STATUS ORDINARY YEAR
```

PROCESSING LOGIC FOR MONTH()

```
IF ( VALUE OF MONTH IS BETWEEN ZERO ( 0 ) AND THIRTEEN ( 13 ) )
      RETURN TO FUNCTION DATE() THAT MONTH IS VALID
ELSE  RETURN THAT MONTH IS INVALID
```

PROCESSING LOGIC FOR DATE_OF_MONTH()

```
IF ( VALUE OF MONTH LIES BETWEEN ONE AND SEVEN )
   BEGIN
       IF ( MONTH IS OTHER THAN FEBURARY )
           IF ( ( DATE - ( (VLUE OF MONTH) MOD 2 )) <= 30 )
               RETURN VALID DATE
       ELSE IF ( (MONTH IS FEBURARY) AND
                  ( ( (YEAR STATUS) + DATE) <= 29 ) )
               RETURN VALID DATE
       ELSE RETURN INVALID DATE
   END
ELSE IF ( (VALUE OF MONTH LIES BETWEEN EIGHT AND TWELVE ) AND
             ( (DATE + ( (VALUE OF MONTH) MOD 2 ) ) <= 31 ) )
         RETURN VALID DATE
ELSE
      RETURN INVALID DATE
```

PROCESSING LOGIC FOR FLOAT_READ()

```
IF ( CURRENT FIELD IS NON EMPTY )
    ASSIGN THE ADDRESS OF CURRENT_FIELD TO VARIABLE FLOAT2
DO WHILE( KEY PRESSED IS NOT IN(ENTER KEY , '.' , KEY HAVING TWO
           ASCII  VALUES) AND ( ( NUMBER OF DIGITS ENTERED  )  <
                                  FIELD LENGTH ) )
   BEGIN
       PRINT CHARACTER C ON SCREEN
       IF ( IS DIGIT CHARACTER C )
            STORE IT IN FLOAT2
   END
END DO
IF ( DIGITS WRITTEN ARE LESS THAN FIELD LENGTH )
   BEGIN
      FEED  CHARACTER ZERO (O) AT THE START OF NUMBER  UNTIL  ITS
      SIZE BECOMES EQUAL TO FIELD LENGTH.
      STORE IT AS A FIELD VALUE AND PRINT ON THE SCREEN
   END
ELSE ASSIGN THE VALUE OF VARIABLE TO FIELD.
IF ( MEMORY IS ALLOCATED TO VARIABLE FLOAT2 )
     FREE IT
IF ( FIELD TYPE IS FLOAT )
   BEGIN
       DO WHILE ( ( NUMBER OF DECIMAL DIGITS ENTERED ) <  DECIMAL
                                                          LENGTH )
          STORE DECIMAL DIGITS ON THE RECORD
       END DO
   END
END IF
```

PROCESSING LOGIC FOR MAIN()

```
RUN FUNCTION SIGNATURE()
IF ( FILE IS ALREADY CREATED )
   THEN CHECK WEATHER TO RETRIEVE INDEX FILE OR DBF FILE
RUN FUNCTION REC_HEADER()
```

PROCESSING LOGIC FOR REC_DES_SIZE()

```
IF ( REQUEST IS TO CREATE DBF FILE )
   BEGIN
       OPEN THE DBF FILE
       WRITE SIZE OF FIELD DESCRIPTION
       WRITE NUMBER OF FIELDS IN THE RECORD
       CLOSE DBF FILE
   END
END IF
```

****

PROCESSING LOGIC FOR REC_SIZE()

```
IF ( REQUEST IS TO CREATE DBF FILE )
   BEGIN
       CONVERT RECORD SIZE INTO INTEGERS
       OPEN DBF FILE IN TEXT MODE
       WRITE RECORD SIZE
       CLOSE DBF FILE
   END
END IF
OPEN FILE IN TEXT MODE
```

****

PROCESSING LOGIC FOR FILE_COPY()

```
WRITE FILE DESCRIPTION IN DBF FILE
IF ( UNABLE TO WRITE )
THEN PRINT ERROR MESSAGE
```

PROCESSING LOGIC FOR FILE_COPY2()

```
IF ( REQUEST IS TO WRITE RECORD AT THE END )
    GO TO THE END OF FILE
IF ( RECORD IS NO EMPTY )
   WRITE IT ON THE FILE
ELSE DON'T WRITE
FREE THE MEMORY ALLOCATED TO RECORD
```

****

PROCESSING LOGIC FOR IS_RECORD_EMPTY()

```
DO WHILE ( COUNTER LESS THAN SIZE OF RECORD AND
           RECORD[COUNTER] != ( SPACE (' ') OR ZERO ('0') )
    BEGIN
        INCREMENT COUNTER BY ONE
    END
END DO
IF ( COUNTER EQUAL TO RECORD SIZE )
     RETURN RECORD IS EMPTY
ELSE RETURN RECORD IS NON EMPTY
```

****

PROCESSING LOGIC FOR RECORD_RETREIVAL()

```
IF ( KEY PRESSED IN FUNCTION REC_HEADER() HAS ONE ASCII VALUE )
     READ RECORD FROM DBF FILE
IF  ( RECORD IS NOT READ CORRECTLY OR KEY PRESSED HAS  TWO  ASCII
                                                       VALUES )
    SET CHECK TO NINE
ELSE MOVE TO THE STARTING POSITION OF NEWLY READ RECORD
DO WHILE ( COUNTER LESS THAN NUMBER OF FIELDS IN THE RECORD )
   BEGIN
       RUN MODIFY
       INCREMENT POINTER IN RECORD TO CURRENTLY ACTIVE FIELD
   END
END DO
```

PROCESSING LOGIC FOR FORMAT()

```
COUNT  THE NUMBER OF SPACES BETWEEN STARTING COLUMN POSITION  AND
ENDIN COLUMN POSITION.
STORE THEM IN VARIABLE J
DO WHILE ( J  > 0 )
   PRINT SPACE ON SCREEN
   DECREMENT J BY ONE
END DO
MOVE THE CURSOR TO STARTING COLUMN POSITION
```

****

PROCESSING LOGIC FOR INDEX_CHECK()

```
IF ( RETREIVAL OF RECORDS ACCORDING TO DBF FILE )
   BEGIN
       ASSIGN FIELD HANDLER OF DBF FILE TO VARIABLE F_DESC2
       ASSIGN RECORD SIZE TO VARIABLE T_F_L2
       ASSIGN  STARTING LOCATION OF FIRST RECORD IN DBF  FILE  TO
       VARIABLE FIL_POS2
   END
ELSE
   BEGIN
      ASSIGN FILE HANDLER OF INDEX FILE TO VARIABLE F_DESC2
      IF ( ENTERED KEY FIELD IS CORRECT )
        BEGIN
            SET  RECORD SIZE OF INDEX FILE TO SUM OF( LENGTH  OF
                      KEY FIELD ON WHICH INDEXING IS TO BE  DONE
                      + SIZE OF (INT) + SIZE OF (LONG) )
            STORE RECORD SIZE IN VARIABLE T_F_L2
        END
      ELSE PRINT ERROR
      END IF
  END
END IF
```

****

PROCESSING LOGIC FOR INDEX_CHECK2()

```
IF ( INDEX FILE IS OPENED )
   BEGIN
       READ THE ADDRESS OF RECORD IN DBF FILE
       GO TO THAT POSITION IN THE DBF FILE
   END
END IF
```

PROCESSING LOGIC FOR KEY_NAME()

INITIALIZE THE VARIABLE NAME AN ARRAY OF CHARACTERS TO SPACES
STORE NULL CHARACTER AT THE END

PROCESSING LOGIC FOR SCREEN

```
DISPLAY THE FIELD NAME ON SCREEN AT SPECIFIED POSITITON.
CALLS TO FUNCTION TYPES() .
```

PROCESSING LOGIC FOR TYPES

```
IF ( FIELD IS OF TYPE "DATE" )
    GENERATE SCREEN OF DATE FORMAT.
ELSE IF ( FIELD IS OF TYPE "FLOAT" )
    BEGIN
          GENERATE SCREEN FOR OF "FLOAT" FORMAT
          WITH DECIMAL AT APPROPRIATE PLACE .
    END
ELSE GENERATE SCREEN FOR "CHARACTER" OR "INTEGER" TYPE
```

PROCESSING LOGIC FOR READ_VALUES

```
GET THE VALUE TYPE FROM THE PARAMETER VALUE_TYPE .
IF ( FIELD TYPE IS "DATE" )
    RUN FUNCTION DATE()
ELSE IF ( FIELD TYPE IS "FLOAT" )
     BEGIN
           SET FIELD_LEN = FIELD_WIDTH - DECIMAL_LEN
           RUN FUNCTION FLOAT_READ()
     END
ELSE IF ( FIELD TYPE IS "INTEGER" )
     RUN FUNCTION FLOAT_READ()
ELSE
    RUN FUNCTION CHAR_READ()
```

PROCESSING LOGIC FOR PAD_CHAR

```
GET CHARACTER TO BE INSERTED FROM PARAMETER ( CH )
INITIALIZE COUNTER TO ZERO (O)
GO TO THE FIRST SPACE CHARACTER IN THE GIVEN WORD
IF ( NO SUCH SPACE CHARACTER )
END IF
ELSE
   BEGIN
          DO WHILE ( (WORD SIZE WITHOUT SPACES) < MAX_SIZE )
             BEGIN
                IF ( STRING HAS SPACES AT CURRENT POSITION )
                   INCREMENT COUNTER BY ONE ( 1 )
             END
          END DO.
          CREATE A NEW STRING RESULT OF SAME SIZE
          INITIALIZE I = O
          DO WHILE ( I < COUNTER )
             BEGIN
                STORE CH2 IN STRING "RESULT"
                INCREMENT I BY ONE
             END
          END DO
          INITIALIZE COUNTER TO ZERO
          DO WHILE ( I < STRING SIZE )
             BEGIN
                RESULT[I] = STR[COUNTER]
                INCREMENT I AND COUNTER BY ONE
             END
          END DO
          STORE THE CONTENTS OF RESULT IN STR.
   END
```

****

PROCESSING LOGIC FOR TRIM_CHAR

```
GET  STRING  CHARACTERS CH & CH2 TO BE INSERTED AND  DELETED  AND
LENGTH OF STRING FROM THE PARAMETER .
INITIALIZE COUNTER1 & COUNTER2 TO ZERO
DO WHILE ( STRING CONTAIN CHARACTER CH )
   INCREMENT STRING PPOSITION BY ONE
END DO
DO WHILE ( COUNTER1 < SIZE OF STRING )
   BEGIN
        STR[COUNTER2] = STR[COUNTER1]
        INCREMENT COUNTER2 AND COUNTER1 BY ONE
   END
END DO
ASSIGN NULL CHARACTER TO END OF STRING
```

PROCESSING LOGIC FOR REC_HEADER

```
IF ( FILE IS ALREADY CREATED )
    APPEND IT BY CALLING THE FUNCTION APPEND()
ELSE
     DO WHILE ( EXIT IS NON ZERO )
        BEGIN
             CALL FIELD_NAME() TO GET FIELD NAME
             CALL FIELD_TYPE() TO GET FIELD TYPE
             IF ( FIELD TYPE IS NOT "DATE" )
               BEGIN
                   CALL FIELD_LENGTH() TO GET LENGTH OF FIELD
                   IF ( FIELD TYPE IS "FLOAT" )
                      CALL FLOAT_TYPE() TO GET DECIMAL WIDTH
                   ELSE INITIALIZE FIELD WIDTH TO ZERO
               END
             ELSE STORE 8 IN FIELD LENGTH
         END
     END DO
END IF
FIND OUT SUM OF( ALL FIELD WIDTHS ) IN THE RECORD
STORE IT IN  T_F_L & TOT_FIELDS_LEN
IF ( FILE OPENED WITH CREAT MODULE )
   BEGIN
        CALL  FUNCTION REC_DES_SIZE TO WRITE  RECORD  DESCRIPTION
        SIZE IN THE DBF FILE
        CALL FUNCTION REC_SIZE TO WRITE SIZE OF RECORD IN THE DBF
        FILE
        INITIALIZE I TO ZERO
        DO WHILE ( I < NUMBER OF FIELDS )
          BEGIN
              WRITE DOWN THE DESCRIPTION OF Ith FIELD IN DBF FILE
              INCREMENT I BY ONE
          END
        END DO
        STORE  THE  LAST  BYTE NUMBER OF  DBF  FILE  IN  VARIABLE
        FIL_POS2
   END
ELSE
  BEGIN
    RUN  INDEX_CHECK()  TO  KNOW THAT FILE  IS  TO  BE  RETREIVED
    ACCORDING TO INDEX FILE OR DBF FILE
  END
END IF
DO
  BEGIN
    ALLOCATTE MEMORY OF RECORD SIZE TO VARIABLE RECORD_S
    ASSIGN RECORD_C = RECORD_S
    DO WHILE ( I < NUMBER OF FIELDS )
       RUN FUNCTION SCREEN() TO DESIGN SCREEN FOR Ith FIELD
       INCREMENT I BY ONE
```

```
END DO
RUN FUNCTION RECORD_RETREIVAL()


DO WHILE ( I < NUMBER OF FIELDS )
   RUN FUNCTION MOVE_MENT() TO MOVE THE CURSOR IN  VARIOUS
   DIRECTIONS
   FUNCTION MOVE_MENT() RETURN PRESSED KEY IN VARIABLE CHECK
   IF ( RETUURN KEY IS UP_ARROW )
      BEGIN
           IF ( CURRENT FIELD IS NOT THE FIRST FIELD )
                MOVE THE CURSOR TO PRECEDING FIELD
           END IF
      END
   ELSE IF ( RETURN KEY IS Pg_Up )
        BEGIN
           IF ( CUURSOR IS NOT AT THE FIRST RECORD )
                MOVE THE CURSOR TO PRECEDING RECORD
           END IF
        END
   ELSE IF ( RETURN KEY IS End )
           POSITION THE CURSOR TO LAST RECORD
   ELSE IF ( RETURN KEY IS Pg_Down )
       BEGIN
           IF ( CURSOR IS NOT AT THE LAST RECORD )
              POSITION THE CURSOR AT PROCEDING RECORD
           END IF
       END
   ELSE POSITION THE CURSOR TO NEXT RECORD
   END IF
   DISPLAY THE RCORD ON SCREEN WHICH IS POSITITONED BY CURSOR
   IN ABOVE STEPS
END DO
WRITE THE RECORD IN DBF FILE
DISPLAY MESSAGE " PRESS ENTER TO EXIT "
GET KEY
RUN INDEX_CHECK2()
END DO WHILE ( ENTERED KEY IS NOT RETURN KEY )
```

PROCESSING LOGIC FOR MAIN()

```
IF ( ARGUMENTS PASSED ARE FOUR ( 4 ) )
   BEGIN
        OPEN  FILE OF NAME GIVEN BY SECOND ARGUMENT IN READ  ONLY
        MODE
        CREATE A NEW FILE OF NAME GIVEN IN THIRLD ARGUMENT
        IF ( THAT FILE ALREADY EXISTS )
             TRUNCATE IT
        STORE THE VALUE OF FOURTH ARGUMENT PASSED AT RUN TIME  IN
        KEY FIELD
        RUN FUNCTION CREATE()
        IF ( KEY_FIELD FOUND IN DBF FILE )
            BEGIN
                DO WHILE ( CURRENT RECORD IS NOT LAST RECORD )
                         RUN INSERT()
                END DO
                RUN FUNCTION WRITE_HEADING()
                RUN FUNCTION RETREIVE()
                RUN FUNCTION FREE_NODE()
             END
        END IF
```

****

PROCESSING LOGIC FOR CREAT()

```
ALLOCATE MEMORY TO THE ROOT OF BINARY TREE
CALL FUNCTION KEY_BUFF()
ALLOCATE MEMORY OF SIZE FIELD LENGTH TO KEY FIELD ( C ) OF ROOT
CALL FUNCTION READ_KEY_FIELD()
```

****

PROCESSING LOGIC FOR KEY_BUFF()

```
RUN FUNCTION APPEND()
STORE THE VALUE RETURN BY APPEND() IN VARIABLE REC_START_LOC
CALL FUNCTION REC_COUNT()
POSITION THE HANDLE TO THE STARTING LOCATION OF FIRST RECORD
```

PROCESSING LOGIC FOR APPEND()

```
OPEN DBF FILE IN READ WRITE MODE
READ NUMBER OF FIELDS IN RECORD FROM DBF FILE
INITIALIZE KEY FIELD LENGTH AND TOTAL FIELD LENGTH TO ZERO
INITIALIZE KEY FIELD NAME TO SPACES
K = 0
DO WHILE ( K LESS THAN NUMBER OF FIELDS IN THE RCORD )
   BEGIN
       READ Kth FIELD FROM DBF FILE
       READ FIELD WIDTH OF Kth FIELD
       TOTAL FIELD LENGTH = TOTAL FIELD LENGTH + FIELD LENGTH  OF
       Kth FIELD
       IF ( ENTERED KEY FIELD NAME IS FOUND IN THE DBF FILE )
          BEGIN
              FLAG2 = K
              STORE FIELD LENGTH OF Kth FIELD IN THE IN KEY_LEN
              STORE  THE  TOTAL LENGTH OF FIRST (K-1)  FIELDS  IN
              VARIABLE INIT_ADDR
          END
       END IF
   END
END DO
INCREMENT TOT_FIELDS_LEN BY ONE
```

PROCESSING LOGIC FOR INSERT()

```
ASSIGN CUR EQUAL TO ROOT
ALLOCATE MEMORY OF SIZE BTREE ie. ( SIZE OF NODE OF BINARY TREE )
ALLOCATE MEMORY OF SIZE KEY FIELD LENGTH TO FIELD C OF STRUCT NEW
CALL FUNCTION READ_KEY_FIELD()
IF ( NEWLY READ VALUE IS LARGER THAN CURRENT VALUE )
    STORE ONE ( 1 ) IN T ELSE STORE ZERO ( 0 ) IN T
DO WHILE ((( NEW RECORD >= CURRENT RECORD ) AND ( RIGHT CHILD OF
            CURRENT  RCORD  IS NOT NULL )) OR (( NEW  RECORD <=
             CURRENT  RECORD  )  AND ( LEFT  CHILD  OF  CURRENT
            RECORD IS NOT  NULL )))
   BEGIN
      INITIALIZE CURRENT RECORD EQUAL TO VAR PRE ie. (  PREVIOUS
                                                       RECORD )
      IF ( NEW RECORD IS LARGER THAN CURRENT RECORD )
         GO  TO  THE RIGHT CHILD OF CUR RECORD AND MAKE  IT  THE
         CURRENT RECORD
      ELSE
         GO TO THE LEFT CHILD OF CURRENT AND MAKE IT THE CURRENT
         RECORD
      END IF
   END
END DO
IF ( NEW RECORD IS LARGER THAN CURRENT RECORD )
   BEGIN
      STORE NEW RECORD AT THE RIGHT CHILD OF CURRENT RECORD
      ASSIGN LEFT AND RIGHT CHILD OF NEW RECORD TO NULL
   END
ELSE IF ( NEW RECORD LESS THAN OR EQUAL TO CURRENT RECORD )
    BEGIN
       STORE NEW RECORD AT THE LEFT CHILD OF CURRENT RECORD
       ASSIGN LEFT AND RIGHT CHILD OF NEW RECORD TO NULL
    END
END IF
```

****

PROCESSING LOGIC FOR WRITE_HEADING()

```
WRITE KEY FIELD NAME ON THE DBF FILE
WRITE KEY FIELD LENGTH ON DBF FILE
```

PROCESSING LOGIC FOR RETREIVE()

```
IF ( LEFT CHILD OF CURRENT RECORD IS NOT NULL )
    CALL TO ITSELF ie.  ( FUNCTION RETREIVE()  )  WITH  PARAMETER
    PASSED TO ITSELF IS LEFT CHILD OF CURRENT RECORD
END IF
WRITE VALUE OF CURRENT RECORD IN INDEX FILE
WRITE ADDRESS OF CURRENT RECORD IN DBF FILE TO INDEX FILE
WRITE THE NUMBER OF THE RECORD IN INDEX FILE
IF ( RIGHT CHILD OF CURRENT RECORD IS NOT NULL )
    CALL TO ITSELF ie.  ( FUNCTION RETREIVE()  )  WITH  PARAMETER
    PASSED TO ITSELF IS RIGHT CHILD OF CURRENT RECORD
EMD IF
```

****

PROCESSING LOGIC FOR FREE_NODE

```
IF ( LEFT CHILD OF CURRENT RECORD IS NOT NULL )
    CALL TO ITSELF ie.  ( FUNCTION FREE_NODE()  ) WITH PARAMETER
    PASSED TO ITSELF IS LEFT CHILD OF CURRENT RECORD
END IF
IF ( RIGHT CHILD OF CURRENT RECORD IS NOT NULL )
    CALL TO ITSELF ie. ( FUNCTION FREE_NODE() )  WITH  PARAMETER
    PASSED TO ITSELF IS RIGHT CHILD OF CURRENT RECORD
EMD IF
FREE THE MEMORY ALLOCATED TO CURRENT RECORD
```

HARDWARE AND SOFTWARE REQUIREMENTS :

H/W REQUIREMENTS :

PC-XT WITH

640KB	RAM
CGA	COLOR GRAPHICS ADAPTER
VDU	VIDEO DISPLAY UNIT

S/W REQUIREMENTS :

MS - DOS VER 3.0 OR Higher.

USER CHARACTERSTICS :

The user should be familiar with C.

He must possess some knowledge of DOS VER 3.0

WORKING ENVIRONMENT :

DEVELOPMENT ENVIRONMENT :

MS-DOS VER 3.0

TURBO-C VER 2.0

OPERATING ENVIRONMENT :

MS-DOS VER 3.0 or higher .

MAINTENANCE ENVIRONMENT :

MS-DOS VER 3.0

UNIX V

TURBO-C VER 2.0

CONCLUSION AND REMARKS

During the course of system devolpment project I was engaged in developing a general purpose "DATA BASE IN C". This database will provide immense help in designing the other related projects such as PAY_ROLL SYSTEM, ACCONTANCY, INCOME TAX or any other project that require some data storage or manipulation. The input format screens of the "DATA BASE" Is compatible with the input format of Dbase. The output format has yet to be design

Utilities of this database can be called in some other program by mere including the neccessary files . The "DATA BASE" developed is flexible enough that further modifications can be done with ease .