Aari_to_Ship.C

11.12.90

```
/*
**  File: Aari_to_Ship.c
**
**  Creation Data: 1990-12-11
**
**  Copying the data from the original Aari_Cruise table
**  to the new table Ship.
**  The Aari_Cruise tabel is renamed to Aari_Cruise_Bck.
*/

/*
**
**  INCLUDE FILES
**
*/

#include <stdio.h>
#include "opendb.h"
#include curses
#include <unixlib.h>
#include math
/*[@#include@]...*/

/*
**
**  MACRO DEFINITIONS
**
*/

/*{@#define@}..*/

/*[@preprocessor directive@]..*/

typedef struct ship_data {
    int     ship_id;
    char    ship [80],
            country [80];
    };

/*[@data type or declaration@]...*/

int result = 0;
```

```
/*
**++
**  FUNCTIONAL DESCRIPTION:
**
**      [@tbs@]...
**
**  FORMAL PARAMETERS:
**
**      [@description or none@]
**
**  IMPLICIT INPUTS:
**
**      [@description or none@]
**
**  IMPLICIT OUTPUTS:
**
**      [@description or none@]
**
**  {@function value or completion codes@}
**
**      [@description or none@]
**
**  SIDE EFFECTS:
**
**      [@description or none@]
**
**--
*/
main (int argc, char *argv[])
MAIN_PROGRAM
{
    extern int err_handler ();
    extern int msg_handler ();

    DBPROCESS * dbproc,
              * dbproc1;

    struct ship_data shipData;

/*    [@block declaration@]...*/

    dberrhandle(err_handler);
    dbmsghandle(msg_handler);

    dbproc  = opendb ("SouthernOceanDB", "sa", NULL);
    dbproc1 = opendb ("SouthernOceanDB", "sa", NULL);

    if ((dbproc == NULL) || (dbproc1 == NULL))
    {
        printf("\nThere is something wrong with the database. Abort!\n");
        exit(1);
/*      [@statement@]...*/
    }

    dbcmd(dbproc,"select distinct Ship, Country from Aari_Cruise");
    dbsqlexec(dbproc);
    dbresults(dbproc);
    dbbind(dbproc,1,STRINGBIND,0,shipData.ship);
    dbbind(dbproc,2,STRINGBIND,0,shipData.country);

    shipData.ship_id = 0;

    while (dbnextrow (dbproc) != NO_MORE_ROWS)
    {
        dbcmd(dbproc1,"insert into Ship");
```

```
        dbfcmd(dbproc1," values (%d,",++shipData.ship_id);
        dbfcmd(dbproc1," '%s',",shipData.ship);
        dbfcmd(dbproc1," '%s')",shipData.country);
        dbsqlexec(dbproc1);
    }
/*     [@statement@]... */

}
```

GORDON_CRUISE.C 05.02.91
Changed 05.01.91
22.11.90 TEST = 1
TODB removed
Run: 12.2.90 BACKFILL removed

SYS[USER.[KURBEISKI
.SOUTHGERMOCEAN
.C]

```
/*
**
**  INCLUDE FILES
**
*/

#include <stdio.h>
#include "opendb.h"
#include curses
#include <unixlib.h>
#include math
/*[@#include@]...*/

/*
**
**  MACRO DEFINITIONS
**
*/

/* TEST          nur Ausgabe der gelesenen Daten
** TODB          Ausgabe der gelesenen Daten und schreiben in die Datenbank
*/

/*#define TEST 1*/  /* Ausdruck der gesamten Daten auf dem Bildschirm */
                    /* Wenn nicht definiert, wird nur eine Auswahl angezeigt */

/* Die Symbole SHIP, CRUISE und BACKFILL steuern das Lesen und Ausgeben
** der Daten. Es gelten folgende Beziehungen
**
**     SHIP    CRUISE   BACKFILL     Definition
**      undef   undef     undef      Daten werden aus HEADERS.FIL gelesen und
**                                   fuer Gordon_Station aufbereitet.
**      undef    def      undef      Daten werden aus SHIPGORD3.DAT gelesen und
**                                   fuer Gordon_Cruise aufbereitet.
**      def     undef     undef      Daten werden aus SHIPGORD3.DAT gelesen und
**                                   fuer Gordon_Ship aufbereitet.
**      def      def      undef      illegal. Es geshieht nichts.
**      x        x          def      Laden der Informatinen aus HEADERS.FIL
**                                   in die Tabelle Gordon_Station_Backfill
*/

/*#define SHIP 1*/
/*#define CRUISE 1*/
#define BACKFILL 1

#define TODB 1 /* Wenn definiert erfolgt Transfer in die Datenbank*/

/*{@#define@}..*/

/*[@preprocessor directive@]..*/

typedef struct cruise_data {
    char    name [9];  /*laenger als die Definition in der DB wegen null-Char*/
    int     cruiseNumber,
            dummy;
    float   latitude,
            longitude;
    char    date [9];
    int     dayOfYear;
    char    time [6],
            comment [101];
    };

typedef struct gordon_cruise_data {
```

```
     int  cruiseNumber,
          shipId;
     char cruiseName [6],
          comment [100];
 };

typedef struct gordon_ship {
     int  shipId;
     char shipName [30],
          country [30];
 };

typedef struct ship {
    int  shipId;
    char shipcode [6],
         shipName [30];
    int  cruiseNumber;
    char country [30];
} ;

typedef char str [4];
typedef str strar [12];

char *strsub ( char * source, int start, int stop );

/*[@data type or declaration@]...*/

int todb = 0;
int result = 0;

static strar ar = {"JAN", "FEB", "MAR", "APR", "MAY", "JUN",
                   "JUL", "AUG", "SEP", "OCT", "NOV", "DEC"};
```

```
/*
**++
**  FUNCTIONAL DESCRIPTION:
**
**      [@tbs@]...
**
**  FORMAL PARAMETERS:
**
**      [@description or none@]
**
**  IMPLICIT INPUTS:
**
**      [@description or none@]
**
**  IMPLICIT OUTPUTS:
**
**      [@description or none@]
**
**  {@function value or completion codes@}
**
**      [@description or none@]
**
**  SIDE EFFECTS:
**
**      [@description or none@]
**
**--
*/
main (int argc, char *argv[])
MAIN_PROGRAM
{
    const char * fileNameFil = "HEADERS.FIL",
               * fileShipName = "SHIPGORD3.DAT";

    extern int err_handler ();
    extern int msg_handler ();

    char c;

    int * shipId,
        * gordonCruiseNumber;

    DBPROCESS * dbproc;

    WINDOW * win, * win1;

    struct cruise_data cruiseData;
    struct gordon_cruise_data gordonCruisedata;
    struct gordon_ship gordonShip;
/*    [@block declaration@]...*/

    gordonCruiseNumber = (int *) malloc(sizeof(int));
    *gordonCruiseNumber = 0;
    shipId = (int *) malloc(sizeof(int));
    *shipId = 0;

#ifdef TODB
    dberrhandle(err_handler);
    dbmsghandle(msg_handler);

    dbproc = opendb ("SouthernOceanDB", "sa", NULL);
    if (dbproc == NULL)
    {
        getchar();
        endwin();
```

if (TODB)
{

}

```
/* Diese Zeilen werden nicht mehr erreicht, da opendb bei einem
   falschem login (Name und/oder Passwort falsch) bereits aussteigt. */
        printf("\nThere is something wrong with the database. Abort!\n");
        exit(1);
/*      [@statement@]...*/
    }
#endif

    initscr();
    win = subwin(stdscr,12,40,5,2);
    win1 = subwin(stdscr,3,40,20,1);


#ifndef BACKFILL
#ifndef SHIP
#ifndef CRUISE
{
    mvwaddstr(win1,1,1,"open file G ");
    mvwaddstr(win1,1,14,fileNameFil);
    wrefresh(win1);

#ifdef TODB
    connectToDB (dbproc,
                 "select max(Ship_Id#) from Gordon_Ship",
                 shipId,
                 INTBIND,
                 0);
#endif

    writeCruiseDataToDB (dbproc, fileNameFil, shipId);

#endif
#endif

#ifndef CRUISE
#ifdef SHIP
    mvwaddstr(win1,1,1,"open file S ");
    mvwaddstr(win1,1,14,fileShipName);
    wrefresh(win1);

#ifdef TODB
    connectToDB (dbproc,
                 "select max(Gordon_Cruise_Id#) from Gordon_Cruise",
                 gordonCruiseNumber,
                 INTBIND,
                 0);
#endif

   . writeShipDataToDB (dbproc, fileShipName, gordonCruiseNumber);

#endif
#endif

#ifdef CRUISE
#ifndef SHIP

    mvwaddstr(win1,1,1,"open file C ");
    mvwaddstr(win1,1,14,fileShipName);
    wrefresh(win1);

#ifdef TODB
    connectToDB (dbproc,
                 "select max(Gordon_Cruise_Id#) from Gordon_Cruise",
                 gordonCruiseNumber,
                 INTBIND,
                 0);
```

```
#endif

    writeShipAndCruiseDataToDB (dbproc, fileShipName, gordonCruiseNumber);
#endif
#endif
#endif

#ifdef BACKFILL

    connectToDB (dbproc,
                 "select max(Gordon_Station_Id#) from Gordon_Station_Backfill",
                 gordonCruiseNumber,
                 INTBIND,
                 0);

    writeBackfillDataToDB (dbproc, fileNameFil, gordonCruiseNumber);

#endif

    endwin();
/*    [@statement@]...*/
}
```

```
/*
**++
**  FUNCTIONAL DESCRIPTION:
**
**      Bindet Daten aus einer Datenbank an eine Variable.
**
**  FORMAL PARAMETERS:
**
**      dbproc      DBPROC      Datenbankreferenz
**      question    char *      Anfrage an die Datanbank
**      idStat      void *      Zeiger auf eine Integer (Nummer)
**      binding     int         Datentype fuer die Antwort
**      len         DBINT       reservierter Platz fuer idStat

**  IMPLICIT INPUTS:
**
**      [@description or none@]
**
**  IMPLICIT OUTPUTS:
**
**      [@description or none@]
**
**  COMPLETION CODES:
**
**      none
**
**  SIDE EFFECTS:
**
**      none
**
**--
*/
 connectToDB (DBPROCESS * dbproc,
                char *question,
                void *idStat,
                int binding,
                DBINT len)
{
/*    [@block declaration@]...*/

    dbcmd(dbproc,question);
    dbsqlexec(dbproc);
    dbresults(dbproc);
    dbbind(dbproc,1,binding,len,idStat);
    dbnextrow(dbproc);

/*    [@statement@]...*/
}
```

```
/*
**++
**  FUNCTIONAL DESCRIPTION:
**
**      writeBackfillDataToDB
**
**  FORMAL PARAMETERS:
**
**      DBPROCESS * dbproc  Datenbankreferenz
**      char * filename     Eingabefile
**      int * id
**      [@tbs@]...
**
**  IMPLICIT INPUTS:
**
**      [@description or none@]
**
**  IMPLICIT OUTPUTS:
**
**      [@description or none@]
**
**  {@function value or completion codes@}
**
**      [@description or none@]
**
**  SIDE EFFECTS:
**
**      [@description or none@]
**
**--
*/
 writeBackfillDataToDB (DBPROCESS * dbproc,
                        char * filename,
                        int * shipId)
{
    struct  cruise_data cruiseData;

    char datetime[22],
        dummy [22],
        sample [3],
        * datetimePnt,
        * datePnt;

    int * gordonCruiseId,
        * gordonStationId,
        month,
        day,
        year,
        hour,
        * depth,
        * sign,
        stationIndex = 0,
        count = 0;


    FILE * fp, * fpo;

    WINDOW * win;

    const * fmt = "%5s %i %i %f %f%*c%8s %i%*c%5s%*2c%11s ";
/*    [@block declaration@]...*/
```

```
    fp = fopen(filename, "ra");
    fpo = fopen ("gcheader.log", "w");

    win = subwin(stdscr,13,40,5,2);

    box(win,'|','-');

    depth = (int *) malloc(sizeof(int));
    *depth = 4;
    sign = (int *) malloc(sizeof(int));
    *sign = 0;
    gordonCruiseId = (int *) malloc(sizeof(int));
    *gordonCruiseId = 0;
    gordonStationId = (int *) malloc(sizeof(int));
    *gordonStationId = 0;

    while (fscanf (fp,
                   fmt,
                   cruiseData.name,
                   &cruiseData.cruiseNumber,
                   &cruiseData.dummy,
                   &cruiseData.latitude,
                   &cruiseData.longitude,
                   cruiseData.date,
                   &cruiseData.dayOfYear,
                   cruiseData.time,
                   cruiseData.comment) != EOF)
    {
        strcpy(datetime,"       19     :  AM\0");
        datetimePnt = datetime + 9;
        strncpy(datetimePnt, strtok(cruiseData.date,"/"), 2);
        datetimePnt = datetime;
        strncpy(datetimePnt, ar[atoi(strtok(NULL,"/")) - 1], 3);
        datetimePnt = datetime + 4;
        strncpy(datetimePnt, strtok(NULL,"/"), 2);
        datetimePnt = datetime + 12;
        strncpy(datetimePnt, cruiseData.time, 5);
        hour = atoi(strtok(cruiseData.time, ":"));
        if ( hour > 12 )
        {
            hour = hour - 12;
            datetime [17] = 'P';
            *depth = 2;
            *sign = 1;
            sample [0] = '\0';
            if (hour >= 10)
            {
                strcat (sample, fcvt((double) hour, 0, depth, sign));
            }
            else
            {
                strcat (strcat(sample, "0"),
                        fcvt((double) hour, 0, depth,sign));
            }
            strncpy(datetimePnt, sample, 2);
        }

#ifdef TODB
        dbcmd (dbproc, " insert into Gordon_Station_Backfill values (");
        dbfcmd (dbproc, "  %d,", cruiseData.cruiseNumber);
        dbfcmd (dbproc, " '%s',", cruiseData.name);
        dbfcmd (dbproc, " %f,", cruiseData.longitude);
        dbfcmd (dbproc, " %f,", cruiseData.latitude);
        if (dbconvert(dbproc, SYBCHAR, datetime, strlen(datetime),
                        SYBDATETIME, dummy, strlen(datetime)) != -1)
```

```
        {
            dbfcmd (dbproc, " '%s',", datetime);
        }
        else
        {
            dbfcmd (dbproc, " '%s',", "");
            fprintf (fpo, "%d \t%s \t %f \t%f \t%d\n",
                        cruiseData.cruiseNumber,
                        cruiseData.name,
                        cruiseData.longitude,
                        cruiseData.latitude,
                        datetime,
                        cruiseData.dayOfYear);
        }
        dbfcmd (dbproc, " %d)", cruiseData.dayOfYear);
        if (dbsqlexec (dbproc) == FAIL)
        {
            getch();
            endwin();
            printf("\nsqlexec failed\n");
            exit(2);
        }
        if ((result = dbresults(dbproc)) == FAIL)
        {
            getch();
            endwin();
            printf("\nresults failed\n");
            exit(2);
        }
#endif

        ++stationIndex;
        *sign = 1;
        fcvt((double)(cruiseData.cruiseNumber),0,depth,sign);
        mvwaddstr(win,2,1,fcvt((double)(cruiseData.cruiseNumber),0,depth,sign));

#ifndef TEST
        if (++count > 100)
        {
#endif
            mvwaddstr(win,1,1,cruiseData.name);
            *sign = cruiseData.dummy >=0 ? 1 : -1;
            mvwaddstr(win,3,1,fcvt((double)cruiseData.dummy,0,depth,sign));
            *sign = cruiseData.latitude >=0 ? 1 : -1;
            *depth = fabs(cruiseData.latitude) >= 100 ? 3 : 2;
            mvwaddstr(win,4,1,fcvt(cruiseData.latitude,4,depth,sign));
            *sign = cruiseData.longitude >=0 ? 1 : -1;
            *depth = fabs(cruiseData.longitude) >= 100 ? 3 : 2;
            mvwaddstr(win,5,1,fcvt(cruiseData.longitude,4,depth,sign));
            mvwaddstr(win,6,1,cruiseData.date);
            *sign = 1;
            mvwaddstr(win,7,1,fcvt((double)cruiseData.dayOfYear,0,depth,sign));
            mvwaddstr(win,8,1,cruiseData.time);
            mvwaddstr(win,9,1,cruiseData.comment);
            *sign = 1;
            mvwaddstr(win,10,1,fcvt((double)stationIndex,0,depth,sign));
            mvwaddstr(win,11,1,datetime);
#ifndef TEST
            count = 0;
        }
#endif
        wrefresh(win);
/*      [@statement@]...*/
    }

    fclose(fpo);
```

if (#TEST || (++count > 100))
{
}

```
    fclose(fp);
/*    [@statement@]...*/
}
```

```
/*
**++
**  FUNCTIONAL DESCRIPTION:
**
**      writeCruiseDataToDB
**
**  FORMAL PARAMETERS:
**
**      DBPROCESS * dbproc  Datenbankreferenz
**      char * filename     Eingabefile
**      int * id
**      [@tbs@]...
**
**  IMPLICIT INPUTS:
**
**      [@description or none@]
**
**  IMPLICIT OUTPUTS:
**
**      [@description or none@]
**
**  {@function value or completion codes@}
**
**      [@description or none@]
**
**  SIDE EFFECTS:
**
**      [@description or none@]
**
**--
*/
 writeCruiseDataToDB (DBPROCESS * dbproc,
                      char * filename,
                      int * shipId)
{
    struct  cruise_data cruiseData;

    char datetime[22],
        sample [3],
        * datetimePnt,
        * datePnt;

    int * gordonCruiseId,
        * gordonStationId,
        month,
        day,
        year,
        hour,
        * depth,
        * sign,
        stationIndex = 0,
        count = 0;


    FILE * fp, * fpo;

    WINDOW * win;

    const * fmt = "%5s %i %i %f %f%*c%8s %i%*c%5s%*2c%11s ";
/*    [@block declaration@]...*/
```

```
    fp = fopen(filename, "ra");
    fpo = fopen ("gcheader.log", "w");

    win = subwin(stdscr,13,40,5,2);

    box(win,'|','-');

    depth = (int *) malloc(sizeof(int));
    *depth = 4;
    sign = (int *) malloc(sizeof(int));
    *sign = 0;
    gordonCruiseId = (int *) malloc(sizeof(int));
    *gordonCruiseId = 0;
    gordonStationId = (int *) malloc(sizeof(int));
    *gordonStationId = 0;

    while ((fscanf (fp,
                    fmt,
                    cruiseData.name,
                    &cruiseData.cruiseNumber,
                    &cruiseData.dummy,
                    &cruiseData.latitude,
                    &cruiseData.longitude,
                    cruiseData.date,
                    &cruiseData.dayOfYear,
                    cruiseData.time,
                    cruiseData.comment) != EOF) && (cruiseData.cruiseNumber < 100))
    {
        strcpy(datetime,"       19     :  AM\0");
        datetimePnt = datetime + 9;
        strncpy(datetimePnt, strtok(cruiseData.date,"/"), 2);
        datetimePnt = datetime;
        strncpy(datetimePnt, ar[atoi(strtok(NULL,"/")) - 1], 3);
        datetimePnt = datetime + 4;
        strncpy(datetimePnt, strtok(NULL,"/"), 2);
        datetimePnt = datetime + 12;
        strncpy(datetimePnt, cruiseData.time, 5);
        hour = atoi(strtok(cruiseData.time, ":"));
        if ( hour > 12 )
        {
            hour = hour - 12;
            datetime [17] = 'P';
            *depth = 2;
            *sign = 1;
            sample [0] = '\0';
            if (hour >= 10)
            {
                strcat (sample, fcvt((double) hour, 0, depth, sign));
            }
            else
            {
                strcat (strcat(sample, "0"),
                        fcvt((double) hour, 0, depth,sign));
            }
            strncpy(datetimePnt, sample, 2);
        }

#ifdef TODB
        dbcmd (dbproc, "select Gordon_Station_Id# from Gordon_Station ");
        dbfcmd (dbproc, " where Latitude between %d and %d ",
                cruiseData.latitude - 0.005, cruiseData.latitude + 0.005);
        dbfcmd (dbproc, "  and Longitude between %d and %d ",
                cruiseData.longitude - 0.005, cruiseData.longitude + 0.005);
        dbfcmd (dbproc, " and Date_Time = '%s'",strsub(datetime,0,10));
```

```
        if (dbsqlexec (dbproc) == FAIL)
        {
            getch();
            endwin();
            printf("\nsqlexec failed\n");
            closedb(dbproc);
            exit(2);
        }
        if ((result = dbresults(dbproc)) == FAIL)
        {
            getch();
            endwin();
            printf("\nresults failed\n");
            closedb(dbproc);
            exit(2);
        }
        dbbind (dbproc, 1, INTBIND, 0, gordonStationId);
        if (dbnextrow (dbproc) != NO_MORE_ROWS)
        {
            fprintf(fpo, "Nr: %d \t%s \t%d \t%s\n",
                *gordonStationId,
                cruiseData.name,
                cruiseData.cruiseNumber,
                strsub(datetime,0,10));
        }
#endif

        ++stationIndex;
#ifndef TEST        if (++count > 100)
        {
#endif
            mvwaddstr(win,1,1,cruiseData.name);
            *sign = 1;
            fcvt((double)(cruiseData.cruiseNumber),0,depth,sign);
            mvwaddstr(win,2,1,fcvt((double)(cruiseData.cruiseNumber),0,depth,sign));
            *sign = cruiseData.dummy >=0 ? 1 : -1;
            mvwaddstr(win,3,1,fcvt((double)cruiseData.dummy,0,depth,sign));
            *sign = cruiseData.latitude >=0 ? 1 : -1;
            *depth = fabs(cruiseData.latitude) >= 100 ? 3 : 2;
            mvwaddstr(win,4,1,fcvt(cruiseData.latitude,4,depth,sign));
            *sign = cruiseData.longitude >=0 ? 1 : -1;
            *depth = fabs(cruiseData.longitude) >= 100 ? 3 : 2;
            mvwaddstr(win,5,1,fcvt(cruiseData.longitude,4,depth,sign));
            mvwaddstr(win,6,1,cruiseData.date);
            *sign = 1;
            mvwaddstr(win,7,1,fcvt((double)cruiseData.dayOfYear,0,depth,sign));
            mvwaddstr(win,8,1,cruiseData.time);
            mvwaddstr(win,9,1,cruiseData.comment);
            *sign = 1;
            mvwaddstr(win,10,1,fcvt((double)stationIndex,0,depth,sign));
            mvwaddstr(win,11,1,datetime);
            wrefresh(win);
#ifndef TEST
            count = 0;
        }
#endif
        dbcanquery (dbproc);
/*      [@statement@]...*/
    }

    fclose(fp);
/*    [@statement@]...*/
}
```

```
/*
**++
**  FUNCTIONAL DESCRIPTION:
**
**      writeShipDataToDB
**      [@tbs@]...
**
**  FORMAL PARAMETERS:
**
**      DBPROCESS * dbproc
**      char * filename
**      int * id
**
**  IMPLICIT INPUTS:
**
**      none
**
**  IMPLICIT OUTPUTS:
**
**      none
**
**  COMPLETION CODES:
**
**      none
**
**  SIDE EFFECTS:
**
**      none
**
**--
*/
 writeShipDataToDB (DBPROCESS * dbproc,
                    char * filename,
                    int * gordonId)
{
    struct ship gordonShip;

    FILE * fp;

    WINDOW * win;

    const * fmt = "%d%*2c%5s%*2c%25c %d%*3c%12c ";

    int * sign,
        * depth,
        count = 0;
/*    [@block declaration@]...*/
```

```
    depth = (int *) malloc(sizeof(int));
    *depth = 4;
    sign = (int *) malloc(sizeof(int));
    *sign = 0;

    fp = fopen(filename, "ra");

    win = subwin(stdscr,12,31,5,2);

    box(win,'|','-');

    while (fscanf (fp,
                   fmt,
                   &gordonShip.shipId,
                   gordonShip.shipcode,
                   gordonShip.shipName,
                   &gordonShip.cruiseNumber,
                   gordonShip.country) != EOF)
    {
        gordonShip.shipcode[5] = '\0';
        gordonShip.shipName[25] = '\0';
        gordonShip.country[12] = '\0';

#ifdef TODB
        dbcmd (dbproc, " insert into Gordon_Ship values ( ");
        dbfcmd (dbproc, "  %d,", gordonShip.shipId);
        dbfcmd (dbproc, " '%s',", gordonShip.shipcode);
        dbfcmd (dbproc, " '%s',", gordonShip.shipName);
        dbfcmd (dbproc, " '%s')", gordonShip.country);
        if (dbsqlexec (dbproc) == FAIL)
        {
            getch();
            endwin();
            printf("\nsqlexec failed\n");
            exit(2);
        }
        if ((result = dbresults(dbproc)) == FAIL)
        {
            getch();
            endwin();
            printf("\nresults failed\n");
            exit(2);
        }
#endif

#ifndef TEST
        if (++count > 10)
        {
#endif
            *sign = 1;
            mvwaddstr(win,1,1,fcvt((double)gordonShip.shipId,0,depth,sign));
            mvwaddstr(win,2,1,gordonShip.shipcode);
            mvwaddstr(win,3,1,gordonShip.shipName);
            *sign = 1;
            mvwaddstr(win,4,1,
                      fcvt((double)gordonShip.cruiseNumber,0,depth,sign));
            mvwaddstr(win,5,1,gordonShip.country);
            wrefresh(win);
            count = 0;
#ifndef TEST
        }
#endif
/*      [@statement@]...*/

    }
```

```
/*      [@statement@]...*/
}
```

```
/*
**++
**  FUNCTIONAL DESCRIPTION:
**
**      writeShipAndCruiseDataToDB
**      [@tbs@]...
**
**  FORMAL PARAMETERS:
**
**      DBPROCESS * dbproc  Referenz zur Datenbank
**      char * filename
**      int * id
**      [@tbs@]...
**
**  IMPLICIT INPUTS:
**
**      none
**
**  IMPLICIT OUTPUTS:
**
**      none
**
**  COMPLETION CODES:
**
**      none
**
**  SIDE EFFECTS:
**
**      none
**
**--
*/
 writeShipAndCruiseDataToDB (DBPROCESS * dbproc,
                     char * filename,
                     int * gordonId)
{
    struct ship gordonShip;

    FILE * fp;

    WINDOW * win;

    const * fmt = "%d%*2c%5s%*2c%25c %d%*3c%12c ";

    int * sign,
        * depth,
        count = 0;
/*    [@block declaration@]...*/
```

```
        depth = (int *) malloc(sizeof(int));
        *depth = 4;
        sign = (int *) malloc(sizeof(int));
        *sign = 0;

        fp = fopen(filename, "ra");

        win = subwin(stdscr,12,31,5,2);

        box(win,'|','-');

        while (fscanf (fp,
                       fmt,
                       &gordonShip.shipId,
                       gordonShip.shipcode,
                       gordonShip.shipName,
                       &gordonShip.cruiseNumber,
                       gordonShip.country) != EOF)
        {
            gordonShip.shipcode[5] = '\0';
            gordonShip.shipName[25] = '\0';
            gordonShip.country[12] = '\0';

#ifdef TODB
            dbcmd (dbproc, " insert into Gordon_Cruise (");
            dbfcmd (dbproc, " Gordon_Cruise_Id#, Ship_Code) ");
            dbfcmd (dbproc, " values ( ");
            dbfcmd (dbproc, "  %d,", gordonShip.cruiseNumber);
            dbfcmd (dbproc, " '%s')", gordonShip.shipcode);
            if (dbsqlexec (dbproc) == FAIL)
            {
                getch();
                endwin();
                printf("\nsqlexec failed\n");
                exit(2);
            }
            if ((result = dbresults(dbproc)) == FAIL)
            {
                getch();
                endwin();
                printf("\nresults failed\n");
                exit(2);
            }
#endif

#ifndef TEST
            if (++count > 10)
            {
#endif
                *sign = 1;
                mvwaddstr(win,1,1,fcvt((double)gordonShip.shipId,0,depth,sign));
                mvwaddstr(win,2,1,gordonShip.shipcode);
                mvwaddstr(win,3,1,gordonShip.shipName);
                *sign = 1;
                mvwaddstr(win,4,1,
                          fcvt((double)gordonShip.cruiseNumber,0,depth,sign));
                mvwaddstr(win,5,1,gordonShip.country);
                wrefresh(win);
                count = 0;
#ifndef TEST
            }
#endif
/*          [@statement@]...*/

        }
```

```
/*      [@statement@]...*/
}
```

```
    }
    return (dest);
}

/*[@function definition@]...*/
```

CRUISE.C

18.6.91

```
/*
**++
**  FACILITY:
**
**      cruise.c
**      [@tbs@]
**
**  ABSTRACT:
**
**      Converting the Aari_Cruise table to the table Ship.
**      Creating and removing of tables and views
**      is done outside of this program.
**
**  AUTHORS:
**
**      Lutz-Peter Kurdelski
**      Alfred-Wegener-Institute for Polar and Marine Research
**      Am Handelshafen 12
**      D-2850 Bremerhaven
**
**  CREATION DATE:
**
**      1990-12-11
**
**  MODIFICATION HISTORY
**
**--
*/

/*
**
**  INCLUDE FILES
**
*/

#include <stdio.h>
#include "opendb.h"
#include curses
#include <unixlib.h>
#include math
/*[@#include@]...*/

/*
**
**  MACRO DEFINITIONS
**
*/

/*{@#define@}..*/

/*[@preprocessor directive@]..*/

typedef struct ship_data {
    int     ship_id;
    char    ship [80],
            country [80];
    };

/*[@data type or declaration@]...*/

int result = 0;
```

C.3

```
/*
**++
**  FUNCTIONAL DESCRIPTION:
**
**      [@tbs@]...
**
**  FORMAL PARAMETERS:
**
**      [@description or none@]
**
**  IMPLICIT INPUTS:
**
**      [@description or none@]
**
**  IMPLICIT OUTPUTS:
**
**      [@description or none@]
**
**  {@function value or completion codes@}
**
**      [@description or none@]
**
**  SIDE EFFECTS:
**
**      [@description or none@]
**
**--
*/
main (int argc, char *argv[])
MAIN_PROGRAM
{
    extern int err_handler ();
    extern int msg_handler ();

    DBPROCESS * dbproc,
              * dbproc1;

    struct ship_data shipData;

/*    [@block declaration@]...*/

    dberrhandle(err_handler);
    dbmsghandle(msg_handler);

    dbproc  = opendb ("SouthernOceanDB", "sa", NULL);
    dbproc1 = opendb ("SouthernOceanDB", "sa", NULL);

    if ((dbproc == NULL) || (dbproc1 == NULL))
    {
        printf("\nThere is something wrong with the database. Abort!\n");
        exit(1);
/*      [@statement@]...*/
    }

    dbcmd(dbproc,"select distinct Ship, Country from Aari_Cruise");
    dbsqlexec(dbproc);
    dbresults(dbproc);
    dbbind(dbproc,1,CHARBIND,0,shipData.ship);
    dbbind(dbproc,2,CHARBIND,0,shipData.country);

    shipData.ship_id = 0;

    while (dbnextrow (dbproc) != NO_MORE_ROWS)
    {
        dbcmd(dbproc1,"insert into Ship");
```

```
        dbfcmd(dbproc1," values (%d,",++shipData.ship_id);
        dbfcmd(dbproc1," '%s',",shipData.ship);
        dbfcmd(dbproc1," '%s')",shipData.country);
        dbsqlexec(dbproc1);
    }
/*    [@statement@]... */

}
```

12.06.91
Biomass.c LPK

```
/*
**++
**  FACILITY:
**
**       biomass.c
**      [@tbs@]...
**
**  ABSTRACT:
**
**      Loading the biomass data into the SouthernOceanDB
**      As given by Mark Thorley, BIOMASS Data Centre from 1991-05-21
**      the following data can be loaded
**
**      FIBEX       HEFX, HOFX, ITFX, ODFX, SIFX
**      SIBEX 1     ACS1, BES1, PSS1, SIS1
**      SIBEX 2     CHS2, BES2, HES2, JBS2, ACS2, KMS2, PSS2
**
**
**      The loading process will be performed as follows:
**      1.      opening the database connection
**      2.      for an collection of files try to load the data
**      2.1.    read the data specification line
**      2.2.    read the data from the file
**      2.3.    write the data as defined in the specification to the db
**      3.      close the database connection
**
**      [@tbs@]...
**
**  AUTHORS:
**
**      Lutz-Peter Kurdelski,
**      Alfred-Wegener-Institute for Polar and Marine Research
**      Am Handelshafen 12
**      D-2850 Bremerhaven
**      [@tbs@]...
**
**
**  CREATION DATE:      1991-05-28
**
**  MODIFICATION HISTORY:
**
**--
*/


/*
**
**  INCLUDE FILES
**
*/

#include <stdio.h>
#include <errno.h>
#include "opendb.h"
#include curses
#include math
#include stdlib
#include <unixlib.h>
/*[@#include@]...*/


/*
**
**  MACRO DEFINITIONS
**
```

```
*/

#define cruisenameLength          5
#define tokenLength              16
#define recordLength            300
#define maximumValues            28
#define falsum                    0
#define verum                     1

#define SalinityLimit          30.0

#define minStationId         800000
#define minDataId           8000000

/*[@#define@]...*/

/*[@preprocessor directive@]...*/

typedef struct data {
    int     Station_Id,          /* for internal use */
            Data_Id,             /* for internal use */
            Cruise_Number,       /* for internal use */
            Depth,               /* 6 */
            PrDepth;             /* 7 */
    char    Cruise_Name [8],     /* cruise, for internal use */
            ObsType [2],         /* 8 */
            DepthType [2],       /* 9 */
            Station [8],         /* 1 */
            Date [8],            /* 2 */
            Time [8];            /* 3 */
    float   Latitude,            /* 4 */
            Longitude,           /* 5 */
            Temp,                /* 10 */
            Salinity,            /* 11 */
            Oxygen,              /* 12 */
            InPhos,              /* 13 */
            Nitrite,             /* 14 */
            Nitrate,             /* 15 */
            Ammonium,            /* 16 */
            Silicate,            /* 17 */
            PH,                  /* 18 */
            Chloride,            /* 19 */
            Chlorophyll_A,       /* 20 */
            Sigma_T,             /* 21 */
            Sigma_T_Obs,         /* 22 */
            Sound_Velocity,      /* 23 */
            Dynamic_Height,      /* 24 */
            Sigma_Theta,         /* 25 */
            Pot_Temp,            /* 26 */
            Spec_Vol_Anomaly,    /* 27 */
            Delta_D              /* 28 */;
};

typedef char str4 [4];
typedef str4 strar [12];

WINDOW * win, * win1;

static strar months = {"JAN", "FEB", "MAR", "APR", "MAY", "JUN",
                       "JUL", "AUG", "SEP", "OCT", "NOV", "DEC"};

strsub ( char *, int, int);

connectToDB (DBPROCESS * dbproc,
             char * question,
             void * idstat,
```

```
int binding,
DBINT i);
```

```
/*
**++
**  FUNCTIONAL DESCRIPTION:
**
**      main
**      Reading the Biomass data and inserting into the database.
**      See also comment at the beginning of the program.
**      [@tbs@]...
**
**  FORMAL PARAMETERS:
**
**      none
**
**  IMPLICIT INPUTS:
**
**      FIBEX       HEFX, HOFX, ITFX, ODFX, SIFX
**      SIBEX 1     ACS1, BES1, PSS1, SIS1
**      SIBEX 2     CHS2, BES2, HES2, JBS2, ACS2, KMS2, PSS2
**      [@tbs@]...
**
**  IMPLICIT OUTPUTS:
**
**      [@description or none@]
**
**  {@function value or completion codes@}
**
**      [@description or none@]
**
**  SIDE EFFECTS:
**
**      [@description or none@]
**
**--
*/
main ()
{
    struct data values;

    char *files [16] = { "HEFX", "HOFX", "ITFX", "ODFX", "SIFX", "ACS1",
                         "BES1", "PSS1", "SIS1", "CHS2", "HES2", "BES2",
                         "JBS2", "ACS2", "KMS2", "PSS2" };

    FILE * fp,
         * log;

    char fn [8],
         c;

    int i,
        result,
        cruiseNr = 0;

    extern bool todb;

    extern int err_handler ();
    extern int msg_handler ();

    DBPROCESS * dbproc;

    i = 0;

    do
    {
        printf("Output to Database ? [Y or N] ");
        c = (char) getc(stdin);
```

```
        c = c & ('\xFF' - ' ');
    } while ((c != 'Y') &&
            (c != 'J') && (c != 'N') && (i++ <= 3));

    if (i > 3)
    {
        printf("Three invalid tries, don't be so silly\n");
        exit(3);
    }

    if (c == 'Y' || c == 'J')
    {
        todb = verum;
    }
    else
    {
        todb = falsum;
    }

    if (todb)
    {
        dberrhandle(err_handler);
        dbmsghandle(msg_handler);

        if ((dbproc = opendb ("SouthernOceanDB", "sa", NULL)) == NULL)
        {
            printf("There was something wrong while accessing the database\n");
            getc(stdin);
            exit(1);
        }
    }

/*    [@block declaration@]..*/

    log = fopen ("biomass.err", "w");

    for (i = 0; i <= 15; i++)
    {
        strcpy(fn,files[i]);
        strcat(fn,".LIS");
        if ((fp = fopen (fn, "r")) == NULL)
        {
            printf("file %s not found\n",fn);
            fprintf(log, "ERROR: file %s not found\n",fn);
        }
        else
        {
            values.Cruise_Number = ++ cruiseNr;
            strcpy (values.Cruise_Name, files [i]);
            if ((result = scanfile (fp, &values, log, dbproc)) > 0)
            {
                fprintf(log,"FILE_ERROR: Some data errors found %s.LIS\n",
                        files [i]);
            };
            if (result < 0)
            {
                fprintf(log,"FILE_ERROR: File %s.LIS\ contains no valid data\n",
                        files [i]);
            };
            fclose(fp);
        }
    }
    fclose(log);
}
```

```
/*
**++
**  FUNCTIONAL DESCRIPTION:
**
**      [@tbs@]...
**
**  FORMAL PARAMETERS:
**
**      [@description or none@]
**
**  IMPLICIT INPUTS:
**
**      [@description or none@]
**
**  IMPLICIT OUTPUTS:
**
**      [@description or none@]
**
**  COMPLETION CODES:
**
**      0   file was scanned without error
**      >0  file was scanned with some error
**      <0  file could not be scanned
**      [@tbs@]...
**
**  SIDE EFFECTS:
**
**      [@description or none@]
**
**--
*/
int scanfile (FILE * fp,
          struct data * values,
          FILE * log,
          DBPROCESS * dbproc)
{
    char * seriousProblem = " SERIOUS PROBLEM. WHY ?",
         * emptyLine      = " empty line",
         * noNumber       = " line contains no number",
         * noValidCruise  = " no valid cruise number",
         * lineEnded      = " line ended before scanning was complete",
         * blankString    = "                                          ";

    extern WINDOW * win,
                  * win1;

    int i,
        k,
        result,
        station_id,
        data_id;

    extern bool todb;
    bool errorFlag = falsum,
         anyCorrectLine = verum,
         anyLine = falsum;

    char line [recordLength + 1],
         statNr [8],
         l [12],
         * ll;

    fprintf(log,"FILE: %s\n", values -> Cruise_Name);

    initscr();
```

```
    win1 = subwin(stdscr,3,6,2,20);
    box(win1,'*','*');
    mvwaddstr(win1,1,1,&(values -> Cruise_Name));
    wrefresh(win1);

    if (todb)
    {
        connectToDB (dbproc,
            "select max(Biomass_Station_Id#) from Biomass_Station",
            &station_id,
            INTBIND,
            0);
        station_id = station_id == 0 ? minStationId : station_id;
        connectToDB (dbproc,
            "select max(Biomass_Standard_Data_Id#) from Biomass_Standard_Data",
            &data_id,
            INTBIND,
            0);
        data_id = data_id == 0 ? minDataId : data_id;
    }

    win = subwin(stdscr,18,71,5,2);
    initwindow(win);
/* Read a line from the input file . */

    i = 0;
    while (fgets(line,recordLength,fp) != NULL)
    {
        sprintf(l,"%d",++i);
        mvwaddstr(win,1,59,l);
        mvwaddstr(win,2,8,blankString);
        if ((result = scanline (line, values, log, i)) >= 0)
        {
            anyLine = verum;
            if (result > 0)
            {
/*              for (k = 32;  k--;  k != 0)
**              {
**                  if (result & 0x8000)
**                  {
**                      fprintf(log," %s",k);
**                  }
**                  result = result << 1;
**              }
**              fprintf(log,"\n");
*/              anyCorrectLine = falsum;
            }
            if (( i % 100 == 1) || errorFlag)
            {
                errorFlag = falsum;
                writeToWindow(win,values);
            }
            if (todb)
            {
                if (strcmp(statNr,values -> Station) != 0)
                {
                    strcpy(statNr,values -> Station);
                    values -> Station_Id = ++ station_id;
                    stationToDB (dbproc, values, log);
                }
                values -> Data_Id = ++ data_id;
                dataToDB (dbproc, values, log);
            }
        }
        else
        {
```

```
                errorFlag = verum;
                fprintf(log,"LINE: [%d] ",i);
                switch (result)
                {
                    case -1:
                        ll = seriousProblem;
                        break;
                    case -2:
                        ll = emptyLine;
                        break;
                    case -3:
                        ll = noNumber;
                        break;
                    case -4:
                        ll = noValidCruise;
                        break;
                    case -5:
                        ll = lineEnded;
                        break;
/*                  [@case constant_expression: statement@]...*/
                    default:
                        ll = NULL;
                        break;
                }
                if (ll != NULL)
                {
                    fprintf(log,"%s\n",ll);
                    mvwaddstr(win,2,8,ll);
                }
                else
                {
                    mvwaddstr(win,2,8,blankString);
                }
            }
            wrefresh(win);
        }
        endwin();

        return (-1 + (anyLine ? (anyCorrectLine ? 1 : 2) : 0));
}

/*
```

```
                                                              */
/*[@function definition@]...*/
```

```
/*
**++
**  FUNCTIONAL DESCRIPTION:
**
**      Scan a line of input for tokens defined by
**      the specification line of the data file
**      [@tbs@]...
**
**  FORMAL PARAMETERS:
**
**      *fp         pointer to the just opened file
**      values      biomass data
**      log         logfile pointer
**      ln          current linenumber in fp
**      [*tbs@]
**
**  IMPLICIT INPUTS:
**
**      [@description or none@]
**
**  IMPLICIT OUTPUTS:
**
**      [@description or none@]
**
**  COMPLETION CODES:
**
**      0           if function was successful
**      < 0         if function recognizes any error
**      -1          an error, that should never occur
**      -2          the line is empty
**      -3          the line contains no digit
**      -4          the function did not find a valid cruise
**      -5          the line ended before scanning was complete
**      >0          a conversion error occurs
**                      a default value is presented in the appropriate field
**                      the integer value represents a bit filed
**                  bit  1  source  28
**                  bit 28  source   1
**      [@tbs@]...
**
**  SIDE EFFECTS:
**
**      [@description or none@]
**
**--
*/

int scanline ( char * line, struct data * values, FILE * log, int ln)
{
    char fn [cruisenameLength] = "\0\0\0\0",
         * flag1 = NULL,
         * token,
         help [20];
    int i,
        flag = 0;

    void convertToInteger();
    void convertToFloat();

/* The filename are always four characters in length. */
    strncpy(fn,values -> Cruise_Name, cruisenameLength - 1);

    if (strlen(line) == 0)
      { return(-2); }
```

```
    if (!containsDigit(line))
        { return(-3); }

/* The first token must be the name of the cruise, i.e. the filename. */

    flag1 = strchr(line,'/');

    token = strtok(line," ");

    if (strcmp(token,fn))
        { return(-4); }
    for (i = 1;  i <= maximumValues;  i++)
    {
        if ((token = strtok (NULL, " /")) == NULL)
        {
            return(-5);
        }
        if ((i == 1) && (flag1 != NULL))
        {   help[0] = '\0';
            strcat(help,token);
            if ((token = strtok (NULL, " ")) == NULL)
            {
                return (-5);
            }
            strcat(help,"/");
            strcat(help,token);
            token = help;
        }

        flag = flag << 1;

        switch (i)
        {
            case 1:
                strcpy(values -> Station, token);
                break;
            case 2:
                strcpy (values -> Date, token);
                break;
            case 3:
                strcpy (values -> Time, token);
                break;
            case 4:
                convertToFloat(&(values -> Latitude), token, &flag);
                break;
            case 5:
                convertToFloat(&(values -> Longitude), token, &flag);
                break;
            case 6:
                convertToInteger(&(values -> Depth), token, &flag);
                break;
            case 7:
                convertToInteger(&(values -> PrDepth), token, &flag);
                break;
            case 8:
                strcpy(values -> ObsType, token);
                break;
            case 9:
                strcpy(values -> DepthType, token);
                break;
            case 10:
                convertToFloat(&(values -> Temp), token, &flag);
                break;
            case 11:
                convertToFloat(&(values -> Salinity), token, &flag);
                if (values -> Salinity < SalinityLimit)
```

```
                {
                    fprintf(log,"LINE [%d] Salinity: %11.4f\n",
                                ln,values -> Salinity);
                }
                break;
            case 12:
                convertToFloat(&(values -> Oxygen), token, &flag);
                break;
            case 13:
                convertToFloat(&(values -> InPhos), token, &flag);
                break;
            case 14:
                convertToFloat(&(values -> Nitrite), token, &flag);
                break;
            case 15:
                convertToFloat(&(values -> Nitrate), token, &flag);
                break;
            case 16:
                convertToFloat(&(values -> Ammonium), token, &flag);
                break;
            case 17:
                convertToFloat(&(values -> Silicate), token, &flag);
                break;
            case 18:
                convertToFloat(&(values -> PH), token, &flag);
                break;
            case 19:
                convertToFloat(&(values -> Chloride), token, &flag);
                break;
            case 20:
                convertToFloat(&(values -> Chlorophyll_A), token, &flag);
                break;
            case 21:
                convertToFloat(&(values -> Sigma_T), token, &flag);
                break;
            case 22:
                convertToFloat(&(values -> Sigma_T_Obs), token, &flag);
                break;
            case 23:
                convertToFloat(&(values -> Sound_Velocity), token, &flag);
                break;
            case 24:
                convertToFloat(&(values -> Dynamic_Height), token, &flag);
                break;
            case 25:
                convertToFloat(&(values -> Sigma_Theta), token, &flag);
                break;
            case 26:
                convertToFloat(&(values -> Pot_Temp), token, &flag);
                break;
            case 27:
                convertToFloat(&(values -> Spec_Vol_Anomaly), token, &flag);
                break;
            case 28:
                convertToFloat(&(values -> Delta_D), token, &flag);
                break;
            default:
                return(-1);
        }
/*      [@statement@]...*/
    }

  return (flag);
}
```

```
/*
**++
**  FUNCTIONAL DESCRIPTION:
**
**      converts an  string to an integer.
**      additional iformation will be placed in flag
**      [@tbs@]...
**
**  FORMAL PARAMETERS:
**
**      int *   result
**      char *  string
**      int *   flag
**      [@tbs@]...
**
**  IMPLICIT INPUTS:
**
**      [@description or none@]
**
**  IMPLICIT OUTPUTS:
**
**      [@description or none@]
**
**  {@function value or completion codes@}
**
**      [@description or none@]
**
**  SIDE EFFECTS:
**
**      [@description or none@]
**
**--
*/
void convertToInteger (int * value, char * token, int * flag)
{
/*    long strtol();*/

    if (!containsDigit(token))
        {
            *value = -9999;
        }
        else
        {
            *value = (int) strtol(token, (char **) NULL, 10);
            if (errno == ERANGE)
            {
                (*flag)++;
                *value = -9999;
            }
        }
}
```

```
/*
**++
**  FUNCTIONAL DESCRIPTION:
**
**      converts an  string to an float.
**      additional iformation will be placed in flag
**      [@tbs@]...
**
**  FORMAL PARAMETERS:
**
**      float * result
**      char *  string
**      int *   flag
**      [@tbs@]...
**
**  IMPLICIT INPUTS:
**
**      [@description or none@]
**
**  IMPLICIT OUTPUTS:
**
**      [@description or none@]
**
**  {@function value or completion codes@}
**
**      [@description or none@]
**
**  SIDE EFFECTS:
**
**      [@description or none@]
**
**--
*/
void convertToFloat (float * value, char * token, int * flag)
{
/*    double strtod(char *, char **, int);*/

    if (!containsDigit(token))
        {
            *value = -9999;
        }
        else
        {
            *value = (float) strtod(token, (char **) NULL);
            if (errno == ERANGE)
            {
                (*flag)++;
                *value = -9999;
            }
        }
}
```

```
/*
**++
**  FUNCTIONAL DESCRIPTION:
**
**      Looks for any occurence of a digit in a string.
**      There is no corresponding function in the libraries.
**      [@tbs@]...
**
**  FORMAL PARAMETERS:
**
**      char * line
**      [@tbs@]...
**
**  IMPLICIT INPUTS:
**
**      Digits '0'...'9'
**      [@tbs@]...
**
**  IMPLICIT OUTPUTS:
**
**      [@description or none@]
**
**  FUNCTION VALUE:
**
**      TRUE    if digits was found
**      FALSE   if no digit was found
**      [@tbs@]...
**
**  SIDE EFFECTS:
**
**      [@description or none@]
**
**--
*/
bool containsDigit (char * line)
{
    static char digits [] = { '0', '1', '2', '3', '4',
                              '5', '6', '7', '8', '9'};
    int i;
    bool flag = falsum;

    for (i = 0;  i <= 9 && !flag;  i++)
    {
        if (strchr(line,digits[i]) != NULL)
        {
            flag = flag | ~0;
        }
    }
    return(flag);
}
```

```
/*
**++
**  FUNCTIONAL DESCRIPTION:
**
**      Initialize printout in the specified window
**      [@tbs@]...
**
**  FORMAL PARAMETERS:
**
**      WINDOW * win
**
**  IMPLICIT INPUTS:
**
**      [@description or none@]
**
**  IMPLICIT OUTPUTS:
**
**      [@description or none@]
**
**  {@function value or completion codes@}
**
**      [@description or none@]
**
**  SIDE EFFECTS:
**
**      [@description or none@]
**
**--
*/
initwindow (WINDOW * win)
{
    box(win, '!', '-');
    mvwaddstr(win,1,1,"Station:");
    mvwaddstr(win,1,40,"LINECOUNT:");
    mvwaddstr(win,2,1,"ERROR:");
    mvwaddstr(win,3,1,"Depth:");
    mvwaddstr(win,3,40,"PrDepth:");
    mvwaddstr(win,4,1,"ObsType");
    mvwaddstr(win,4,40,"DepthType:");
    mvwaddstr(win,5,1,"Date:");
    mvwaddstr(win,5,40,"Time:");
    mvwaddstr(win,6,1,"Latitude:");
    mvwaddstr(win,6,40,"Longitude");
    mvwaddstr(win,7,1,"Temp:");
    mvwaddstr(win,7,40,"Salinity:");
    mvwaddstr(win,8,1,"Oxygen:");
    mvwaddstr(win,8,40,"InPhos:");
    mvwaddstr(win,9,1,"Nitrite:");
    mvwaddstr(win,9,40,"Nitrate:");
    mvwaddstr(win,10,1,"Ammonium:");
    mvwaddstr(win,10,40,"Silicate:");
    mvwaddstr(win,11,1,"PH:");
    mvwaddstr(win,11,40,"Chloride:");
    mvwaddstr(win,12,1,"Chlorophyll_A:");
    mvwaddstr(win,12,40,"Sigma_T:");
    mvwaddstr(win,13,1,"Sigma_T_Obs:");
    mvwaddstr(win,13,40,"Sound_Velocity:");
    mvwaddstr(win,14,1,"Dynamic_Height:");
    mvwaddstr(win,14,40,"Sigma_Theta:");
    mvwaddstr(win,15,1,"Pot_Temp:");
    mvwaddstr(win,15,40,"Spec_Vol_Anomaly:");
    mvwaddstr(win,16,1,"Delta_D:");
    wrefresh(win);

}
```

```
/*
**++
**  FUNCTIONAL DESCRIPTION:
**
**      Printout in the specified window
**      [@tbs@]...
**
**  FORMAL PARAMETERS:
**
**      WINDOW *      win
**      struct data * values
**
**  IMPLICIT INPUTS:
**
**      [@description or none@]
**
**  IMPLICIT OUTPUTS:
**
**      [@description or none@]
**
**  {@function value or completion codes@}
**
**      [@description or none@]
**
**  SIDE EFFECTS:
**
**      [@description or none@]
**
**--
*/
writeToWindow (WINDOW * win, struct data * values)
{
    char l [recordLength + 1];

    sprintf(l,"%d",values -> Cruise_Number);
    mvwaddstr(win,1,36,l);
    sprintf(l,"%s",values -> Station);
    mvwaddstr(win,1,32,l);
    sprintf(l,"%d",values -> Depth);
    mvwaddstr(win,3,17,l);
    sprintf(l,"%d",values -> PrDepth);
    mvwaddstr(win,3,59,l);
    mvwaddstr(win,4,17,values -> ObsType);
    mvwaddstr(win,4,59,values -> DepthType);
    mvwaddstr(win,5,17,values -> Date);
    mvwaddstr(win,5,59,values -> Time);
    sprintf(l,"%11.4f",values -> Latitude);
    mvwaddstr(win,6,17,l);
    sprintf(l,"%11.4f",values -> Longitude);
    mvwaddstr(win,6,59,l);
    sprintf(l,"%11.4f",values -> Temp);
    mvwaddstr(win,7,17,l);
    sprintf(l,"%11.4f",values -> Salinity);
    mvwaddstr(win,7,59,l);
    sprintf(l,"%11.4f",values -> Oxygen);
    mvwaddstr(win,8,17,l);
    sprintf(l,"%11.4f",values -> InPhos);
    mvwaddstr(win,8,59,l);
    sprintf(l,"%11.4f",values -> Nitrite);
    mvwaddstr(win,9,17,l);
    sprintf(l,"%11.4f",values -> Nitrate);
    mvwaddstr(win,9,59,l);
    sprintf(l,"%11.4f",values -> Ammonium);
    mvwaddstr(win,10,17,l);
    sprintf(l,"%11.4f",values -> Silicate);
```

```
    mvwaddstr(win,10,59,l);
    sprintf(l,"%11.4f",values -> PH);
    mvwaddstr(win,11,17,l);
    sprintf(l,"%11.4f",values -> Chloride);
    mvwaddstr(win,11,59,l);
    sprintf(l,"%11.4f",values -> Chlorophyll_A);
    mvwaddstr(win,12,17,l);
    sprintf(l,"%11.4f",values -> Sigma_T);
    mvwaddstr(win,12,59,l);
    sprintf(l,"%11.4f",values -> Sigma_T_Obs);
    mvwaddstr(win,13,17,l);
    sprintf(l,"%11.4f",values -> Sound_Velocity);
    mvwaddstr(win,13,59,l);
    sprintf(l,"%11.4f",values -> Dynamic_Height);
    mvwaddstr(win,14,17,l);
    sprintf(l,"%11.4f",values -> Sigma_Theta);
    mvwaddstr(win,14,59,l);
    sprintf(l,"%11.4f",values -> Pot_Temp);
    mvwaddstr(win,15,17,l);
    sprintf(l,"%11.4f",values -> Spec_Vol_Anomaly);
    mvwaddstr(win,15,59,l);
    sprintf(l,"%11.4f",values -> Delta_D);
    mvwaddstr(win,16,17,l);
}
```

```
/*
**++
**  FUNCTIONAL DESCRIPTION:
**
**      Transfer station data to the database
**      [@tbs@]...
**
**  FORMAL PARAMETERS:
**
**      WINDOW *      win
**      struct data * values
**
**  IMPLICIT INPUTS:
**
**      [@description or none@]
**
**  IMPLICIT OUTPUTS:
**
**      [@description or none@]
**
**  {@function value or completion codes@}
**
**      [@description or none@]
**
**  SIDE EFFECTS:
**
**      [@description or none@]
**
**--
*/
stationToDB (DBPROCESS * dbproc, struct data * values, FILE * log)
{
    char l [recordLength + 1];

    sprintf(l,"%s %s 19%s %s:%s:%s",
            strsub(values -> Date,4,5),
            months[atoi(strsub(values -> Date,2,3)) - 1],
            strsub(values -> Date,0,1),
            strsub(values -> Time,0,1),
            strsub(values -> Time,2,3),
            strsub(values -> Time,4,5));
    dbcmd (dbproc, " insert into Biomass_Station values(");
    dbfcmd(dbproc," %d,",values -> Station_Id);
    dbfcmd(dbproc," %d,",values -> Cruise_Number);
    dbfcmd(dbproc," \"%s\",",values -> Station);
    dbfcmd(dbproc," %11.4f,",values -> Latitude);
    dbfcmd(dbproc," %11.4f,",values -> Longitude);
    dbfcmd(dbproc," \"%s\",",l);
    dbfcmd(dbproc," -9999,");
    dbfcmd(dbproc," \"%s\")",values -> Cruise_Name);
    if (dbsqlexec (dbproc) == FAIL)
    {
        fprintf(log,"DBERROR: sqlexec failed in stationToDB\n");
        printf("\nsqlexec failed in stationToDB\n");
        printValues (values);
        waitForAccept();
        endwin();
        exit(2);
    }
    if (dbresults (dbproc) == FAIL)
    {
        fprintf(log,"DBERROR: results failed in stationToDB\n");
        printf("\nresults failed in stationToDB\n");
        printValues (values);
        waitForAccept();
```

```
            endwin();
            exit(2);
        }
    }
```

```
/*
**++
**  FUNCTIONAL DESCRIPTION:
**
**      Transfer data to database
**      [@tbs@]...
**
**  FORMAL PARAMETERS:
**
**      WINDOW *      win
**      struct data * values
**
**  IMPLICIT INPUTS:
**
**      [@description or none@]
**
**  IMPLICIT OUTPUTS:
**
**      [@description or none@]
**
**  {@function value or completion codes@}
**
**      [@description or none@]
**
**  SIDE EFFECTS:
**
**      [@description or none@]
**
**--
*/
dataToDB (DBPROCESS * dbproc, struct data * values, FILE * log)
{
    dbcmd (dbproc, " insert into Biomass_Standard_Data values(");
    dbfcmd(dbproc," %d,",values -> Data_Id);
    dbfcmd(dbproc," %d,",values -> Station_Id);
    dbfcmd(dbproc," %d,",values -> Depth);
    dbfcmd(dbproc," %d,",values -> PrDepth);
    dbfcmd(dbproc," \"%s\",",values -> ObsType);
    dbfcmd(dbproc," \"%s\",",values -> DepthType);
    dbfcmd(dbproc," %11.4f,",values -> Temp);
    dbfcmd(dbproc," %11.4f,",values -> Salinity);
    dbfcmd(dbproc," %11.4f,",values -> Oxygen);
    dbfcmd(dbproc," %11.4f,",values -> InPhos);
    dbfcmd(dbproc," %11.4f,",values -> Nitrite);
    dbfcmd(dbproc," %11.4f,",values -> Nitrate);
    dbfcmd(dbproc," %11.4f,",values -> Ammonium);
    dbfcmd(dbproc," %11.4f,",values -> Silicate);
    dbfcmd(dbproc," %11.4f,",values -> PH);
    dbfcmd(dbproc," %11.4f,",values -> Chloride);
    dbfcmd(dbproc," %11.4f,",values -> Chlorophyll_A);
    dbfcmd(dbproc," %11.4f,",values -> Sigma_T);
    dbfcmd(dbproc," %11.4f,",values -> Sigma_T_Obs);
    dbfcmd(dbproc," %11.4f,",values -> Sound_Velocity);
    dbfcmd(dbproc," %11.4f,",values -> Dynamic_Height);
    dbfcmd(dbproc," %11.4f,",values -> Sigma_Theta);
    dbfcmd(dbproc," %11.4f,",values -> Pot_Temp);
    dbfcmd(dbproc," %11.4f,",values -> Spec_Vol_Anomaly);
    dbfcmd(dbproc," %11.4f)",values -> Delta_D);
    if (dbsqlexec (dbproc) == FAIL)
    {
        fprintf(log,"DBERROR: sqlexec failed in dataToDB\n");
        printf("\nsqlexec failed in dataToDB\n");
        printValues (values);
        waitForAccept();
        endwin();
```

```
            exit(2);
        }
        if (dbresults (dbproc) == FAIL)
        {
            fprintf(log,"DBERROR: results failed in dataToDB\n");
            printf("\nresults failed in dataToDB\n");
            printValues (values);
            waitForAccept();
            endwin();
            exit(2);
        }
    }
```

```
/*
**++
**  FUNCTIONAL DESCRIPTION:
**
**      Printout values
**      [@tbs@]...
**
**  FORMAL PARAMETERS:
**
**      WINDOW *      win
**      struct data * values
**
**  IMPLICIT INPUTS:
**
**      [@description or none@]
**
**  IMPLICIT OUTPUTS:
**
**      [@description or none@]
**
**  {@function value or completion codes@}
**
**      [@description or none@]
**
**  SIDE EFFECTS:
**
**      [@description or none@]
**
**--
*/
printValues (struct data * values)
{
    printf("The data set read last:\n");
    printf("Cruise_Name %s",values -> Cruise_Name);
    printf(" Cruise_Number %d",values -> Cruise_Number);
    printf(" Station %s\n",values -> Station);
    printf("Depth %d",values -> Depth);
    printf(" PrDepth %d\n",values -> PrDepth);
    printf("ObsType %s",values -> ObsType);
    printf(" DepthType %s\n",values -> DepthType);
    printf("Date %s",values -> Date);
    printf(" Time %s\n",values -> Time);
    printf("Latitude %11.4f",values -> Latitude);
    printf(" Longitude %11.4f\n",values -> Longitude);
    printf("Temperature %11.4f",values -> Temp);
    printf(" Salinity %11.4f\n",values -> Salinity);
    printf("Oxygen %11.4f",values -> Oxygen);
    printf(" InPhos %11.4f\n",values -> InPhos);
    printf("Nitrite %11.4f",values -> Nitrite);
    printf(" Nitrate %11.4f\n",values -> Nitrate);
    printf("Ammonium %11.4f",values -> Ammonium);
    printf(" Silicate %11.4f\n",values -> Silicate);
    printf("PH %11.4f",values -> PH);
    printf(" Chloride %11.4f\n",values -> Chloride);
    printf("Chlorophyll_A %11.4f",values -> Chlorophyll_A);
    printf(" Sigma_T %11.4f\n",values -> Sigma_T);
    printf("Sigma_T_Obs %11.4f",values -> Sigma_T_Obs);
    printf(" Sound_Velocity %11.4f\n",values -> Sound_Velocity);
    printf("Dynamic_Height %11.4f",values -> Dynamic_Height);
    printf(" Sigma_Theta %11.4f\n",values -> Sigma_Theta);
    printf("Pot_Temp %11.4f",values -> Pot_Temp);
    printf("Specific_Volume_Anomaly %11.4f\n",values -> Spec_Vol_Anomaly);
    printf(" Delta_D %11.4f\n",values -> Delta_D);
}
```

```
/*
**++
**  FUNCTIONAL DESCRIPTION:
**
**      Wait for the entry of the string 'accept'
**      [@tbs@]...
**
**  FORMAL PARAMETERS:
**
**      [@description or none@]
**
**  IMPLICIT INPUTS:
**
**      [@description or none@]
**
**  IMPLICIT OUTPUTS:
**
**      [@description or none@]
**
**  {@function value or completion codes@}
**
**      [@description or none@]
**
**  SIDE EFFECTS:
**
**      [@description or none@]
**
**--
*/
 waitForAccept ()
{
    char l [20];

    do
    {   printf("Type in 'accept': ");
        gets(l);
    } while (strcmp(l,"accept") != 0);
}

/*[@function definition@]...*/
```

24.07.1991

```
/*
**++
**  FACILITY:
**
**      brandini.c
**      main
**
**      Program to select special values for graphics from
**      BIOMASS for Frederico Brandini.
**
**      Special handling of NULL values in the database is provided.
**      [@tbs@]...
**
**  ABSTRACT:
**
**      [@tbs@]...
**
**  AUTHORS:
**
**      Lutz-Peter Kurdelski
**      Alfred-Wegener-Institute
**      Am Handelshafen 12
**      D-2850 Bremerhafen
**      Germany
**
**
**  CREATION DATE:      1991-07-05
**
**  MODIFICATION HISTORY:
**
**--
*/

/*
**
**  INCLUDE FILES
**
*/

#include <stdio.h>
#include <curses.h>
#include <math.h>

/*
** This is for the database only.
*/
#include <sybfront.h>
#include <sybdb.h>

/*
** some special for readability
*/
#define and &&
#define or ||
/*
**
*/
#define tinyDef      -127
#define smallDef     -32767
#define intDef       -9999
#define floatDef     -9999.0
#define databaseName 31
#define maxCruises   15

typedef char dbnames [databaseName];
```

C-5

```
typedef char str5 [5];
typedef char str8 [8];

typedef struct p {
    float       Lat,
                Lon;
} point;

typedef struct r {
    point       origin,
                corner;
} rect;

typedef struct s {
    int     stationId;
    struct station * next;
} station;

typedef struct database {
    dbnames dbase,
            user,
            password;
} db;

typedef struct rv {
    int     depthCalc;
    double  degreeCalc;
} retVal;

rect * enterPositions ();
station * getStation (DBPROCESS *, rect *, char *);

void writeData( DBPROCESS *, char *, rect *, db *, BOOL, int);
int writeDensityFile (DBPROCESS *, station *, char *, int, db *);
int writeIntegrationFile (DBPROCESS *, station *, char *, int, db *, int);
double sigma_t (double, double, double);
int getDefault ();
void setDefaults (DBPROCESS *);
void printDefault (FILE *, double);
retVal * degreeOfInstabilityThreePoints (int, int, db *);
double grad_t (double, double, double, double, double, double);

const char * defaultDB = "SouthernOceanDB",
           * defaultDBO = "sa";
char * cruises [] = { "HOFX", "ITFX", "HEFX", "ODFX", "SIFX",
                      "PSS1", "SIS1", "BES1", "ACS1",
                      "PSS2", "KMS2", "JBS2", "ACS2", "BES2", "HES2" };
```

```
/*
**++
**  FUNCTIONAL DESCRIPTION:
**
**      main
**
**  FORMAL PARAMETERS:
**
**      none
**
**  IMPLICIT INPUTS:
**
**      none
**
**  IMPLICIT OUTPUTS:
**
**      none
**
**  COMPLETION CODES:
**
**      none
**
**  SIDE EFFECTS:
**
**      none
**
**--
*/
main (int argc, char *argv[])
MAIN_PROGRAM
{
    str8 experiment;

    str5 all,
         intOnly;

    DBPROCESS * dbproc;

    extern int err_handler();
    extern int msg_handler();

    rect * pos;

    db dbo;

    int noNullValues,
        i;

    BOOL integratedValue,
         allExp;

    dberrhandle(err_handler);
    dbmsghandle(msg_handler);

    strcpy(dbo.dbase,defaultDB);
    dbo.user[0] = '\0';
    dbo.password[0] = '\0';

    if ((dbproc = opendb(dbo.dbase, dbo.user, dbo.password)) == NULL)
    {
        printf("Database specification error. See Message from SQL server.\n");
        exit(-1);
    }
/*
**  The user must specify the experiment.
```

```
*/
    printf("All experiments ? [y/n] : ");
    gets(all);
    if (strlen(all) != 0 and toupper(all[0]) != 'Y' and toupper(all[0]) != 'N')
    {
        printf("Please answer 'yes' or 'no'\n");
        exit(-1);
    }
    allExp = strlen(all) == 0 or toupper(all[0]) == 'Y'
                ? TRUE
                : FALSE;
    if (!allExp)
    {
        printf("Enter the experiment : ");
        gets(experiment);
    }
/*
**  The user must specify two points on the sphere.
*/
    pos = enterPositions ();
/*
**  If pos is null (this happens if any position is an empty string)
**  AND experiment is an empty string the program stopps.
*/
    if (!allExp and (strlen(experiment) == 0 and pos == NULL))
    {
        printf("Neither position nor experiment given!\n");
        printf("There is nothing for me to do. Bye.\n");
        exit(-1);
    }
/*
**  Tell whether NULLs in the database shoulb be surpressed
**  or serve a default value that might be wrong.
**  Otherwise the default value can be set to a wise
**  nether used value like -9999 which will than
**  be transferred to '?' if specified.
*/
    if ((noNullValues = getDefault()) < 0 or ( noNullValues > 3))
    {
        printf("I do not know how to handle NULL value\nsorry...\n");
        exit(-1);
    }
/*
**  Sometimes the user wants only the integrated values.
*/
    printf("Print integrated values ONLY : [y/n] ");
    gets(intOnly);
    integratedValue = strlen(intOnly) == 0 or toupper(intOnly[0]) == 'N'
                        ? FALSE
                        : TRUE;
/*
**  According to the positions and the experimnet
**  the stations are selected
**  and stored in a station list.
*/
    if (allExp)
    {
        for ( i = 0; i < maxCruises; i++)
            writeData(dbproc,
                        cruises [i],
                        pos,
                        &dbo,
                        integratedValue,
                        noNullValues);
    }
    else
```

```
    {
        writeData(dbproc,
                  experiment,
                  pos,
                  &dbo,
                  integratedValue,
                  noNullValues);
    }
    closedb();
    printf("\nDone\n");
}
/*
** Now all has been done
** ---------------------------------------------------
*/
```

```
/*
**++
**  FUNCTIONAL DESCRIPTION:
**
**      writeData
**
**  FORMAL PARAMETERS:
**
**      DBPROCESS * dbproc
**      char * experiment
**      rect * pos
**      db * dbo
**      BOOL integratedValue
**      int noNullValues
**
**  IMPLICIT INPUTS:
**
**      [@description or none@]
**
**  IMPLICIT OUTPUTS:
**
**      [@description or none@]
**
**  {@function value or completion codes@}
**
**      [@description or none@]
**
**  SIDE EFFECTS:
**
**      [@description or none@]
**
**--
*/
void writeData( DBPROCESS * dbproc,
                char * experiment,
                rect * pos,
                db * dbo,
                BOOL integrateOnly,
                int noNullValues)
{
    int maxDepth;

    station * stations = NULL;

    stations = getStation (dbproc, pos, experiment);

    if (stations != NULL)
    {
        if (!integrateOnly)
            writeDensityFile (dbproc,
                    stations,
                    experiment,
                    noNullValues,
                    dbo);

        maxDepth = 50;
        writeIntegrationFile (dbproc,
                    stations,
                    experiment,
                    noNullValues,
                    dbo,
                    maxDepth);

        maxDepth = 100;
        writeIntegrationFile (dbproc,
```

```
                        stations,
                        experiment,
                        noNullValues,
                        dbo,
                        maxDepth);
        }
    }
    /*
    ** end of writeData
    */
```

```
/*
**++
**  FUNCTIONAL DESCRIPTION:
**
**      getDefault
**
**      Get a parameter which specifies how to deal with
**      database default values is the field in the database
**      contains a NULL value.
**
**  FORMAL PARAMETERS:
**
**      none
**
**  IMPLICIT INPUTS:
**
**      none
**
**  IMPLICIT OUTPUTS:
**
**      none
**
**  FUNCTION VALUE:
**
**      0   no specific handle
**              may result in some errors in computing
**              dependent values
**      1   surpress ALL NULL values in the database
**              may result in no values at all
**      2   setting specail defualt values
**      3   same as 2 and translating into special default value
**          an undefined value for the piped program
**
**  SIDE EFFECTS:
**
**      none
**
**--
*/
int getDefault ()
{
    int i = 0;
    str5 s;

    printf("Enter a value between 0 and 3 to select the specified\n");
    printf("handling of NULL values in the database.\n\n");
/*    printf("\t0\tno specific handle\");
    printf("\t\t\tmay result in some errors in computing\n");
    printf("\t\t\tdependent values, i.e. internal default\n");
    printf("\t\t\tfor integer and real values is 0\n");
*/    printf("\t1\tsurpress ALL NULL values in the database\n");
    printf("\t\t\tmay result in no values at all\n");
    printf("\t2\tsetting specific default values\n");
    printf("\t3\tas 2 but allowing to translate this values into\n");
    printf("\t\tan undefined value for the piped program\n");
    printf("Enter : ");
    gets(s);
    sscanf(s,"%d",&i);
    return (i);
}
/*
** end of getDefault
*/
```

```
/*
**++
**  FUNCTIONAL DESCRIPTION:
**
**      enterPosition
**
**      returns a rectandle on a sphere
**      defined by the user
**
**  FORMAL PARAMETERS:
**
**      none
**
**  IMPLICIT INPUTS:
**
**      none
**
**  IMPLICIT OUTPUTS:
**
**      none
**
**  FUNCTION VALUE:
**
**      rect * pos      pointer to a structure containing
**                      two points
**
**  SIDE EFFECTS:
**
**      none
**
**--
*/
rect * enterPositions ()
{
    rect * pos;

    char s [20];

    pos = (rect *) malloc (sizeof(rect));

    printf("Enter the position values.\n");
    printf("\tValues of northern hemisphere and eastwards Greenwich > 0\n");
    printf("\tValues of southern hemisphere and wsetwards Greenwich < 0\n\n");
    printf("North West corner longitude : ");
    gets(s);
    if (strlen(s) == 0)
    {
        return (NULL);
    }
    if (sscanf(s,"%f", &((pos -> origin).Lon)) == 0)
    {
        return (NULL);
    }
    printf("North West corner latitude : ");
    gets(s);
    if (strlen(s) == 0)
    {
        return (NULL);
    }
    if (sscanf(s,"%f", &((pos -> origin).Lat)) == 0)
    {
        return (NULL);
    }
    printf("South East corner longitude : ");
    gets(s);
```

```
    if (strlen(s) == 0)
    {
        return (NULL);
    }
    if (sscanf(s,"%f", &((pos -> corner).Lon)) == 0)
    {
        return (NULL);
    }
    printf("South East corner latitude : ");
    gets(s);
    if (strlen(s) == 0)
    {
        return (NULL);
    }
    if (sscanf(s,"%f", &((pos -> corner).Lat)) == 0)
    {
        return (NULL);
    }

    return (pos);
} /* end of enterPosition */
```

```
/*
**++
**  FUNCTIONAL DESCRIPTION:
**
**      getStation
**
**      constructing a list of station nodes
**      containing all station within the rectangle
**      pos and according to the the experiment
**      experiment.
**      This is done by asking the database dbproc.
**
**  FORMAL PARAMETERS:
**
**      DBPROCESS * dbproc      reference to the database
**      rect * pos              the rectangle
**      char * experiment       the name of the experiment
**
**  IMPLICIT INPUTS:
**
**      none
**
**  IMPLICIT OUTPUTS:
**
**      none
**
**  FUNCTION VALUE:
**
**      Pointer to a list of Stations
**      NULL otherwise
**
**  SIDE EFFECTS:
**
**      none
**
**--
*/
station * getStation (DBPROCESS * dbproc,
                      rect * pos,
                      char * experiment)
{
    station * stat = NULL,
            * lstat = NULL;
    int statId;

    dbcmd(dbproc,"select Biomass_Station_Id# from Biomass_Station");
    dbcmd(dbproc," where");
    if (pos != NULL)
    {
        dbfcmd(dbproc," Longitude between %11.4f and %11.4f",
                (pos -> origin).Lon, (pos -> corner).Lon);
        dbfcmd(dbproc," and Latitude between %11.4f and %11.4f",
                (pos -> origin).Lat, (pos -> corner).Lat);
    }
    if (pos != NULL and strlen(experiment) != 0)
    {
        dbcmd(dbproc," and");
    }
    if (strlen(experiment) != 0)
    {
        dbfcmd(dbproc," Cruise_Name like \"%s\"",
                experiment);
    }
    if (dbsqlexec(dbproc) == FAIL)
    {
```

```
        return(NULL);
    }
    if (dbresults(dbproc) == FAIL)
    {
        return(NULL);
    }
    if (dbbind(dbproc,1,INTBIND,0,&statId) == FAIL)
    {
        return(NULL);
    }
    while (dbnextrow(dbproc) != NO_MORE_ROWS)
    {
        if (lstat == NULL)
        {
            stat = (station *) malloc(sizeof(station));
            lstat = stat;
        }
        else
        {
            lstat -> next = (station *) malloc(sizeof(station));
            lstat = lstat -> next;
        }
            lstat -> next = NULL;
            lstat -> stationId = statId;
    } /* end of while (dbnextrow(dbproc) != NO_MORE_ROWS) */
    return (stat);
} /* end of getStation */
```

```
/*
**++
**  FUNCTIONAL DESCRIPTION:
**
**      writeDensityFile
**      [@tbs@]...
**
**  FORMAL PARAMETERS:
**
**      DBPROCESS * dbproc      refernce to the database
**      station * stations      the list of the stations
**      char * experiment       the name of the experiment
**      int noNullValues        surpress NULL values in the databse
**      db * dbo                user characteristics
**
**  IMPLICIT INPUTS:
**
**      none
**
**  IMPLICIT OUTPUTS:
**
**      none
**
**  COMPLETION CODES:
**
**      [@description or none@]
**
**  SIDE EFFECTS:
**
**      none
**
**--
*/
int writeDensityFile (DBPROCESS * dbproc,
                      station * stations,
                      char * experiment,
                      int noNullValues,
                      db * dbo)
{
    const char * delim = "/. ,;:";

    station * lstat = NULL;

    int statId,
        cruise,
        i,
        j,
        err = SUCCEED;

    DBFLT8  density,
            depth,
            temperature,
            salinity,
            chlorophyll,
            lat,
            lon;

    DBCHAR stat [9];

    char s [255],
         * ss;

    FILE * fp;

    retVal * degree;
```

```
lstat = stations;

if (noNullValues >= 2)
{
    setDefaults(dbproc);
}

while (lstat != NULL)
{
    dbcmd(dbproc,"select Cruise_Number, Station_Number,");
    dbcmd(dbproc," Latitude, Longitude");
    dbcmd(dbproc," from Biomass_Station");
    dbfcmd(dbproc," where Biomass_Station_Id# = %d",lstat -> stationId);
    dbfcmd(dbproc," order by Biomass_Station_Id#");
    err = dbsqlexec(dbproc);
    if (err != FAIL)
        err = dbresults(dbproc);
    if (err != FAIL)
        err = dbbind(dbproc,1,INTBIND,0,&cruise);
    if (err != FAIL)
        err = dbbind(dbproc,2,STRINGBIND,0,&stat);
    if (err != FAIL)
        err = dbbind(dbproc,3,FLT8BIND,0,&lat);
    if (err != FAIL)
        err = dbbind(dbproc,4,FLT8BIND,0,&lon);
    if (err != FAIL)
    {
        while (dbnextrow(dbproc) != NO_MORE_ROWS)
        {
            ss = strtrr(stat, delim);
            sprintf(s,"%s_%d_%s.dat",experiment,cruise,ss);
            printf("Writing file '%s'\n",s);
            degree = degreeOfInstabilityThreePoints(lstat -> stationId,
                                noNullValues,
                                dbo);
            dbcmd(dbproc,"select Depth, Temperature, Salinity,");
            dbcmd(dbproc," Chlorophyll_A");
            dbfcmd(dbproc," from Biomass_Data_View");
            dbfcmd(dbproc," where Biomass_Station_Id# = %d",
                    lstat -> stationId);
            if (noNullValues == 1)
            {
                dbcmd(dbproc," and Temperature is not null");
                dbcmd(dbproc," and Salinity is not null");
                dbcmd(dbproc," and Chlorophyll_A is not null");
            }
            dbcmd(dbproc," order by Depth");
            err = dbsqlexec(dbproc);
            if (err != FAIL)
                err = dbresults(dbproc);
            if (err != FAIL)
                err = dbbind(dbproc,1,FLT8BIND,0,(BYTE *) &depth);
            if (err != FAIL)
                err = dbbind(dbproc,2,FLT8BIND,0,(BYTE *) &temperature);
            if (err != FAIL)
                err = dbbind(dbproc,3,FLT8BIND,0,(BYTE *) &salinity);
            if (err != FAIL)
                err = dbbind(dbproc,4,FLT8BIND,0,(BYTE *) &chlorophyll);
            if (err != FAIL)
            {
                if (DBROWS(dbproc) != FAIL)
                {
                    fp = fopen(s,"w");
                    fprintf(fp,"  Station  Lon      Lat      DCc");
                    fprintf(fp,"  Inst.-Calc\n");
```

```
                        fprintf(fp,"  %s     %7.2f  %7.2f  %3d  %9.6f\n",
                                stat,
                                lon,
                                lat,
                                degree -> depthCalc,
                                degree -> degreeCalc);
;                       fprintf(fp,"  Depth          Temperature   Salinity     ")♦
                        fprintf(fp,"  Chlorophyll    Density\n");
                        while (dbnextrow(dbproc) != NO_MORE_ROWS)
                        {
                            if (noNullValues >= 2 and
                                (temperature == floatDef or
                                salinity == floatDef))
                            {
                                density = floatDef;
                            }
                            else
                            {
                                density = (float) sigma_t (0,
                                                           temperature,
                                                           salinity);
                            } /* end of noNullValues >= 2) */
                            if (noNullValues == 2)
                            {
                                fprintf(fp,"  %12.6f");
                                printDefault(fp,temperature);
                                printDefault(fp,salinity);
                                printDefault(fp,chlorophyll);
                                fprintf(fp,"  %12.6f",density);
                                fprintf(fp,"\n");
                            }
                            else
                            {
                                fprintf(fp,"  %12.6f  %12.6f  %12.6f",
                                        depth,
                                        temperature,
                                        salinity);
                                fprintf(fp,"  %12.6f  %12.6f\n",
                                        chlorophyll,
                                    density);
                            } /* end of (noNullValues == 2) */
                        } /* end of while (dbnextrow(dbproc) == ROW) */
                        fclose(fp);
                    } /* end of if (err != FAIL) (inner) */
                } /* end of while (dbnextrow(dbproc) == ROW) */
            } /* end of if (DBROWS(dbproc) != FAIL) */
        } /* end of if (err != FAIL) (outer) */
        lstat = lstat -> next;
    } /* end of while (lstat != NULL) */
    return (err);
} /* end of writeDensityFile */
```

```
/*
**++
**  FUNCTIONAL DESCRIPTION:
**
**      writeIntegrationFile
**
**  FORMAL PARAMETERS:
**
**      DBPROCESS * dbproc      refernce to the database
**      station * stations      the list of the stations
**      char * experiment       the name of the experiment
**      int noNullValues        surpress NULL in the database
**      db * dbo                user characteristics
**      int maxDepth            maximum depth
**
**  IMPLICIT INPUTS:
**
**      none
**
**  IMPLICIT OUTPUTS:
**
**      none
**
**  COMPLETION CODES:
**
**      [@tbs@]...
**
**  SIDE EFFECTS:
**
**      none
**
**--
*/
int writeIntegrationFile (DBPROCESS * dbproc,
                          station * stations,
                          char * experiment,
                          int noNullValues,
                          db * dbo,
                          int maxDepth)
{
    station * lstat;

    int err = 0;

    DBFLT8  lat = 0,
            lon = 0,
            integratedValue = 0,
            chloro = 0,
            chloroLast = 0,
            firstChloro = 0;

    DBINT   cruise = 0,
            statId = 0,
            depth = 0,
            depthLast = 0;

    DBCHAR  cruiseName [9],
            stat [9],
            dt [28];

    FILE * fp;

    char s [255],
         * ss,
         * year,
```

```
      * mon,
      * day;

retVal * degree;

lstat = stations;

sprintf(s,"%s_IC%d.dat",experiment,maxDepth);
fp = fopen(s,"w");

fprintf(fp,"  Longitude  Latitude  Chloroph  IChlorop  IDc");
fprintf(fp,"  Inst.-Calc  Date        CName  Station\n");
printf("Writing file %s\n",s);

while (lstat != NULL)
{
    dbcmd(dbproc,"select Longitude, Latitude, Date_Time, Cruise_Name,");
    dbcmd(dbproc," Station_Number, Cruise_Number");
    dbcmd(dbproc," from Biomass_Station");
    dbfcmd(dbproc," where Biomass_Station_Id# = %d", lstat -> stationId);
    err = dbsqlexec(dbproc);
    if (err != FAIL)
        err = dbresults(dbproc);
    if (err != FAIL)
        err = dbbind(dbproc,1,FLT8BIND,0,(BYTE *) &lon);
    if (err != FAIL)
        err = dbbind(dbproc,2,FLT8BIND,0,(BYTE *) &lat);
    if (err != FAIL)
        err = dbbind(dbproc,3,STRINGBIND,0,(BYTE *) dt);
    if (err != FAIL)
        err = dbbind(dbproc,4,STRINGBIND,0,cruiseName);
    if (err != FAIL)
        err = dbbind(dbproc,5,STRINGBIND,0,stat);
    if (err != FAIL)
        err = dbbind(dbproc,6,INTBIND,0,(BYTE *) &cruise);
    if (err != FAIL)
    {
        if (dbnextrow(dbproc) == REG_ROW)
        {
            dbcmd(dbproc,"select Chlorophyll_A, Depth");
            dbcmd(dbproc," from Biomass_Data_View");
            dbfcmd(dbproc,
                    " where Biomass_Station_Id# = %d",
                    lstat ->stationId);
            if (noNullValues == 1)
            {
                dbcmd(dbproc," and Chlorophyll_A is not null");
            }
            dbcmd(dbproc," order by Depth");
            err = dbsqlexec(dbproc);
            if (err != FAIL)
                err = dbresults(dbproc);
            if (err != FAIL)
                err = dbbind(dbproc,1,FLT8BIND,0,(BYTE *) &chloro);
            if (err != FAIL)
                err = dbbind(dbproc,2,INTBIND,0,(BYTE *) &depth);
            if (err != FAIL)
            {
                integratedValue = 0;
                if (dbnextrow(dbproc) != NO_MORE_ROWS)
                {
                    firstChloro = chloro;
                    chloroLast = chloro;
                    depthLast = depth;
                    while (dbnextrow(dbproc) != NO_MORE_ROWS)
                    {
```

```
                        if ((chloro >= 0) and (depth <= maxDepth))
                        {
                            integratedValue = integratedValue +
                            (chloro + chloroLast) / 2 * (depth - depthLast);
                            chloroLast = chloro;
                            depthLast = depth;
                        } /* end of if (chloro > 0) */
                    } /* end of while (dbnextrow(dbproc) != NO_MORE_ROWS) */
                } /* end of if (dbnextrow(dbproc) != NO_MORE_ROWS) */
            } /* */
        } /* end of if (err != FAIL) */
        else
        {
            integratedValue = -1;
        }
    } /* end of if (err != FAIL) */
    else
    {
        integratedValue = -1;
    }

    ss = strtok(dt," ");
    mon = (char *) malloc (strlen(ss) + 1);
    strcpy(mon,ss);
    ss = strtok(NULL," ");
    day = (char *) malloc (strlen(ss) + 1);
    strcpy(day,ss);
    ss = strtok(NULL," ");
    year = (char *) malloc (strlen(ss) + 1);
    year = strcpy(year,ss);
    ss = (char *) malloc(strlen(mon) + strlen(day) + strlen(year) + 3);
    *ss = '\0';
    strcat(ss,day);
    strcat(ss,"-");
    strcat(ss,mon);
    strcat(ss,"-");
    strcat(ss,year);
    free(year);
    free(day);
    free(mon);

    degree = degreeOfInstabilityThreePoints(lstat -> stationId,
                                            noNullValues,
                                            dbo);

    if (integratedValue != 0 and
        degree != 0 and
        firstChloro != 0)
    {
        fprintf(fp,"  %9.4f  %8.4f  %8.6f  %8.4f  %3d  %9.6f",
                    lon,
                    lat,
                    firstChloro,
                    integratedValue,
                    degree -> depthCalc,
                    degree -> degreeCalc);
        fprintf(fp,"  %s  %s   %s\n",
                    ss,
                    cruiseName,
                    stat);
    }
    lstat = lstat -> next;
} /* end of while (lstat != NULL) */
fclose(fp);
}
/*
```

```
** end of writeIntegrationFile
*/
```

```
/*
**++
**  FUNCTIONAL DESCRIPTION:
**
**      setDefaults
**
**      Set special default values other than the predefined
**      values by SYBASE, i.e.
**
**          Type        default     set to
**          TINYBIND    0           -127
**          SMALLBIND   0           -32767
**          INTBIND     0           -9999
**          FLT8BIND    0.0         -9999.0
**
**  FORMAL PARAMETERS:
**
**      DBPROCESS * dbprocess
**
**  IMPLICIT INPUTS:
**
**      none
**
**  IMPLICIT OUTPUTS:
**
**      none
**
**  FUNCTION VALUE:
**
**      none
**
**  SIDE EFFECTS:
**
**      none
**
**--
*/
void setDefaults (DBPROCESS * dbproc)
{
    char tiniDeflt          = tinyDef;
    short int smallIntDeflt = smallDef;
    int intDeflt            = intDef;
    double floatDeflt       = floatDef;

    dbsetnull(dbproc, TINYBIND, 0, (BYTE *) &tiniDeflt);
    dbsetnull(dbproc, SMALLBIND, 0, (BYTE *) &smallIntDeflt);
    dbsetnull(dbproc, INTBIND, 0, (BYTE *) &intDeflt);
    dbsetnull(dbproc, FLT8BIND, 0, (BYTE *) &floatDeflt);
}
/*
** end of setDefaults
*/
```

```
/*
**++
**  FUNCTIONAL DESCRIPTION:
**
**      printDef
**
**      print a value on a file as a value
**      or a default value.
**
**  FORMAL PARAMETERS:
**
**      FILE * fp
**      double value
**
**  IMPLICIT INPUTS:
**
**      none
**
**  IMPLICIT OUTPUTS:
**
**      none
**
**  COMPLETION CODES:
**
**      none
**
**  SIDE EFFECTS:
**
**      none
**
**--
*/
void printDefault (FILE * fp,
            double value)
{
    if (value == floatDef)
    {
        fprintf(fp,"             ?");
    }
    else
    {
        fprintf(fp,"  %12.6f",value);
    }
}
/*
** end of printDefault
*/
```

```
/*
**++
**  FUNCTIONAL DESCRIPTION:
**
**      degreeOfInstabilityThreePoints
**
**      evaluating of the depth of maximum stability
**      using three points from the table
**      [@tbs@]...
**
**  FORMAL PARAMETERS:
**
**      DBPROCESS * dbproc
**      int stationId
**      int noNullValues
**      [@tbs@]...
**
**  IMPLICIT INPUTS:
**
**      none
**
**  IMPLICIT OUTPUTS:
**
**      none
**
**  FUNCTION VALUE:
**
**      double stability
**
**  SIDE EFFECTS:
**
**      none
**
**--
*/
retVal * degreeOfInstabilityThreePoints (int stationId,
                        int noNullValues,
                        db * dbase)
{
    DBPROCESS * dbproc;

    DBFLT8 depth0 = 0.0,
        depth1 = 0.0,
        depth2 = 0.0,
        depthInit = 0.0,
        sigmaCalc2 = 0.0,
        sigmaCalc1 = 0.0,
        sigmaCalc0 = 0.0,
        salinity = 0.0,
        temp = 0.0,
        sigmaCalcInit = 0.0,
        degreeCalc = 0.0,
        degreeCalcLast = 0.0;

    retVal * returnValue;

    int err = 0;

    bool flag = FALSE,
         cflag = FALSE;
char s[1];

    returnValue = (retVal *) malloc (sizeof(retVal));

    returnValue -> depthCalc = 0;
```

```
    returnValue -> degreeCalc = 0.0;

    dbproc = opendb(dbase -> dbase, dbase -> user, dbase -> password);

    dbcmd(dbproc,"select Depth,");
    dbcmd(dbproc," Salinity, Temperature from Biomass_Data_View");
    dbfcmd(dbproc," where Biomass_Station_Id# = %d",stationId);
    if (noNullValues == 1)
    {
        dbcmd(dbproc," and Temperature is not null");
        dbcmd(dbproc," and Salinity is not null");
    }
    dbcmd(dbproc," order by Depth");
    err = dbsqlexec(dbproc);
    if (err != FAIL)
        err = dbresults(dbproc);
    if (err != FAIL)
        err = dbbind(dbproc,1,FLT8BIND,0,(BYTE *) &depth2);
    if (err != FAIL)
        err = dbbind(dbproc,2,FLT8BIND,0,(BYTE *) &salinity);
    if (err != FAIL)
        err = dbbind(dbproc,3,FLT8BIND,0,(BYTE *) &temp);
    if (err != FAIL)
    {
        if (dbnextrow(dbproc) != NO_MORE_ROWS)
        {
            depthInit = depth2;
            depth0 = depthInit;
            sigmaCalcInit = sigma_t (0,temp,salinity);
            sigmaCalc0 = sigmaCalcInit;
            if (dbnextrow(dbproc) != NO_MORE_ROWS)
            {
                depth1 = depth2;
                sigmaCalc2 = sigma_t (0,temp,salinity);
                sigmaCalc1 = sigmaCalc2;
                while (dbnextrow(dbproc) != NO_MORE_ROWS)
                {
                    degreeCalcLast = degreeCalc;
                    sigmaCalc2 = sigma_t (0,temp,salinity);
                    degreeCalc = grad_t (depth0, depth1, depth2,
                                    sigmaCalc0, sigmaCalc1, sigmaCalc2);
                    if (fabs((double) degreeCalc) <
                        fabs((double) degreeCalcLast)
                            and !cflag)
                    {
                        returnValue -> degreeCalc =
                                (sigmaCalc2-sigmaCalcInit) /
                                    (depth2-depthInit);
                        returnValue -> depthCalc = (int) depth2;
                        cflag = TRUE;
                    }
                    depth0 = depth1;
                    depth1 = depth2;
                    sigmaCalc0 = sigmaCalc1;
                    sigmaCalc1 = sigmaCalc2;
                } /* end of while (dbnextrow(dbproc) != NO_MORE_ROWS) */
            } /* end of if (dbnextrow(dbproc) != NO_MORE_ROWS) */
        } /* end of if (dbnextrow(dbproc) != NO_MORE_ROWS) */
    }
    dbclose(dbproc);
    return(returnValue);
}
/*
** end of degreeOfInstabilityThreePoints
*/
```

```
/*
**++
**  FUNCTIONAL DESCRIPTION:
**
**      grad_t
**
**      slope of sigma_theta
**
**  FORMAL PARAMETERS:
**
**      double  depth0
**      double  depth1
**      double  depth2
**      double  sigma0
**      double  sigma1
**      double  sigma2
**
**  IMPLICIT INPUTS:
**
**      none
**
**  IMPLICIT OUTPUTS:
**
**      none
**
**  FUNCTION VALUE:
**
**      double  slope
**
**  SIDE EFFECTS:
**
**      none
**
**--
*/
double grad_t (double depth0,
               double depth1,
               double depth2,
               double sigma0,
               double sigma1,
               double sigma2)
{
    double  deltaZ1,
            deltaZ2,
            deltaZ,
            deltaS1,
            deltaS2,
            rho1,
            rho2;

    deltaZ2 = depth2 - depth1;
    deltaZ1 = depth1 - depth0;
    deltaS2 = sigma2 - sigma1;
    deltaS1 = sigma1 - sigma0;
    rho2 = deltaS2 * deltaZ1 / deltaZ2;
    rho1 = deltaS1 * deltaZ2 / deltaZ1;
    deltaZ = depth2 - depth0;
    return (( rho2 + rho1 ) / deltaZ);
}
/*
** end of grad_t
*/
```

24.07.1991

```
#module ALPHAM A

/*
**++
**  FACILITY:                                    ALPHA.C
**
**      [@tbs@]...
**
**  ABSTRACT:
**
**      Routines to evalute specific values from measured
**      oceanographic data as there is Simga_T
**      [@tbs@]...
**
**  AUTHORS:
**
**      Lutz-Peter Kurdelski
**      Alfred-Wegener-Institute
**      Am Handelshafen 12
**      D-2850 Bremerhaven
**      [@tbs@]...
**
**
**  CREATION DATE:      1991-07-09
**
**  MODIFICATION HISTORY:
**
**--
*/

/*
**
**  INCLUDE FILES
**
*/

#include <math.h>
#include <stdio.h>
double alpha (double, double, double);
double sigma_t (double, double, double);

main ()
{
    printf ("P T S"); scanf("%f %f %f",&p,&t,&s);
    printf ("alpha %8e sigma_t %8e\n",alpha(p,t,s),sigma_t(p,t,s));
}
```

C-6

```
/*
**++
**  FUNCTIONAL DESCRIPTION:
**
**      alpha
**
**      Reprogrammed from a FORTRAN routine by the
**      Institute for Marine Research, Kiel and
**      Alfred-Wegener-Institute, Bremerhaven
**
**      Equation of state for seawater proposed by JPOTS 1980
**      UNITS:
**          pressure    P       bars
**          temperature T       deg Celsius (IPTS-68)
**          salinity    S       NSU (IPSS-78)
**          density     RHO     kg/M^3
**          spec. Vol.  ALPHA   m^3/kg
**
**    Check value:
**          ALPHA   =   9.435561E-4 M^3/kg
**
**          For
**              S = 40 NSU
**              T = 40 deg c
**              P = 1000 bars
**
**      Testet by this routine:
**      ALPHA   =   9.435561e-4 m^3/kg
**
**  FORMAL PARAMETERS:
**
**      double  pressure
**      double  salinity
**      double  temperature
**
**  IMPLICIT INPUTS:
**
**      none
**
**  IMPLICIT OUTPUTS:
**
**      none
**
**  FUNCTION VALUE:
**
**      double  alpha
**
**  SIDE EFFECTS:
**
**      none
**
**--
*/
double alpha ( double pressure,
               double temperature,
               double salinity)
{
    double A = 0,
           B = 0,
           C = 0,
           D = 1.91075e-4,
           E = 0,
           A1 = 0,
           B1 = 0,
           AW = 0,
```

```
            BW = 0,
            K = 0,
            KO = 0,
            KW = 0,
            RHO = 0,
            SR = 0,
            R1 = 0,
            R2 = 0,
            R3 = 0,
            R4 = 4.8314e-4,
            retValue = 0;

    SR = sqrt(fabs(salinity));
    /* pure water density at atm pressure */
    R1 = ((((6.536332e-9 * temperature - 1.120083e-6) * temperature
        + 1.001685e-4) * temperature - 9.095290e-3) * temperature
        + 6.793952e-2) * temperature + 999.842594;
    /* seawater density at atm pressure */
    R2 = (((5.3875e-9 * temperature - 8.2467e-7) * temperature + 7.6438e-5)
        * temperature - 4.0899e-3) * temperature + 8.24493e-1;
    R3 = (-1.6546e-6 * temperature + 1.0227e-4) * temperature - 5.72466e-3;
    RHO = (R4 * salinity + R3 * SR + R2) * salinity + R1;
    /* specific volume at atm pressure */
    retValue = 1.0 / RHO;
    if (pressure != 0)
    {
        /* compute secant bulk modulus k(p,t,s) */
        E = (9.1697e-10 * temperature + 2.0816e-8) * temperature - 9.9348e-7;
        BW = (5.2787e-8 * temperature - 6.12293e-6) * temperature + 8.50935e-5;
        B = BW + E * salinity;
        C = (-1.6078e-6 * temperature - 1.0981e-5) * temperature + 2.2838e-3;
        AW = ((-5.77905e-7 * temperature + 1.16092e-4) * temperature
            + 1.43713e-3) * temperature + 3.239908;
        A = (D * SR + C) * salinity + AW;
        B1 = (-5.3009e-4 * temperature + 1.6483e-2) * temperature + 7.944e-2;
        A1 = ((-6.1670e-5 * temperature + 1.09987e-2) * temperature
            - 0.603459) * temperature + 54.6746;
        KW = (((-5.155288e-5 * temperature + 1.360477e-2) * temperature
            - 2.327105) * temperature + 148.4206) * temperature + 19652.21;
        /* compute k(0,t,s) */
        KO = (B1 * SR + A1) * salinity + KW;
        /* evaluate k(p,t,s) */
        K = (B * pressure + A) * pressure + KO;
        retValue = retValue * (1.0 - pressure / K);
    }
    return (retValue);
}
```

```
/*
**++
**  FUNCTIONAL DESCRIPTION:
**
**      sigma_t
**
**      needs function alpha
**
**  FORMAL PARAMETERS:
**
**      double  pressure
**      double  salinity
**      double  temperature
**
**  IMPLICIT INPUTS:
**
**      none
**
**  IMPLICIT OUTPUTS:
**
**      none
**
**  FUNCTION VALUE:
**
**      double  sigma_t
**
**  SIDE EFFECTS:
**
**      none
**
**--
*/
double sigma_t ( double pressure,
            double temperature,
            double salinity)
{
    return (1 / alpha (pressure, temperature, salinity));
}
```

Gordon_Cruise.C

```
/*
**++
**  FACILITY:
**
**      gordon_cruise.c
**
**  ABSTRACT:
**
**      The special variables SHIP, CRUISE, and BACKFILL are used to
**      to control the data flow. The following conditions are valid:
**
**     SHIP   CRUISE  BACKFILL   Definition
**     false  false    false     Read data from HEADERS.FIL compare to values
**                                  in Gordon_Station table.
**     false   true    false     Read data from SHIPGORD3.DAT and prepare
**                                  for Gordon_Cruise table.
**     true   false    false     Read data from SHIPGORD3.DAT and prepare
**                                  for Gordon_Ship table.
**     true    true    false     illegal. Do nothing.
**      x       x       true     Load data from HEADERS.FIL into the table
**                                  Gordon_Station_Backfill
**
**      Some other variables are used to document testing and loading.
**
**     TEST  TODB                Definition
**     true   x                  Test reading
**      x    true                Transfer to the database
**
**  AUTHORS:
**
**      Lutz-Peter Kurdelski
**      Alfred-Wegener-Institute for Polar and Marine Research
**      Am Handelshafen 12
**      D-2850 Bremerhaven
**
**  CREATION DATE:
**
**      1990-11-23
**
**  MODIFICATION HISTORY
**
**      1991-02-01  minor changes
**      1991-08-19  switch from compiler/ preprocessor option to
**                  variable controlled program execution
**
**--
*/

/*
**
**  INCLUDE FILES
**
*/

#include <stdio.h>
#include "opendb.h"
#include curses
#include <unixlib.h>
#include math

/*
**
**  MACRO DEFINITIONS
**
*/

typedef struct cruise_data {
    char    name [9];
```

```
    int     cruiseNumber,
            dummy;
    float   latitude,
            longitude;
    char    date [9];
    int     dayOfYear;
    char    time [6],
            comment [101];
    };

typedef struct gordon_cruise_data {
     int  cruiseNumber,
          shipId;
     char cruiseName [6],
          comment [100];
 };

typedef struct gordon_ship {
     int  shipId;
     char shipName [30],
          country [30];
 };

typedef struct ship {
    int  shipId;
    char shipcode [6],
         shipName [30];
    int  cruiseNumber;
    char country [30];
} ;

typedef char str [4];
typedef str strar [12];

char *strsub ( char * source, int start, int stop );

BOOL BACKFILL = FALSE;
BOOL CRUISE = FALSE;
BOOL SHIP = FALSE;
BOOL TEST = FALSE;
BOOL TODB = FALSE;
int result = 0;

static strar ar = {"JAN", "FEB", "MAR", "APR", "MAY", "JUN",
                   "JUL", "AUG", "SEP", "OCT", "NOV", "DEC"};
```

```
/*
**++
**  FUNCTIONAL DESCRIPTION:
**
**      main
**
**      See abstract of the module.
**
**  FORMAL PARAMETERS:
**
**      none
**
**  IMPLICIT INPUTS:
**
**      none
**
**  IMPLICIT OUTPUTS:
**
**      none
**
**  COMPLETION CODES:
**
**      none
**
**  SIDE EFFECTS:
**
**      none
**
**--
*/
main (int argc, char *argv[])
MAIN_PROGRAM
{
    const char * fileNameFil = "HEADERS.FIL",
               * fileShipName = "SHIPGORD3.DAT";

    extern int err_handler ();
    extern int msg_handler ();

    char c;

    int * shipId,
        * gordonCruiseNumber;

    DBPROCESS * dbproc;

    WINDOW * win, * win1;

    struct cruise_data cruiseData;
    struct gordon_cruise_data gordonCruisedata;
    struct gordon_ship gordonShip;

    gordonCruiseNumber = (int *) malloc(sizeof(int));
    *gordonCruiseNumber = 0;
    shipId = (int *) malloc(sizeof(int));
    *shipId = 0;

    if (TODB)
    {
        dberrhandle(err_handler);
        dbmsghandle(msg_handler);

        if ((dbproc = opendb ("SouthernOceanDB", "sa", NULL)) == NULL)
        {
            getchar();
            endwin();
/*
```

```
** Because the database system abortsin this case the following lines are
** never reached.
*/
            printf("\nThere is something wrong with the database. Abort!\n");
            exit(1);
        }
    }

    initscr();
    win = subwin(stdscr,12,40,5,2);
    win1 = subwin(stdscr,3,40,20,1);

    if (!BACKFILL)
    {
        if (!SHIP && !CRUISE)
        {
            mvwaddstr(win1,1,1,"open file G ");
            mvwaddstr(win1,1,14,fileNameFil);
            wrefresh(win1);

            if (TODB)
            {
                connectToDB (dbproc,
                        "select max(Ship_Id#) from Gordon_Ship",
                        shipId,
                        INTBIND,
                    0);
            }

            writeCruiseDataToDB (dbproc, fileNameFil, shipId);

        }

        if (!CRUISE && SHIP)
        {
            mvwaddstr(win1,1,1,"open file S ");
            mvwaddstr(win1,1,14,fileShipName);
            wrefresh(win1);

            if (TODB)
            {
                connectToDB (dbproc,
                        "select max(Gordon_Cruise_Id#) from Gordon_Cruise",
                        gordonCruiseNumber,
                        INTBIND,
                    0);
            }

            writeShipDataToDB (dbproc, fileShipName, gordonCruiseNumber);

        }

        if (CRUISE && !SHIP)
        {
            mvwaddstr(win1,1,1,"open file C ");
            mvwaddstr(win1,1,14,fileShipName);
            wrefresh(win1);

            if (TODB)
            {
                connectToDB (dbproc,
                        "select max(Gordon_Cruise_Id#) from Gordon_Cruise",
                        gordonCruiseNumber,
                        INTBIND,
                    0);
            }

            writeShipAndCruiseDataToDB (dbproc,
```

```
                                        fileShipName,
                                        gordonCruiseNumber);
        }
    }
    else
    {
        connectToDB (dbproc,
                "select max(Gordon_Station_Id#) from Gordon_Station_Backfill",
                gordonCruiseNumber,
                INTBIND,
                0);

        writeBackfillDataToDB (dbproc,
                               fileNameFil,
                               gordonCruiseNumber);
    }

    endwin();
}
```

```
/*
**++
**  FUNCTIONAL DESCRIPTION:
**
**      writeBackfillDataToDB
**
**  FORMAL PARAMETERS:
**
**      DBPROCESS * dbproc  Datenbankreferenz
**      char * filename     Eingabefile
**      int * id
**
**  IMPLICIT INPUTS:
**
**      TEST
**      TODB
**      BACKFILL
**      SHIP
**      CRUISE
**
**  IMPLICIT OUTPUTS:
**
**      none
**
**  COMPLETION CODES:
**
**      none
**
**  SIDE EFFECTS:
**
**      none
**
**--
*/
 writeBackfillDataToDB (DBPROCESS * dbproc,
                        char * filename,
                        int * shipId)
{
    struct  cruise_data cruiseData;

    char datetime[22],
        dummy [22],
        sample [3],
        * datetimePnt,
        * datePnt;

    int * gordonCruiseId,
        * gordonStationId,
        month,
        day,
        year,
        hour,
        * depth,
        * sign,
        stationIndex = 0,
        count = 0;

    FILE * fp, * fpo;

    WINDOW * win;

    const * fmt = "%5s %i %i %f %f%*c%8s %i%*c%5s%*2c%11s ";
```

```
fp = fopen(filename, "ra");
fpo = fopen ("gcheader.log", "w");

win = subwin(stdscr,13,40,5,2);

box(win,'|','-');

depth = (int *) malloc(sizeof(int));
*depth = 4;
sign = (int *) malloc(sizeof(int));
*sign = 0;
gordonCruiseId = (int *) malloc(sizeof(int));
*gordonCruiseId = 0;
gordonStationId = (int *) malloc(sizeof(int));
*gordonStationId = 0;

while (fscanf (fp,
                fmt,
                cruiseData.name,
                &cruiseData.cruiseNumber,
                &cruiseData.dummy,
                &cruiseData.latitude,
                &cruiseData.longitude,
                cruiseData.date,
                &cruiseData.dayOfYear,
                cruiseData.time,
                cruiseData.comment) != EOF)
{
    strcpy(datetime,"       19     :  AM\0");
    datetimePnt = datetime + 9;
    strncpy(datetimePnt, strtok(cruiseData.date,"/"), 2);
    datetimePnt = datetime;
    strncpy(datetimePnt, ar[atoi(strtok(NULL,"/")) - 1], 3);
    datetimePnt = datetime + 4;
    strncpy(datetimePnt, strtok(NULL,"/"), 2);
    datetimePnt = datetime + 12;
    strncpy(datetimePnt, cruiseData.time, 5);
    hour = atoi(strtok(cruiseData.time, ":"));
    if ( hour > 12 )
    {
        hour = hour - 12;
        datetime [17] = 'P';
        *depth = 2;
        *sign = 1;
        sample [0] = '\0';
        if (hour >= 10)
        {
            strcat (sample, fcvt((double) hour, 0, depth, sign));
        }
        else
        {
            strcat (strcat(sample, "0"),
                    fcvt((double) hour, 0, depth,sign));
        }
        strncpy(datetimePnt, sample, 2);
    }

    if (TODB)
    {
        dbcmd (dbproc, " insert into Gordon_Station_Backfill values (");
        dbfcmd (dbproc, "  %d,", cruiseData.cruiseNumber);
        dbfcmd (dbproc, " '%s',", cruiseData.name);
        dbfcmd (dbproc, " %f,", cruiseData.longitude);
        dbfcmd (dbproc, " %f,", cruiseData.latitude);
        if (dbconvert(dbproc, SYBCHAR, datetime, strlen(datetime),
                    SYBDATETIME, dummy, strlen(datetime)) != -1)
        {
```

```
            dbfcmd (dbproc, " '%s',", datetime);
        }
        else
        {
            dbfcmd (dbproc, " '%s',", "");
            fprintf (fpo, "%d \t%s \t %f \t%f \t%d\n",
                    cruiseData.cruiseNumber,
                    cruiseData.name,
                    cruiseData.longitude,
                    cruiseData.latitude,
                    datetime,
                    cruiseData.dayOfYear);
            wrefresh(win);
        }
        dbfcmd (dbproc, " %d)", cruiseData.dayOfYear);
        if (dbsqlexec (dbproc) == FAIL)
        {
            getch();
            endwin();
            printf("\nsqlexec failed\n");
            exit(2);
        }
        if ((result = dbresults(dbproc)) == FAIL)
        {
            getch();
            endwin();
            printf("\nresults failed\n");
            exit(2);
        }
    }

    ++stationIndex;
    *sign = 1;
    fcvt((double)(cruiseData.cruiseNumber),0,depth,sign);
    mvwaddstr(win,2,1,fcvt((double)(cruiseData.cruiseNumber),0,depth,sign));

    if (TEST || (++count > 100))
    {
        mvwaddstr(win,1,1,cruiseData.name);
        *sign = cruiseData.dummy >=0 ? 1 : -1;
        mvwaddstr(win,3,1,fcvt((double)cruiseData.dummy,0,depth,sign));
        *sign = cruiseData.latitude >=0 ? 1 : -1;
        *depth = fabs(cruiseData.latitude) >= 100 ? 3 : 2;
        mvwaddstr(win,4,1,fcvt(cruiseData.latitude,4,depth,sign));
        *sign = cruiseData.longitude >=0 ? 1 : -1;
        *depth = fabs(cruiseData.longitude) >= 100 ? 3 : 2;
        mvwaddstr(win,5,1,fcvt(cruiseData.longitude,4,depth,sign));
        mvwaddstr(win,6,1,cruiseData.date);
        *sign = 1;
        mvwaddstr(win,7,1,fcvt((double)cruiseData.dayOfYear,0,depth,sign));
        mvwaddstr(win,8,1,cruiseData.time);
        mvwaddstr(win,9,1,cruiseData.comment);
        *sign = 1;
        mvwaddstr(win,10,1,fcvt((double)stationIndex,0,depth,sign));
        mvwaddstr(win,11,1,datetime);
    }
    wrefresh(win);
}

fclose(fpo);
fclose(fp);
}
```

```
/*
**++
**  FUNCTIONAL DESCRIPTION:
**
**      writeCruiseDataToDB
**
**      Make a selection on Gordon_Station and Gordon_Station.
**      Than compare the station due to an interval of [-0.05,0.05] degrees.
**      Print out stations which are not within this intervals.
**
**  FORMAL PARAMETERS:
**
**      DBPROCESS * dbproc  Datenbankreferenz
**      char * filename     Eingabefile
**      int * id
**
**  IMPLICIT INPUTS:
**
**      TEST
**      TODB
**      SHIP
**      CRUISE
**      BACKFILL
**
**  IMPLICIT OUTPUTS:
**
**      none
**
**  COMPLETION CODES:
**
**      none
**
**  SIDE EFFECTS:
**
**      none
**
**--
*/
 writeCruiseDataToDB (DBPROCESS * dbproc,
                      char * filename,
                      int * shipId)
{
    struct  cruise_data cruiseData;

    char datetime[22],
        sample [3],
        * datetimePnt,
        * datePnt;

    int * gordonCruiseId,
        * gordonStationId,
        month,
        day,
        year,
        hour,
        * depth,
        * sign,
        stationIndex = 0,
        count = 0;


    FILE * fp, * fpo;

    WINDOW * win;

    const * fmt = "%5s %i %i %f %f%*c%8s %i%*c%5s%*2c%11s ";
```

```
fp = fopen(filename, "ra");
fpo = fopen ("gcheader.log", "w");

win = subwin(stdscr,13,40,5,2);

box(win,'|','-');

depth = (int *) malloc(sizeof(int));
*depth = 4;
sign = (int *) malloc(sizeof(int));
*sign = 0;
gordonCruiseId = (int *) malloc(sizeof(int));
*gordonCruiseId = 0;
gordonStationId = (int *) malloc(sizeof(int));
*gordonStationId = 0;

while ((fscanf (fp,
                fmt,
                cruiseData.name,
                &cruiseData.cruiseNumber,
                &cruiseData.dummy,
                &cruiseData.latitude,
                &cruiseData.longitude,
                cruiseData.date,
                &cruiseData.dayOfYear,
                cruiseData.time,
                cruiseData.comment) != EOF)
        &&
         (cruiseData.cruiseNumber < 100))
{
    strcpy(datetime,"       19     :  AM\0");
    datetimePnt = datetime + 9;
    strncpy(datetimePnt, strtok(cruiseData.date,"/"), 2);
    datetimePnt = datetime;
    strncpy(datetimePnt, ar[atoi(strtok(NULL,"/")) - 1], 3);
    datetimePnt = datetime + 4;
    strncpy(datetimePnt, strtok(NULL,"/"), 2);
    datetimePnt = datetime + 12;
    strncpy(datetimePnt, cruiseData.time, 5);
    hour = atoi(strtok(cruiseData.time, ":"));
    if ( hour > 12 )
    {
        hour = hour - 12;
        datetime [17] = 'P';
        *depth = 2;
        *sign = 1;
        sample [0] = '\0';
        if (hour >= 10)
        {
            strcat (sample, fcvt((double) hour, 0, depth, sign));
        }
        else
        {
            strcat (strcat(sample, "0"),
                    fcvt((double) hour, 0, depth,sign));
        }
        strncpy(datetimePnt, sample, 2);
    }

    if (TODB)
    {
        dbcmd (dbproc, "select Gordon_Station_Id# from Gordon_Station ");
        dbfcmd (dbproc, " where Latitude between %d and %d ",
            cruiseData.latitude - 0.005, cruiseData.latitude + 0.005);
        dbfcmd (dbproc, "  and Longitude between %d and %d ",
            cruiseData.longitude - 0.005, cruiseData.longitude + 0.005);
        dbfcmd (dbproc, " and Date_Time = '%s'",strsub(datetime,0,10));
```

```
            if (dbsqlexec (dbproc) == FAIL)
            {
                getch();
                endwin();
                printf("\nsqlexec failed\n");
                closedb(dbproc);
                exit(2);
            }
            if ((result = dbresults(dbproc)) == FAIL)
            {
                getch();
                endwin();
                printf("\nresults failed\n");
                closedb(dbproc);
                exit(2);
            }
            dbbind (dbproc, 1, INTBIND, 0, gordonStationId);
            if (dbnextrow (dbproc) != NO_MORE_ROWS)
            {
                fprintf(fpo, "Nr: %d \t%s \t%d \t%s\n",
                    *gordonStationId,
                    cruiseData.name,
                    cruiseData.cruiseNumber,
                    strsub(datetime,0,10));
            }
        }

        ++stationIndex;
        if (TEST || (++count > 100))
        {
            mvwaddstr(win,1,1,cruiseData.name);
            *sign = 1;
            fcvt((double)(cruiseData.cruiseNumber),0,depth,sign);
            mvwaddstr(win,2,1,fcvt((double)(cruiseData.cruiseNumber),0,depth,sig
n));
            *sign = cruiseData.dummy >=0 ? 1 : -1;
            mvwaddstr(win,3,1,fcvt((double)cruiseData.dummy,0,depth,sign));
            *sign = cruiseData.latitude >=0 ? 1 : -1;
            *depth = fabs(cruiseData.latitude) >= 100 ? 3 : 2;
            mvwaddstr(win,4,1,fcvt(cruiseData.latitude,4,depth,sign));
            *sign = cruiseData.longitude >=0 ? 1 : -1;
            *depth = fabs(cruiseData.longitude) >= 100 ? 3 : 2;
            mvwaddstr(win,5,1,fcvt(cruiseData.longitude,4,depth,sign));
            mvwaddstr(win,6,1,cruiseData.date);
            *sign = 1;
            mvwaddstr(win,7,1,fcvt((double)cruiseData.dayOfYear,0,depth,sign));
            mvwaddstr(win,8,1,cruiseData.time);
            mvwaddstr(win,9,1,cruiseData.comment);
            *sign = 1;
            mvwaddstr(win,10,1,fcvt((double)stationIndex,0,depth,sign));
            mvwaddstr(win,11,1,datetime);
            wrefresh(win);
        }
        dbcanquery (dbproc);
    }

    fclose(fp);
}
```

```
/*
**++
**  FUNCTIONAL DESCRIPTION:
**
**      writeShipDataToDB
**
**  FORMAL PARAMETERS:
**
**      DBPROCESS * dbproc
**      char * filename
**      int * id
**
**  IMPLICIT INPUTS:
**
**      none
**
**  IMPLICIT OUTPUTS:
**
**      none
**
**  COMPLETION CODES:
**
**      none
**
**  SIDE EFFECTS:
**
**      none
**
**--
*/
 writeShipDataToDB (DBPROCESS * dbproc,
                    char * filename,
                    int * gordonId)
{
    struct ship gordonShip;

    FILE * fp;

    WINDOW * win;

    const * fmt = "%d%*2c%5s%*2c%25c %d%*3c%12c ";

    int * sign,
        * depth,
        count = 0;
```

```
    depth = (int *) malloc(sizeof(int));
    *depth = 4;
    sign = (int *) malloc(sizeof(int));
    *sign = 0;

    fp = fopen(filename, "ra");

    win = subwin(stdscr,12,31,5,2);

    box(win,'|','-');

    while (fscanf (fp,
                   fmt,
                   &gordonShip.shipId,
                   gordonShip.shipcode,
                   gordonShip.shipName,
                   &gordonShip.cruiseNumber,
                   gordonShip.country) != EOF)
    {
        gordonShip.shipcode[5] = '\0';
        gordonShip.shipName[25] = '\0';
        gordonShip.country[12] = '\0';

        if (TODB)
        {
            dbcmd (dbproc, " insert into Gordon_Ship values ( ");
            dbfcmd (dbproc, "  %d,", gordonShip.shipId);
            dbfcmd (dbproc, " '%s',", gordonShip.shipcode);
            dbfcmd (dbproc, " '%s',", gordonShip.shipName);
            dbfcmd (dbproc, " '%s')", gordonShip.country);
            if (dbsqlexec (dbproc) == FAIL)
            {
                getch();
                endwin();
                printf("\nsqlexec failed\n");
                exit(2);
            }
            if ((result = dbresults(dbproc)) == FAIL)
            {
                getch();
                endwin();
                printf("\nresults failed\n");
                exit(2);
            }
        }

        if (TEST || (++count > 10))
        {
            *sign = 1;
            mvwaddstr(win,1,1,fcvt((double)gordonShip.shipId,0,depth,sign));
            mvwaddstr(win,2,1,gordonShip.shipcode);
            mvwaddstr(win,3,1,gordonShip.shipName);
            *sign = 1;
            mvwaddstr(win,4,1,
                        fcvt((double)gordonShip.cruiseNumber,0,depth,sign));
            mvwaddstr(win,5,1,gordonShip.country);
            wrefresh(win);
            count = 0;
        }
    }
}
```

```
/*
**++
**  FUNCTIONAL DESCRIPTION:
**
**      writeShipAndCruiseDataToDB
**
**  FORMAL PARAMETERS:
**
**      DBPROCESS * dbproc  Referenz zur Datenbank
**      char * filename
**      int * id
**
**  IMPLICIT INPUTS:
**
**      none
**
**  IMPLICIT OUTPUTS:
**
**      none
**
**  COMPLETION CODES:
**
**      none
**
**  SIDE EFFECTS:
**
**      none
**
**--
*/
 writeShipAndCruiseDataToDB (DBPROCESS * dbproc,
                    char * filename,
                    int * gordonId)
{
    struct ship gordonShip;

    FILE * fp;

    WINDOW * win;

    const * fmt = "%d%*2c%5s%*2c%25c %d%*3c%12c ";

    int * sign,
        * depth,
        count = 0;
```

```
    depth = (int *) malloc(sizeof(int));
    *depth = 4;
    sign = (int *) malloc(sizeof(int));
    *sign = 0;

    fp = fopen(filename, "ra");

    win = subwin(stdscr,12,31,5,2);

    box(win,'|','-');

    while (fscanf (fp,
                   fmt,
                   &gordonShip.shipId,
                   gordonShip.shipcode,
                   gordonShip.shipName,
                   &gordonShip.cruiseNumber,
                   gordonShip.country) != EOF)
    {
        gordonShip.shipcode[5] = '\0';
        gordonShip.shipName[25] = '\0';
        gordonShip.country[12] = '\0';

        if (TODB)
        {
            dbcmd (dbproc, " insert into Gordon_Cruise (");
            dbfcmd (dbproc, " Gordon_Cruise_Id#, Ship_Code) ");
            dbfcmd (dbproc, " values ( ");
            dbfcmd (dbproc, "  %d,", gordonShip.cruiseNumber);
            dbfcmd (dbproc, " '%s')", gordonShip.shipcode);
            if (dbsqlexec (dbproc) == FAIL)
            {
                getch();
                endwin();
                printf("\nsqlexec failed\n");
                exit(2);
            }
            if ((result = dbresults(dbproc)) == FAIL)
            {
                getch();
                endwin();
                printf("\nresults failed\n");
                exit(2);
            }
        }

        if (TEST || (++count > 10))
        {
            *sign = 1;
            mvwaddstr(win,1,1,fcvt((double)gordonShip.shipId,0,depth,sign));
            mvwaddstr(win,2,1,gordonShip.shipcode);
            mvwaddstr(win,3,1,gordonShip.shipName);
            *sign = 1;
            mvwaddstr(win,4,1,
                      fcvt((double)gordonShip.cruiseNumber,0,depth,sign));
            mvwaddstr(win,5,1,gordonShip.country);
            wrefresh(win);
            count = 0;
        }
    }
}
```