

PROJECT REPORT (VOL II)

**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**

```

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <ctype.h>
#include <sys\stat.h>
#include <fcntl.h>
#define MAX_VAR_LEN 15
#define FIELD_SPECIF_SIZE 24
#define Up_Arrow -1
#define Pg_Up 2
#define End 4
#define Pg_Down 3
typedef struct {
    char var_name[MAX_VAR_LEN+1];
    char var_type[2];
    char var_length[3];
    char var_decimal[3];
} var_rec;

char file[] = "tst.c";
char file_flag = 'f';
int f_desc,f_desc2;
long file_position,fil_pos2,record_start,t_f_l2;
void types (char *var_type,int *var_length,int*var_dec);
int file_copy(var_rec *v1);
void file_copy2(char *record,int *buff_size,int fd,int check);
void float_length(char float_len[2],int *var_len,int *v_col_start,
    int *v_row_start,int *flt_len,int *v_col_end);
int date(int *inc,char *record);

```

```

int date_of_month(int flag_y,int m,int d);
int year(int y);
int month(int m);
void float_read(int *i,int *i_dupl,int *num,int *len,int *dec,
                char *record,char *c,int *col,int *inc,char *field_type );
void char_read(int *i,int *i_dupl,int *len,char *record,int *col,
               int *inc, char *c);
void read_values(int *i,int *idupl,int *value_len,int *value_dec,
                char *c,char *value_type,int *inc,char *record);
void text_setter(void);
void move_ment(int *i,int *i_dupl,int *field_len,int *flt_len,int *check,
               char *var_type,int *inc,char *record,char *c);
void movements(int *i,int *inc,int *col,char *record);
void screen( char *var_name,char *var_type,int *var_length,
             int *var_dec,int *k);
void rec_header(char *argv[],char index_flag);
void format ( int *v_col_start,int *v_row_start,int *v_col_end);
void field_length(char var_len[2],int *v_col_start,int *v_row_start,
                  int *field_len,int *v_col_end);
void rec_size(int record_size);
void rec_des_size(int num_of_fields);
void modify(int *field_len,int *flt_len,char *var_type,
            int *inc,char *record);
char is_record_empty (char *record, int buf_size);
void record_retreival( char *record_s,char *record_c, int tot_fields_len,
                      long t_f_l,int j,int field_len[20],int flt_len[20],
                      var_rec var_buf[20],int *check,int enter );
void index_check( char *argv[], char *index_flag,long t_f_l,long *d );
void key_name(char name[MAX_VAR_LEN + 1] );
void index_check2( char index_flag);

```

```

#include "ldcl2.c"

void field_name(char var_name[15], int *v_col_start,int *v_row_start,
                int *v_col_end,int *exit )

/*****
* This function reads the field name of 15 characters that start with an *
* alphabet and may contain alpha numeric characters or '_'. *
* It takes address of array as input ie.(address of array is passes as *
* parameter and store the field name in that array. *
*****/

{
    int i,i_dupl = 0;
    char c = ' ',flag = 'f';
    for(i=0;i<MAX_VAR_LEN;i++)
        var_name[i] = ' ';
    var_name[i] = '\0';
    i=0;
    format( v_col_start,v_row_start,v_col_end);
    var_name[i] = getche();
    if (var_name[0] != '\r')
    {
        while ((var_name[i] != '\r') && ( i <(MAX_VAR_LEN-1)))
        {
            flag = 'f';
            if (var_name[i] == 0)
            {
                var_name[i] = c;
                gotoxy(*v_col_start+i,*v_row_start);
                putchar(c);
                if (i_dupl < i)
                    i_dupl = i;
            }
        }
    }
}

```

```

    c = move_lr( &i,*v_col_start,*v_row_start,&i_dupl,var_name);
    if (c != 0)
        var_name[i] = c;
    flag = 't';
}
else if ((i==0)&&(!(isalpha(var_name[i]))))
    {
        --i;
        gotoxy(*v_col_start,*v_row_start);
    }
else if(!(isalnum(var_name[i]))&&(var_name[i] != '_'))
    {
        --i;
        gotoxy(*v_col_start+1+i,*v_row_start);
    }
if (flag == 'f')
    {
        c = var_name[i+1];
        var_name[++i] = getch();
    }
if (var_name[i] == '\r')
    {
        if (i == 0)
            *exit = 0;
    }
}
} /* ----- if loop terminates ----- */

```

```
    else *exit = 0;
    if (var_name[i] == '\r')
        var_name[i] = ' ';
}
```

```
move_lr(int *i, int v_col_start, int v_row_start, int *i_dupl,
        char var_name[MAX_VAR_LEN])
```

```
{
    char c;
    int d, temp;
    do
    {
        d = getch();
        switch(d)
        {
            case 77:
                {
                    if (*i < *i_dupl)
                        (*i)++;
                    break;
                }
            case 75:
                {
                    if (*i > 0)
                        (*i)--;
                    break;
                }
            case 83:
                {
```

```

        for(temp = *i;temp < *i_dupl; temp++)
            {
                var_name[temp] = var_name[temp + 1];
                putchar(var_name[temp]);
            }
        if (*i < *i_dupl)
            (*i_dupl)--;
    }
    default : break;
}
gotoxy (v_col_start + *i,v_row_start);
c = getch();
} while ( c == 0);
if (isalnum(c)||!(c=='_'))
    {
        putchar(c);
        return c;
    }
else return 0;
}

```

```

void field_type(char var_type[10], int *v_col_start,int *v_row_start,
                int *v_col_end)
{
    char choice;
    int i=0,j;

```

```
char *field_type[4]={"character","integer ","float ",
                    "date "};
```

```
*v_col_start +=20;
```

```
*v_col_end = *v_col_start + 9;
```

```
format( v_col_start,v_row_start,v_col_end);
```

```
cprintf("character");
```

```
choice = getch();
```

```
while(choice != '\r')
```

```
{ /* while loop begins */
```

```
  if (choice != ' ')
```

```
  {
```

```
    choice = getch();
```

```
    continue;
```

```
  }
```

```
  if(i==3)
```

```
  i=0;
```

```
  else i++;
```

```
  gotoxy(*v_col_start,*v_row_start);
```

```
  cprintf("%s",field_type[i]);
```

```
  choice = getch();
```

```
} /* while loop ends */
```

```
  var_type[0] = *(field_type[i]);
```

```
var_type[1]='\0';
```

```
}
```

```
void field_length(char var_len[2],int *v_col_start,int *v_row_start,
                  int *field_len, int *v_col_end)
```

```

{
    int i;
    char c,c2,c_dupl=0;
    var_len[0] = var_len[1] = ' ';
    *v_col_start += 15;
    *v_col_end = *v_col_start + 2;
    i=*v_col_start;
    format( v_col_start,v_row_start,v_col_end);
do
    {
        back:  c= getch();
        if ((c == 0)&&(c_dupl != 0))
            {
                if ((c = getch()) == 77)
                    {
                        c = c_dupl;
                        break;
                    }
            }
        gotoxy(i,*v_row_start);
    } while(!isdigit(c));
    if (isdigit(c))
        c_dupl = c;
    putchar(c);
    i++;
do
    {
        c2 = getch();

```

```

if (c2 == 0)
{
    if ((c2 = getch()) == 75)
    {
        gotoxy(--i, *v_row_start);
        goto back;
    }
}
else if(isdigit(c2))
{
    putchar(c2);
    i++;
    var_len[0] = c;
    var_len[1] = c2;
    *field_len = (c-'0')*10 + c2-'0';
}
else if (c2 == '\r')
{
    gotoxy(i - 1, *v_row_start);
    putchar('0');
    i++;
    putchar(c);
    i++;
    var_len[0] = ' ';
    var_len[1] = c;
    *field_len = c-'0';
}
}while((c2 != '\r') && (i != *v_col_end));

```

```

        var_len[2] = '\0';
    }

int append(var_rec record[20],int field_len[20],int flt_len[20] )
{
    int fd,no_of_fields,k=0;
    char rd_from_file[3];
    fd = open(file,O_RDWR,0);
    f_desc = fd;
    lseek(fd,201,0);
    if ((read(fd,rd_from_file,3))!=3)
        printf("ERROR");
    no_of_fields = atoi(rd_from_file);
    lseek(fd,31,1);
    do
    {
        read(fd,record[k].var_name,16);
        read(fd,record[k].var_type,2);
        read(fd,record[k].var_length,3);
        read(fd,record[k].var_decimal,3);
        field_len[k] = atoi(record[k].var_length);
        flt_len[k] = atoi(record[k].var_decimal);
        k++;
    }
    while(k < no_of_fields);
    lseek(fd,01,2);
    record_start = (26 + no_of_fields * 24);
}

```

```

    file_position = lseek(f_desc,record_start,0);
    return(no_of_fields);
}

```

```

void rec_header(char *argv[], char index_flag)
{
    int j,i,exit=1,count,enter = 0,check =0,d1,d2,fd,chk_dupl = 0;
    int field_len[20],flt_len[20],tot_fields_len = 0;
    long t_f_l,d,gh,size_i_l;
    int v_col_start,v_row_start,v_col_end;
    var_rec var_buf[20];
    char *record_s,*record_c,c,cha[100];
    size_i_l = sizeof(int) + sizeof(long);
    v_col_start=5;
    v_row_start =2;v_col_end=20;
    j=0;
    if (file_flag == 'f')      {
        j = append(var_buf,field_len,flt_len);
        printf("%d",field_len[0]);}
    else
    {
        do
        {
            v_col_start=5;
            v_col_end = 20;
            field_name(var_buf[j].var_name,&v_col_start,&v_row_start,
                &v_col_end,&exit);
            if (exit != 0)

```

```

{
  field_type(var_buf[j].var_type,&v_col_start,&v_row_start,
            &v_col_end);

  if (var_buf[j].var_type[0] != 'd')
  {
    field_length(var_buf[j].var_length,&v_col_start,
                &v_row_start,&field_len[j],&v_col_end);
    if(var_buf[j].var_type[0] == 'f')
    {
      float_length(var_buf[j].var_decimal,&field_len[j],
                  &v_col_start,&v_row_start,&flt_len[j],&v_col_end);
    }else
    {
      var_buf[j].var_decimal[0] = ' ';
      var_buf[j].var_decimal[1] = ' ';
      var_buf[j].var_decimal[2] = '\0';
    }
  }else
  {
    var_buf[j].var_length[0] = '0';
    var_buf[j].var_length[1] = '8';
    var_buf[j].var_decimal[0] = ' ';
    var_buf[j].var_decimal[1] = ' ';
    field_len[j] = 8;
    var_buf[j].var_decimal[2] = '\0';
    var_buf[j].var_length[2] = '\0';
  }

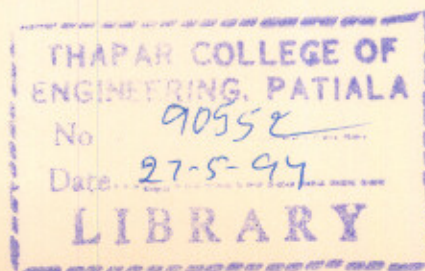
  v_row_start +=2;

```

```

        j++;
    }
}while (exit != 0);
}
for(i=0;i<j;i++)
    tot_fields_len = tot_fields_len + field_len[i];
++tot_fields_len;
t_f_l = (long) tot_fields_len;
if (file_flag != 'f')
    {
        rec_des_size(j);
        rec_size(tot_fields_len);
        for(i=0;i<j;i++)
            file_copy(&var_buf[i]);
        close(f_desc);
        f_desc = open(file,O_RDWR,0);
        file_position = lseek(f_desc,0l,2);
        lseek(f_desc,file_position,0);
    }
    index_check( argv,&index_flag,t_f_l,&d );
do
    /***** DO LOOP BEGINS *****/
    {
        record_s = (char *)malloc(tot_fields_len);
        for(count =0;count < tot_fields_len-1;count++)
            *(record_s + count) = ' ';
        *(record_s+count) = '\0';
        record_c = record_s;

```



```

normvideo();
clrscr();
textbackground(0);
textcolor(15);

                /****** file_copy() writes the record   *****/
                *****/      description into file      *****/

for(i=0;i<j;i++)
{
    /****** function screen() makes screen
    for data entry *****/
    screen(var_buf[i].var_name,var_buf[i].var_type,
           &field_len[i], &flt_len[i],&i);
}

    /****** f_desc is declared as a global variable **/
record_retreival( record_s,record_c,tot_fields_len, t_f_1, j,field_len,
                 flt_len, var_buf,&check,enter );

/****** IT IS USED FOR RETREAVING AND MODIFICATION OF A RECORD *****/
for(i=0;i<j;)
{
    c = '\r';
    d1 = d2 = 0;
    chk_dup1 = check;
    move_ment(&d1,&d2,&field_len[i],&flt_len[i],&check,
             var_buf[i].var_type,&i,record_s,&c);

    switch(check)

```

```

{
case Up_Arrow :
    {
        if(i > 0)
            {
                i--;
                record_s = record_s - field_len[i];
            }
        break;
    }
case Pg_Up :
    if (fil_pos2 == lseek(f_desc2,0l,1) )
        break;
    else {
        lseek(f_desc2,-t_f_12,1);
        index_check2( index_flag);
    }
case End :
    {
        if (check == 4)
            {
                if (index_flag == 's')
                    lseek(f_desc2,-size_i_1,2);
                else lseek(f_desc2,-t_f_12,2);
                index_check2( index_flag);
            }
    }
}

```

```

case Pg_Down :
    {
        if(check == 3)
        {
            /* d = lseek(f_desc2,0l,1); */
            if ((gh = (lseek(f_desc2,0l,1)))
                != (d - size_i_1 ))

                lseek(f_desc2,t_f_12,1);

            index_check2( index_flag);
        }
        if((read(f_desc,record_s,tot_fields_len))
            != tot_fields_len)

            printf("Error");
        record_c = record_s;
        lseek(f_desc,-t_f_1,1);
        for(i=0; i < j;i++)
        {
            modify( &field_len[i],&flt_len[i],
                var_buf[i].var_type,&i,record_s);

            record_s = record_s + field_len[i];
        }
        record_s = record_c;
        i = 0;
        break;
    }

default :
    {
        record_s = record_s + field_len[i];
        i = i++;
    }

```

```

                break;
            }
        }
    }

    /*****
    *          function read_values() store values from *
    *****/
    screen into records *****/

file_copy2(record_c,&tot_fields_len,f_desc,chk_dup1);

clrscr();

printf("\n\n\t press enter to exit :");

enter = getche();

if (enter == 0)
    enter = getch();

clrscr();

if (file_flag == 't')          /***** == 'f' *****/
    lseek(f_desc2,t_f_12,1);

index_check2( index_flag);

}while(enter != 13);          /***** DO LOOP TERMINATES *****/
}

```

```

void index_check( char *argv[], char *index_flag,long t_f_1,long *d )

```

```

/*****
*   called by      :   rec_header()
*   calls to      :   key_name()
*   purpose       :   it links index file with dbf file to
*                   perform editing .
*****/

```

```

{
    long k,len;
    int index_field_len;
    char key_field[MAX_VAR_LEN +1];
    if ( *index_flag != 't' )

```

```

    {
        f_desc2 = f_desc;
        t_f_l2 = t_f_l;
        fil_pos2 = file_position;
    }
else
    {
        f_desc2 = open(argv[2],O_RDONLY|O_BINARY,0);
        *d = lseek(f_desc2,0l,2);
        lseek(f_desc2,0l,0);
        len = MAX_VAR_LEN + 1;

        read(f_desc2,key_field,len);
        key_name( argv[3] );
        if (strcmp( argv[3],key_field ) == 0)
            {
                read(f_desc2,&index_field_len,sizeof(int));
                *index_flag = 's';
                k = (long) index_field_len;
                fil_pos2 = lseek( f_desc2,k,1);
                t_f_l2 = index_field_len + sizeof(int) + sizeof(long) ;
            }
        else printf(" ERROR incorrect field name ");
    }
}          /***** function index_check() ends *****/

void index_check2(char index_flag )
{

```

```

long k,j;
if ( index_flag == 's' )
{
    j = sizeof(long);
    read(f_desc2,&k,j);
    lseek(f_desc2,-j,1);
    lseek(f_desc,k,0);
}
}

```

```

void key_name(char name[MAX_VAR_LEN + 1] )

```

```

{
    int len;
    len = strlen(name);
    while( len < MAX_VAR_LEN )
    {
        name[len] = ' ';
        len++;
    }
    name[len] = '\0';
}

```

```

void text_setter(void)
{
    textbackground(15);
    textcolor(0);
}

```

```

void record_retreival( char *record_s,char *record_c, int tot_fields_len,
                      long t_f_l,int j,int field_len[20],int flt_len[20],
                      var_rec var_buf[20],int *check,int enter )

```

```

    {
        int i;
        if(((read(f_desc,record_s,tot_fields_len))
            != tot_fields_len) && (enter != 0))
            {
                *check = 9;
            }
        else
            {
                lseek(f_desc,-t_f_l,1);
                record_c = record_s;
                for(i=0; i < j;i++)
                    {
                        modify( &field_len[i],&flt_len[i],
                            var_buf[i].var_type,&i,record_s);
                        record_s = record_s + field_len[i];
                    }
                record_s = record_c;
            }
    }

```

```

void format ( int *v_col_start,int *v_row_start,int *v_col_end)
    {
        int i,j;
        gotoxy(*v_col_start,*v_row_start);
        j = *v_col_end - *v_col_start;
        for(i=0;i<=j;i++)

```

```
    cprintf(" ");
    gotoxy(*v_col_start,*v_row_start);
}
```

```
void float_length(char float_len[2],int *var_len,int *v_col_start,
                 int *v_row_start,int *flt_len,int *v_col_end)
{
    int i;
    char c,c2;
    float_len[0] = float_len[1] = ' ';
    *v_col_start += 5;
    *v_col_end = *v_col_start + 2;
    i=*v_col_start;
    format( v_col_start,v_row_start,v_col_end);
do
    {
        c= getch();
        gotoxy(i,*v_row_start);
    }
    while(!isdigit(c));
    putchar(c);
    i++;
do
    {
        c2 = getch();
        if(isdigit(c2))
        {
```

```

    putch(c2);
    i++;
    float_len[0] = c;
    float_len[1] = c2;
    *flt_len = (c-'0')*10 + c2-'0';
}
else if (c2 == '\r')
    {
        gotoxy(i - 1, *v_row_start);
        putch('0');
        i++;
        putch(c);
        i++;
        float_len[0] = ' ';
        float_len[1] = c;
        *flt_len = c-'0';
    }
if (*var_len < *flt_len)
    {
        gotoxy(*v_col_start, *v_row_start);
        putch(' ');
        putch(' ');
        do
            {
                gotoxy(*v_col_start, *v_row_start);
                c=getch();
            }
        while(!isdigit(c));
    }

```

```

        putchar(c);
        i = *v_col_start + 1;
    }
    }while((c2 != '\r')&&(i != *v_col_end));
float_len[2] = '\0';
}

```

```

int file_copy(var_rec *v1)
{
    lseek(f_desc,0l,2);
    if(write(f_desc,v1,FIELD_SPECIF_SIZE) != FIELD_SPECIF_SIZE)
        printf("error");
}

```

```

void file_copy2(char *record,int *buff_size,int fd,int check)
{
    int s;
    long r;
    fd = f_desc2;
    if(check == 9)
        s = lseek(fd,0l,2);
    else {
        r = lseek(fd,0l,1);
        if ( r < fil_pos2 )
            s = lseek(fd,0l,2);
    }
    if ( ( is_record_empty( record, *buff_size) ) == 'n' )

```

```

        {
            if (write(fd,record,*buff_size)!= *buff_size)
                printf("error");
        }
        free(record);
    }

char is_record_empty (char *record, int buf_size)
    {
        /*****
        *
        *   called by      : file_copy2
        *
        *   function      : it tells weather a record which it takes as
        *                   input is empty or not ?
        *
        *   return value   : it returns 'y' if record is empty else 'n'.
        *
        *****/

        int count;

        for(count = 0;((count < buf_size ) && (*(record + count)
            == ' ' || *(record + count) == '0') ); count++);

        if ( count == (buf_size -1) )
            return 'y';

        else return 'n';
    }

int signature()
    {
        int fd;

        char sign[] = "bhavdeep 17/04/71";

        fd = open(file,O_RDWR,0);

        if (fd <=0)

```

```

    {
        fd = open(file,O_RDWR;O_CREAT;S_IREAD;S_IWRITE;O_APPEND,0);
        if (write(fd,sign,17)!= 17)
            printf("error");
        file_flag = 't';
    }
    close(fd);
    return fd;
}

```

```
void rec_des_size(int num_of_fields)
```

```

{
    int fd;
    char rec_des[3];
    itoa(FIELD_SPECIF_SIZE,rec_des,10);
    if(file_flag != 'f')
    {
        fd = open(file,O_WRONLY;O_APPEND,0);
        write(fd,rec_des,3);
        itoa(num_of_fields,rec_des,10);
        write(fd,rec_des,3);
        close(fd);
    }
}

```

```
void rec_size(int record_size)
```

```

{
    int fd;

```

```

char record[3];
itoa(record_size,record,10);
if(file_flag != 'f')
    {
        fd = open(file,O_WRONLY;O_APPEND,0);
        write(fd,record,3);
        close(fd);
    }
printf(" %s",record);
fd = open(file,O_RDWR,0);
}

void main(int argc ,char *argv[])
{
char index_flag=' ';
textbackground(0);
clrscr();
text_setter();
f_desc = signature();
if (file_flag == 'f')
    {
        if ((argc == 4) && ( strcmp( argv[1],"INDEX") == 0) || (strcmp( argv[
            index_flag = 't';
    }
rec_header(argv,index_flag);
normvideo();
}

```

```

#include "dh2.c"

#define NEXT 20

#define ROW 3

#define C c1= values[count2-1]; c2 = values[count2];
#define D (10*(c1-'0') + (c2-'0'))

void types(char *var_type,int *var_length,int *var_dec);
void screen( char *var_name,char *var_type,int *var_length,
            int *var_dec,int *k)

/*****
*   function : screen()
*   This function generates the screen for entering data in the
*   records.
*   call by : rec_header();
*   calls to : types()
*
*****/
{
    int row,col;

    row = 3;

    col = 5;

    gotoxy(col,row + *k);
    cprintf("          : ");
    gotoxy(col,row + *k);
    cprintf("%s",var_name);
    gotoxy(col+NEXT,row + *k);
    types(var_type,var_length,var_dec);
/*   for(counter=0;counter<*var_length;counter++)
    cprintf(" "); */
}

```

```
void types(char *var_type,int *var_length,int *var_dec)
```

```
/*
 *
 *   called by : screen()
 *   calls to  : none
 *
 */
```

```
{
    int counter,num;
    if (*var_type == 'd')
        cprintf(" / / ");
    else if (*var_type == 'f')
    {
        num = *var_length - *var_dec;
        for(counter=0;counter<num;counter++)
            cprintf(" ");
        cprintf(".");
        for(counter=0;counter<*var_dec;counter++)
            cprintf(" ");
    }
    else
    {
        for(counter=0;counter<*var_length;counter++)
            cprintf(" ");
    }
}
```

```
void read_values(int *i,int *i_dupl,int *value_len,int *value_dec,
                char *c,char *value_type,int *inc,char *record)
```

```
{
```

```

/*****
*
* It reads the record from the screen generated by function screen()
* called by : rec_header()
* calls to : float_read(),char_read(),date()
*
*****/

int flag_d,*inc_dup;

int col = 5;

int row = 3,num;

char *record_dup;

gotoxy(col+NEXT + *i,row + *inc);

if(*value_type == 'd')
{
    inc_dup = inc;
    flag_d = 1;
    do {
        inc = inc_dup;
        if(flag_d == 0)
        {
            gotoxy(col+NEXT,row + *inc);
            cprintf(" / / ");
        }
        flag_d = date(inc,record);
    } while(flag_d == 0);
}

else if(*value_type == 'f')
{
    num = *value_len - *value_dec;
    float_read(i,i_dup1,&num,value_len,value_dec,record,
              c,&col,inc,value_type);
}

```

```

    }
    else if(*value_type == 'i')
    {
        num = *value_len;
        float_read(i,i_dupl,&num,value_len,value_dec,record,
                  c,&col,inc,value_type);
    }
    else char_read(i,i_dupl,value_len,record,&col,inc,c);
}

```

```

void pad_char( char str[], char ch, int field_length )

```

```

{
    typedef unsigned char BYTE;
    BYTE ptr, i, j, length = 0;
    char result[80];
    while ( str[length] != ' ' && length < field_length )
        length++;
    ptr = field_length - length;
    setmem( result, ptr, ch );
    for ( j = 0, i = ptr; i < field_length; i++, j++ )
        result[i] = str[j];
    str = result;
    result[i] = '\0';
    cprintf("%s",result);
}

```

```

void trim_char( char str[], char ch, char ch2, int len )
{
    typedef unsigned char BYTE;
    BYTE i = 0, j = 0;
    char k;
    while ( str[i] == ch )
        i++;
    while ( i < len )
        str[j++] = str[i++];
    while ( j < len )
        str[j++] = ch2;
    k = str[j];
    str[j] = '\0';
    printf("%s",str);
    str[j] = k;
}

```

```

void move_ment(int *i,int *i_dupl,int *field_len,int *flt_len,int *check,
               char *var_type,int *inc,char *record,char *c)
{
    int d,temp;
    gotoxy (NEXT + 5 ,ROW + *inc);
    if ( *var_type == 'i')
        trim_char(record,'0',' ',*field_len);
    else if(*var_type == 'f')
        trim_char(record,'0',' ',( *field_len - *flt_len ));

    gotoxy (NEXT + 5 + *i,ROW + *inc);
}

```

```

do
{
    *check = 0;
    if (*c == 0)
        d = getch();
    else
        d = 0;
while((*i_dupl) != ' ' && (*i_dupl < (*field_len - 1)))
    (*i_dupl)++;
    switch(d)
    {
        case 77:
            {
                if((*var_type == 'f') &&
                    (*i == (*field_len - *flt_len - 1)))
                    *i = *i + 2;
                else if (*i < *i_dupl)
                    (*i)++;
                break;
            }
        case 75:
            {
                if((*var_type == 'f') &&
                    (*i == (*field_len - *flt_len + 1)))
                    *i = *i - 2;
                else if (*i > 0)
                    (*i)--;
                break;
            }
    }

```

```
case 83:
    {
        if (*var_type != 'd')
            {
                *(record + *i_dupl) = ' ';
                for(temp = *i; temp < (*i_dupl); temp++)
                    {
                        *(record+temp) = *(record + temp +1 );
                        putchar(*(record+temp));
                    }
                putchar(*(record + *i_dupl));
                if (*i < *i_dupl)
                    (*i_dupl)--;
            }
        break;
    }
case 72:
    {
        *check = -1;
        break;
    }
case 80:
    {
        *check = 1;
        break;
    }
case 73:
    {
        7
```

```

        *check = 2;
        break;
    }
    case 81:
        {
            *check = 3;
            break;
        }
    case 79:
        {
            *check = 4;
            break;
        }
    default :
        break;
}

gotoxy (NEXT + 5 + *i,ROW + *inc);
if (*check == 0)
    *c = getch();
else *c = '\r';
if ((*c != '\r')&&>(*c != 0))
    read_values(i,i_dupl,field_len,flt_len,c,var_type,inc,record);
} while ( *c == 0);
gotoxy (NEXT + 5 ,ROW + *inc);
if (*var_type == 'i')
    pad_char(record, '0' , *field_len);
else if (*var_type == 'f')
    pad_char(record, '0' , (*field_len - *flt_len));

```

```

}

void modify(int *field_len,int *flt_len,char *var_type,int *inc,char *record)
{
    int i;
    gotoxy (NEXT + 5 ,ROW + *inc);
    switch(*var_type)
    {
        case 'd': *field_len = 8;
        case 'i':
        case 'c':
            {
                for ( i= 0; i< *field_len; i++)
                {
                    cprintf("%c",*(record + i));
                }
                break;
            }
        case 'f':
            {
                for ( i= 0; i < (*field_len - *flt_len); i++)
                    cprintf("%c",*(record + i));
                for ( i= (*field_len - *flt_len); i< *field_len; i++)
                    cprintf("%c",*(record + i));
                break;
            }
    }
}

```

```
int date(int *inc,char *record)
```

```
/******  
*  
*   it reads date from the screen  
*   called by : read_values()  
*   calls to  : year(),month(),date_of_month()  
*  
*****/  
  
{  
  
    int yr,mon,dat,flag_y,flag_m,flag_d = 0;  
    int count,count2,col = 25;  
    char c,c1,c2,*r_dup,values[8];  
    for(count2 = 0;count2<7;)  
        {  
            for(count = 0;count <2;count++)  
                {  
                    gotoxy(col+count+count2,3 + *inc);  
                    c = getche();  
                    if(isdigit(c))  
                        values[count+count2]=c;  
                    else --count;  
                }  
            if(count2 != 6)  
                {  
                    count2 += 2;  
                    values[count2++] = '/';  
                }  
            else count2++;  
        }  
}
```

```

for(count=0;count<8;count++)
    *(record+count) = values[count];
*(record+count) = '\0';
count2 = 7;
C
yr = D;
count2 -= 3;
C
mon = D;
count2 -= 3;
C
dat = D;
flag_y = year(yr);
flag_m = month(mon);
if (flag_m)
    flag_d = date_of_month(flag_y,mon,dat);
return(flag_d);
}

```

```

void float_read(int *i,int *i_dupl,int *num,int *len,int *dec,char *record,
               char *c,int *col,int *inc,char *field_type )
{
/*****
*
*   called by : read_values()
*
*   calls to  : none
*
*   it reads float & integer type of data
*****/
int k,l,flt_pos,flt_pos_dupl;

```

```

char *float2,flag='t';
if(isdigit(*record))
{
    float2 = record;
    flag = 'f';
}
else float2 = (char *)malloc(*len);
while ((*i<*num)&&((*c!='\r')&&(*c!='.')&&(*c != 0)))
{
    putchar(*c);
    if(isdigit(*c))
        *(float2 +(*i)++) = *c;
    else gotoxy(*col+NEXT + *i,ROW + *inc);
    *c = getch();
    if (*i > *i_dup1)
        *i_dup1 = *i;
}
*i = *i_dup1;
if(flag != 'f')
{
    if(*i<*num)
    {
        for(k=0;k<(*num-*i);k++)
            *(record +k) = '0';
        for(l=0;l<*i;l++)
            *(record +k++) = *(float2 +l);
        gotoxy(*col+NEXT,ROW + *inc);
    }
}

```

```

        for(k=0;k<*num;k++)
            putchar(*(record +k));
    }
else
    {
        for(k=0;k<*i;k++)
            *(record+k) = *(float2+k);
    }
}
if (flag != 'f')
    free(float2);
if(*field_type == 'f')
    {
        if (flag == 'f')
            k = *i -1;
        else k = 0;
        gotoxy(*col+NEXT + k + 1 + *num,ROW + *inc);
        while ((k<*dec)&&(*c!='\r'))
            {
                putchar(*c);
                if(isdigit(*c))
                    *(record + *num +k++) = *c;
                else
                    gotoxy(*col+NEXT + *num +k+1 ,ROW + *inc);
                *c = getch();
                if (*c == 0)
                    *c = '\r';
            }
    }

```

```

        *i = k+1;
    }
}

```

```

void char_read(int *i,int *i_dupl,int *len,char *record,int *col,
              int *inc, char *c)

```

```

{
/*****
*
* It reads character type of data
* called by : read_values()
* calls to : none
*
*****/
    while ((*i<*len)&&(*c!='\r')&&(*c != '\x0'))
    {
        if(isalnum(*c))
        {
            *(record + (*i)++) = *c;
            putchar(*c);
        }
        *c = getch();
        if (*i > *i_dupl)
            *i_dupl = *i;
    }
}

```

```

void movements(int *i,int *inc,int *col,char *record)

```

```

{
/***** for the movements of left-arrow &
* right-arrow in character type fields.
*****/
}

```

```

int asci=1;
asci = getch();
switch(asci)
    {
        case 77 :{
            if(*(record + *i +1) != ' ')
                {
                    (*i)++;
                    gotoxy(*col + NEXT + *i,3 + *inc);
                }
            break;
        }
        case 75 :{
            if (*(record + *i) == ' ')
                putchar(*(record + *i));
            (*i)--;
            gotoxy(*col +NEXT + *i,3 + *inc);
        }
    }
}

```

```

int year(int y)
{
    if (y%4 == 0)
        return(0);
    else return(1);
}

```

```
int month(int m)
```

```
{
```

```
    if (m>0 && m<13)
```

```
        return(1);
```

```
    else return(0);
```

```
}
```

```
int date_of_month(int flag_y,int m,int d)
```

```
{
```

```
    if (m>0 && m<8)
```

```
    {
```

```
        if (((d - m%2 ) <= 30 ) && (m != 2)) || ((m == 2) && ((flag_y + d ) <= 29
```

```
            return(1);
```

```
        else return(0);
```

```
    }
```

```
    else if ((m > 7 && m < 13 ) && (( d + m%2 ) <= 31 ))
```

```
        return(1);
```

```
    else return(0);
```

```
}
```

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <io.h>
#include <fcntl.h>
#include <sys/stat.h>
#define MAX_VAR_LEN 15
#define FIELD_SPECIF_SIZE 24
extern unsigned _stklen = 36499;
extern unsigned _heaplen = 0;
typedef struct {
    char var_name[MAX_VAR_LEN+1];
    char var_type[2];
    char var_length[3];
    char var_decimal[3];
} var_rec;

#define BUFF (btree *)malloc(sizeof (btree))
#define cur_pre ( strcmp( cur->c , pre->c ))
#define new_cur ( strcmp( new->c , cur->c ))
#define WHILE while(((( new_cur >= 0 ) && ( cur->rch != NULL )) ||
                    (( new_cur <= 0 ) && ( cur->lch != NULL )))

char file[] = "dpt.c" ;
char file2[] = "cats.c";
typedef struct tree {
    char *c;
    long rec_addr;
    int rec_no;
    struct tree *lch;

```

```

        struct tree *rch;
    } btree;

btree *root, *new, *pre, *cur;
int x,t,y,key_len,rec_counter = 1,fd,fd2,no_of_fields,len;
int no_of_records, s_counter = 1;
char flag = 't';
char *key_name;
long skip_bytes , init_addr,tot_fields_len,rec_start_loc,rec_addr;
int flag2 = 21;
    /** flag2 checks wheather key field exists or not ****/
/***** if flag2 remains 0 then it implies that key field
        does not exists in the dbf file *****/

void creates();
void write_heading();
void delete();
void insert ();
void retrieve();
void free_node(btree *cur );
long append(var_rec record[20],int field_len[20] )
{
    int k=0,len2;
    char rd_from_file[3];
    lseek(fd,201,0);
    if ((read(fd,rd_from_file,3))!=3)
        printf("ERROR");
    no_of_fields = atoi(rd_from_file);
    lseek(fd,31,1);
    key_len = tot_fields_len = 0;

```

```

len2 = strlen(key_name);
while (len2 < MAX_VAR_LEN )
    {
        *(key_name+len2) = ' ';
        len2++;
    }
*(key_name+MAX_VAR_LEN) = '\0';
while (k < no_of_fields)
    {
        read(fd,record[k].var_name,(MAX_VAR_LEN + 1));
        lseek(fd,21,1);
        read(fd,record[k].var_length,3);
        field_len[k] = atoi(record[k].var_length);
        tot_fields_len = tot_fields_len + field_len[k] ;
        len = strlen(record[k].var_name);
        if (strncmp(key_name,record[k].var_name,len) == 0 )
            {
                flag2 = k;
                key_len = field_len[k];
                init_addr = tot_fields_len - key_len;
            }
        lseek(fd,31,1);
        k++;
    }
    tot_fields_len++;
    return lseek(fd,01,1);
}
void rec_count()

```

```

{
    long end;
    end = lseek(fd, 0l, 2);
    no_of_records = (end - rec_start_loc) / tot_fields_len;
    skip_bytes = tot_fields_len - key_len;
}

void main(int argc, char *argv[])
{
    if (argc == 4)
    {
        fd = open(argv[1], O_RDONLY|O_BINARY, 0);
        fd2 = open(argv[2], O_RDWR|O_CREAT|S_IRREAD|S_IWRITE|O_TRUNC, 0);
        key_name = (char *)malloc(MAX_VAR_LEN + 1);
        key_name = argv[3];
        creates();
        if (flag2 != 21)
        {
            while (rec_counter <= no_of_records)
                insert();
            cur = root;
            rec_counter = 1;
            puts(" enter character ");
            getchar();
            write_heading();
            close(fd2);
            fd2 = open(argv[2], O_RDWR|O_BINARY|O_APPEND|O_CREAT
                |S_IRREAD|S_IWRITE, 0);
        }
    }
}

```

```

        retrieve(cur);
        close(fd2);
        close(fd);
        free_node(cur);
    }
}
}
void write_heading()
{
    write(fd2,key_name,(len+1));
    write(fd2,&key_len,sizeof(int));
}
void key_buff()
{
    var_rec rec1[20];
    int field_len[20],j,width;
    rec_start_loc = append( rec1,field_len );
    rec_count();
    rec_addr = lseek(fd,rec_start_loc,0);
    lseek(fd,init_addr,1);
}
void read_key_field(char *c, long *address)
{
    int width;
    if (read( fd,c,key_len ) != key_len )
        flag = 'f';
    else

```

```

        {
            *address = rec_addr;
            rec_addr = rec_addr + tot_fields_len;
            lseek(fd, skip_bytes, 1);
            rec_counter++;
        }
    }

void creates()
{
    root = BUFF;
    root->lch = root->rch = NULL;
    key_buff();
    if (( root->c = (char *) malloc ( key_len ) ) == NULL )
    {
        printf("out of memory");
        printf("%d", s_counter);
    }
    read_key_field( root->c, &(root->rec_addr) );
}

void insert()
{
    long skip;
    btree *new;
    cur = root;
    new = BUFF;

```

```

s_counter++;

if ( (new->c = (char *) malloc ( key_len )) == NULL )
{
    printf(" out of memory ");
    printf("%d",s_counter);
}
read_key_field( new->c,&(new->rec_addr) );
t = ( new_cur > 0 ) ? 1 : 0;
WHILE )          /***** WHILE is define as a macro *****/
{
    pre = cur;
    if (new_cur > 0)
        cur = cur->rch;
    else if (new_cur <= 0)
        cur = cur->lch;
/*
    else {
        printf(" data already present ");
        break;
    }
*/
}
if (new_cur > 0)
{
    cur->rch = new;
    new->lch = new->rch = NULL ;
}
else if (new_cur < 0)
{

```

```

    cur->lch = new;
    new->lch = new->rch = NULL;
}
}          /***** insert function terminates *****/

```

```

void retrieve( btree *cur )
{
    s_counter++;
    if ( cur->lch != NULL )
        retrieve( cur->lch );
    write ( fd2,cur->c,key_len);
    write ( fd2,&cur->rec_addr,sizeof(long));
    write ( fd2,&rec_counter,sizeof(int));
    if ( cur->rch != NULL )
        retrieve( cur->rch);
}

```

```

void free_node(btree *cur )
{
    if ( cur->lch != NULL )
        free_node( cur->lch );
    if ( cur->rch != NULL )
        free_node( cur->rch);
    free(cur);
}

```

